V.E.D.I.C.

Virtual Environment for Developing Interactive Code

Group #13 – Spring 2016

Developers

William Funk
Jorge Rodriguez
James Vinson

Sponsor

Dr. Richard Leinecker
# Table of Contents

Executive Summary ............................................................................................................ 1
Executive Summary ............................................................................................................ 1
Project Significance .......................................................................................................... 3
Motivations ........................................................................................................................ 5
  William Funk .................................................................................................................. 5
  Jorge Rodriguez ............................................................................................................. 5
  James Vinson .................................................................................................................. 5
Broader Impacts ................................................................................................................ 7
Competing Technologies ................................................................................................... 13
Required Technologies ..................................................................................................... 15
Technical Objectives ....................................................................................................... 16
  Objective #1: Big Data Visibility .................................................................................. 16
  Objective #2: Old Functionality Retained .................................................................... 17
  Objective #3: Object-Oriented With Objects ............................................................... 17
  Objective #4: Full Manipulation ................................................................................... 18
  Objective #5: Functional Puzzles .................................................................................. 18
Modes of Operation .......................................................................................................... 20
  Database View Mode ..................................................................................................... 20
  Database Edit Mode ...................................................................................................... 21
  Programming View Mode .............................................................................................. 23
  Programming Edit Mode ............................................................................................... 23
Technical Requirements ................................................................................................. 25
  Must Haves .................................................................................................................... 25
  Would Like to Have ....................................................................................................... 25
Inspirations Behind VEDIC's Design ............................................................................... 27
  Take away from Unreal: (Structuring system) ............................................................... 27
  Take away from LYRA VR: (Object Interaction) .......................................................... 28
User Experience Design in Virtual Reality ................................................................. 31
  Content Degree Ranges ............................................................................................... 31
  Downward Content Zones ............................................................................................ 32
  Follow Content Ranges ............................................................................................... 33
  Limitations of Leap Motion Controller ......................................................................... 34
  UX: From Physical to Psychological ............................................................................ 35
  Use Constant Head-Tracking ....................................................................................... 36
  User Movement Control ............................................................................................... 36
  User Can't Experience Velocity .................................................................................... 36
  Grounding the User ....................................................................................................... 37
  Moving Objects (without user interaction) ................................................................... 38
  Text Guidelines ............................................................................................................ 38
  Use of Directional Sounds ............................................................................................ 39
  Universal Usability (Avoiding the "Phantom Leg") ....................................................... 39
  Distinguishing Interactions (Static V.S. Dynamic Objects) ........................................ 40
  Ergonomics of Human Interaction .............................................................................. 40
  Use Lighting to Guide Users ....................................................................................... 41
Design Phase.................................................................................................84
Implementation Phase - Database.................................................................86
Implementation Phase - Programming.........................................................89
Sources..........................................................................................................93
Sources w/ URL.............................................................................................94
Executive Summary

VEDIC is a programming development platform implemented in virtual reality. Its primary purpose is to visualize data, and organize code in 3D space. Unlike most virtual reality products, which consist of games or user experiences, VEDIC is primarily a development tool for VR.

This tool's power manifests itself into two areas of the information technology industry: the visualization of and interaction with databases, and object-oriented programming. Its singular goal is to leverage the power and depth of VR technology into areas ordinarily limited by a two-dimensional text plane. Where tabs of data once obscured one another to the viewer's eye, it is now possible to offset them in a larger, almost infinite landscape. The possibilities, to put it plainly, are unlimited.

Ordinarily when dealing with database information, an information technician is limited to viewing one or two tables at a time, displayed in row after row of data that is often lost to the human brain's desire to shut out repetition. The deeper connection between multiple tables, and the scope of those underlying threads are often overlooked unless hours of mapping and cataloging can be performed.

Robust design is not always in the forefront of the mind for those practicing this craft. Some databases accumulate tables that number in the dozens and even hundreds despite best practices. Instead of redesigning or shifting data as the need arises, database technicians will often patch the already existing content with a new column, or table. Sometimes joins are made for temporary purposes, but left to rot in the tangled mess of overgrown projects.

Virtual Reality can disentangle that mess, and give the developer the perspective needed to better wade through that database content; perhaps even finding unintentional connections that would have otherwise never been found in a strictly 2D format.

Drifting away from databases and big data, there are areas of computer science that benefit directly from VEDIC. It is often difficult for a new programmer to understand the ideas and concepts behind object-oriented programming. Typically, the novice is trained in procedural languages first, like C, and are later introduced to OOP languages such as Java and C#. The notion that code can represent the physical objects of our three-dimensional world confuses many.

VEDIC creates an intuitive environment for the learning programmer to experiment and better understand how object-oriented programming works as a concept. Programs can be created, following the syntactical rules of an OOP language (C# in this version of VEDIC), transpiled into a cs file, and exported for external compiling.
It doesn't stop at education, either. VEDIC possess the capacity to create all but the most advanced C# programs, and may rival the competing Interactive Development Environments currently on the market.

The goals and objectives of the database portion of this project range from efficiency to education. It will represent SQL databases and allow C.R.U.D. (Create, Read, Update, Delete) operations to be applied to these databases in a visually intuitive way. It must identify and display connections between SQL tables and their contents in a way not easily obtainable with two dimensions.

In the area of programming, it will represent a series of C# files, classes, objects, etc. in a 3D virtual reality context that better portrays an object-oriented language. This 3D programming environment shall allow the user to create/manipulate the environment in the development of their code/systems.

A user must be able to assemble function declarations using a list of designated components much like a jigsaw puzzle, while still identifying dependencies to a class in a visual context (interfaces and abstract classes).

The technical environment for accomplishing these goals will be within the Unity 5 Engine for rendering virtual reality. C# will be the primary programming language used, and MySQL the standard SQL syntax.

VEDIC will not only use the user experience (UX) rules developed over decades with two-dimensional formats, but explore the variety of ideas and experiments currently taking place with the advent of affordable VR. In the end, this project will endeavor to create a new standard for UX as it navigates this unexplored frontier.
Project Significance

No longer will programmers or data technicians be restricted by two dimensions. The dawn of affordable virtual reality is here at last, giving us unlimited depth within which to play, develop, and explore. An entire world was discovered, and hardware like the Oculus Rift, HTC Vive, and Sony's PlayStation VR—to name only a few—are the vessels to take us there.

Now that we've set foot on those sandy shores, pioneers staring at verdant horizons, ripe with possibilities, there are necessities to consider. Where will we set up camp (virtual reality operating systems)? What resources will we seek out first (Leap motion, Myo band, traditional mouse and keyboard)? Through what lens will we peer into its depths (3D web browsers, immersive virtual worlds)?

VEDIC is a proof of concept, a tool to provide solid ground for student and professional alike. Its aim is to redesign the way the world looks at data; to bring object-oriented programming back to its roots with actual objects. The significance of this tool will be to provide a foothold in an otherwise uncertain domain.

Designers and developers are racing to be the ones who standardize how everyone views and interacts with this new space. Tools spring up and fall into obscurity. New designs revolutionize how humans can use sight to navigate, but in turn cause unforeseen side-effects (nausea and crossing eyes). Many of these attempts are mere theories yet to be put to the test.

The significance of VEDIC won't be to simply add another design to this growing market, but to create a place for all other designs to take root. Some of the already proposed user experience and user interface (UI) designs will be incorporated into this project, and built upon. Others will be of the sole creation of its developers.

In the image below is a piece of preliminary concept art unique to the look and feel of the user interface and use of depth and space to incorporate multiple pieces of different data, the tools a user has at their disposal, and the visually intuitive nature of the experience.
Concept Art - VEDIC Look and Feel [15]
Motivations

William Funk:

It was one of my earliest childhood memories where I had first been introduced to virtual reality 25 years ago. In the carnival games section at Busch Gardens, there was a featured attraction. It had the size of a small pool, a headset that threatened to topple me over from the sheer weight of it, and a hand-held blaster reminiscent of the science fiction movies of that era. I can still remember that 3D world of polygons, low resolution pterodactyls swooping in, and my cube projectiles spitting forth at a whopping 40 frames per second. For a moment, as a child, I had stepped into the future; an exciting future with endless possibilities. Now two-and-a-half decades later, I've finally caught up to that future. I can create the reality around me with the code from my fingertips. What's left? Someone or some group to make this technology more relevant in the here and now. This is why VEDIC is important. This is our contribution to that dream.

Jorge Rodriguez:

My initial motivation to pursuing this senior design project was to create a playground where people can come in and interact with a system that we provide on a social level. This eventually molded itself, with the help of my colleagues, into VEDIC. I love the idea of working with an engine like Unity, the inclusion of VR equipment such as the Oculus, and other tech to make something extraordinary. I have always been drawn to videogames, and working in Unity for this project was an obvious candidate for projecting that lifelong joy into a profession. Of course, this goes beyond just working in Unity, because the project is trying to utilize the potential of Unity as a "game-engine" to create a platform for developing code.

James Vinson:

I joined this team because I have confidence in each of my teammates' abilities, and I didn't want to provide free labor to some corporate entity only not to own it after I finished. If I contributed in the creation of something awesome, I want the right to be able to talk about it and the right to do whatever I want with it after senior design is done. Going back to the idea of creating something awesome, nothing compares to working with Virtual Reality. Virtual Reality is a hot topic currently in the tech industry. The reason, Virtual Reality brings new possibilities for developers that were once thought unobtainable. I want to work with Virtual Reality because I see a growing need for it in the future and none of the other projects that were pitched during senior design made use of the technology. This is the first thing that attracted me to the group, each member
wanted to work with Virtual Reality. The thing that is keeping my interest with this group is that we are building something in Virtual Reality that has the potential to advance its developmental potential. We are not building a game that uses Virtual Reality, we are building a development tool.
Broader Impacts

Creating applications for use inside virtual reality is an extensive topic that normally would convey certain stigmas as they are portrayed in the media. Usually the way that the entertainment industry envisions it is as a platform which creates disjunction within society. A society that participates in using this technology, normally illustrated in "futuristic" movies such as *Gamer (2009)* and to a lesser extent *Surrogate (2009)*, are conveyed as a typical dystopia where humans have become so accustomed to that technology, that traditional values of communication are always changed or broken.

Another piece of media in the form of manga called *Sword Art Online (2009)* later to be produced into anime in 2012, depicts another angle of the Virtual Reality topic by exaggerating technology, yet again, to illustrate how this type of thinking can be both harmful and beneficial to society. Of course, these are just works of fiction. What can be taken from these pieces of entertainment is helpful, however, when documenting what these devices can do for us as a species, good or bad.

As it stands now, news over virtual reality is not necessarily being broadcast all over national television. However, this has not stopped the conversation from partaking in the vastness of the internet along with professionals such as Computer Scientists or leading User Experience Designers. Part of the conversation is trying to understand what sort of role virtual reality can truly take as both a piece of entertainment and as a tool. In a show like *Sword Art Online*, the headsets used allow the user's mind to become the tool that allows them to manipulate a generated world space as virtual avatars. Albeit, in the show, this is not simply virtual reality, but contextually it implies similar situations that can be advantageous when applied to our own headsets. The show depicts use of this headset for patients who are unable to move their body properly. They are able to partially experience what it was once like to be able to move around, and connect with people on a more personal level than what they are currently able to do. Part of what virtual reality can do is bridge the gap between environments that were once flat into a world where things have weight and depth. This opens a ton of possibilities for very applicable pieces of technology, such as therapeutic treatment applications or simulating new kinds of learning environments.

What currently peaks our interest as a group is the effect that technologies like virtual reality can have on society as a whole. Not just for entertainment, but more as an application that can be used to increase efficiency in different departments. Now that environments can be generated in a 3D space that exists in a seemingly 3D environment with virtual reality, things can now be conceptualized quite differently. Virtual reality now has massive potential to develop a system that can provide a lot of beneficial
properties. Thus far, technology has generally led to adapting ideas and concepts into two dimensional flat rectangular spaces. Designing and finding intuitive ways to convey information has always had an expected outcome of being a two dimensional product that, generally speaking, had no real depth nor orientation to the user. There is always some form of disconnect. It's perfectly logical to think that 3D environments are already being conveyed in 2D spaces. And, they have been. For many years with cinematography being what it is and videogames simulating 3D worlds, applications have had a lot of practice in trying to visualize the environments of different applications for 2D spaces. With virtual reality, humans are once again bridging a gap of disconnect that exists between the plane of the monitor and the user. This is done by forcing those monitors to be in such close proximity to the user, that it can simulate the illusion of being in another world. This, along with motion control that these headsets come built with, allows the medium to have so much more to work with than what a standard monitor can provide.

As stated earlier, professionals in the fields related to Computer Science have come to acknowledge the potential that virtual reality can have as a tool. They are currently determining how applications can reach desired threshold outcomes of productivity. Humans live and understand everything in three dimensions. We are instinctively hardwired by nature to be able to interpolate information in a 3D environment as opposed to a 2D environment. Virtual reality can take advantage of this instinctive human intuition whereas a 2D environment has to extrapolate and determine clever ways of tapping into these human instincts with an already limiting mode of presentation. Leap Motion, a company that focuses on products that allow human limbs to be represented in a virtual space, calls this idea "Spatial Semantics" [1]. According to them, the spaces that virtual reality provides can allow "users to spatially arrange objects in order to make sense of data and its meaning, thereby revealing relationships and making connections" [1]. They also go on to say that "external memory" is an efficient way to recall remembrance on certain things. This means that when information is presented in a true 3D environment, it is more than likely to be easily recalled. VEDIC attempts to find a solution to presenting visuals in a 3D environment like virtual reality that are meaningful to the user. This can help serve as a platform for implementing other systems that promote user efficiency.

Having access to a 3D environment means taking advantage of the cognitive power of the human mind. Allocating meaning to a 3D space can tap into these instincts and provide a smoother way to aggregate data. Microsoft has developed two projects, Pivot and Sandance, that helps to illustrate this point. These images were presented by Leap Motion.
Now it can be shown with the two figures above that 3D space can play a strong role in providing factors for relationships. In the image representing alphabetical ordering and
time ordering, we can see that objects can exists in this 3D plane and allow for quick recognition to what is oldest, and what is the alphabetical ordering. The second image provides a similar construct, except it orders these objects by frequency and recency. Thinking of how you would order applications in terms of recency and frequency provides a view that allows us to quickly understand which objects were used the most, and most recently. Given that both Pivot and Sandance were developed for standard monitors, having an entire 3D space provided in VR can change this equation to become even more encompassing. Attempting to create 3D relationships with wider ranges of spaces to convey information is currently one of virtual reality's working topics. It would allow for more data to be presented to the user that can actually be digested. However, it's a matter of finding the most efficient and intuitive way of doing so. VEDIC would ultimately be joining this process and attempt to lay the ground work for others that are attempting to do handle data representation in virtual reality.

The impact that VEDIC can apply on the world of database and the world of programming is the increase of efficiency. Everything that has been discussed in this section thus far has already provided some evidence on how VR increases efficiency. In a presentation done by Jody Medich, a UX designer, studies show that when it comes to multitasking, users who are presented with "large monitors or multi-monitor environments" to show "huge productivity gains." Speaking strictly numbers, those same studies showed an increase of 40% in productivity [3]. She goes on to explain that a large reason is because users can separate tasks and events into different areas of their environment, allowing for an easier spatial way to split tasks. It is something that humans can handle conceptually with ease rather than having various tabs and applications be displayed on a taskbar that has to be sifted through in order to continue a certain task. Now this is not even taking into consideration what virtual reality can bring to the world of multitasking. If the studies show any conclusive evidence, it's that virtual reality can surely benefit the same efficiency that multi-monitor environments have, if not more. The reason being is because virtual reality encompasses you 360 degrees. You have been given the mega monitor. Cue theme music from 2001, A Space Odyssey.

For VEDIC, we are attempting to recreate the way we can understand and manipulate data and code. This has a lot of practical uses. VEDIC can be used as a system to help teach people who are new to the subject of databases and coding concepts. This can help by visualizing ideas more easily. This is not the first time that a simulation has been adopted for teaching concepts and technology to people who are relatively new to subjects. Back in 2014, Microsoft bought out Mojang's block world game, Minecraft, for a whopping $2.5 billion dollars [2]. Besides the obvious reason of capitalizing on their gains for their own benefit, Microsoft took the investment to a whole other level. Back in that same year, Microsoft incorporated a new system of Minecraft we call Minecraft: Education Edition. This effectively is taking the game of Minecraft, all of its resources
and scripting, and conforming it to be an environment where instructors can teach their students different lessons in school through the use of an interactive world. According to an article by CNN, Minecraft is currently being used to teach students "…coding, math, history, geography, and civil engineering" [2]. This even goes further back, where students would be taught by instructors the complexities of circuitry through Minecraft's internal electrical system of Redstone (wires) and switches. Given Minecraft's success in this department, it's not far-fetched that something that attempts to visualize databases and code can go a long way to teaching data relationships and object-oriented programming.

One of the impacts that we hope to take away from something like VEDIC is how our application can translate into the world of running an Operating System entirely in virtual reality. The work and effort we put into this system of manipulating data and code all leads to feeding the beast that is "how do we get an operating system in VR to work?" Currently, all the designers and programmers that are hammering away at figuring out how to get virtual reality to work are at the forefront. Once virtual reality gets to an acceptable state by consumers, it's only a matter of time before some company begins releasing versions of their OS systems that are fully integrated to work in virtual reality. This is looking far into the future, but VEDIC, for the team and for the world, is another ticket in the pot of features and systems that have a chance in changing the way VR development is handled. The world of virtual reality operating systems may even be closer than we think. Google is currently working on deploying an application that acts as a wrapper for their operating system. It allows navigation of their platform with the Samsung Gear VR Headset [4].

This project also lays foundation work for creating a system to handle big data appropriately. Although the project is rooted at getting our requirements of a database to come to life, there's no reason why a system that can visually represent data can't be expanded to interpret the data and generate visual representations of those results. Of course, this is a subject that will be discussed. To put it in conventional terms, though, VEDIC can be a system to run big data in order to come to quick conclusions about the data received.

For us, the impact that the project creates will be a new learning experience that will help us dip our feet in the pool that is virtual reality. Becoming invested in this process and learning from it as it starts to grow its legs is a worthy investment of our time. We hope that, from the project, we gain the necessary tools and skills that will allow us to help in the development of virtual reality.
Overtime, VEDIC, along with a long list of other applications being developed for VR, is laying the foundation for what new concepts and cues would make the most sense in this new set-up. I think it's important to note that plenty of the design decisions that make up an operating system, a web page, a videogame, anything technology related, are taken for granted. It's a system that had to be learned, and now has established the norm for what specific operations do on our monitors and on our systems. We know that when the discoloration of a tab is highlighted, that means that it is currently the application selected in our view. We know that the top right is meant for closing applications and that the top left provides a means to go back. These things were not intuitive at first. Someone had to come up with these designs that eventually the public adopted and they became widely accepted. Virtual reality takes no exception from this "growing pains" process. It's up to people like us working on VEDIC that will attempt to solve these questions and provide examples of what to do or maybe, even more importantly, what not to do. This will help people to learn from our mistakes, use the things that work, and help build off of the rapidly growing knowledge base of how to build applications for virtual reality.
Competing Technologies

VEDIC is one of a kind at what it strives to do. A fully immersive, 3D database visualization and interaction environment has yet to surface, and visual programming is more theoretical than applicable...until now. Some companies and their software have solved parts of the puzzle, however, and those attempts will be discussed here as they helped shape the direction of this project.

The first notable piece of software is Epic Games' Unreal Engine 4 VR programming environment. It 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 its 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.

The one advantage this software provides is a second point of user input. The second hand-held laser pointing device acts like a second mouse pointer, but with a better ambidextrous quality that a second mouse wouldn't offer. Angles on the red laser beams also one-up their mouse counterparts because the developer can get a better feel for the change of direction and rotation on the object they attempting to move.

No doubt this application will serve game development well. Its immersive 3D environment and interactive game elements allow the developer a more hands-on approach. This however, doesn't deal directly with database visualization, and the coding in programming is still done in the 2D text realm rather than a visual evolution.

Another competing product is Unity's Node Based Editor in Unity 3D. Here is an application that takes 2D UML components and strings them together with flexible wires. This in many ways is similar to the way VEDIC intends to create its 3D nodes with their various inputs and outputs stringing them together. Unlike VEDIC, however, these nodes are simply 2D planes with data printed on them. The 3D aspect is more aesthetic than advantage as depth isn't used in any more capacity than the positions of those planes in relation to one another.
There are many 2D UML tools available on the market that accomplish the same level of detail, but without the need for a 3D rendering engine. Unreal and Unity have used 2D nodes to visualize programming, but never in a 3D context. This Node Based Editor is not a further evolution on these, nor is it at the level VEDIC strives to achieve.

The final competitor is not software at all, but an idea. A Masters thesis titled, *A Framework for Design and Implementation of Visual Languages* by Ando Saabas. In this paper, the language of choice to visualize is Java—not a far stretch from C#. In order to compete with what VEDIC is trying to accomplish, the paper would have had to define specific syntax, graphical attributes, or at least some abstract syntax with grammar. The paper blatantly skips these in favor of a one-size-fits-all mentality, to provide a *guideline* for how one might design a visual language.

This thesis may yet inspire some creative choices when developing VEDIC’s visual syntax and graphical attributes, but as it stands, it was an attempt to theorize about the topic rather than develop anything applicable. Barring any new products along the way, VEDIC stands to be the only fully 3D programming and database environment available to date.
Required Technologies

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. Its 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. On the level of cost, XBOX/PS4 controllers are most accessible as all three project developers already possess one or more of these. In addition to that, the Leap Motion Controller can be obtained for as little as $30.

The Leap Motion Controller has the added benefit of allowing a greater level of interaction for the user as the hand is represented virtually in the environment, following every movement and inflection along the way. It is for these reasons that VEDIC will use the Leap for main use, and on occasion, the XBOX/PS4 controllers for development purposes.

Further down in this document, each of these chosen tools will be described in greater detail and the benefits of each device expounded upon. All chosen tools mentioned above have already been purchased by the team. Research and experimentation with each is ongoing, the results of which are promising.
Technical Objectives

Objective #1: Big Data Visibility

It is the number one mission for VEDIC to be able to present a large amount of data to the user in an easily digestible manner. In current forms, the database (SQL) is in the form of tables with a variable number of columns and any number of rows/records. To view more than one table at the same time with current implementations, a developer requires multiple screens, each to display a separate table. If the tables were small enough, the developer could fit more than one table on each screen by resizing the respective windows.

This is a cumbersome and often expensive process. Whether the issue is a limitation on funds or space, it is inefficient to require extra hardware to look at something as banal as a window of tabular data. An alternative is to not view the data simultaneously, but rather conceal lesser important tables in tabs on the same windows.

Multiple tabs may save on hardware, space, and cost, but it then detracts from the developer’s ability to view and absorb as much data as they can in order to form a better picture of what the overall database might be telling them.

In the context of programming, these same problems persist. Each file, class, or even function can take up an entire pane on the developer's screen. They can use one of the two methods described above, but they are again limited by hardware, space, funds, and the amount of the program they are able to visually process at one time.

Concept Art - Multiple Data Objects & Their Relationships [15]
Through shapes, colors, and connecting pipes/wires, VEDIC strives to remove the need to read in order to understand every aspect of the database or C# code. Its objective is to place all of the tables, columns, and relationships in plain view of the developer without the need for flipping from panel to panel, screen to screen, or tab to tab. Selecting an individual table will populate a panel with specific tabular data, or the touch of a particular node will highlight the function's conditional statement.

With a virtual environment, it's possible to create multiple panels to display more than one source of data in the same field of view. As a stretch goal to this objective, VEDIC strives to make available to the user, the ability to create new panels and populate each with a selected table or programming node depending on the mode of operation at that time.

**Objective #2: Old Functionality Retained**

Ensure all of the same functionality that any existing SQL database IDE has to offer. This is to include, but not be limited to, C.R.U.D. operations. It's imperative that the transition from two dimensions to a third does not lose any of the benefits afforded by the former. The mission is to expand on what is already present, not diminish it.

To accomplish this task, the user must be able to assemble any of the basic MySQL query statements and have them be executed in a timely manner to be displayed in both 2D and 3D. Ideally, a user should be able to expand upon this by using manual gestures connecting one or more tables, select columns or rows with a single touch, and swipe quickly through large amounts of data with relative ease.

To accomplish this, the typical MySQL keywords will be available to the user, and a complex graph theory algorithm can determine whether or not the query statement assembled is syntactically correct, or if adjustment needs to be made. In many ways, this error checking will be vastly superior to the typical 2D method as the string is broken into logical nodes in VEDIC, whereas most SQL IDE's simply allow a text string to be inputted and basic grammar to be checked.

**Objective #3: Object-Oriented With Objects**

In current development environments, a programmer is relegated to constructing conceptual objects using text in two-dimensions. The inputs and outputs of these objects, their methods and variables all obscured by an alphanumeric language that doesn't accurately represent them. It is the goal of VEDIC to revolutionize how object-oriented programming is visualized by using actual objects.

Using nodes (translucent cubes) to represent classes, objects, components, and methods, a user can see the object they are assembling. With designated input and output spheres located around the cube's six sides, it becomes apparent how many, and of what type of,
connections exist per component. With the additional use of color, and proper adherence to UX guidelines, further meaning can be derived from the scene constructed as dozens or more nodes are interconnected by wires/pipes to form a bigger picture.

The added benefit of visualizing these connections would be that class dependencies would then reveal themselves, negating the need to comb through hundreds and sometimes thousands of lines of code to discover those same clues. How many classes inherit from one specific parent class? How many are running a particular interface? Which are inner classes of others? These are the questions that seeing code in object form can hopefully answer.

As a stretch goal attached to this objective, it would be beneficial to permit a rudimentary runtime environment inside VEDIC. This would require compilation internally rather than externally, but could result in a new visual dimension. As the program runs, light could flow from one node to the next along the various pipelines connecting them. From this, the user could understand what their code is actually designed to do beyond their intention or expectation.

Objective #4: Full Manipulation

It isn't enough to simply see the objects floating about in space. The developer must have the capacity to fully manipulate the nodes and connections. VEDIC strives to provide the most advanced tools to accomplish this. Using the Leap Motion Controller and its Orion library, virtual hands can appear in the position of the developer's actual hands. It traces the movement of the arm, hand, and fingers to a realistic level of accuracy.

With this technology, VEDIC can use preexisting gestures, and ones of new design, to provide the user with the ability to manipulate their objects as though they existed within real space, minus haptic feedback. Hands will be able to hold nodes in their palm. Fingers can pinch and drag objects from one point to another. A forefinger can trace lines through space to literally draw connections from one component to the next.

No longer will it be necessary to separate the creator from the creation with unnatural devices like the mouse and keyboard. The hands and fingers humans take for granted can now be directly accessed for the purpose of programming.

Objective #5: Functional Puzzles

One of the more challenging aspects of visually programming is the construction of the methods themselves. Diverse in their seemingly infinite number of combinations, these functions can create dozens, if not hundreds of unique patterns. To successfully achieve a more intuitive way of displaying this process, VEDIC will use a sort of jigsaw puzzle of components. Each piece connects to the others to form a larger whole, much like a puzzle would.
Again the graph theory algorithm is necessary to ensure errors are outed the moment they're made, and make remedy required before continuing further. The final result will resemble a 3D circuit board with cube nodes receiving inputs through wires, and funneling their output off to other nodes through additional wires. Nodes can be deleted, wires severed, and new combinations constructed all in the same fluid process.
Modes of Operation

Database View Mode

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. Users 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 the scene (increase distances between objects), implode the scene (decrease distances between objects), and move through the scene in a flying-style of motion.

This mode allows the user to view all of their database information at once. Unlike its 2D counterparts, VEDIC demonstrates the size and scope of the data involved. In a panel along the user's peripheral view, an individual table's columns and rows can be seen in the typical manner to bridge the gap between developer habits born of older technologies and for easier reading as virtual reality still has yet to find a way to successfully display text in a 3D environment.

<table>
<thead>
<tr>
<th>Index</th>
<th>Company</th>
<th>PaidDate</th>
<th>Category</th>
<th>TaxLocal</th>
<th>TaxFederal</th>
<th>Country</th>
<th>Cost</th>
<th>DateEntered</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>KNIGHTSTOP</td>
<td>2016-01-13</td>
<td>medical</td>
<td>0.10</td>
<td>0.00</td>
<td>usa</td>
<td>1.59</td>
<td>2016-01-28 08:24:41</td>
</tr>
<tr>
<td>6</td>
<td>ORLANDO UTILITIES COMMISSION</td>
<td>2016-01-19</td>
<td>electricity</td>
<td>7.88</td>
<td>0.00</td>
<td>usa</td>
<td>66.07</td>
<td>2016-01-28 08:35:24</td>
</tr>
<tr>
<td>7</td>
<td>ORANGE COUNTRY UTILITIES DEPAR</td>
<td>2016-01-18</td>
<td>water</td>
<td>1.33</td>
<td>0.00</td>
<td>usa</td>
<td>37.45</td>
<td>2016-01-28 08:41:58</td>
</tr>
<tr>
<td>8</td>
<td>BRIGHTHOUSE</td>
<td>2016-01-18</td>
<td>internet</td>
<td>0.00</td>
<td>1.36</td>
<td>usa</td>
<td>81.38</td>
<td>2016-01-28 08:49:55</td>
</tr>
<tr>
<td>9</td>
<td>UWM</td>
<td>2016-01-15</td>
<td>mortgage</td>
<td>0.00</td>
<td>0.00</td>
<td>usa</td>
<td>1185.25</td>
<td>2016-01-28 08:47:09</td>
</tr>
<tr>
<td>10</td>
<td>HILLSBOROUGH COUNTY TAX COLLEC</td>
<td>2016-01-06</td>
<td>other</td>
<td>0.00</td>
<td>0.00</td>
<td>usa</td>
<td>92.20</td>
<td>2016-01-28 08:49:03</td>
</tr>
<tr>
<td>11</td>
<td>ROSS</td>
<td>2016-01-16</td>
<td>clothing</td>
<td>0.33</td>
<td>0.00</td>
<td>usa</td>
<td>5.32</td>
<td>2016-01-28 08:53:39</td>
</tr>
<tr>
<td>12</td>
<td>TJ MAXX</td>
<td>2016-01-16</td>
<td>clothing</td>
<td>0.07</td>
<td>0.00</td>
<td>usa</td>
<td>16.95</td>
<td>2016-01-28 08:54:09</td>
</tr>
<tr>
<td>13</td>
<td>SUNPASS</td>
<td>2016-01-03</td>
<td>transport</td>
<td>0.00</td>
<td>0.00</td>
<td>usa</td>
<td>20.00</td>
<td>2016-01-28 08:56:04</td>
</tr>
<tr>
<td>14</td>
<td>7-ELEVEN - FUEL</td>
<td>2016-01-19</td>
<td>transport</td>
<td>0.00</td>
<td>0.00</td>
<td>usa</td>
<td>42.00</td>
<td>2016-01-28 08:56:23</td>
</tr>
<tr>
<td>15</td>
<td>HUMANA</td>
<td>2016-01-05</td>
<td>healthInsurance</td>
<td>0.00</td>
<td>0.00</td>
<td>usa</td>
<td>99.60</td>
<td>2016-01-29 10:55:35</td>
</tr>
<tr>
<td>19</td>
<td>AT AND T</td>
<td>2016-01-03</td>
<td>phones</td>
<td>112.50</td>
<td>0.00</td>
<td>usa</td>
<td>112.00</td>
<td>2016-01-28 10:59:13</td>
</tr>
<tr>
<td>20</td>
<td>BEST BUY (PROTECTIVE COVER)</td>
<td>2016-01-04</td>
<td>officeSupplies</td>
<td>1.75</td>
<td>0.00</td>
<td>usa</td>
<td>26.74</td>
<td>2016-01-29 11:03:16</td>
</tr>
<tr>
<td>21</td>
<td>US POST OFFICE</td>
<td>2016-01-04</td>
<td>hair</td>
<td>0.00</td>
<td>0.00</td>
<td>usa</td>
<td>16.95</td>
<td>2016-01-29 11:05:50</td>
</tr>
<tr>
<td>22</td>
<td>SUPERCUTS</td>
<td>2016-01-09</td>
<td>hygiene</td>
<td>0.00</td>
<td>0.00</td>
<td>usa</td>
<td>16.00</td>
<td>2016-01-29 11:08:38</td>
</tr>
<tr>
<td>23</td>
<td>CVS</td>
<td>2016-01-14</td>
<td>hygiene</td>
<td>1.80</td>
<td>0.00</td>
<td>usa</td>
<td>29.35</td>
<td>2016-01-29 11:09:01</td>
</tr>
<tr>
<td>24</td>
<td>CVS</td>
<td>2016-01-14</td>
<td>hygiene</td>
<td>2.85</td>
<td>0.00</td>
<td>usa</td>
<td>47.11</td>
<td>2016-01-29 11:09:28</td>
</tr>
<tr>
<td>25</td>
<td>NATIONAL BIKE REGISTRATION</td>
<td>2016-01-06</td>
<td>entertainment</td>
<td>0.00</td>
<td>0.00</td>
<td>usa</td>
<td>10.00</td>
<td>2016-01-29 11:14:25</td>
</tr>
<tr>
<td>26</td>
<td>PUBLIC</td>
<td>2016-01-08</td>
<td>groceries</td>
<td>2.30</td>
<td>0.00</td>
<td>usa</td>
<td>102.66</td>
<td>2016-01-29 11:19:44</td>
</tr>
<tr>
<td>27</td>
<td>PUBLIC</td>
<td>2016-01-12</td>
<td>groceries</td>
<td>1.04</td>
<td>0.00</td>
<td>usa</td>
<td>67.64</td>
<td>2016-01-29 11:21:26</td>
</tr>
</tbody>
</table>

Typical MySQL Table - Example of 2D Panel Content
Database Edit Mode

The creation and/or manipulation of databases mode will have the user seated at a virtual work-bench. On a wall along the user's left peripheral, and within arms-reach, will be a series of shelves with cabinets, bins, drawers, or boxes labeled with the corresponding object(s) they contain, 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).

Directly in front of the user, above the desk/work-bench will be the complex object they are building or manipulating (much like a 3-dimensional circuit diagram). This SQL table will hover above the work-bench and can be connected using specifically colored wires/pipes to other tables and their columns. The 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 or dropped, as well. The system will implement one query at a time as the user submits them.
Along the user's right peripheral will be the same 2D panel where column and row data can be seen from the selected table. A user can scroll through this data much in the same way they would with a touchscreen device, only their hand can pass though the panel with no resistance. This panel doubles as the console log output for errors, warnings, and other useful developer information.

On the desk/workbench is a readout of the query under construction. A series of interactive buttons will appear and change as the user's options adjust to the choices they make.
Programming View Mode

Third VEDIC mode is the viewing of a C# project. This mode will contain colored shapes and connections similar to the viewing of the database 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 these objects possess.

This mode is where the user can view their program as a whole. Lines of inheritance and extension will be visible with colors that relay this meaning. Functions will be logical chains of nodes using graph theory to check whether connections are legitimate, or if correction is required. Colored shapes representing variable types sit in logical places within these nodes to demonstrate encapsulation.

A user can expand the view, zooming in and out, exploding or imploding the nodes to lengthen or shorten their connections. Nodes will float free-form in an invisible fluid environment, movable by the user's hand until they've found a position suitable to their understanding.

Programming Edit Mode

The 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 desk/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 the user outside of the VEDIC system.
All objects, whether complex classes or simple functions, will be broken into components. Each component will have a series of input and output spheres where a user can bridge connections. Much like a functional programming language takes output from one function and directly feeds it into the input of the next function, the output from these VEDIC nodes will feed directly into the input of the next.

Each subsequent connection will trigger the error checking mechanism contained within their parent object (explained in greater detail in later sections). If an invalid connection is made, the user is alerted to the error with both visual and audible cues. If the connection is valid, a colored wire bridges the two with a hue that represents the nature of that connection.

A user may delete a node at will. This will cause the nodes further on in the chain to separate and float freely in the workspace, while the nodes earlier in the chain to maintain their connections. The user can then reconnect the free floating nodes rather than recreate them from scratch. Those new connections will still be governed by the same error checking mechanism used in original connections.

When ready, the user can save, submit, or export the selected code. Each node, and their connections, are scanned, tokenized, parsed, and converted to a text-based equivalent. This text is transported externally from VEDIC in the form of a cs file for later compilation.

As a stretch goal, this environment will also be where the user can import C# code that exists outside of the VEDIC environment. This code would be scanned, tokenized, parsed, and transpiled into the 3D mesh factory where equivalent nodes and connections would be spawned.
Technical Requirements

Must Haves

- Must have a scanner/parser for SQL JSON import/export.
- Ability to receive data from local and remote databases.
- Must have a transpiler to convert SQL tokens into 3D objects.
- Shall have two database modes of operation: view and edit/create.
- Must have a relationship guideline that defines how objects are constrained to/with one another.
- Must have a position manager that correlates user 2D input to 3D movement.
- Shall use virtual reality environment.
- Must be able to perform basic C.R.U.D. queries; to include but not be limited to, allowing alteration of tables, columns, and data contained within individual rows.
- Ability to create new tables from two or more table joins/unions.
- Shall display existing database in 3D space, with table and column relationships visible.
- Need to recognize, visually, all the components involved in the database through colors, shapes, and labels.
- Ability to automate generation of 3D objects from data string.
- Must have system to identify intended child/parent relationships.
- Shall be able to save the state of the program for later use.
- Must have a detailed logging system for debug purposes.
- Required to have an informational website for all aspects related to VEDIC.
- Shall have thorough documentation for all phases of project.
- Must implement unit tests for each component of the project.
- Needs to use keyboard, mouse, and leap controller.

Would Like to Have

- Shall be able to construct classes for an object-oriented programming language.
- Must be able to view all created classes in 3D space.
- Required to have function comprised of modular components.
- Needs to have a save state to later revisit project in the environment.
- Must have detailed logging system for the programming modes of operation.
- Shall have the ability to scan and parse to export classes to C# files.
- Needs to have the ability to scan, parse, and transpile C# files into 3D objects.
- Ability to create new, edit, or delete old classes and their functions/variables.
- Automated generation of 3D objects from those C# files.
- Must show relationship between implemented and extended classes.
- Has to illustrate private, public, and protected protection levels.
- 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.
- Must be able to convert 3D objects into C# files and package them for export.
- 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 self-contained, localized database within VEDIC environment.
- Show object instantiation of program to demonstrate size and scope of project.
- Include a separate runtime environment for C# programs.
Inspirations Behind VEDIC's Design

In order for the VEDIC team to achieve the task of building a virtual reality development environment we first needed to know how to visualize code? Code visualization is at the heart of VEDIC. The reason for this is that visualization gives developers greater insight to the flow of a program and what interaction their programs are making. In the beginning, the group did a lot of research on the subject. Each developer scanned research papers, tested virtual reality applications, and talked with several people about different possibilities of how to solve this problem. One of the biggest inspirations for the solution came from talking to one of the senior design students from a previous semester about the Unreal Engine and a program it uses called *Blueprints*.

Take away from Unreal: (Structuring system)

*Blueprints* is a very interesting program because of its method of visualization. This method of visualization became one of the driving forces behind VEDIC's design. *Blueprints* is a visual scripting/node editor used to develop code. In *Blueprints*, users create events, functions, and variables with the use of nodes and wires. Each node represents a portion of code and each wire represents a transition/step of execution within the program. The code that gets generated for the Unreal Engine is C++. An example of a Blueprint can be seen in the figure below:

![Blueprint's Node Structure](image)

Each of the nodes in *Blueprints* is a visual representation of a component or keyword that is used within code. For example the input node seen at the beginning of the above figure represents the input parameters to a function. The output nodes represent what the function produces when called. This can be taught as a return statement. The branching node represents if and else statements. The thing that chooses which statement is taken during execution is the conditional which is represented by the >= node.
Each of these nodes together form a weighted graph. We wanted to achieve a node-like system similar to this and decided that this idea of nodes, and what they would represent, would be the driving force behind VEDIC's design.

One of the biggest differences between VEDIC and Blueprints is that VEDIC is planning to use the C# programming language instead of C++, which is what the Unreal Engine uses. VEDIC will also handle MySQL database scripting, a feature not found in the Unreal Engine. It is also important to note that unlike Unreal, VEDIC will be generating a cs/sql file for the user. The VEDIC team believes that the generation of these files is important because the code that users generate inside the VEDIC application needs to be of use outside of the VEDIC application.

Below is an example of a cs file. This is the code that will be generated if a user wants to create a graph like the one seen above.

```csharp
bool CheckScore(int x)
{
    int scoreNeeded = 100;
    if(x >= scoreNeeded)
    {
        return winGame = true;
    }
    else
    {
        return winGame = false;
    }
}
```

In order for us to create nodes that would represent code, it was necessary to identify the components that make up database queries and C# functions. Once accomplished, abstraction of those components to objects would be possible for a user could interact with.

**Take away from LYRA VR: (Object Interaction)**

Unlike the Unreal Engine, VEDIC nodes are not going to be 2D representations, but instead, 3D objects in world space. The reason for this is that users should be able to
interact with and manipulate nodes in a natural way. LYRA VR became the biggest influence on how VEDIC's 3D nodes will look, the environment which the nodes will reside in, and the types of interactions users will make within 3D space.

Examples of this can be seen in the VR Application known as LYRA VR and in the two figures below:

![LYRA VR - Touch Interaction With Nodes](image1)

**LYRA VR - Touch Interaction With Nodes [17]**

![LYRA VR - Moving Nodes Through Space](image2)

**LYRA VR - Moving Nodes Through Space [17]**

LYRA VR is virtual reality music development application. It was created during the 2015 leap motion game jam. The purpose of that game jam was to showcase the Leap Motion's development potential. In the application, developers used their hands to interact with objects that represent notes, drum beats, and other musical artifacts to create
songs. The user creates a song by connecting the objects together to construct a sequence graph. An example of this can be seen in the image below:

*LYRA VR - Connecting Nodes [17]*
User Experience Design in Virtual Reality

In order to make sure that VEDIC was not going to end up being this unappealing and unintuitive platform, a lot of systematic research had to be done in order to ensure that the VEDIC team knew absolutely everything there was to know about the history of virtual reality. More specifically, focusing on the current pieces of functionality that exist for VR as well as design do's and don'ts that would govern particular features. It's all about trying to find a proper fit for managing the application while at the same time adhering to established ground rules created by a community of developers that have been working on VR already.

There are a few points that have to be discussed before delving into user experience in VEDIC. First off, the importance of UX design in virtual reality. Like with anything else, when it comes to the usability of an application, it all centers on this idea that a user's input and response to the system is a careful relationship between user and program. The way that the user communicates and understands a system is ultimately the way that the user will judge the system. It does not matter if the system can perform amazing feats under the hood, or that it is the most advanced piece of technology. What matters in this scenario is how the user picks up and understands what is being presented to them. User Experience to many programmers is normally cast aside when building applications, fostering the idea that once the functionality exists, the experience can then be tailored to the functionality. Of course, for more segmented applications like web content, maybe this separation is more clear cut, and approaches like these may not be too detrimental to the usability of the application. But when working with virtual reality, an experience where the user literally has to use 100% of their mental processing power, user experience should not be taken lightly. It's a merging of user manipulation and program functionality. The relationship just got a whole lot closer. The discussion of user experience design within VEDIC is a topic that needs to be handled with great care. For VEDIC, it starts with the basis of the Oculus Rift DK2 headset and the Leap Motion Controller, since these are the main tools being worked with.

Content Degree Ranges

Let's start with the actual physical requirements that a virtual reality headset impose on the user. These are probably the most native user experience rules that should be followed as a guideline for discussing anything else. For the sake of this project, and overall consistency, VEDIC will use only certain guidelines governed by the use of an Oculus Development Kit 2 and Leap Motion controller, mounted on the front end of the DK2 device. With the headset on the user, there are immediate concerns that need to be addressed about how the user visualizes the 3D virtual environment. As it stands now, the Oculus DK2 has a field of view of roughly 100 degrees [13]. Note that this is 10 degrees
fewer than their more sophisticated counterpart (Oculus Rift), which is the full retail consumer version provided by Oculus by the end of March 2016. This translates into data about user accessibility in viewing objects within their immediate environment. There's a presentation provided by Alex Chu, lead designer at Samsung Research America, that discusses studies done that determined optimal user performance with neck rotation limits. Below is an image from that presentation in regards to comfortable head movements and limits.

With the data provided by Mr. Chu, we can try and establish boundaries for where user attention will be most concentrated on, and develop an understanding of how much the user can see. Given that users can comfortably move their heads from a resting forward position 30 degrees in either direction, we can calculate that with a field of view (FOV) of around 100 degrees. This translates to a comfortable front view of 160 degrees, combining both directions. Doing the same for the maximum head rotation where users have to strain, results in an FOV that roughly encompasses 210 degrees. These numbers have been rounded for simplicity. From these two established FOV's, it's known that there is generally a field of view of 160 degrees for main content objects. There are things users will be manipulating throughout most of their experience. These area are reserved past these field of views where users have to strain their head rotation to access. The use of events in these areas should be interacted with sparingly.

**Downward Content Zones**
The data also shows us that users are more comfortable looking upwards than they are looking downwards. Chu regards this as being "due to the physique" of the human body [7]. However, there is some conflicting information found that implies the opposite of what Chu presented. According to general office ergonomics, monitors are meant to be under eye level of the user [5]. This is because a resting position of looking partially downward is more comfortable than having to look straight ahead, neck vertical. Fusing these two snippets of information together brings us to a consensus that focusing content for the user slightly downward so that the user is not looking directly forward may be more comfortable. Allowing for extra event areas to be positioned upwards is more friendly than forcing content under the users' vision where according to the studies by Mr. Chu, would strain the user more [7].

Follow Content Ranges

When regarding depth and distance of objects and events in orientation with the user, some relevant numbers come out of the same presentation by Alex Chu. Given that this is a simulated 3D space, objects that are rendered 20m away from the user begin to drastically lose their appearance of being a 3D object [7]. The reason for this is because the headset begins to lose its ability to trick the user into thinking what they are seeing is actually in 3D. Recall old movie 3D tricks where images of red and blue are slightly separated in order to create the illusion of objects coming out of the theater screen. This is similar to the same principle, just more complex. Oculus works by producing two similar images on both of the eye-pieces like monitors that are used to view the virtual world in the headset. They skew objects and images to trick the user's eyes into focusing on an object that appears to be distant and possessing depth. Once an object's range exceeds 20m, the illusion of skewing the image to make something appear 3D is lost. Both screens are rendering the same image with no room to skew it in order to create the illusion because of the objects relative distance to the user. This establishes that any scene object needed to be understood by the user would be best to not exceed this limit.
Similar restraints are also described when objects appear too close to the user. The reasoning behind this is the simple explanation of becoming cross-eyed. When objects have to be rendered in close proximity to the user, it asks these users to focus on them, and thereby causes the eyes to cross. It's the same sensation as trying to look at your own nose. The world in front of you becomes out of focus to stay attentive to an event persisting within this boundary of 0.5m. That is why, for design constraints of developing in VR, no content should exist within this range unless absolutely necessary. This would only cause the user to disconnect from the environment, and the illusion shattered. For VEDIC, this is something to avoid.

Limitations of Leap Motion Controller

Just to take a step back here, VEDIC will be accepting user input with the Leap Motion Controller. It is important to go over a few of the restrictions and design points pertaining to it. First off, the explanation of VR content zones is fine, but when talking about a stationary experience controlled by Leap Motions, a developer is confronted with limitations. First off, there's a restriction to how far the extension of user's arms can properly manipulate objects. When talking content zones of 20 meters long, it's not viable to have objects at a distance the user cannot interact with. Rather, objects users will be able to handle are things that are close to them. However, given the way Leap Motion
works, it's possible to manipulate objects that are further than a user's reach with innovative extending features. This must be taken into account when creating objects that users are expected to interact with.

Another limitation of the medium is the inability to cross the arms. Through testing and in Leap documentation, it's impossible at this state to cross the arms and still have leap recognize them individually. The reason for this is because they use "infrared silhouette", which determines arms in the field and renders them as so [12]. When one arm crosses the other, the controller cannot distinguish the meshes being received by the infrared reader and collapses the process. This forces arms to stay on separate sides of the user-space at all times.

The range of view that Leap Motion bolsters is a 150 degrees horizontally [12]. This mounted on to the display of the user's Oculus ensures that no matter where they are looking, their arms will always be recognizable.

UX: From Physical to Psychological

Getting past the physical limitations of something like the Oculus DK2 is one matter, but dealing with the psychological turmoil that comes with VR is an entirely different subject. Given that the user will be using a device that's goal is to trick the user's mind into believing that they are currently somewhere else, it's easy to see how problems could develop with this sort of an experience. The running trials of trying to develop on VR platforms in recent years has come with a certain amount of learning and adapting to new forms of "VR sickness" that becomes an issue when boundaries or unforeseen ground rules are broken by the developer. It's necessary that the mistakes others have made in the past are analyzed. Preventing any form of psychological annoyance or harm to the user is crucial for providing a positively potent experience within VEDIC.

One of the most prominent VR sicknesses known by anyone that has used VR equipment is the issue of induced motion sickness. When a world is presented in a VR environment, the user will most likely feel as if they have entered that world. It's a sensation that can lead to a lot of displacement for the user. The sensation of being so engrossed with the environment can lead the mind to believe that you instinctively are present in this environment. Therefore, the mind will assume the same rules apply as they do in the real world. Now unfortunately, there is no guaranteed way of ensuring 100% no motion sickness to users using VEDIC. Using a headset that places the mind into another world is a harder pill for some to swallow than others. What can be done to alleviate some of this is to help facilitate the expected outcomes that are instinctively triggered in the human mind by adjusting different context and conditions with appropriate runtime
changes.

Use Constant Head-Tracking

One of the easiest connections to instinctive expectation is the notion of head motion. In order for the user to view the environment presented in VR, the user must be moving their head around. This leads to the expected reaction that their movement is changing the orientation of the camera that constructs that environment for them. If this synchronization is broken for whatever reason, the user can begin to feel disconnected from their environment. This strange sensation can easily lead to the user experiencing nausea caused by perceived motion sickness. The rules that govern this dilemma are simple. Whenever the user is wearing the headset, their head movements must always be tracked and translated in real-time to the screen [8]. At no time should the application ever disable head-tracking because maybe they opened a menu or looked away from the event. The same goes for the other end of the spectrum. Users should never be forced to look in a certain direction. If a user is looking down at the ground and then immediately looking back forward, it would feel very strange that their head is not where it should be.

User Movement Control

Users should always be in charge of their movements. If a user has the liberty to move in an environment, then it should be a rule of thumb to never prohibit movement, and same goes for the alternative. It's similar to the head-tracking, but it has a different context because we are not talking about the head where feedback of the visual image is in direct relation to how the user's head is orientated. This refers to having the user feel like they are in control of their location, and are not being moved against their will. Teleportation is used in VR environments to express the movement over large spaces in limited playing areas. Users know ahead of time that they will be moving from point A to point B. Forcing a change in the user location without addressing it will become nauseating.

User Can't Experience Velocity

Another UX design point is to always be aware of users' velocity in whatever context they are in. Google has an article depicting the physiological considerations for designing VR headsets. Simply put, they state that "in real life, we feel acceleration and deceleration, but we do not feel velocity" [8]. It's the sensation of feeling complacent even though the person is moving with a car that is going at 70mph on a highway. The solution to this dilemma is simple. When trying to emulate speeds oriented with user movement, it's important to ensure that the user is going at a constant speed instead of
being constantly accelerated and decelerated. In VEDIC, users will be fixed into a position that will be unmovable by input, and only have head orientation to use as a point of movement. The idea of translating movement to view large data is still an idea that can hold some potential, but not currently a pending feature.

Grounding the User

Grounding a user to their experience is an essential component when trying to trick the mind into not feeling displacement in the world being presented [8]. What VEDIC has done is provide context to this sensation by naming these sort of elements in the world as "Anchor objects". The user needs to feel that in some way they are connected to this generated world and are not just free floating. In order to do this, the environment should have some form of consistency so that the user can rely on this focal point being presented. That way, when users move, they have a place to judge their positional movement based off of that of their anchor points. It's a relationship of trust with the user; disclosing to the user that you are safely behind this counter and you can always come back and see it, whether it be in your peripheral vision or locked in an area around you, like a cockpit of an airplane. Because VEDIC is mostly an experience where you are manipulating objects and working with an area that does not change, our anchor object will be the command console that appears before the user as a desk and helps them understand where they exist in the world along with everything else. An example of this can be seen in the free demo Geometric published by Leap Motion. Their console helps orient the user to the center of their workspace, giving the user a sense of location in the
Moving Objects (without user interaction)

When it comes to dictating motion for users the rules are simple enough. Ensure that the user feels like they are in control. But when we switch the subject over to objects in the user's surroundings, the user may or may not have control over their movement, depending on the desired effect. With this in mind, it's important to differentiate what the user can and can't interact with to keep the user in a state of understanding. Not only this, but how users see objects must be a tailored experience as well. If a user was meant to concentrate on particular objects, then great care should be taken into consideration when having those objects move without the users' permission. Movement should not be stagnated, but extremely fluid, and slow as well. Moving objects in and out of the user's cone of vision can become disorienting if it happens consistently without any pattern or rhythm. Violent object movements can also lead to a sense of dizziness and nausea that should be avoided. One easy example is changing a user's scenery without movement. If a different view needs to be presented to the user, shifting the entire environment will leave the user disconnected with the world. However, notifying the user and translating the movement by momentarily changing the view to something static for the user to re-emerge somewhere else would not violate those rules.

Text Guidelines

Text is also an interesting feat to figure out in virtual reality. When you are dealing with standard environment objects that are naturally 3D, it is easy to understand and interpret what these objects mean and how they are in orientation to your surroundings. However, when dealing with text, this is an entirely different story. Text by nature is normally a 2D flat presentation of characters. In virtual reality, displaying these 2 dimensional pieces in front of the user creates some issues. Because with the way people work with VR, the eyes are tricked into moving properly in the VR space on an all-encompassing curved surface being created by two screens. The tricks that foster this 3D environment are natural to the system rendering. However, when this focus point is translated into a 2D space in the environment for long stretches, the eyes find difficulty in actually following along those 2D surfaces. VR headsets are trying to create images of that text, that has no real depth. Jodie Medich explained it best during the presentation on the good and bad of VR. She says "The focal depth is … in an arch, and you are asking people to read a flat piece of paper. My eyes can't adjust to read that… because you really only have this one focal plane". [3]. Now this has been improved in recent iterations of VR equipment by increasing graphical qualities and refresh rates, but issues can still arise making it important to follow some of these rules when approaching the display of text in VR. According to the presentation by Plemmens, it's important to avoid "typefaces that will have anti-aliasing issues" [3]. This means making sure that letters are straight and clean.
to avoid any blurriness normally seen in fancier texts, like cursive writing. Plemmens also states that if the user must read a lot of text, putting them on concave surfaces will help assist the whole 3D rendering process and allow the user to have focus points [3]. Making sure that text is contrasted as well with its background makes it easier to focus on.

Use of Directional Sounds

An advantage of a VR environment is its ability to have a space where there is practically a 360-degree screen view happening all around you, even if you can't see that 360 degrees all at once. This all sounds great, but how exactly does it really sound? Despite those play on words, audio cues in virtual environments are really important to get right. It could mean the difference between the user being properly directed to turn in a direction where their attention is needed, to completely being lost in their world space. This puts a heavy emphasis on incorporating full "directional audio in the 3D space to direct attention where you want them to go" [9]. This accommodates for the change in view mode from a 2D monitor to a 3D simulated space. If the user is currently focused on a task on the right, but subsequently a task that they started previously has just ended on the left, it would make sense that the audio cue take place to their left then being just a mono or even stereo left sound. Surround sound has already been doing this, but the experience in a VR environment must be tailored to use these systems of manipulating sound in a field.

Universal Usability (Avoiding the "Phantom Leg")

There are other design decisions that seem small but are actually big in terms of keeping the user from having any form of "VR sickness." One in particular that VEDIC as a team prefers is one that we call the "Phantom Leg." This comes with the idea that users can all be of different heights and disabilities. When constructing an environment to be used by different sized users, you have to compensate for how the environment will take those factors into consideration. For example, some of the VEDIC developers felt a sensation during some of the play tests in VR games that made legs feel strange. It felt like being a lot taller or shorter than what really was. If a person is standing and feels their feet touch the ground, but alternatively in the VR space they are only a few feet off the ground, this can cause feelings of having a weird numbing sensation in the legs. The same can be applied if sitting on a stool where feet can't touch the ground, and in the VR environment the viewer can see the ground where normally their feet would touch, it makes it feel like their legs are strangely disconnected. In order to compensate for this, it would probably be useful to make note to the user to be sitting down when their avatar is seated, as well as to never really indicate a floor directly under the user. In this way, the user can just fill in the pieces of where the floor should be, and they can move on.
Distinguishing Interactions (Static V.S. Dynamic Objects)

Part of an issue that users would encounter when being given an environment in VR is deciding which parts of that environment provide a form of interaction. There can be a slew of tools and objects placed in different positions from the user, and they are not quite sure what each component does. Exempting the possibilities of providing a tutorial of the application, the user needs to be able to understand the distinction between usable and non-usable. Normally, a sort of feedback to the user would be expected so that users know what is allowed. Highlighting objects in the environment, whether due to close proximity of a hand, or just in general, is an effective means of conveying this functionality to the user. The two images above show the VR game *Job Simulator* where a user approaches an object and is provided feedback that the object can be moved by highlighting it with a light blue color.

Ergonomics of Human Interaction

Human bodies don't function in straight lines. Our bodies are meant to interact with the world in a series of arches, especially when restrained only to one spot [11]. It's easy to design a piece of environment in virtual reality to represent what a user needs to see for a 2D monitor. However, this is not a practical approach. If a user needs to interact with the equipment that is being presented in the VR world than it should be tailored so that it can be easily manipulated. This is to say, that a human's arms, which currently are the main method of interaction with VEDIC, work in a series of arches, governed by the elbow and shoulder. If a user needs to drag something from right to left, making sure that their surfaces for doing so also flow in an arched manner. This will prevent any unnecessary uneasiness when it comes to wanting to drag objects across. This will avoid interactions where users cannot reach the edge of the space due to a flat surface spanning passed the arch of an arm.
Use Lighting to Guide Users

Now that we are moving passed the 2D into the 3D, it's important to focus on the use of building your environment in a way that flows. A lot of the ideal gaming design points and architectural design points now apply to the world of user experience for VR. Just like in gaming and architecture, we now have to act as mini architects when it comes to creating a world where light sources now affect the way information is conveyed. If user interaction is required in specific areas, a change in the way light is being displayed to convey this message is absolutely necessary. We can't leave users in the dark about how anything in the environment functions, so it's important to use lights (shining bulbs, directed light) to reveal the necessary actions versus the ones that don't currently matter. We can't assume that everyone owns a form of audio output. Accounting for different setups with as much directional design as possible is a must.

Object Scaling

In a VR setting, it's easy to break the virtual reality users are experiencing through the application. Object scaling is an idea that provides a few guidelines as to how users recognize different objects, and ways that they could make them feel. For example, small objects such as mini-cubes the size of a thumb will "feel cute and toy-like" declaring that it can be picked up with your hands [9]. On the other hand, bigger objects will make users feel blocked. If a user is to manipulate an item, it helps to convey that the object is small, or at least smaller than it seems.

Simulating "life-size" objects within the game world can also help the user feel comfortable and connected with the experience. A life-sized desk, or a few props that are of life-sized proportions can help trick users into feeling that the environment that they are in is not that strange. This way, users can stop thinking about where they are, but start focusing more time in actually doing something with it.

Expose Users Gradually

Generally, something like a tutorial inside of an application tends to administer some doubt to whether the product was designed properly. Some would argue that if the system was designed intuitively, then it would not be necessary to illustrate the functionality of the application through self-contained demos. And in some instances this could be true. However, there are scenarios where the functionality being described is fairly new to users, and therefore it would behoove a developer to provide some sort of guided explanation as to why certain things are the way they are. For virtual reality, this is a necessary must.
Coming back to Daniel Plemmens presentation on Designers + Geeks: Building Virtual Reality, some very interesting points were made that holds a lot of truth about the current state of VR acceptance by the public, and how designing incremental tutorials would be very beneficial.

"We have these incredibly powerful devices... There are a lot of folks that believe that if we don't get the design of the VR application correct right now, we are going to see another example of the 1990's where VR came up and then sputtered out a bit... Make sure to ease people into the experience... Give people time to acclimate."

– Daniel Plemmens [3]

The idea that Plemmens is presenting is that people need to be "scaffolded" into understanding the environment that is around them. There are a lot of users out there who have never experienced VR before. They don't know a lot of the language cues that we have learned from using VR in development. We can't assume that people understand the basics of VR. Therefore, developers should design the applications so that they encourage a mode of tutorial that users can get up to speed with, what things they should know, and hopefully not feel overwhelmed as a result of this. The general consumer is still new to this hardware, and the last thing that applications should do is discourage or make users uncomfortable by expecting a lot from the user. Assume that the user has never used a VR headset and build a gradual scaffold there. For VEDIC, the team wants to make sure that they follow these same guidelines.
User Interface Design

Twenty years ago, UX and UI designers were confronted with a new and exciting medium—the web browser; a two dimensional screen, effectively a plane, on which all content must be intuitively displayed, interacted with, and filled with meaningful content. Much like this medium that we now take for granted, virtual reality will one day be as commonplace as the internet. VEDIC, like other attempts to standardize and explore VR, is the NetScape of this new frontier. A third dimension has unfolded onto our technical world and given us pseudo-depth.

No longer must we paint a picture, but rather design a container, a room of unlimited size and shape. The only restraints are those of the human mind to grasp its contents in a meaningful way.

While UX designers are racing to define, catalog, and shape this new space, UI developers must create the tools born from the rules they discover along the way. Is grasping an object with the whole hand more effective a method of interaction than pinching it with thumb and forefinger? Can pushing a button with virtual fingers have the same intuitive feedback in an environment free of haptics than that of a mouse click? Questions like these number in the hundreds, and continue to multiply as the number of physical tools for VR become available.

A set of UX design guidelines were mentioned and described earlier in this document. Building on, and complying with, those rules are the individual UI components that VEDIC will use in its normal operation. Each gesture, movement, and response is listed in detail below. Each item will have a technical description to distinguish it from any other, one or more images to illustrate that item, and then further details as to the reason and purpose behind the design.

Terminology

Unless explicitly stated otherwise, the following sections will refer to the hands as left hand and right hand, identified from the perspective of the person on whom they are attached. Their left hand is located on their left side, and their right hand located on their right side. When the palm of either hand is facing down toward the ground, this position will furthermore be referred to as prone. The palm facing up toward the sky will be referred to as supine.
Concept Art - Demonstration of the Prone Hand Position [15]

Concept Art - Demonstration of the Supine Hand Position [15]
The five fingers will be henceforth referred to by their number. The forefinger will be the first finger. Middle finger is the second, ring finger the third, pinky as the fourth, and the thumb will continue to be called the thumb.

Distance will be measured qualitatively, starting at the user's face and increasing along a positive axis as distance from the user increases. The negative axis starts at the back of the user's head and increases along the negative axis as distance from the user's backside increases.
Five zones exist in the user's visible realm, and shall be referred to by the following names:

**The Safety Zone** is the area immediately in front of the user's face where no content should appear. The user shall not have the ability to interact with this area, nor shall their virtual limbs be rendered in this area should they bring them close.

**The Rear Zone** is the area directly behind the user, starting from one ear and extending 180 degrees around until the other ear. Any content rendered here shall be non-essential. This area shall be populated by content of an aesthetic nature, meant to complete the scene rather than add functionality to it.

**The Interactive Zone** is located in front of the user, starting at the Safety Zone's furthest edge, and contains the area within the user's reach. This is the area where the user interacts with the environment. Buttons, sliders, node elements, all should exist in this space.

**The Peripheral Zones** are located at either side of the user, just in view of their peripheral vision. This space is contained within the Interactive Zone, but characterized by their position between the forward space and the Rear Zone. In this area, information panels can hang in mid-air with data that isn't immediately pertinent, but can be quickly
viewed with the turn of the head as easily as glancing at a second monitor would have been.

**The Backdrop** is the zone far out of reach and serves more as set dressing than legitimate environment. It can be used to set a certain mood, interchangeable by the user if multiple skins are available to them. Where programming at the beach would be impractical as an example, due to the sand, bright sunlight, and copious amounts of salt water, it wouldn't be an issue in virtual reality. A sun rising in the distance; waves rolling across the shore yet never reaching your feet; perfect white sand that doesn't track into your home or muck up the spaces beneath the buttons on your keyboard.

**Gestures**

**The pinch:** Performed with either the left or right hand, the user folds fingers two, three, and four into their palm. With thumb and first finger extended, the tips connect to complete this gesture. See the following image for a better visual:

![Concept Art - The Pinch Gesture](image-url)
This gesture will most often be used to grab a single object, and lock it to the first finger and thumb until the gesture is broken, or the hand moves out of view of the Leap's camera. Unlike the grab gesture described later, this is more of a precision gesture, where placing a specific component in a specific location is more important than quantity of movement.

**The grab:** Performed by first positioning the palm next to one or more elements. The first four fingers fold around the element(s) and into the palm. This gesture is concluded when the thumb folds over the four coiled fingers. See the depiction below for a better visual:

![Concept Art - The Grab Gesture](image)

This gesture will most often be used to lock onto multiple objects in order to move them in bulk from one place to another. This will be less precise than the pinch gesture described above, but has the added benefit of quantity.

**The circle:** Performed by folding fingers two, three, and four into the palm of the hand with the first finger extended. Tracing the first finger through the air in a clockwise gesture is indicative of increase or positive, while counterclockwise indicates decrease or negative. See the depiction below for a better visual:
This gesture will serve two purposes, distinguished by context. When near a dial of any kind, this movement will turn the dial clockwise when the first finger rotates in the clockwise direction and counterclockwise when the first finger moves in the counterclockwise direction. When not near a dial, this gesture will rotate certain objects around the depth axis.

The touch/tap: Performed when fingers two, three, and four are folded into the palm, and the thumb is either perpendicular to the folded fingers or folder over them. The first finger remains extended and the gesture is complete when the tip of the first finger makes contact with a valid object. See depiction below for a better visual:
This gesture is used to select an option in a list or carousel, a node, or to trace a connection from an output to an input of one node to the next. Perhaps the most prolific of gestures that will be used in VEDIC, and also the most intuitive to the user as touch screen devices have made it commonplace.

**The push/poke:** Performed in the exact same manner as the touch/tap gesture above with the added exception that the tip of the first finger continue to push forward.

Most often used to press buttons, this gesture can also move an object through space in small bursts, and poke through pliable surfaces.

**The swipe/slide:** Performed with all four fingers extended and touching side by side with the thumb along side the others, or pointed up and away from them. See the depictions below for a better visual:
Concept Art - The Swipe/Slide Gesture (Finger Positions) [15]

Concept Art - The Swipe/Slide Gesture (Hand Motion) [15]
This gesture is used to pass partially through a surface and slide through content, much like the human hand could be used to move a hanged shirt along the rack. This same movement could be used to haphazardly move large numbers of objects from one area of the environment to another. This would be done when precision is of no concern.

The gestures mentioned above are only the main few that will affect the bulk of interactions the developer performs in the course of their session. Other gestures can be developed (trained) as the need arises later as new functionality is desired, planned, and programmed into VEDIC.
The Abstraction/Structuring Problem

This abstraction/structuring process for VEDIC’s nodes was one of the more difficult problems that the team had to solve. As an example, a database query can be separated into different types. Those types are updates, inserts, deletes, and selects just to name a few. Each of these types of queries are composed of a different sequence of strings. That sequence of strings can range from something simple to complicated. The complexity and therefore difficulty of this problem comes from structuring conditional operands that could or could not follow each of these database queries. Below are a few select queries that demonstrate differing levels of complexity ranging from easiest to the more complicated.

- Select from * suppliers;
- Select suppliers.snum, suppliers.sname from suppliers;
- Select snum, sname from suppliers where snum in
  ( select snum
  from shipments natural join parts
  where color='black'
  and jnum in
    ( select jnum
      from jobs
      where city = 'Orlando'
    )
  );

It's also important to note that not only was it necessary to make sure that the appropriate database query nodes get attached to one another, but when the VEDIC team starts implementing the code generation module for C#, they need to make sure that the structuring system prevents a database query node from interacting with a coding node. For example, VEDIC should prevent any connection a user tries to make between and database select query and a C# class.
Designing the Structuring System

First Design

The first structuring system that the VEDIC team designed represented each string as a node and each node as a different color and shape. As an example the word select would be a green triangle node. This is more closely related to the notes found in LYRA VR.

The original idea was to structure nodes in something called expressions. When a node makes a connection with another node, the two nodes would then become an expression. This expression is a new node that contains references to the other node that created it. Similar to a sperm and an egg coming together to form a new human. The genetic information of the parents are contained within the child. The type of expression to be created will be based on the type of nodes that get connected together much like the child type is dependent on the genetic makeup of its parents. The user would then build upon that expression with other nodes creating a new expression each time a new node is added. This is similar to a child getting older and becoming parents themselves by mating with another person. This process will then create a binary tree structure which could be traversed to generate the text file representative of it. It's important to note that there will be an object used to determine what nodes can and cannot be connected together. This object would essentially play matchmaker and will be called whenever the user tries to make a connection.

Comical Illustration of the Forced Matchmaking in Old Design
The problem with this design is an issue with usability and informational overload. By representing each string as a node users will be bombarded with a huge number of choices. This leads to users being intimidated by our application. Users will not know where to start and therefore more likely not to use VEDIC.

Second Design

The second structuring system the VEDIC team came up with was based on the idea of context free grammars and solving the inaccessibility problem of our first design. Instead of each node representing an individual string, each node would represent an individual expression based on our own context free grammar.

For those unfamiliar, context free grammars is a set of recursive rewriting rules used to generate patterns of strings. An example of a context free grammar can be seen below.

Example 1-1

Let's say that the following context free grammar uses an expression called MathProblem. MathProblem is defined by the following 2 expressions, Var and Operation.

<MathProblem> => <Var> = <Operation>
<Var> => a | b | c
<Operation> => <Var> + <Operation> |
            => <Var> * <Operation> |
            => ( <Operation> ) |
            => <Var>

A valid MathProblem would be a = b * (a + c). Below shows the series of steps that prove this:

<MathProblem> => <Var> = <Operation>
   => a = <Operation>
   => a = <Var> * <Operation>
   => a = b * <Operation>
   => a = b * ( <Expression> )
   => a = b * ( a + <Operation> )
   => a = b * ( a + <Var> )
As we stated above, each object the user would interact with will be an expression from our context free grammar. The user would then combine these expressions with other expressions until the expression is expressed with it ending at a terminal expression. An example of a terminal in the expression MathProblem is Var. Var can only be expressed as an a, b or c. A good example of a non-terminal expression is Operation. Operation is an expression that contains other expressions. What also makes Operation unique is that it's recursive. This means it's an expression defined by itself, potentially creating a looping structure. We can see this looping taking place when we showed the steps that $a = b \times (a + c)$ is a valid MathProblem Expression.

The Inspiration for this design came from the child's toy/game of square pegs and round holes. Each expression would represent a page and some expressions would contain holes. The ideas that the user can only put the appropriate expression in the appropriate hole. Doing this we help guide the user in creating valid expressions. An expression is not completed until there are no holes remaining.

While trying to write the documentation for VEDIC, the team came into some difficulties with this design. The difficulties came from how we were going to implement the node that would represent expressions in code. The class diagrams for the expressions became too cumbersome and massive for an individual to understand or keep track of. This is when the team decided to come up with a different approach to the node connecting system.

Third design (Current design)

The third design of the node connection system was inspired by binary search trees, a conversation about complexity with VEDIC's sponsor Dr. Richard Leinecker, and the node editor in Blueprints.

A binary search tree is a data structure typically taught in an introductory computer science class or in an introductory data structures course. The idea behind a binary search tree is that you are structuring data in such a way that a graph is created so that it's easy to find data. The graph is composed of nodes and each node has references to two other nodes. These references are known as branches and they are called the left and the right branch.

This binary search tree data structure was mentioned by Dr. Leinecker as a way to help simplify the complexities we were facing with the second design. He realized that each
node in the *Blueprints* editor had two distinct sides, a left and a right. Everything on the left side of a node was an input and everything on the right side was an output. This is when the problem was determined to be a graph related problem and one of the best ways to solve a graph problem is in using a binary search tree structure.
Implementation Vid-Objects

The following sections will discuss the implementation of the current Vid_Object design. The class diagram of a Vid_Object can be seen in the figure further below.

What is a Vid_Object?

Vid_Object stands for Visual Interactive Developer Object. The Vid_Object is the base model for VEDIC and is the core logic behind its 3D nodes. A Vid_Object is everything a user can interact with or manipulate. It can be thought of as the atom of the VEDIC. Just like there are different atoms in real life such as hydrogen, helium, and carbon, so to are there different Vid_Objects. The two most important types of Vid_Object are Vid_Nodes and Vid_Expressions.

Class Diagram of a Vid_Object

What is a Vid_Node?
A Vid_Node is a Vid_Object that contains relational data from another Vid_Node. That relational data can be found in the Vid_Node properties. These properties are inputs, outputs, left, right, and expression. These properties help in transforming each node into a text file.

The left property stores all Vid_Nodes treated as inputs. The right property stores what node the current node is connected to. It is important to note that the right and left property can contain multiple lists of Vid_Nodes. The reason for this is to account for multiple input parameters for functions and for multiple references to the output of that function.

This property is similar to that of the left property and the right property. The distinction between the two is the type of object that the property and the property store. Instead of the two properties storing Vid_Nodes, They store NodeOutputs.

What is a NodeOutput and what is its purpose?

A NodeOutput is an object that represents the output of a node. Its purpose is to help simplify node connections. One of the biggest flaws with the old designs were that a lot of specific code would have to be created in order to make sure that one type of node doesn't connect with another type of node. What this means is that for each time a connection was attempted, it was necessary to have a look at the classes for each node and then check to make sure that the desired connection was valid.

With NodeOutputs it isn't necessary to know what type of node the current node is trying to connect with. That particular value is just heard. This make tasks such as implementing recursion much easier both for the VEDIC developers and for users. Essentially NodeOutputs opens up a lot of development possibilities and user creativity.

What is a Vid_Expression?

A Vid_Expression is a special Object in VEDIC. Originally Vid_Expressions were meant to contain other Vid_Objects. Only a valid Vid_Object would be added to a Vid_Expression. The process of adding a Vid_Object to a Vid_Expression was governed by an evaluator/matchmaker object. Its job was to detect when a user tries to add an invalid Vid_Object to a selected Vid_Expression. The Evaluator would reject that Vid_object, preventing it from being added to the expression.
After many redesigns behind the structuring system of VEDIC, Vid_Expressions ended up being the starting node/root node for each node graph. The reason why having a starting node is important is that it simplifies graph traversal. By knowing where to start, it's easier to then follow the starting nodes right property to a new node which will eventually lead the developer to a leaf node, and thus the end of a valid MySql query expression or C# function.

What is the Stringifyable interface?

The Stringifyable interface provides a method called stringify. Each Vid_Node will implement this interface. This method is used in the conversion of a Vid_Node into text. Because all the nodes are connected together they essentially form a parse tree. During the traversal of that tree the stringify method gets called on each Vid_Node which appends formatted text to an existing string or text file. It's important to note that only nodes connected to a Vid_Expression or any node already connected to the expression will be stringified.
User Interactions With Vid_Objects

Interactions with Vid_Objects are the catalyst for many events within VEDIC. When a user touches a Vid_Object, the interaction causes a sequence of events that encompasses three major objects. Those three objects are the selected Vid_Object, Vid_DeveloperUtils, and the Vid_ModeManager.

The logic that handles interaction of Vid_object starts when the user touches the currently selected Vid_Object. That Vid_Object makes a call to the Vid_ModeManager to check what state VEDIC is in. The Vid_ModeManager will then return the current state of the application back to the Vid_Object that called it. That Vid_Object would then use the current state of VEDIC in selecting what type of logic should be executed based on the user touching the Vid_Object.

The following diagram shows a sequence of events that takes place when a user touches a Vid_Object:

I mentioned before that Vid_Object uses the current state of VEDIC in selecting what type of logic should be executed based on the user touching the Vid_Object. Below are the Sequence diagrams for states in development for VEDIC.
Separation of Logic Through States

Why does VEDIC need States?

Users of VEDIC are going to be constantly touching, building, and connecting Vid_Objects in the application. There needs to be a way to distinguish moving objects around in the world space from deleting them. Our solution to this problem is to separate user actions into discrete states.

After many redesigns behind the structuring system of VEDIC, the Vid_Expression ended up being the starting node/root node for each node graph. The reason why having a starting node is important is that it simplifies graph traversal. By knowing where to start, we can then follow the starting nodes right property to a new node which will eventually lead us to a leaf node And thus the end of a valid MySql query expression or C# function.

What are the different states in VEDIC?

Currently there are 7 states in development for VEDIC.
1. Move
2. Delete
3. Query connection
4. Expression Manipulation
5. Class Building
6. Function Building
7. Database Manipulation

What controls the state VEDIC is in?

Each state will be controlled by the ApplicationStateManager. This ApplicationStateManager is a singleton and it keeps track of the state, currently selected Object, previous selected Object, and currently selected Expression. These objects are key to the underlying logic in each VEDIC state.
What is the importance of currently selected Object and previous selected Object?

In order to make a connection from one object to another object the user would have to select two objects. This means that in order for a connection to take place, a reference to the first object has to be stored somewhere else in code other than the currently selected object variable, otherwise the object that was currently residing within the currently selected object location would get overridden by the newly selected object. This is why we have to have a place in code to store the previously selected object.

The following sections will explore the different states of VEDIC in more detail.
The Move State

The movement state is probably the most straightforward in terms of logic for VEDIC. When a user selects the state no state logic will be activated. This is clearly seen in the sequence diagram below. Thanks to the Leap Motion API, objects that get touched by the hand object move automatically. The only requirement the object needs is a Unity collider object. This is all thanks to Unity's physics engine.

![Move State Diagram](image-url)
Delete State

The delete state is a step above in terms of complexity compared to the move state. It was mentioned before about the importance of keeping track of the currently selected object and the previously selected object under the heading entitled "What is the importance of currently selected Object and previous selected Object". This is one of those times where keeping track of the objects that the user selects becomes necessary.

We can see in the above figure that the VEDIC state logic is looking for the user to touch the same object twice. This requires us to check the currently selected object with the
incoming object. If the two match, VEDIC will perform the logic to delete the selected Vid_Object. The logic that is in charge of deleting the node is located in the DeveloperUTIL class.

The DeveloperUTIL class is a class that contains a collection of functions that do not make sense anywhere else. These types of functions are meant to help VEDIC developers perform a specific task such as deleting the connections that make references to a node that a user wishes to delete.
Connection State

The sequence diagram in the previous section does not represent an official state in VEDIC but instead many states. Those states are Query Builder, Class Building, and Function Building. Each of these states are going to require the connection of the one Vid_Object to another Vid_Object. The type of object that gets connected together is dependent on VEDIC's state.

![Connection State Diagram]
Just like in the delete state, this is one of those times where we need to keep track of the objects that the user selects. We can see in the figure above that the VEDIC state's logic is looking for the user to touch two different objects. When the second object is selected, The Vid_ModeManager tries to connect the two objects together.

If the two objects are Vid_Nodes they will have references to the inputs they're expecting and outputs they produce. A quick check can be done to make sure that the connection is legal and if it is, VEDIC makes that connection and then sets the previously selected node to null. If the connection is not valid, VEDIC makes the previously selected node null.

Types of Vid_Nodes and their respective states

VEDIC restricts the types of nodes a user interacts with. It choose the types of nodes it restricts based on VEDIC's state. The following are pairs of VEDIC states and the nodes that they can inhabit in that state.

Query Building
- Update
- Select
- Delete
- Insert
- Drop
- Join
- Conditionals/Operands

Class Building
- Function/Methods
- Imports
- Other classes
- Basic data types

Function Building
- Function/Methods
- Other classes
• Basic data types
• Conditionals/Operands
Query (send) Statements

We mentioned in the previous section that the query state should handle connecting Vid_Objects together, but there is another function that state needs to handle. That function is sending a query to a remote database.

The above diagram shows the sequence of events when the user sends the query statement they've been working on to a select database. The process starts when the user presses a button that tells VEDIC that it's ready to send the query that they've been working on to the database.

The query is then sent to an object known as the Query_Controller. The Query_Controller is an object that the user can't interact with. The Query_Controller's responsibility is to get into contact with the user's database of choice and format the query to a string so that it can be read by the database.
The server that the database will be located on will need to have a special php that we will create so that the query that we sent to that database can be processed correctly. One of the other reasons for this, users of VEDIC need to put the specialty php file on their server that holds their database so that the file can return a JSON object.

That JSON object will be sent back to the Query_Controller and the Query_Controller will determine whether the query passed or failed based off of that object. It's also important to note that JSON objects will also contain metadata needed to perform particular tasks on the 3D object that represents the user's database. The Query_Controller informs the database object of the metadata it needs to update itself/perform its designated task.
Tools

VEDIC uses a variety of tools to accomplish its goals. Each was chosen for a number of reasons: cost, availability, power, and ability to integrate with the other tools.

Oculus Rift Development Kit 2.0:

The Oculus DK2 is a lightweight headset that tethers to a personal computer for a personalized virtual reality experience. It comes with a positional tracking unit to detect location changes of the headset down to small degrees of rotation or movement. Extra lenses and a wide assortment of power adapters suggested it was designed for a world market. It requires two USB inputs, preferably one at a 3.0 level, and can achieve a frame rate in excess of seventy per second.

The company Oculus has been at the forefront of the affordable VR push since the beginning. Their development kits have kindled massive interest in the technology across multiple industries, from military simulations to gaming and even medical training. Until now, human beings could only move through virtual worlds in their own minds. Now they can use multiple senses to externalize this unreal space.

Our team chose this company and this device because it had been available on the market when its competitors were still trying to push out their first device. Due to that time already in use, the DK2 was available through second-hand avenues such as Amazon and eBay. The VEDIC group needed to verify the plausibility of this product as quickly as
possible, and the DK2 proved the answer in both cost and time to receive. It's for these reasons the DK2 was purchased and is now in use in developing the product.

**Leap Motion Controller:**

Perhaps even more impressive than the VR headset, is the company Leap Motion with their compact controller that combines infrared and machine learning to map the real human hand into the virtual environment. Coupled with their Orion software, these virtual hands can interact with the elements in virtual space in general terms and also with specific, predetermined gestures.

There were a number of competing products for VEDIC to use. The Myo band measures the flex and release of muscles in the forearm or bicep. The XBOX and PS4 controllers offer multiple buttons and two miniature joysticks that allow the user a wide arrange of input without needing to ever look at the device. Emotiv Epoc is a device worn on the head, translating EEG into usable computer input. Lastly, the tried and true mouse and keyboard.

The Myo band was quickly ruled out as a viable option as its price was five times that of the Leap, and offered a fraction of the input in return. The XBOX and PS4 controllers are still in the running as tools for our developers version, but didn't fit our collective vision for an immersive IDE. A user would have trouble using these devices to traverse data and program in 2D. The trouble would only be exacerbated in a third dimension.

Mouse and keyboard are still useful tools for the programmer, but they rely on one
important physical sense—sight. The moment a user lets go of either of these devices in order to move or adjust their VR head set, they're forced to feel around until they've found them again.

Though programming with thought patterns would be an evolutionary jump in this endeavor, there were three important factors that kept the Epoc from being a worthy candidate: the cost was exorbitant—as much as the DK2—it would require significant training time for one user to do rudimentary tasks in VEDIC, and since it's a headset, it would be competing for the space that the DK2 already occupies.

Unity 5 Engine:

There were many 3D graphics engines with VR capabilities to choose from, but the Unity Engine had a number of characteristics beneficial to VEDIC: Its native use of the C# language, a wide and knowledgeable support network, and numerous tutorials made learning the system faster than it would have been with its counterparts. Also, all three VEDIC developers have had some experience working with the Unity Engine in the past while little to none on the others.

Visual Studios:

The irony in needing an IDE when creating an IDE does not escape the developers of VEDIC. Nevertheless, a robust IDE was required, and one that could work with C# at that. Visual Studios, though difficult to acclimate to initially, presented as the best choice because of its ability to integrate with both Github and the Unity Engine. With built-in crossover, it's anticipated that the project will benefit from reduced development time.

PHP:

This dynamically typed language was selected to serve as a supporting server-side business logic layer for receiving the queries constructed from within VEDIC and sent to the database. Asp.Net was a competing choice, and while its native C# connection would seem to make it the logical choice, it was decided that PHP could provide a modular interface in the event that this software expanded to allow multiple languages and query syntax.

MySQL:
SQL remains unchallenged as the most commonly used database architecture and MySQL stands as one of the most widely accepted dialects for that structure. Since VEDIC is meant to benefit as many developers and data technicians as possible, we chose MySQL to ensure fewer people would need to change the structure of their data in order to take full advantage of this system.
Challenges

Oculus Runtime Environment Issues

In order for the VEDIC development process to be as friendly and efficient as possible, the consideration of having a portable environment to run the Oculus headset is a serious factor for the team. In order for realistic testing of code within the Unity Engine, the ability to be able to test in real time the changes that we make in the environment as they are experienced inside the headset is crucial. A lot of headache and turmoil was spent in attempting to get it all ready and willing to work. As it stands, the team currently doesn't have a "portable" means of doing so. Many of these reasons stems from a lack of budget, while others are reasonable issues with Oculus themselves.

It's without a doubt that virtual reality is a heavy beast to run properly. This is not due to the actual physical configuration of their product being difficult to understand. Rather it is the changes of the Oculus direction as a company that has put a big unexpected roadblock in the middle of VEDIC's path toward easy development. So far, Oculus has managed through what seems like unnecessary deliberation to cut compatibility to a community of developers and consumers alike by restricting access to their platform due to sub-par set ups and unsupported hardware configurations. There are many factors that play into this situation, so it's important to document what is known thus far in order to help prevent others who may stumble upon similar issues, as well as provide the collaborated groundwork that will hopefully lead to a solution.

The issue that the VEDIC team is having in particular is establishing a solid configuration of a portable laptop environment to work properly with the Oculus Runtime Environment and the Oculus Development Kit 2.0. Setting up the system is fairly simple, although the instructions for properly installing and configuring this set-up is surprisingly sparse. As it stands now, the Oculus Website has terrible ways of managing all the installation pieces that have to be put together in order to get this working. Their documentation is lost in the web with no intuitive way of actually finding it through browsing their interweb. Normally, we would have to Google search actual applications and SDK's to install because their website has not shown us any means of accessing those pages starting from the homepage. This lack of organization that Oculus seems to foster with their documentation is rather annoying, but it was managed.

The current system that has to be installed in order to get Oculus to work is an installation
of their latest SDK for Oculus development, and one of their many Oculus Runtime Environment versions. At the time of setup and attempt to configure, the latest Runtime environment was version 0.8. The issue with using these runtime environments is that what they weren't readily advertising is their lack of support or compatibility with users who were running systems with unsupported graphics cards and architectures. They would rather users did not know of this unfriendliness towards their user base until it is too late.

Oculus Then and Now

In order to understand why our attempts at laptop configuration are not functioning properly requires a small look at the history of Oculus development over the last few years. More importantly, the way Oculus has increased performance requirements to run their system and how that has evolved. As it stands now, Oculus has had many versions of their headset, most worked as prototypes, where only three really had any consumer availability. The three main headsets are the Oculus Development Kit 1.0, Oculus Development Kit 2.0, and The Oculus Rift (Current Retail Version). During the earlier days of Oculus, it seems that a lot of the technology of what was running the Oculus didn't have to break someone's bank. Earlier iterations of their headsets only allowed working minimum requirements that standard gaming PC's were able to hit. However, as Oculus began building newer and newer prototypes, those requirements began to change.

Oculus Headset Comparisons

<table>
<thead>
<tr>
<th></th>
<th>Oculus DK1</th>
<th>Oculus DK2</th>
<th>Oculus Rift</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution (Total)</td>
<td>1280 x 800</td>
<td>1920 x 1080</td>
<td>2160 x 1200</td>
</tr>
<tr>
<td>OLED</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Refresh Rate</td>
<td>60Hz</td>
<td>75Hz</td>
<td>90Hz</td>
</tr>
<tr>
<td>FOV</td>
<td>110</td>
<td>100</td>
<td>110</td>
</tr>
</tbody>
</table>

*Formatted information comparing iterations of Oculus Headsets [6, 13]*

Current Oculus Requirements
It makes sense that as a product begins its development and gets to later and later stages, that the technology that can run them gets greater and greater. There was a time where Oculus was able to run at a DK1 standard where it was only required to have 60 frames per second as well as having a low resolution output. So effectively, you would conclude that at one point in time, Oculus was able to run on lower end machines than the current specifications. Even before the May 2015 announcement of Specifications for the Oculus Rift Experience, Oculus had a working environment that had relatively no issues with different architectures and lower spec PCs. They were able to supplement, even if a sub-par experience was what the user was getting. However, the pains that are currently being experienced now is a change of mentality that came soon after the May 2015 Specifications announcement for their retail version of Oculus Rift, even more so at the beginning of 2016.

It seems that Oculus started to realize that an experience that only ran at lower frame rates and graphical fidelity was not going to cut it. To them, in order to experience virtual reality in its true glory, machines were expected to be pushed to their limits. Along with announcing their new specifications, which now will hold for all future iterations of devices, Oculus had this to say.

"The goal is for all Rift games and applications to deliver a great experience on this configuration. Ultimately, we believe this will be fundamental to VR's success, as developers can optimize and tune their game for a known specification, consistently achieving presence and simplifying development." – Oculus Blog, May 2015 [10]

This also appears solid. They are drawing a concise baseline so that developers now know what tools they have to work with on this new Oculus Platform. However, there is a lot of discrepancy and issues that arose as the launch of the Oculus Rift started to draw
nearer. This is especially noticeable in the following year, early in 2016, when Oculus began hauling major updates to their current application environments. These environments started to prevent older clients of Oculus that developed on the DK1 and DK2 to meet with the newer requirements as a minimum instead of as an optimal. This not only effects low spec users, but also laptop architectures. It seems that the reasoning behind changing the way their runtime environment worked was in order to prevent sub-par experiences from ever being able to occur.

Earlier this year, news about being unable to run setups of the Oculus Environment with new updates to their older runtimes started to become an issue. Zak Lyons, the human-computer interaction team at Bath University, says that "most modern machines are capable of running VR" but goes on to say that not all "...can run it well" [14]. Sam Watts, a developer in VR added that due to peoples complains and issues of Oculus not working properly on sub-par machines that Oculus "just stopped supporting it" [14]. It is to a point where Oculus Runtime 0.8 will stop working if it recognizes any sort of rig setup that it no longer supports anymore. This change in flow has hurt our ability, along with many others, to be able to develop on a platform like Oculus.

How this Affects VEDIC

Now with laptops concerned, the reason why these are now currently not suitable setups are their inability to be able to push recommended graphical thresholds to an external device. Most of this stems from laptops' native hardware limitations. Laptops are meant for portability, which normally means using cheap components and battery saving techniques within their architecture that prohibits power consuming tasks, like rendering past 60hz. Also, laptops normally have an integrated graphics card from the supplier to handle basic operations while reserving a dedicated graphics card to render to heavier stuff like videogames and movies. This internal logic that the laptop was designed in order to compensate tasks for two systems now raises flags inside the Oculus Runtime environment that ceases actual use of the Oculus from ever happening. For example, some laptop setups have a hardwired connection from their Intel integrated graphics card directly to the external HDMI port, which Oculus utilizes to run. Other annoyances that laptops generally have are internal components and software that determines which graphics card will be running which application. Oculus has decided to remove any support for devices that run like this, even if their specifications match the requirements listed.

One of the hurtful things about this is that one of the developers of VEDIC invested in a new laptop in order to run the Oculus DK2. As it turns out, news of this was just starting
to surface, and the team was not aware of these issues. The team knew that the laptop environment may not have been best, but getting something roughly close to a working experience would allow for ease of testing in any setting. Instead, when the Oculus is plugged in, and all systems seem to be running fine, the dreaded issue many people are facing is this.

There is a scattering of workarounds that attempt to solve this issue. Because everyone's situation is unique from laptop to laptop, many are never working solutions. Many solutions actually have warnings, something along the lines of making sure the system is backed-up before trying this trick. VEDIC understands the need for Oculus to force certain restrictions, but to completely remove any form of compatibility that existed for older iterations feels unjust. What makes this even tougher is that since the only compatible runtime environment with windows 10 is the 0.8, the team is left with limiting options to roll back to an older environment, unless a workaround exists. One of the developers currently had the Oculus Rift working on a Windows 7 laptop, but unknowingly had been forcefully upgraded to Windows 10 OS. The Oculus then stopped working on that laptop as well. The team is working at the possibility of rolling back this upgrade of Windows 10 and seeing if that works.

Now as a fallback plan, one of the VEDIC developers, has a desktop at home that although being far away, can run the Oculus Runtime Environment fine. It may get to the point where the team may need to discuss bringing in the Tower to the Senior Design Lab in order to get proper testing accomplished. However, this is a last resort. The team is currently working on other solutions that may provide the desired results. One of the current tests to run is using an alternative DVI port from the laptop. According to a few
sources, using a *Mini Display Port to Dual-Link DVI Adapter* from Apple may help facilitate the process of pushing out faster frame rates of 120 hz, more than enough to satisfy Oculus environments. (YouTube Video) This will be tested in the weeks approaching summer.
Budget & Expenses

Assets Acquired

- Oculus Development Kit 2.0  –  $550.00
- Leap Motion Controller x 2  –  $70.00
- PS4 Controller  –  $60.00
- XBOX One Controller  –  $60.00
- Total  –  $740.00

Projected Acquisitions

- Code School Tutorials  –  $50.00
- Leap Motion Keyboard  –  $100.00
- Myo Band Controller  –  $200.00
- 3D Assets  –  $100.00
- Total  –  $450.00

Projected Total

- All Assets  –  $1190.00
Milestones

Research Phase (March 13th, 2016)

**Unity (All Developers)** - February 28th, 2016 [Complete]
Learning the Unity environment through tutorials and the creation of demo projects.

**Visual Studios (All Developers)** - February 28th, 2016 [Complete]
Learning the Visual Studios IDE by using it alongside the above mentioned Unity projects.

**Visual Programming Languages (All Developers)** - March 6th, 2016 [Complete]
Reading the existing literature and research papers on the subject and looking at "current system" examples.

**Unit Tests (All Developers)** - March 6th, 2016 [Complete]
Learning how to program unit test in general C#, as well as how to do so specifically in Unity.

**3D Math & 3D Graphics (All Developers)** - March 6th, 2016 [Complete]
Re-learning the basics of math and graphics in three dimensions through reading academic literature.

**Unity Customization (All Developers)** - March 6th, 2016 [Complete]
Watch tutorial Videos on how to customize the Unity edit environment, and follow along in demo projects.

**Parsers & Scanners (All Developers)** - March 13th, 2016 [Complete]
Re-learn how to program parsers and scanner through class notes, and additional reading on C# versions.

**Transpilers (All Developers)** - March 13th, 2016 [Complete]
Learn how transpilers are programmed, and research if SQL and/or C# versions already exist.

**Visual Parsers & Scanners (All Developers)** - March 13th, 2016 [Complete]
Reading the existing literature and research papers on the subject and looking at "current system" examples.

**Visual Transpilers (All Developers)** - March 13th, 2016 [Complete]
Reading the existing literature and research papers on the subject and looking at "current system" examples.
Oculus, Leap, & Myo APIs (All Developers) - March 13th, 2016 [Complete]
Read the official documentation on these systems' API, and watch tutorials on current uses of them.

UX Design for 3D Environments (All Developers) - March 13th, 2016 [Complete]
Read existing literature, and watch tutorial Videos on how to design intuitive user interfaces in 3D.

Design Phase (April 17th, 2016)
Target Audience & Wireframe (All Developers) - March 13th, 2016 [Complete]
Layout the look, feel, and expected use of the VEDIC home site.

Activity & Sequence Diagrams (All Developers) - March 20th, 2016 [Complete]
Expected activities of users, and the order of events they activate/encounter, followed by expected outcomes.

UML Diagrams (All Developers) - March 20th, 2016 [Complete]
Programmatic outline of classes, attributes, and methods of the program every step of the way (all states).

Parser/Scanner SQL (All Developers) - March 27th, 2016 [Complete]
Outline the flow and functions involved in the scanning and then parsing chain of events to/from an JSON string.

Transpiler SQL (All Developers) - March 27th, 2016 [Complete]
Outline the flow and functions involved in the transpiling of parsed tokens into quantities of prefab object types.
**Parser/Scanner C# (All Developers) - March 27th, 2016  [In Progress]**
Outline the flow and functions involved in the scanning and then parsing chain of events to/from a C# file.

**Transpiler C# (All Developers) - March 27th, 2016  [In Progress]**
Outline the flow and functions involved in the transpiling of parsed tokens into quantities of prefab object types.

**Function Graphs (All Developers) - April 3rd, 2016  [In Progress]**
Identify the modular components in a function declaration, design each in C#, and outline their relationships.

**Logger (All Developers) - April 3rd, 2016  [In Progress]**
Outline specific components of a log, and where the data is stored. Outline the infrastructure for those logs.

**Concept Art (All Developers) - April 3rd, 2016  [Complete]**
Commission a series of high-end, graphic designs of the four modes of operation in stylized fashion.

**Visual Design & Workflow (All Developers) - April 10th, 2016  [Complete]**
Create an organized matrix of visual shapes/colors that associate with specific classes/tables/variables/etc.

**User Input Handler (All Developers) - April 10th, 2016  [Complete]**
Pre-define the keyboard keys, mouse events, and hand gestures to their respective in-program responses.

**Error Detection Handler (All Developers) - April 10th, 2016  [Complete]**
List out the predicted scenarios a user could go wrong, and prevent those actions or design the responses.

**Unit Tests (All Developers) - April 17th, 2016  [In Progress]**
Outline the template pattern of a unit test, and layout the specific instantiations of all other tasks.

**Documentation - Forms (All Developers) - April 17th, 2016  [In Progress]**
Outline the template of the forms used in our future documentation. An online form could handle all tasks.
Implementation Phase - Database (June 12th, 2016)

**HTML/CSS Website (William Funk)** - March 13th, 2016 [Complete]
Build the layout and style of VEDIC informational landing site.

**JavaScript Website (William Funk)** - March 13th, 2016 [Complete]
Implement the front-end functionality of the VEDIC informational landing site.

**Backend Website (William Funk)** - March 13th, 2016 [Complete]
Create the php backend for the VEDIC informational landing site, and create the rudimentary SQL database.

**Demo Database (William Funk)** - March 13th, 2016 [In Progress]
Setup and populate an SQL database for use in the VEDIC database modes during testing and presentation.

**Database Communication Layer (William Funk)** - April 3rd, 2016 [In Progress]
Create the php/.NET backend layer for the demo database for testing and presentation.

**UI View Mode (William Funk)** - May 1st, 2016
Create assets for, and implement said assets, into the database view mode.
UI Edit Mode (William Funk) - May 15th, 2016
Create assets for, and implement said assets, into the database edit/create mode.

Create the functionality that manages how objects move through the 3D space relative to a user input.

Create Visual Cue Resources (William Funk) - May 15th, 2016
Develop animated visuals to illustrate the various events taking place within VEDIC’s 3D environment.

Implement Visual Cue Resources (William Funk) - May 29th, 2016
Place the visual cue resources in their respective event handlers.

Documentation (William Funk) - June 12th, 2016
Complete the requisite forms throughout the life of this phase.

Landscape View Mode (Jorge Rodriguez) - May 1st, 2016
Create the non-interactive 3D backdrop for the view modes.

Object Relationship Matrix (Jorge Rodriguez) - May 1st, 2016
Program the matrix against which all objects assume their relative positions and relationships with each other.

Landscape Edit Mode (Jorge Rodriguez) - May 15th, 2016
Create the non-interactive 3D backdrop for the edit/create modes.

**3D Mesh Factory (Jorge Rodriguez)** - May 15th, 2016
Implement the automated creation of 3D objects associated to the transpiled JSON string.

**Function Graph (Jorge Rodriguez)** - May 29th, 2016
Program SQL logic involved in the function graph that will attach to the GUI components in its UI counterpart.

**Error Detection Handler (Jorge Rodriguez)** - May 29th, 2016
Implement the catch logic for user errors, and relay relevant response to user visually/audibly.

**Create Sound Cue Resources (Jorge Rodriguez)** - June 12th, 2016
Develop audio assets to signal to a user that various events have taken place within VEDIC's 3D environment.

**Implement Sound Cue Resources (Jorge Rodriguez)** - June 12th, 2016
Place the visual cue resources in their respective event handlers.

**Database Implementation Phase – Work Time Estimate (Jorge Rodriguez)**

[Diagram showing work time estimate]

**Keyboard/Mouse Handler (James Vinson)** - May 1st, 2016
Program all listeners relevant to valid keyboard keys, mouse clicks, and mouse movements.

**Parser/Scanner (James Vinson)** - May 1st, 2016
Implement the SQL scanner and parser used to tokenize the JSON string incoming, and the 3D objects outgoing.

**Transpiler (James Vinson)** - May 1st, 2016
Implement the SQL transpiler used to turn tokenized SQL data into 3D prefab object types.

**Parent/Child Adoption (James Vinson) - May 29th, 2016**
Implement the structure to associate an object placed inside another object as child of parent.

**Application State Manager (James Vinson) - May 29th, 2016**
Program the transitions from one mode to the next based off of a user-instigated event.

**Logger (James Vinson) - May 29th, 2016**
Code and set in place the system for log creation for debugging purposes.

**Session Storage (James Vinson) - June 12th, 2016**
Create a saved state system for the VEDIC environment to make loadable later.

---

**Database Implementation Phase – Work Time Estimate (James Vinson)**

---

**Implementation Phase - Programming (November 13th, 2016)**

**Leap Handler (William Funk) - July 3rd, 2016**
Implement hand gestures and their resulting outcomes. Backtrack to add these to the database modes, too.

**Position Management System (William Funk) - July 17th, 2016**
Create additional functionality that manages how objects move through the 3D space relative to a user input.

**Demo C# Project (William Funk) - July 17th, 2016**
Create a C# project to be used later as input for loading C# files into VEDIC.
Create Visual Cue Resources (William Funk) - July 31st, 2016
Develop additional animated resources illustrating various events taking place within VEDIC's 3D environment.

Implement Visual Cue Resources (William Funk) - July 31st, 2016
Place the additional visual cue resources in their respective event handlers.

UI View Mode (William Funk) - June 26th, 2016
Create additional assets for, and implement said assets, into the view mode for the programming environment.

UI Edit Mode (William Funk) - July 17th, 2016
Create more assets for, and implement said assets, into the edit/create mode for programming environment.

Presentation Script (William Funk) - November 6th, 2016
Scripted lines for all three developers to use during the final presentation.

Object Relationship Matrix (Jorge Rodriguez) - July 3rd, 2016
Program the matrix against which all objects assume their relative positions and relationships with each other.

Error Detection Handler (Jorge Rodriguez) - July 24th, 2016
Implement the catch logic for user errors, and relay relevant response to user visually/audibly.

Function Graphs (Jorge Rodriguez) - August 21st, 2016
Program C# logic involved in the function graph that will attach to the GUI components
in its UI counterpart.

3D Mesh Factory (Jorge Rodriguez) - September 11th, 2016
Implement the automated creation of 3D objects associated to the transpiled C# files.

Create Sound Cue Resources (Jorge Rodriguez) - September 25th, 2016
Develop more audio assets to signal to a user that events have taken place within VEDIC's 3D environment.

Implement Sound Cue Resources (Jorge Rodriguez) - September 25th, 2016
Place the visual cue resources in their respective event handlers.

Documentation (Jorge Rodriguez) - October 9th, 2016
Complete the requisite forms throughout the life of this phase.

Presentation Montage (Jorge Rodriguez) - October 23rd, 2016
Assemble a Video montage of clips and images from dev team, in VEDIC environment, etc., for presentation.

Presentation Demo Database (Jorge Rodriguez) - November 6th, 2016
Create database for specific use in final presentation to perform specific queries during live demo.

Presentation C# Demo (Jorge Rodriguez) - November 13th, 2016
Create C# project for specific use in final presentation to perform specific queries during live demo.

Programming Implementation Phase – Work Time Estimate (Jorge Rodriguez)
Keyboard/Mouse Handler (James Vinson) - June 26th, 2016
Program all additional listeners relevant to valid keyboard keys, mouse clicks, and mouse movements.

Parser/Scanner Input (James Vinson) - September 11th, 2016
Implement the C# scanner and parser used to tokenize incoming C# files.

Transpiler (James Vinson) - October 30th, 2016
Implement the C# transpiler used to turn tokenized C# data into 3D prefab object types.

Parent/Child Adoption (James Vinson) - August 7th, 2016
Implement the structure to associate an C# object placed inside another object as child of parent.

Application State Manager (James Vinson) - August 14th, 2016
Program the transitions from one mode to the next based off of a user-instigated event.

Logger (James Vinson) - September 4th, 2016
Code and set in place the modified system for log creation for debugging purposes.

Session Storage (James Vinson) - October 2nd, 2016
Create a saved state system for the VEDIC programming environment to make loadable later.

Parser/Scanner Output (James Vinson) - November 13th, 2016
Implement the C# scanner and parser used to tokenize the outgoing 3D objects, converting them to C# files.
Sources


Acosta Duarte, Andrea. Concept Artist and Web Designer [15]

Sources w/ URL:

http://blog.leapmotion.com/designing-vr-tools-good-bad-ugly/ [3]
http://alexchu.net/Presentation-VR-Design-Transitioning-from-a-2D-to-a-3D-Design-Paradigm [7]
https://www.google.com/design/spec-vr/designing-for-google-cardboard/physiological-considerations.html# [8]