CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

SERIOUS AND PURPOSEFUL

VIDEO GAME ENVIRONMENTS

A graduate project submitted in partial fulfillment of the requirements
For the degree of Master of Science
in Computer Science

By

Jeremy Staples

May 2011

The graduate project of Jeremy Staples is approved:

Peter Gabrovsky, Ph.D. Date

Robert McIlhenny, Ph.D. Date

G. Michael Barnes, Ph.D., Chair Date

California State University, Northridge

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

SERIOUS AND PURPOSEFUL
VIDEO GAME ENVIRONMENTS

By

Jeremy Staples

Master of Science in Computer Science

Games that aid people in a serious way are commonly called Serious Games. A number of publicly released and highly available Serious Games exist today. These games are specifically designed to provide people with entertainment, but at the same time also assist them in their important and beneficial duties. The focus of this graduate project is on video game systems that are within the realm of Serious Games. Additional focus is given to a new genre of video game that falls within the broader scope of Serious Games called “Games with a Purpose”. Purposeful games are designed to collect data that is generally difficult to generate using computer algorithms without human interaction (playing). An in-depth look at the online video game Second Life is also presented. Second Life is a massively multiplayer online video game that has applications to Serious Gaming. A look at the future of social networking aspects of the 3D online environment is presented, as well as other interesting aspects of Second Life including its building process. An analysis of the Second Life scripting language is provided from the unique aspect of a programmer new to Second Life, who wants to use it to develop a Serious Game. A first-person account of developing a 3D game simulation in Second Life is presented, along with its relationship to computer science and its potential educational benefits as a Serious Game.

Introduction

This section is an introduction to serious and purposeful video game environments from the standpoint of someone who is totally unfamiliar with the two topics. It assumes no previous knowledge of gaming, but does bring up a small number of computer science-related topics that are assumed to be understood in order to preserve the relevance of the graduate project to the field of computer science. Topics that are discussed briefly in this section include a short description of what serious and purposeful video game environments are, what makes them special and also a number of examples of serious and purposeful video games are presented. We'll discuss why serious and purposeful video games are potentially important to society and how they might affect the world as we know it in the future.

Serious Games

Flight simulators used today by commercial and military pilots effectively test a pilot's flying skills without ever having to leave the ground. These flight simulators, along with tank operating simulators, train conducting simulators and other vehicle simulations are just one small genre (or sub-category) of Serious Games called Simulation Games [21]. Shooting games intended for law enforcement to test an officer's accuracy and reaction time, as well as his or her abilities in proper judgment and restraint are another example of this particular genre of Serious Games. However, a wide range of genres within the Serious Games genre exist [20]. There are games that teach players non-violent alternatives to conflict [4] and on the other end of the spectrum there are games that teach strategies and train players in the art of self defense [25]. Some Serious Games educate players in certain helpful and humane ways of addressing the numerous struggles of the third-world, such as hunger, sanitation concerns, health, housing and other major concerns that affect humanity [2]. There is a Serious Game that teaches civilians how to keep their homes and properties safe from flooding [9] and even a Serious Game that simulates a Darfur refugee camp [19]. In fact, Serious Games can be considered a humanitarian subject to some extent - simply because so many eco-friendly and humane Serious Games have been developed [1]. However, the real definition of a Serious Game is any game that is designed for a primary purpose other than pure entertainment [18].

Games with a Purpose

Games with a Purpose are another sub-category of Serious Games. The first known Game with a Purpose was called ESP. The game is shown in Figure 1 on page 8. It was imagined and created by Dr. Luis Von Ahn and then further developed by a group of Von Ahn's colleagues at Carnegie Mellon University. The ESP game is available for play at the GWAP.com website along with many other purposeful games [8]. The GWAP website is shown in Figure 2 on page 8. The game pits two randomly chosen players against each other in a race to find the best words to describe the various images that appear randomly on the screen. The game is played over the Internet and all of the players have absolutely no means of direct communication with each other. However, each player is looking at and describing the exact same image on the screen as the other player or players in the game (although not necessarily at the very same time). Whenever both players use the same word to describe an image, it is considered a match and each player in the game receives points. The matched words between the players are recorded by the system. Rounds are short, the pace is fast and the points earned during each round are added to the players' overall score.

Behind the scenes, the "purposeful" goal behind the ESP game is covertly achieved. This is because the game is actually a rouse for a functional system that works continuously in the background during game play to label each of the images in the game. The images that are used for the content of the game get linked together with the keyword associations that are provided by the players as answers in the game. These associations are made whenever two or more players score a match for a word they answer. All of the images are systematically provided by Google Images and the images are rotated amongst the players automatically.

A tangible and beneficial output is derived out of game play automatically by the system. The ESP game discovers the digital associations between descriptive keywords and images. By storing the words in a database and using the words for later searches, the system "mines" the visual and analytical talents of the human players. The result is a real and measurable benefit to human beings in general.

The keywords are associated with the images automatically in the Google Images search page, so progressively the act of searching for images using the highly popular Google search engine becomes faster, more effective and more accurate over time. Keywords are continuously created for more and more images as they are added to Google Images and as the players continue to play the game using the new content. This continuous process of updating the game with new images and updating the search engine with more accurate keywords makes both the players of the game and the users of Google Search happy.

Human Computation

State of the Art

Since their creation, video game systems have primarily been used by human beings for the simple purpose of entertainment and just to “pass the time”. Actual real-world productivity very rarely has been one of the major concerns of video game producers, manufacturers or developers. However, this may be changing. The work of Dr. Luis Von Ahn in the field of human computation (also known as machine learning) has ignited interest in the topic of purposeful video game environments and a wide variety of these purposeful video games have been created. Von Ahn’s contributions to the genre initially stemmed from academic research along with several published papers on the CAPTCHA security system along with its applications to cryptography [3]. In 2004, he and a team of students at Carnegie Mellon University developed the previously described ESP online video game system - known today as the Google Image Labeler. His work culminated in 2006 with a series of specially-designed, “purposeful” video games. His research in the field of human computation, as well as the art of building purposeful video games continues to this day. For more information on the work of Dr. Luis Von Ahn, visit Von Ahn’s homepage [13] and research page [14] at Carnegie Mellon University.

Human computation can be thought of as the somewhat simple concept of outsourcing to human beings all or part of the data processing needed for tasks that computers are currently unable to solve adequately on their own [27]. It is well known that some problems and tasks cannot be solved by computers, computer algorithms or computer programs. Various other problems and tasks can be solved or performed to some degree by computers, but not efficiently or accurately. Computer systems have poor perceptual recognition capabilities.

Some simple, real-world logic that is easily understood by human beings (even young children), cannot be understood by the most advanced computer systems in the world. Perhaps the most popular example of such a problem is the CAPTCHA user recognition systems employed throughout the Internet for use in resolving whether a remote user that is logging into a system is an actual human being or rather a computer system disguised to falsely portray a human being. Humans can usually differentiate the letters and numbers in the images that are produced by the CAPTCHA system relatively easily. Computers however, cannot adequately perform this differentiation - at least in general not accurately or efficiently enough so that the CAPTCHA system performs its simple function with a high level of success. It logically follows from the generalization of the CAPTCHA system that many of today’s computer systems fail to solve perceptual and analytical problems that human beings typically consider to be easy.

For example, many modern-day robots still stumble over obstacles that are strewn across the floor when walking and many of the common digital voice recognition software systems used for telephone answering systems are often only mildly dependable and are frequently bypassed immediately by their users out of pure frustration. However, human beings can almost effortlessly navigate through a cluttered floor and can in most cases, easily differentiate simple, clearly-spoken directives voiced over a telephone connection. That is not to say many advances in both of these areas haven’t been made. On the contrary, much progress is being made in mitigating each of the limitations to these systems on a number of computer systems. However there is still much progress required in order to find reliable and consistent solutions to these problems.

Fig. 1: The ESP Game (Adopted by Google as the Google Image Labeler) [22]

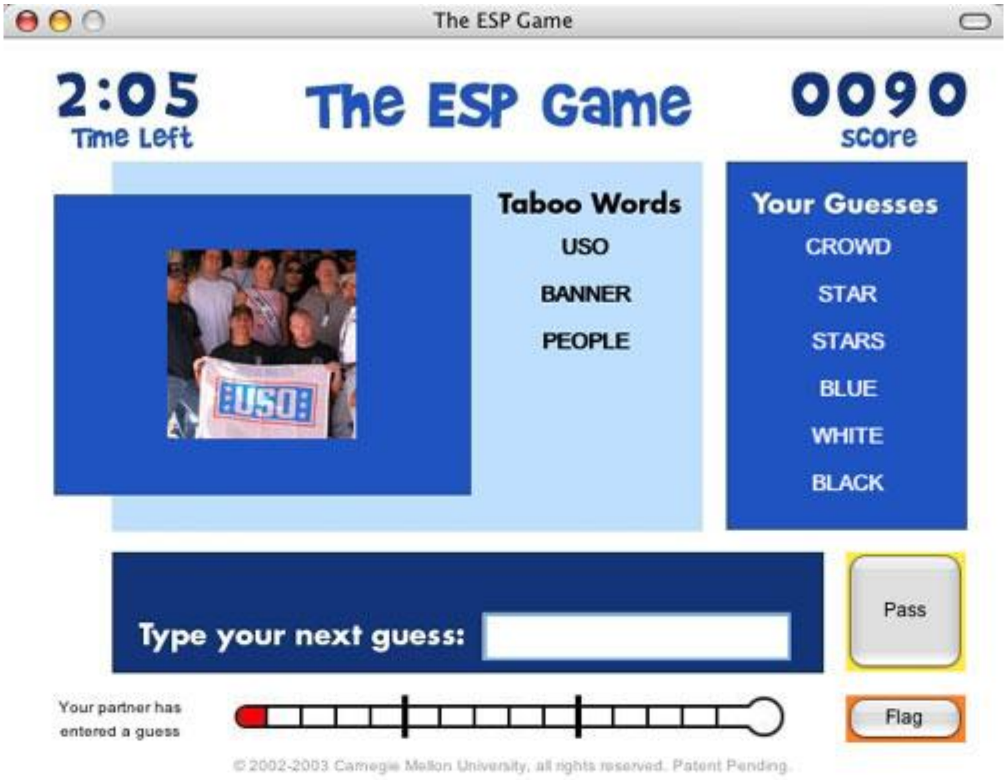
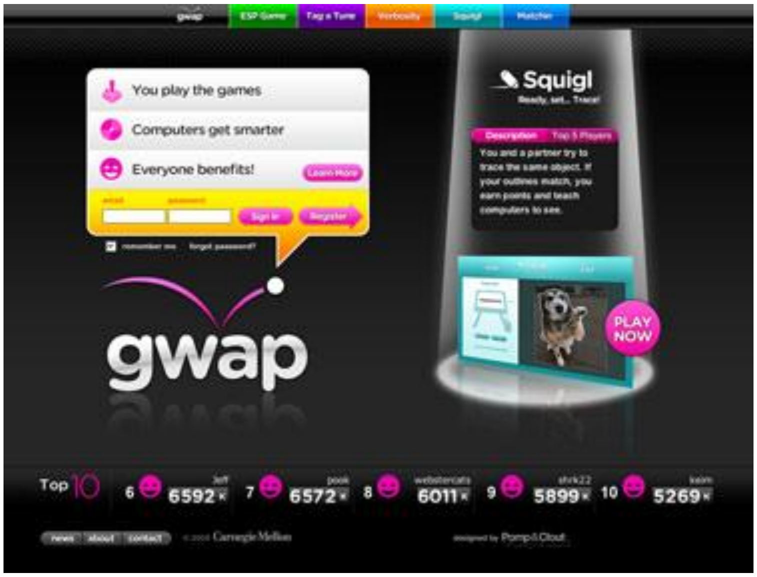


Fig. 2: The GWAP.com Website [23]



The Potential for Human Computation

Still other well known problems and tasks on the other end of the spectrum have been proven to be unsolvable by any Turing Machine or other computer device. The Church-Turing Thesis states that if a problem cannot be solved by a Turing Machine, it can never be solved by any computational machine [5]. These unsolvable problems are collectively known as “undecidable problems”. Undecidable problems can never be solved by a computer system, no matter how much we improve our understanding of computers or advance our technical or computational resources. Computational theorists have historically backed up this claim and the Church-Turing Thesis has been generally, if not universally, accepted within the computer science community. To be perfectly clear, there is no evidence presently that suggests human computation will ever make any known undecidable problems decidable.

In purposeful games, the useful data that is “mined” from human beings is stored by the computer system for further analysis and processing of the results. Human beings interface with computers when playing the game and the computer system collects and processes the results of the human computation. In this way, problems that were perhaps not seen as computer science problems previously because of their complexity can ultimately return to the general realm of computer science and computational theory. This might occur because some aspects of the problem that were not solvable by any existing computer system could potentially bottleneck the overall task as a whole. The simple application of human computation may lead to solutions to problems that have one or more resolvable bottlenecking task. Thus these tasks might potentially become solvable via machine learning.

Quantifiable units of human work as units of “human computational resources” might even be developed theoretically. “Joules” of human work can arguably be quantified as a standardized unit of human computational energy, just as the work done by computers can be quantified in units of computation time, levels of resource usage or as a percentage of the overall computational load. Perhaps a unit would be difficult to establish and agree upon at first - as can arguably be said for any standardized unit - but this notion is possible. Unfortunately, the work performed by human beings through human computation has largely been ignored in the past. This is because, at least in today’s world (and in the past as well), whenever a person plays a game or solves a puzzle, the computational energy expended by the player (the energy of human computation) has always been inherently wasted. Any idea of recapturing that energy has been of little or no concern. This happened in the past quite simply because there was never a perceivable benefit or tangible positive result that could come from solving any ordinary game puzzle or by playing any ordinary game. One exception to this rule might be the mental benefit of learning something new or practicing a technique within the game - whether it is truly useful in a tangible way or not. The fact that no real benefit is produced simply by playing a video game arguably has always been the case throughout history. However, with a game that is both fun and at the same time purposeful, we might finally yield a quantifiable positive outcome from play. This may occur simply by utilizing the inherent human computational assets that have always been there, but have up until now always been “wasted” when playing the game.

Direct human-computer interaction is usually an essential aspect needed for purposeful games to work because purposeful games are specifically designed to monitor and collect the data that can potentially be “mined” when humans make decisions. Presently, computers alone are not capable of systematically processing digital photographs in order to give a real world, “human context” to the images. In other words, a computer today simply cannot analyze a picture in any way and say for example, “Okay, this is a picture of a cat”, or “In this picture there are three people playing poker on the beach”. Human beings however can do this rather easily and accurately and are in fact very good at this type of simple problem solving. Thus through purposeful games and machine learning, the computer systems interface with the images themselves via human beings. The cycle completes when the humans interface again with the computers to access the results of a search. The outcome of this process is making the search engine experience considerably more enhanced at Google, as the system has been permanently adopted by the company.

Of course, one obvious limiting factor to the theory of human computation is that unlike computers, human beings usually require some form of compensation for their work. They demand some kind of reward for their time. However, human entertainment in the form of playing a video game for fun solves this problem fairly well. Therefore, one may consider the entertainment factor of a purposeful video game to be a catalyst for human computation and machine learning. In other words, the more fun that a purposeful video game is to play, the more likely it is that people will want to play the game. It follows that the more likely the purposeful video game is to be played, the more likely it is that some potentially useful data will be collected from the system, and thus more problems can be solved through human computation [28]. The potential level of computational power that is possible in the future if a purposeful video game were to be adopted as one of the most popular video games played around the world could indeed be quite staggering.

Second Life as a Serious Game

Second Life [26] is a massively-multiplayer, online, OpenGL-based, virtual 3D video game environment that allows its players to not only explore and interact with its immersive in-game world, but also create and develop most of the online 3D content in the game by building, modeling, sculpting, customizing, sharing and expanding upon the majority of the game environment itself. Players extend the games as they play. They can interact with each other’s creations and explore each other’s creativity through interactions with the creations that they make. All newly-created and modified objects or other user content in the Second Life game are updated almost immediately for all other players to see in the game world and thus new content becomes ready for interaction in near real-time. Second Life gives its players the ability to script the actions of the objects that they create, providing the behaviors that determine how their creations will interact in the virtual world. Players create all of the specific actions that their objects exhibit in the virtual world by using the state and event-driven LSL scripting language (Linden Scripting Language). The LSL scripting language gives script writers the option to react to any of the typical events that occur within the game. For example, if a player touches an object that is scripted to change colors when it is touched by a player, the touch event will fire an action controlled by an event handler and the object will change colors.

The LSL scripts in Second Life give players and developers the rare opportunity to create virtual 3D game environments. These game environments can then be experienced by other players. Game environments in Second Life can even be developed to demonstrate and teach basic learning concepts to other players in an online educational setting. Going a step further than just advocating Second Life as a potential educational environment, the initial hypothesis proposed by this graduate project is that Second Life may even be an excellent environment for quickly and easily creating Serious Games. Like all other Serious Games, these games can be played for fun, but at the same time they can also be used to teach valuable lessons to the players. Furthermore, purposeful games may be quite suitable for Second Life. For the applied portion of this graduate project, an attempt is made to teach computer science basic concepts to beginning and prospective computer science students through the use of simulations in Second Life.

Second Life itself can arguably be considered a Serious Game that teaches basic computer science concepts. This is simply because Second Life introduces the LSL scripting language to its users as a way to create considerably more advanced and stimulating in-game content compared to unscripted 3D models in the game alone. For example, a player might want to script a pet dog that follows them around on walks in Second Life. Playing the game entices its players to learn to program in the language. However, in an attempt to go one step further than this, we might also be able to expand on the Serious Games-related aspects of Second Life by writing various computer science concepts into the LSL script code and then demonstrating these basic concepts to players within the Second Life world. We may also be able to create objects that visually demonstrate some of the concepts in computer science that might otherwise be difficult to grasp for new students.

Another interesting view of Second Life in the context of a Serious Game arises when we consider the unique potential for Second Life to potentially become the next great paradigm for social networking. Presently, the Facebook social networking website greatly dominates the Internet in terms of popularity insofar as social networking is concerned, as well as in regards to the levels of sheer traffic that the site generates online. Facebook’s market share is expected to hold steady at around 60-70% according to a study by DreamGrow Social Media [24]. It is perhaps safe to assume from this fact that as of today the average person prefers their social networking activities to be done in a 2D menu-driven graphical user interface environment. However, in the future a large number of people may adopt a 3D game engine-like environment such as Second Life for their social networking environment of choice [30]. Second Life and IMVU [10] currently hold the majority of the market share for online 3D social networking websites that are not primarily the forum for a more traditional style of online video game. If the future trend in social networking really does move in the favor of the online 3D game engine-like paradigm, then it follows reasonably that a large number of users and a large amount of profitability might also end up shifting away from the 2D Facebook world and inevitably gravitate toward the companies that manage to dominate the market for the 3D social networking paradigm first. It is quite likely that Facebook itself will invest heavily into this paradigm seeking a competitive stake in this new market as well.

If we examine this concept just a little further we may attempt to consider all angles of our potential vision for the Second Life environment in becoming a leading paradigm for social networking environments. Indeed there are several additional aspects that make Second Life the perfect fit for a 3D social networking environment. One aspect that makes Second Life attractive as an emerging social networking environment is that Second Life provides a conduit for its users to think creatively to build and script “outside” of the 2D box. Clearly, a 2D social networking paradigm like Facebook is a logical place to start for a populace still in the process of fully adopting the idea of social networking in general and slowly transitioning from a 2D interface to a 3D interface. As users gain experience they may migrate heavily to the virtual world of interactive 3D social networking environments.

Second Life certainly has more to offer its users than Facebook as an online social networking environment. Second Life provides a deep and expansive world to explore amongst friends with an enormous number of potential experiences available to its users. Facebook, with its simple 2D paradigm, limits its users to a rather small world comparatively – a world consisting mainly of the simple facets that have been put in place to facilitate its fairly standard online communications outlets through social networking. Second Life gives its users an additional opportunity to shape their own private area of the Second Life world, so other users are empowered to visualize any concept or idea that their imaginations might put forward. This is why learning is not only possible, but also highly facilitated in Second Life.

Real-time voice chat capabilities already exist in Second Life. Video chat is still another logical progression for the foreseeable future in Second Life. Organizing people in groups or rallies in Second Life mimics the same processes in real life. Sports events and other games involving teams or large numbers of people are possible in the game. Social debates and meetings are similar to those occurring in real life. However in Second Life each of the participants can be thousands of miles away from each other. The serious and important social aspects for remote 3D virtual interactions in real-time that are potentially possible are far-reaching. These capabilities and interactions make Second Life inherently a Serious Game.

The Process of Creating a Serious Game

The intent for this graduate project is to examine the process of developing a Serious Game in a reasonably timely and efficient manner. In the beginning of the design phase, it was decided that Second Life would make a productive environment for developing a Serious Game. At the same time it would also allow us to explore the principles of Serious Gaming even further through its constructs. However, other options for the proper video game environment were considered as well. Programming the game using XNA Game Studio [32] using Microsoft Visual Studio C# was the first idea. Then the possibility of using OpenGL [17] with C++ in Microsoft Visual Studio was considered as a second option. The use of one of a number of popular 3D game engines was still another option that was considered. However, Second Life was found to be advantageous over all of these choices for a development environment, because as opposed to many of the other options available, the lower-level functions of the game would not have to be developed. Instead, the existing low-level functionality and the relative reliability of the development tools already available in Second Life (and free to use for its players) could be used for rapid development and powerful demonstration of the main ideas behind Serious Gaming.

The following section is a general account of the development experience written in a first person context. The overall goals were to describe the major difficulties encountered in designing the project, to identify the recognizable computer science-related principles in Second Life and to show what was accomplished during the development of the project.

From Humble Beginnings

Second Life was described to me a very short time before beginning work on the graduate project. At that time I knew I needed a development environment for putting together a serious or purposeful video game demonstration. I opened an account just to see what Second Life was all about. As a self-professed "gamer", I had for many years intentionally avoided massively multiplayer online video game environments. That is because in order to be a functioning member of society, I simply could not justify playing such addictive, time-consuming and expansive video games. To engage in such an environment for a graduate project appeared to me as a rare and delightfully unusual opportunity. One of the first things that I realized when I played the game was that I certainly had a lot to learn if I was going to make significant progress in the time I had allotted to me for completing the graduate project. I had to quickly come up to speed in the Second Life world and at the same time I needed to have enough material to demonstrate the core principles that I wanted to address for the graduate project.

Second Life begins for all new players in a small tutorial area of the Second Life world that is intended to show new players the "basics" of Second Life. The tutorial area shows players how to view objects, walk, run, dress, chat, sit, touch, etc. You quickly learn that Second Life has a vernacular of its own. It is a completely separate world than our own with its own unique language and culture. Your player is called an avatar or "avi". As far as general appearances are concerned, you can make your player look any way you want for the most part. All standard human races, genders, body shapes and hair colors are available for you to customize your avatar. You have an endless number of choices available for dressing your avatar as well. However, even these mildly unconstrained considerations may still be thinking “inside” of a very small box that Second Life really doesn’t restrict its avatars to fit into.

In fact, if you want to be an enormous demon with giant devil horns, you can be one. If however you instead want to be a miniature purple and green dinosaur with cartoonish features that breathes fire and blows plumes of smoke out of its nostrils, you can be this avatar as well. You can be almost anything and everything that you want to be in Second Life, as long as it is within a number of guidelines for functionality (as well as decency). Really it is this relative lack of restrictions on the player that is one of the main points of the game.

Why walk around in Second Life when you can fly? That’s right. Another useful thing to know in Second Life is that gravity holds no bounds for those who choose to take to the air. This ability gives players an excellent opportunity to explore the tallest skyscrapers, fly above the trees or go anywhere else in the world that they might want to go - even high above the clouds. The sky is quite literally the limit. You can easily see over the roofs of houses and fly through open windows several floors up. These abilities, as well as many other abilities that one might consider somewhat strange or unusual for people to do in the real world, are all very normal things in Second Life. You can do a lot of other things in the Second Life world that aren’t possible or easy to do in the real world as well, such as teleport to new places, move through certain objects and move objects remotely without ever touching them.

However, at the same time you have a lot of limitations on your form that don’t really exist in real life. For example, sitting is one of the most important things you can do in the Second Life world. Sitting on objects in Second Life can change your whole view of the world, start a chain of scripted animation processes or just allow you some time to get off your feet. Sitting is actually very powerful in Second Life, because it is one of a limited number of events that both an avatar and an object share together. This allows the event, the object and the avatar to be associated together so that the event can be handled in the object script using the LSL event handling functions. The same principle applies to touching objects. Sitting and touching objects is a common way for avatars and scripts to interact with each other and the rest of the world. So needless to say, avatars in Second Life do a lot of sitting and are constantly touching things.

If you want to have something in the Second Life world, you can probably have it - but it might cost you. Second Life has an economic system that is based upon the Linden Dollar. Linden Dollars can be purchased in the Second Life world using real world currency. Once an avatar has a number of Linden Dollars, they can purchase an enormous number of objects for their own personal virtual enjoyment. Players can buy articles of clothing, jewelry, shoes, accessories, swords, guns, motorcycles, cars, furniture, buildings, etc. Anything in Second Life for the most part can be bought using Linden Dollars, including land. Auctions take place regularly to sell off items and parcels of land. Shops and store fronts abound for avatars to buy items with whatever expendable Linden Dollars they might have and players can spend vast fortunes in Linden Dollars on a seemingly endless selection of things to shop for.

But why pay real money for Linden Dollars to buy things in Second Life? It’s not really necessary. All you really need in Second Life if you don’t have any real money on hand is some talent. Generally speaking, a few rather useful talents that you’ll want to have if you want to make a decent amount of Linden money in Second Life are artistic talents, 3D modeling talents and scripting skills. This is because all of the content in Second Life is user-created in some way or another. Let’s say you just bought the perfect kitchen table for your Second Life home. Well, you could have just made it instead of buying it and as a matter of fact, before you bought that table, somebody did make it and then most likely sold it to somebody. It is true that in Second Life, all Linden money and virtual capital is essentially created out of thin air, albeit virtual thin air. This is why the creators of Second Life (known collectively as the Lindens) are able to make real money in the real world.

Every player in Second Life is essentially a “god” of sorts. You can build objects by simply materializing a number of primitive objects out of thin air and then modifying these objects to your liking. Building an object in this way is called “rezzing” the object. The word originally appeared in the first Disney’s Tron movie. (The Tron universe obviously resembles the Second Life universe in a number of ways.) Every object in Second Life is made up of one or more of these simple shapes called “prims”. Prims are the smallest unit of digital matter in Second Life. Objects can be highly complex, consisting of many hundreds of prims, or as simple as a single prim object with no colors or textures applied. The basic prim in Second Life, once created, has a default light yellow wooden texture.

I learned the basics of Second Life as quickly as I could by following the Second Life Wiki [29]. I explored several regions and chatted with many other players in the game to get my bearings as quickly as possible. In fact, I tried to make friends with as many other advanced players as possible - at least the best that I could. Many of these players immediately informed me that I was an obvious “noob”. For those of you that are not familiar with this term, being a “noob” means you are a newbie at the game and you are (usually quite obviously to the elite players), not very familiar with or comfortable with your online game surroundings. I was told that I lacked the proper clothing, the hair, the features, the walk, the talk, the objects, accessories and the animations of an advanced Second Life player. I didn’t have a powerfully customizable viewer in order to experience the world in its full glory and I didn’t have a single Linden Dollar in my virtual pocket. I definitely had a lot to learn if I was going to make the necessary progress I needed to make in this brand new world, and I needed quite a lot of time for exploration.

Building in Second Life

Building things in Second Life is a relatively simple, yet highly powerful process. Prim objects can be rezzed and then modified from their original state by stretching, scaling, moving, coloring, texturing and more. Building in Second Life is shown in Figure 3 on page 26. Linking prims together allows for many highly complex objects to be created in the virtual 3D space. Gravitational forces and other Newtonian physical rules can also be applied to objects as well as some otherwise impossible physical rules. In addition to building simple inanimate objects, scripts can be written and added directly into the contents folder of any object in Second Life. Editing a script is shown in Figure 4 on page 26.

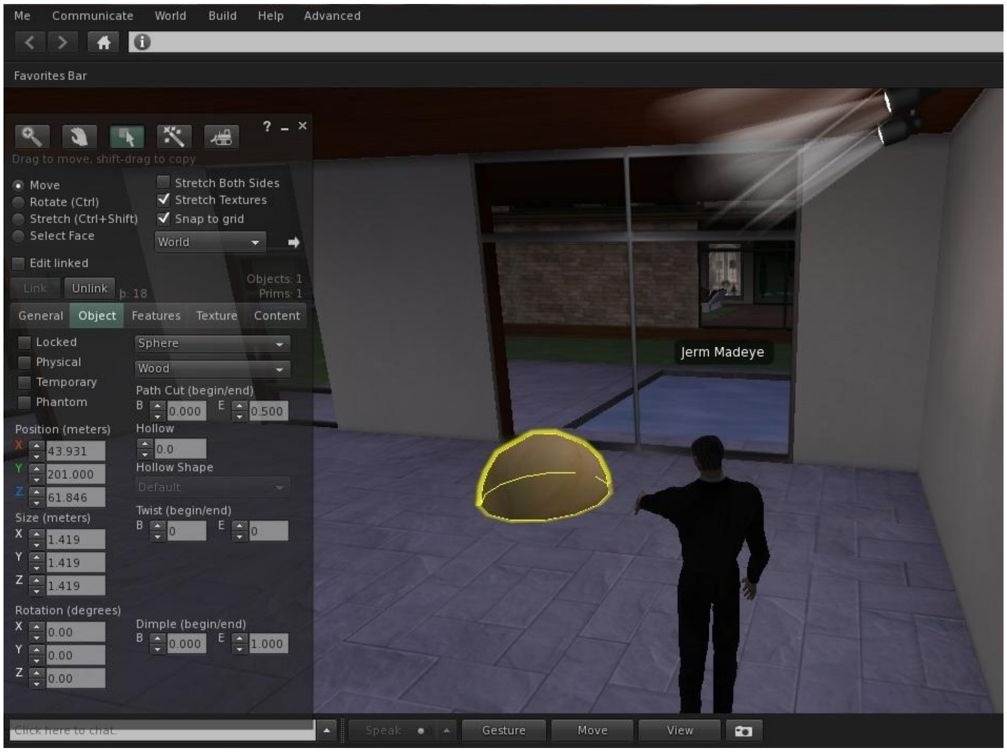
Not only can the scripts themselves be added to the contents folder of an object, but also other objects (which can have their own internal scripts) can be added to the contents folder of an object. When a script is added to the contents of an object, the script will be automatically executed when the object is rezzed. Objects that contain other objects in their contents folder can rez those objects as separate objects in the world at any time and this process is possible any number of times. However, these objects can never technically be linked together as a part of the rezzing parent object. In other words, parent prims cannot rez child prims. It is possible to create child prims and then link these prims to the parent, but the child prims must exist at the same time as the parent prim it is linked to and this cannot be done as part of a script. An avatar can link the objects together but an object script cannot. Rezzing is essentially an instantiation of one complete object with all of its child prims. This is a lesson that I eventually learned the hard way in my scripting attempts, because I spent a lot of time trying in vain to create an "object rezzer" script that could be contained in the contents folder of my parent prim yet would rez and link the child prims that were needed for the script whenever it was needed. What I was attempting to do was unfortunately impossible. (It makes perfect sense why this is not allowed if we consider the possibilities of recursion and its potentially unwanted side effects when not properly handled.)

Basic Functionality

Various special communications methods exist in LSL for message passing between child prims, between child prims and their parent prim and between the parent prim of one object and the parent prim of a separate object. In addition to this, non-chat methods of communication are also possible in Second Life using LSL. One such method of chat-free communication is through the use of the LSL library function llSetText(). This function displays information in a textual form above the object in the Second Life world. This simple method of information communication can be highly useful in debugging scripts and can also be used effectively for presenting information to an avatar visually. Objects that contain LSL scripts can chat messages back and forth with other communicative scripts running separately in the same object, or an object script can chat outwards to scripts that are contained in other objects in a number of ways.

Some initial scripts were developed for the graduate project specifically with the intention of aiding students in their understanding of computer science concepts. These scripts were written for Second Life using the LSL scripting language and each was intended to demonstrate, either visually or fundamentally, a number of basic concepts in computer science. The scripts were written with the intent of publically demonstrating the objects in Second Life as well as having the objects shared, transferred or modified. It was also intended for certain sections of the script code to be openly displayed around the object to explain the concepts behind the code. While several scripts preceded this LSL script, the following script called the Prim Color Sort Game (in-game, known as the “ButtonConsole” object), became the main focus for the applied portion of the graduate project.

Fig. 3: Building an Object in Second Life

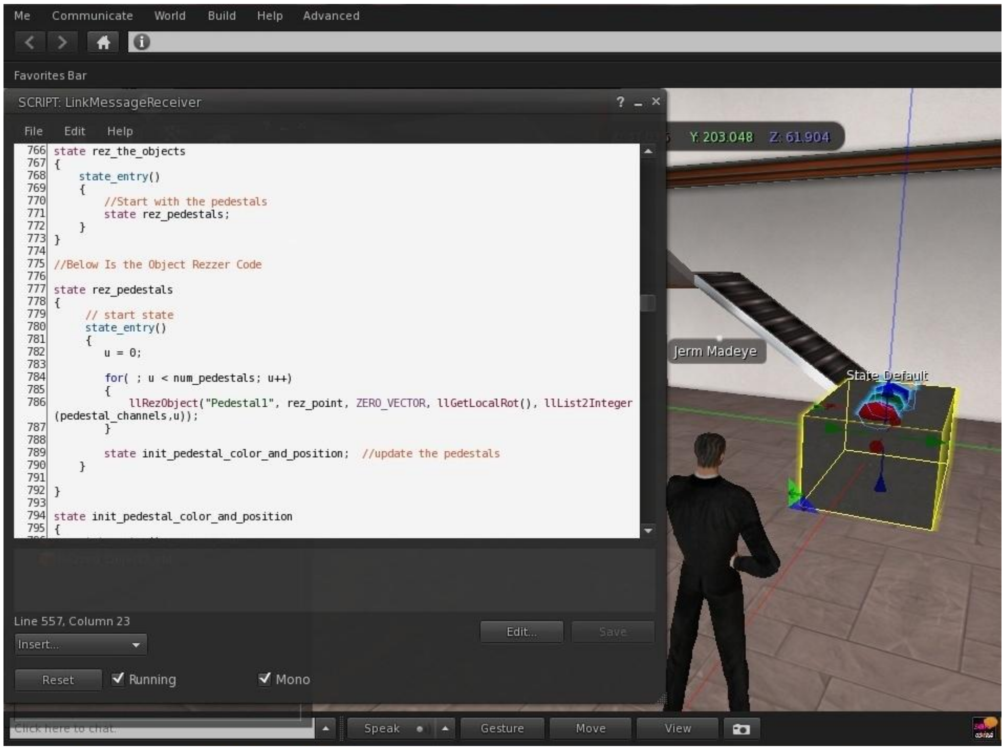


● Move
○ Rotate (Ctrl)
○ Stretch (Ctrl+Shift)
○ Select Face
□ Stretch Both Sides
☑ Stretch Textures
☑ Snap to grid
World
□ Edit linked
p: 18
Objects: 1
Prims: 1

□ Locked
□ Physical
□ Temporary
□ Phantom
Sphere
Wood
Path Cut (begin/end) B 0.000 E 0.500
Hollow 0.0
Hollow Shape Default
Twist (begin/end) B 0 E 0
Dimple (begin/end) B 0.000 E 1.000

Position (meters) X 43.931 Y 201.000 Z 61.846
Size (meters) X 1.419 Y 1.419 Z 1.419
Rotation (degrees) X 0.00 Y 0.00 Z 0.00

Fig. 4: Editing an Object Script in Second Life



SCRIPT: LinkMessageReceiver

```
state rez_the_objects
{
    state_entry()
    {
        //Start with the pedestals
        state rez_pedestals;
    }
}

//Below Is the Object Rezzer Code

state rez_pedestals
{
     // start state
     state_entry()
     {
        u = 0;

        for( ; u < num_pedestals; u++)
        {
            llRezObject("Pedestal1", rez_point, ZERO_VECTOR, llGetLocalRot(), llList2Integer
(pedestal_channels,u));
        }

        state init_pedestal_color_and_position;  //update the pedestals
    }

}

state init_pedestal_color_and_position
{
```

Line 557, Column 23

The Prim Color Sort Game

The Prim Color Sort Game is a simulation in Second Life that was created to demonstrate the feasibility of building a Serious Game within the world of Second Life. The simulation was built in a number of stages over the period of one semester. The following section describes the simulation, how it functions and what it attempts to prove as a simulation specifically designed with the intention of being built and scripted as a Serious Game. A view of the Prim Color Sort Game can be seen in Figure 5 on page 30.

Initial Functions

To play the Prim Color Sort Game in Second Life, move your avatar to the location of the Prim Color Sort Game’s control box via teleport or other method. Walk to a good viewing area. A distance of about 2-3 meters from the control box is preferred for game play.

Located on top of the control box is a series of buttons. One button is cylindrical and consists of a stone-like texture. This button is the “GO” button. The “GO” button has three basic functions. One of its functions is to start the game after an RGB color is selected. Another function of the “GO” button is to solve the puzzle automatically for the player at any time during game play using the sort algorithm. Finally, resetting and ending the game can be done using the “GO” button. This action will stop the game from running until the “GO” button is pressed again.

The three “RGB” buttons located on top of the control box are round and colored red, green and blue. To start the game simulation, touch one of the “RGB” buttons to make a color selection of red, green or blue. You can do this by left clicking the “RGB” button with the mouse or right clicking the “RGB” button and selecting the “touch” option from the popup menu that appears. Whenever an “RGB” button is pressed, it will glow in the color of the selection made. After an “RGB” selection has been made, press the “GO” button to start the game.

Game Play

The game begins when the “GO” button is pressed. Eight pedestals and eight prim sort objects are rezzed in a row across the front of the control box. Each of the pedestals has a prim sort object on top of it. The prim sort objects are semi-randomly colored in a shade of the selected RGB sort color. The object of the game is for the avatar to order the prim sort objects in a progressive order from left to right using the lightest shades of the color on the left of the stage area and the darkest shades of the color on the right of the stage area.

As the avatar moves the prim sort objects around, the pedestals underneath them turn either red or green. A green colored pedestal indicates that the prim sort object sitting on top of the pedestal is in the correct position for the properly sorted order. A red colored pedestal indicates that the object on top of it is not in the proper position to solve the puzzle. The object is to sort the objects and make each of the pedestals turn green.

The avatar can swap any two prim sort objects at a time by selecting the pedestals underneath them. To select a pedestal, the avatar simply needs to touch the pedestal. When selected, a pedestal will glow. When two pedestals are selected, the prim objects on top of them will swap spaces automatically. The avatar can continue swapping objects as much as they want. Once the proper order is achieved, a message will appear indicating that the game has been won.

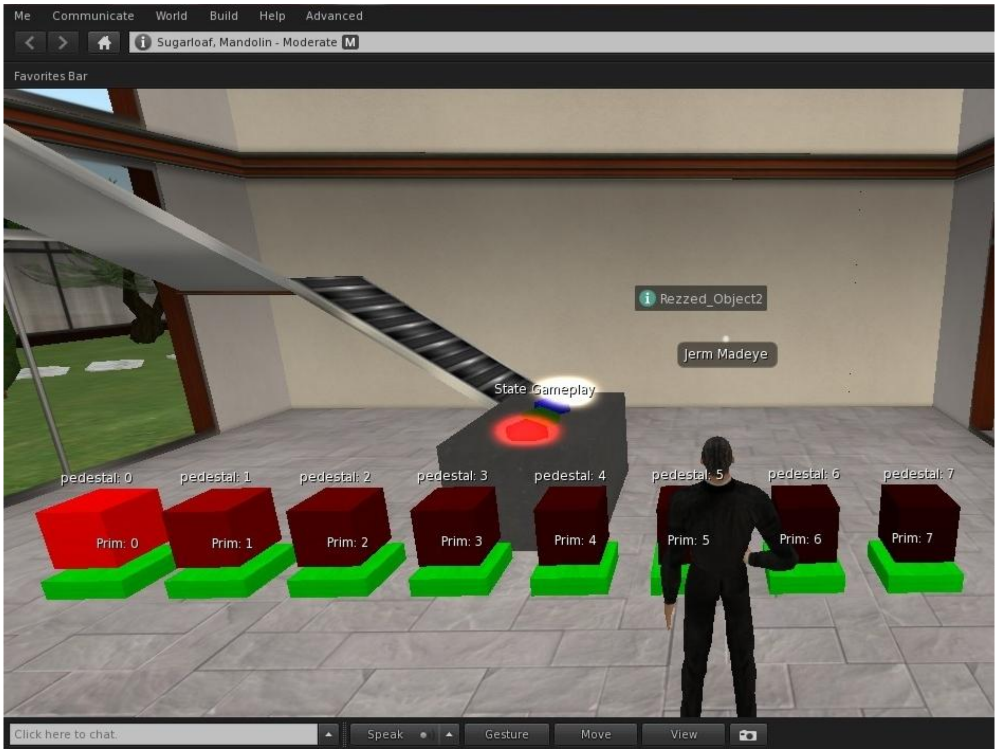
If at any time the player wishes to solve the puzzle automatically, they can press the “GO” button and the prim sort objects will be sorted using the sort algorithm. When the game is solved, pressing the “GO” button again will end the game. When the game is over, all the pedestals and prim sort objects will disappear and the control box will go back to its initial state.

Purpose

The game is intended to teach basic concepts of computer science to beginning and prospective students that might be interested in the subject. Since the game is constructed using many of the same principles that computer science students learn, portions of the code from the game’s script are displayed on the whiteboards around the game. Avatars can view these displays to see the concepts involved in making the game and their fundamental purpose in computer science and computer programming. In addition to this, the game is available for copying and modification in case any other player wants to build on the structure of the game or view the game code. This allows computer science students to further discover the concepts involved in building the game.

The following section details how the project was envisioned from the very beginning of its design and shows the rough draft concepts and ideas that were examined. It also shows how several of the ideas were imagined and described before working through the major details and debugging many of the problems that were encountered during development.

Fig. 5 – Playing the Prim Color Sort Game



Drafting the Prim Color Sort Game

This section details the initial requirements of the Prim Color Sort Game. These requirements were described before any of the work was done on the project. The intent for this section is to show the entire process of building the game and the difficulties that were encountered during its development. Mainly we can see from this section many of the initial game requirements, some of which created problems during the development of the game. Later we will see what strategies were taken to resolve these issues.

Summary

The Prim Color Sort Game will sort a number of basic prims (such as cubes, spheres and pyramids) according to their color. The user will choose a red, green or blue value to sort (order) the prims according to, and then tell the in-game “machine” (or control console) that it should sort the prims in order by their value (from 0 to 255) for the selected color choice by pressing a button on the machine.

Model

The model for the prim sort algorithm will include a control box which may have buttons modeled into it for choosing the sort color. The sort color will be red, green or blue. The colored buttons will be activated when “touched” by the user and a light will glow at the top of the panel to indicate the color selected. A “GO” button will activate the sort procedure. Once activated, the “sort prims” will be ordered according to the selected RGB value. There will likely be a “stage” or platform area that each of the sort prims will order themselves on. This platform may even be enclosed to keep the user from reaching or interacting with the stage area. (Perhaps interacting with the sort prims will be restricted so that the algorithm will work properly. It’s unclear if this is necessary at this time.)

The main focus for choosing a sort algorithm is to use simple algorithms that even beginners can understand. This approach should be seen as an important theme of the overall development and presentation of the graduate project. If we can teach simple computer science concepts in our simulation, then our simulation game can successfully be considered a Serious Game.

Sort Algorithm

The algorithm for the sorting process will be a basic sort algorithm that is simple enough for beginners to understand. The sort algorithm may simply sort the object ID’s of the sort prims. Each time the “GO” button is pressed, the algorithm will be activated and will order the “sort prims” according to which of the color selection buttons has been pressed.

The sort algorithm should be posted on a “whiteboard” somewhere near the demonstration. The whiteboards in Second Life can easily present to the player a small textual capture that can be used for teaching. The whiteboards may also present a slideshow presentation as well for teaching the concepts in the project that can be demonstrated in a slideshow.

Arrangement Algorithm

Once the proper order for placing the sort prims is determined by the sorting algorithm, the next logical thing to do is to physically arrange the sort prims in their proper order on the stage. Moving or “translating” the sort prims seems to be the best idea, but we may alternatively want to delete the sort prims and then redraw them in order (as a simple backup option, in case translations present physical or spatial problems). Lower RGB numbers will move to the beginning or “left” of the stage and higher RGB numbers will move to the end or “right” of the stage. The algorithm may call each of the object ID’s iteratively and place each sort prim on the stage in its proper position. A sort prim in the spot that the algorithm is trying to move to will simply be moved to an empty swap location.

Proposed Solutions

This section describes some of the potential solutions that were put together midway through the project. Each of the ideas outlined in this section were proposed as possible solutions for achieving the goals that we initially set out as requirements for the graduate project during the requirements phase of the development.

ButtonConsole

ButtonConsole is an object created to perform the main functions of the Prim Color Sort Game simulation in Second Life. ButtonConsole will use a bubble sort algorithm called llListSort() to solve the game automatically or will allow the avatar to alternatively solve the puzzle manually. The construction of the game uses a number of computer science principles provided by the LSL scripting language itself. These include a unique 3D spatial-object-oriented programming paradigm, the use of computer algorithms, listeners, message passing communications, events, event handlers and also several basic programming constructs such as Lists, If-Then-Else statements, While loops, For loops, Functions, etc.

The ButtonConsole object will require a sort algorithm. This was previously considered in our design as a bubble sort that would be developed into the script. One simple candidate considered was the slow bubble sort llListSort() that is already included in the library of pre-built LSL script functions. A more in-depth description of this sort algorithm follows.

The llListSort() list sort is available through the LSL library. It performs a slow bubble sort of the list passed to the function. It returns a list in the properly sorted order. The following line shows a call to llListSort(…).

```
list newlist = llListSort(myList, 1, TRUE);
```

The 1 in the parameter list is the “level” of the “strided list” being passed. (We’ll discuss these later.) A 1 makes it a strided list of 1 which is just a regular list, but 0 or any integer below 1 works for all unstrided lists as well. The “TRUE” in the parameter list means that the sorted list will be returned in ascending order. “FALSE” would sort the list in descending order [12].

We can effectively sort the list of RGB values using the llListSort() algorithm, but this will not help us to have adequate material for demonstration in the learning portion of the requirements that we have imposed on the project. It was thought previously that a merge sort would be the best candidate for a sort algorithm because it is more efficient than the bubble sort and at the same time it is better for demonstrating our initial hypothesis in the project. Alternatively we may want to break down the process of the llListSort() function into pseudo code that can be demonstrated in our sim. (This approach became the method of choice in the graduate project, although it may not have been the best solution.)

Pedestal

The pedestal object is an object that is to be contained inside the contents folder of the ButtonConsole object. It is basically a platform intended to hold the prim sort objects but which also acts as a medium for the avatar to interact directly with the sim. There will be a pedestal object for every prim sort object and possibly one or more empty pedestal objects available in the sim as a temporary swapping space for the avatar or sort algorithm to make room when moving other objects around in the game play area.

When an avatar touches a pedestal object during game play, the pedestal object should glow to indicate that it has been selected. When two pedestal objects are selected at a time, the prim sort objects on top of the pedestal objects should swap. If an empty pedestal object and a non-empty pedestal object are selected, the object should move from its current occupied spot over to the empty spot above the other pedestal. The pedestals each get rezzed by the ButtonConsole object whenever the “Go” button is touched by the avatar.

Prim Sort Object

The prim sort object is another object contained in the contents folder of the ButtonConsole object. It is a simple prim object that gets colored to a semi-random shade of either red, green or blue, depending on which button is selected for the game. Prim sort objects cannot be interacted with by the avatar directly. Instead, the pedestal objects are used for any and all interactions by the avatar. The idea is to have the avatar sort the prim sort objects in order from the lightest shade of the selected color to the darkest shade of the selected color. However, if the player chooses instead, they can have the ButtonConsole object perform the sort.

State Diagrams

A set of state diagrams were drawn up in the middle stages of the development of the Prim Color Sort Game. These diagrams are located in Appendix A. The diagrams show the higher-level functions that the object was originally envisioned to perform and the states that it was meant to transition through. With so many states and with so many variables that could be affected while passing between each of the different states, it was felt that some documentation was needed for understanding what the envisioned state transitions would look like.

The first thing that needed to be documented in these state diagrams was the overall view of the ButtonConsole itself and its buttons. How would it handle button presses to setup the game, rez the pedestals and prim sort objects, as well as allow for game play by the avatar? At this particular point in the development, it was still undecided whether or not the game should also sort the objects by the direct command of the avatar through chat. It was also unclear whether the sort should be activated through the console controls or whether the sort routine would simply be used to check if the ordering was correct. This portion of the game was left out of the diagram.

The next thing that was needed for the state diagrams was a high-level diagram that would show the potential states the ButtonConsole object could be in when the player makes a selection for an RGB mode and begins the game. This view was separate from the view of all of the pedestals and prim sort objects. This diagram would help to sort out the various states needed in the script in order to properly execute button presses, which could be rather complicated when implemented with If-Then-Else statements. (Note that Second Life does not currently support switch statements.)

Finally, it was decided that state diagrams would be necessary to map out the state transitions for the initial stages of development of the pedestals and prim sort objects. The state diagrams for these objects were shown to have a half-completed functional model. At the time this was a good start at allowing for further development of the entire sim.

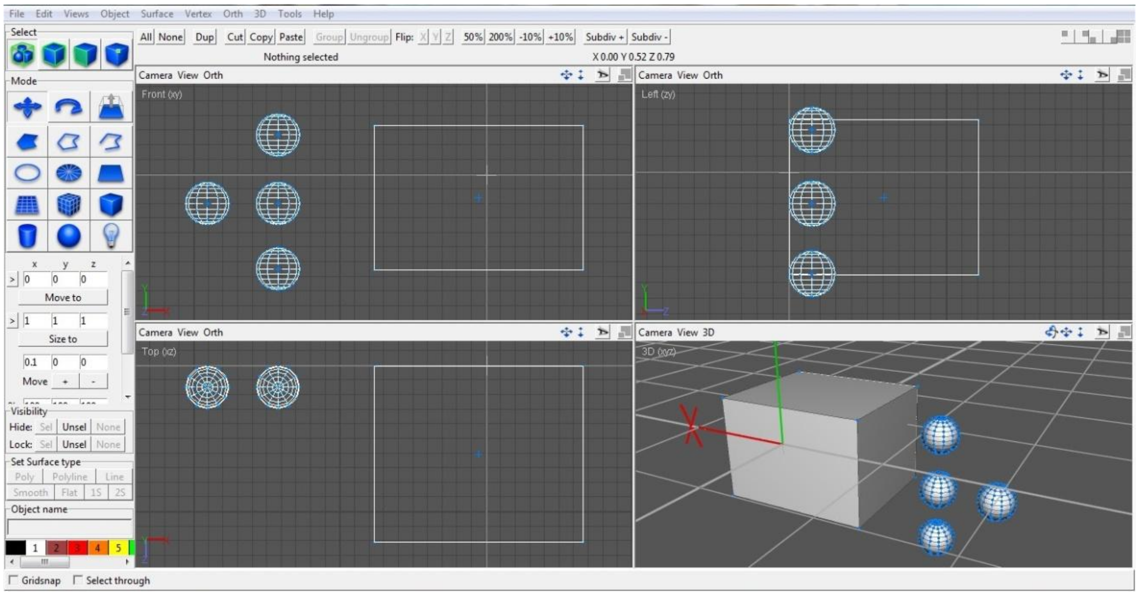
Final Project Solutions

Modeling in AC3D

The AC3D graphical 3D modeling program, which is downloadable from the Inivis webpage [31], was used to create each of the objects represented in the Prim Color Sort Game simulation. AC3D allows for its models to be exported directly as complete binary files for easy import into Second Life. Even complex, sculpted models known as "sculpties" can be imported into Second Life.

For the use of the files in the graduate project, we simply export from AC3D the various primitive objects that make up the complete physical objects in the game, The AC3D environment allows for much easier building and sculpting of the 3D models than the native build capabilities and environment that is automatically given to the players of Second Life in-game. Intensive sculpting development can be done to produce impressive 3D models that can be imported into Second Life. Each of these complex models can be exported directly from AC3D and imported into Second Life. A view of the process of building a 3D model in AC3D is shown in Figure 6 on page 40.

Fig. 6: Building 3D Models in AC3D



Lessons Learned

In the final stages of development of the graduate project, many lessons were learned and many aspects that were previously envisioned for the sim were emphasized. Others were de-emphasized or even scrapped completely during development. Some of the solutions that were decided upon appear in the following sections showing the process of development. We see many of the accomplishments that were made and implemented in the graduate project, as well as many of the complications encountered in the process.

Chat

Chatting in the Prim Color Sort Game became far more complex than was originally envisioned. In fact, it wasn’t even realized in the beginning of development that chatting would even be needed at all for this sim. However, it was quickly realized while building the control box that the buttons would need to be made into child prims of the parent object. The control box itself would become the parent of these prims. These prims would then need to be linked together and implemented to chat back and forth the various aspects of the game state and other status information about events occurring within the control box itself.

Problems Rezzing Child Prims

In addition to the control box itself, pedestals would need to be created with the prim sort objects sitting on top of them. These objects would be rezzed as separate objects from the ButtonConsole object. It was thought at the time that this would be a necessary consequence resulting from the unfortunate restriction within Second Life that no prim can be rezzed as a child prim of the rezzing object. The rezzed objects would have to be their own separate entities. Therefore two different forms of communication would be required. One form would handle chat internally within the ButtonConsole object and the other form of chat would be utilized for external chat between separate objects in the simulation.

Internal Chat

Internal chat within objects was accomplished through the use of link messages. Link messages are a form of chat that is available for all LSL scripts to communicate information between a set of prims linked together into a single object called a “link set”. Enumerated types are available that represent messages to and from the parent prim, between the child prims or between the entire set of prims in the linked set. Internal messages have several advantages over external chat messages. One major advantage is the removal of any potential for crosstalk. Crosstalk occurs whenever one object causes undesired effects in another object because both use or compete for the same message channel. When separate objects chat through an external chat conversation, crosstalk can often complicate things significantly or cause messages to be dropped when too many chat messages are being passed at one time for the system to handle. In addition to this, avatars and other scripts cannot listen in on any internal chat channels. Although negative chat channels can be used to accomplish this goal for avatars in external chat conversations, chat message channels themselves can be bypassed as a concern altogether using internal chats, since message channels are not used for internal chat conversations.

External Chat

External chat communication between separate objects was accomplished through the use of external chat messaging. To avoid potential crosstalk issues, a method for establishing random dedicated communication channels between the parent control box and the pedestal and prim sort objects was scripted. First, each of the pedestals and prim sort objects listen on all open chat channels for any message that they can hear. If these objects receive a chat message of any type, the script will check the UUID object key of the sender of the message and if it matches the object key of the object that rezzed it, then it will accept the message and parse out of the message contents. The selected message channel to use in all further messaging is contained in this message, as well as a number of other initial values that the control box needs to send out, such as the initial positioning vector and color information. After establishing a dedicated message channel, the prim sort object or pedestal stops listening on all other message channels and instead listens only on the established dedicated channel from that point forward. Any future commands that might be passed to it from the control box via external chat message will be sent on the dedicated chat channel.

Animation and Communication Concerns

Translating and animating objects became far more of a concern than was ever initially envisioned for the project. First attempts at rezzing the pedestals and moving them into their proper position in the sim failed for unknown reasons. The original idea was that the parent control box should store variables for the locations, spacing and numbering of the pedestals and prim sort objects. Before bringing the prim sort objects into the picture, it was felt to be a good idea to start out only working with the pedestals and then when that part was working, the prim sort objects could be introduced into the sim. Inheritance was the motivation in a sense, although inheritance is not supported in the LSL basic language. (Note that the “Mono” scripting feature is currently being developed to further support C++-like object-oriented programming constructs [16].) The idea was to copy and paste the basic functions from the pedestals over to the prim sort objects and then expand on their unique functionality.

One simple exercise that led to more experience with the animation functions of Second Life was to insert a simple sequence of Euler rotations into the script for handling touch events on the pedestals. After performing three Euler rotations, an added step to call llDie() in the script was included. The llDie() command deletes an object that has been rezzed. This helped with debugging the scripts as well, because whenever a pedestal would go “rouge” during debugging and stop responding to commands issued by the control box, it was easy enough to simply touch the pedestal and it would rotate a few times and then disappear. This was useful for putting together the script and was also a helpful exercise for understanding transformations in Second Life.

Initial attempts to move status information around for the intended states of the pedestals failed. Crosstalk and chat message buffer overflows were suspected causes. It was understood that communications would have to be streamlined in such a manner that it would allow for the least amount of information exchange necessary while using external chat messages and that all negotiations between the control box (as the rezzing object) and the pedestals would need to be kept to a minimum. It was decided first to package all of the information that was to be transmitted as a character-delimited string which would be compact, yet could still be parsed by the receiver to transfer a lot of useful information in just one message exchange. This technique yielded improved results, but still other spatial and communicative complications arose in further testing. One problem that was encountered was that objects outside of a short radius would sometimes not hear messages from the control box. Another issue that was found was that the pedestals would most often “walk” to their designated locations, but occasionally some dropped messages would cause stragglers to become abandoned by the process and then become lost to the overall simulation. Attempts at more complex animation sequences only complicated things further and were de-emphasized.

With some research, it was found that animations could be aided by using several open source scripts that are widely available on the Internet. These scripts seemed like a good guide, at least to help in animating the pedestals and prim sort objects in the face of some of the complications that had already been encountered. In fact, while researching these scripts further, it became obvious that spatially translating remote objects is a challenging aspect of scripting movable objects in Second Life. Luckily, these animation scripts were available, open source, and could be incorporated into the project in order to accomplish desired animations. These animations could really liven up the project, so despite the many problems encountered, it seemed like a good idea to keep animations in the overall plan structure for the final project. Maya 2011 [15] could even be used to prototype the animations. Then the Maya animation sequences could be used to help script out the final animations in the project using the available open source animation scripts.

For the next stage of the development, the scripts needed to be altered to remotely change the color of the pedestals using the chat commands from the control box. For the sake of efficiency and also to avoid dropping any chat messages, the position, color, glow intensity, object ID and llDie() status information could be sent in a single “packaged” message or separately. That way the ButtonConsole control box would be capable of sending multiple status messages if needed. When a string is passed to the pedestal via one of the pre-established, dedicated chat channels, the pedestal script parses out the message and assigns the correct primitive parameters to its own appearance. Depending on which color selection is made on the control box, the prim sort objects on top of the pedestals change to a semi-random shade of red, green or blue values.

The pedestals themselves need to change from red to green when the object that is on top of the pedestal is in the correctly sorted order and then they also need to go from green to red when the object is removed from the properly sorted order. The pedestals also need to glow when the avatar selects (touches) a single pedestal. When the player selects two pedestals in a row, both pedestals should glow for a moment, then the two objects should swap positions and the pedestals should stop glowing to show the player that a new choice is allowed and an action is being requested of the player. The objects on top of the pedestals should swap whenever any two pedestals are selected, but the pedestals should always remain in the same place. By implementing this simple sequence of primitive parameter changes, the player should know intuitively that swapping the objects is the required method for sorting the objects by color. Use of the pedestal glow could also be accompanied by a rotation of the object selected, so that the player feels a notion of control even further through the movement of the objects. The player should realize that the objects can be sorted simply by swapping pairs of objects in succession.

Since the player is given the chance to sort the objects manually, some idea of the difference between sorting the prim sort objects algorithmically versus sorting them using a person’s own mental intuition may become apparent. Sort algorithms don’t see the overall changes being made across all of the objects all at once. Instead they use an iterative movement across the objects to process each of the pairs or groups (buckets) of sortable objects together. It would be useful to point out to the player of the game that the algorithmic method of sorting has the distinct advantages of brute force processing and simplistic calculation methods, whereas when sorting using their own mental judgment, an actual human being has the advantage of intuition and past experience.

Despite the safeguards put in place to avoid them, communication problems persisted in the event-driven nature of the sim. It became clear that a state-driven message passing architecture was needed. A controlled, procedural and state-driven communication method was needed in order to ensure that for each of the potential states the objects could be in, the proper messages were being passed between the objects. It was also thought to be extremely helpful if the messages were acknowledged by the receiver.

All of the messages needed to go through to the receiver successfully before moving forward in the algorithm. To add this change to the script would be similar to using TCP rather UDP in OSI layer 2 computer networking communications [7]. The main difference between a controlled version of the message passing architecture (with message acknowledgements) versus any uncontrolled method is that messages are much more likely to be received and correctly handled on the receiving end. Message numbers could also be added to identify dropped messages. If acknowledgements are not received at any time, the object should resend the message repeatedly for a specific period of time. When eventually timing out, the system can safely return to a pre-determined fail state and then the failed objects should broadcast that they had failed to all the other objects in the sim.

To assist with the sorting of the objects it was decided that every pedestal would have a variable to represent the prim sitting on top of it. The reverse statement would also be true so that every prim sort object would have a pedestal number place holder as well. The control box parent script carries a list of the pedestal numbers in their standard order along with a list of the prim sort objects in their proper order. Knowing this order allows the parent to recover if any of the pedestals or objects should stop communicating. This method of timed communications is a sort of “dead man’s switch” to alert the sim of any failures. This method should prevent the total failure of the sim due to loss of communications with a pedestal or object. This simple principle can also be described on whiteboards around the sim as one of the required educational goals in the learning requirements imposed on the sim.

A message passing architecture was also created so that the parent or other objects in the sim can understand messages coming in dynamically. A message can be passed to change the color for example and then another message can be passed to change the glow of the object. The receiver parses the message with the special delimiter character (or separator) “/” to receive a message that alternates between an identifier message and a value message. So for example, if the parent script wants to change the pedestal color to green, it sends the new color vector “<0, 1.0, 0>” - where each of the values are a floating point value (rather than the 0-255 integers that were expected in earlier models) to the pedestal as “c/0/1.0/0”. This message after transit gets parsed to become the string “c 0 1.0 0” and through a simple control construct, the pedestal then knows to change its color to the new color as it is directed to do.

However, if the script needs to, it can also change the pedestal number and the glow intensity of the pedestal in the very same message by following the first message with “/g/1.0” which would then be parsed as “g 1.0” which represents a glow intensity of 1.0. The pedestal or object can then change both attributes for its color and glow intensity simultaneously. This method aims to prevent too many messages from being passed at the same time and in addition to this, the basic principle of parsing can be emphasized as a learning topic for students.

Listeners do not persist continuously through different states in Second Life. Regardless of whether the listener is needed or not for the next state, all of the listeners must be instantiated upon entry into a state and must then be removed when the state is exited. Listeners must always be deleted and then recreated upon entry to the next state, even if it is intended to do the exact same function in the very next state. Some adjustments to the listeners had to be made to properly handle the state transitions that occur in the sim due to this fact.

We swap the prim sort objects on top of the pedestals whenever an avatar selects two pedestals in a row. The pedestals glow when selected and whenever a pedestal is selected it sends a message to the parent using the pre-established message passing architecture. The message passing architecture is structured such that the sender sends a message to the receiving object and then establishes itself in a state where the sending object is then waiting to hear back from the receiver with a message that both represents an acknowledgement and also provides any requested information. For example, the acknowledgement may be an object number to trade with, the pedestal number that is selected or a simple acknowledgement.) The acknowledgement and information transfer only occur after the receiving object is notified to send the information from the sending object. In order to function properly, the control box must receive a message from both pedestals holding the prim sort objects to be swapped within a given timeout period. The necessary acknowledgements that are needed in order to achieve such a controlled technique would not be possible without a controlled message passing architecture.

State Diagrams

To determine the best working method of operation in the final stages of development for the overall sim, it was considered beneficial to create some basic state diagrams for each of the objects in each state within the overall flow of operation in the sim. The state diagrams in Appendix B. were developed in order to assist in the understanding of the state transitions as well as the communication requirements for the sim.

Deconstructing the LSL Scripting Language

The following section presents a brief deconstruction and analysis of the LSL scripting language. The effectiveness of LSL is evaluated by its intended purposes within Second Life. We also look at the advantages and disadvantages of the language in comparison to other well known and popular scripting languages. We'll look at which of its features are generally considered important in a scripting language and which important features are missing. We'll look at several of the unique aspects of the language that make it worth discussing along with its merits and misgivings. Note that many of the language constructs will be written with a capital letter in order to more clearly indicate its identity, whereas in the language it may be written in lowercase. So for example, a For loop as it is written in this graduate project appears as the lowercase word "for" when written in the LSL language.

LSL Fundamentals

LSL is a scripting language that includes hundreds of built-in functions for the script writer to implement directly in their scripts. The language itself is almost entirely C-like in its construction. Many of the typical C-based control flow statements (also known as control statements or guard statements) exist in the LSL language, such as If-Then-Else statements, For loops, While loops and Do-While loops. One notably missing statement however out of the common C-based control flow statements is the Switch statement and its generally associated sub-constructs.

Also missing from LSL are collection-controlled statements such as Foreach (read as the conjunction for-each) and Forall (read as for-all) which are found in C#, Perl, Java, Visual Basic, JavaScript, Ruby and Python [6]. However, the absence of these statements makes sense because “collection data structures” such as Structs, Arrays, Collections, Jagged-Arrays and Objects (in the object-oriented sense) are all not available in LSL. Instead of these data structures, the LSL language relies on Lists to supplement the user in cases where the previously-mentioned data structures might need to be used. Strided lists are an additional option that is available to LSL script writers to mitigate this problem. We’ll discuss strided lists some more in a moment.

The data structures that are available in LSL are the Integer, Float, String, UUID (key), Vector and Quaternion [11]. Integers and Floats are used for numeric calculations. Strings are used to hold ASCII words and sentences. UUID’s are unique variables given to Second Life objects that cannot change and can be used to identify the origin of chat messages sent by objects and avatars, amongst other things. Vectors can hold 3D coordinates within the Second Life world as either regional or local coordinates. Vectors can also represent RGB values. Quaternion values are used to specify the angle of rotation for calculations dealing with orientation on both objects and avatars. Lists can contain any of these datatypes in any order. Several built-in functions are available for working with lists. A strided list is a list of items that is subdivided into “strides” of a matching series of datatypes. So for example, the list {1, “one “ <1,1,1>, 2, “two”, <2,2,2>, 3, “three”, <3,3,3>} has a stride level of three. Each stride consists of three variables in a row. The first variable is an Integer, the next is a Float and then a 3-

Dimensional Vector comes last. The system certainly has its drawbacks in comparison to using Structs, object-oriented Objects and other related constructs.

LSL is a strongly typed language. Thus explicit type conversion through typecasting is required to perform many operations. These operations mimic those that are found in a number of C-like languages. In other cases, built-in functions allow for explicit type conversion such as the List2Vector(), Vector2String() and List2String() functions. Type conversions not explicitly handled in the code are illegal and will return an error in LSL. LSL's strong typing is somewhat unfortunate in the author's opinion, because a weakly typed language (such as PHP) seems to be a more natural choice for the language. If the Lindens want LSL to gain in popularity, they must pitch to modern, young programmers and the next generation of programmer appears to prefer the weakly typed languages such as PHP. (Again, this is just the opinion of the author. Many others will obviously disagree with this statement.)

Further Analysis of LSL

LSL already exhibits an interesting object-based paradigm. While this paradigm is object-related in a 3D modeling sense, it is not truly object-oriented in the typical computer science sense of the word. Every 3D object in the game can have a number of scripts running in parallel (through time sharing - not parallel processing) within the object, but object-oriented programming like that of C++ is not possible at this time. There is work being done currently to make object-oriented programming available in LSL and this will be a giant step forward for the language in the opinion of the author. Object-oriented programming in combination with the 3D modeling, object-related paradigm in Second

Life would be a very interesting thing to experience. The idea of inheritance in its OOP sense would make a lot of sense as would many other OOP principles. For students to see a combination of these paradigms would be extremely impressive. Hopefully this will become a reality soon, as it would certainly improve the language (and Second Life in general) quite significantly.

The state and event-driven nature of the LSL language is another notable aspect that is worth mentioning. Scripts (and thus the objects that contain the scripts) are always in some well-defined and finite state. A script begins its execution in the default state and for every script in LSL a default state must be created. Depending on what state the script is in, outside influences can have various (and often very different) effects on the object and internally, a scripted object can behave in several (again, often very different) ways depending on what state it is currently in. Essentially, the objects in the game follow the flow of a finite state machine. The object may listen for different inputs in different states or alter their own appearance from one state to another. Each state has both entry and exit functions. This helps empower the programmer to fire different events for different state transitions into and out of a state.

Quite frankly, many frustrating things occurred while programming the graduate project in LSL. For one, many scripts can execute at the same time, but the size and memory allocated to a single script is limited. At about 2000 lines of code, problems occurred that weren’t handled correctly. While writing a segment of code and saving the file to compile the script, sometimes the compiler would complain about an error at the end of the file. This was because at the end of the file, code had simply disappeared when the memory thresholds of the object had been exceeded. No warning was provided. The editor just clipped the end off of the code rather quietly. The same problem happened when cutting and pasting sections of code from one place to another on a number of occasions. A particular sound was made whenever this occurred and perhaps this was a warning, but this was of little condolence when the file had to be re-written. Writing multiple files is a work-around for this issue, but it can be annoying and difficult to manage when it is preferred to just work from a single file.

Conclusion

It was discovered through a considerable amount of time in development, as well as through a large amount of trial and error in debugging, that although Second Life is a good place to start for the casual development of Serious Games, much of the gratification lies in developing simple simulation scripts - rather than more complex Serious Games scripts. As the complexity of a simulation grows, it appears that the complexity of the script grows faster and becomes unreliable.

The conclusion of the author is that Second Life is certainly poised to become a much more favorable environment in the near future for the development and demonstration of Serious Games. However, the Second Life world is young and growing rapidly, and thus it is currently so expansive that the overhead required for the development and debugging of its low-level functions and player-controlled build tools is under-represented by the number of Linden staff members (the Linden Labs programmers and QA engineers). The underlying functions that are in place for script writers is lacking in reliability. The absence of many of the commonly-used control structures that are available in many other scripting languages, as well as the need for more advanced programming paradigms such as object-oriented programming and pre-established message passing architectures or control schemes leave much to be desired from an advanced script writer or computer programmer.

Despite its drawbacks, Second Life has become a pioneer in the world of Serious Games. It allows its users to build simulations quickly and easily that can be appreciated by other users with all of the distribution handled by Second Life inherently. Its potential for distance learning-based education and other Serious Games topics is promising. It is groundbreaking in many of its aspects for social networking and it is the author's opinion that in the future, Second Life or a similar environment will become a dominant social networking media.

It is the opinion of the author that as far as serious and purposeful games are concerned, we are merely seeing the beginning of the long-term development of these game genres. In the future, Serious Games will almost undoubtedly become the de facto standard for the way people teach and learn many common subjects - at least those subjects that facilitate this type of learning. Commonly learned subjects will almost certainly be further standardized and through Serious Games people will have an opportunity to learn efficiently in a structured environment that will help level the playing field for students around the world.

Machine learning will continue to thrive through purposeful games. Whenever there is an opportunity for a purposeful game to contribute to machine learning, we can be certain that one or more purposeful games will be developed. We will learn with time many more effective and exciting methods for building these games and in general, the popularity of purposeful games will almost certainly increase.

Video games are already an enormous influence on mankind. This is certainly not something that is set to change any time in the near or even distant future. If we as human beings can learn to not only accept video games as a way of modern life, but also embrace them and evolve with them as a civilization accordingly, we will find that video games can not only be a fun and exciting way to disassociate ourselves from the real world, but they can also be a way to advance our civilization forward in a number of ways - while at the same time having a considerably good time doing so. Serious and purposeful games are a win-win situation for both mankind and technology that deserves much further recognition and support by not only the gaming community alone, but also the world at large.

References

[1] "26 Learning Games for Change | Serious Games | Online Learning Games." *Mission to Learn – Lifelong Learning Blog — Know Better. Live Better.* Web. 28 Apr. 2011. <http://www.missiontolearn.com/2008/04/learning-games-for-change/>.

[2] *3rd World Farmer: A Simulation to Make You Think.* Web. 28 Apr. 2011. <http://www.3rdworldfarmer.com/index_test.html>.

[3] Captcha, Using A. *The Official CAPTCHA Site*. Web. 28 Apr. 2011. <http://www.captcha.net/>.

[4] "Check out the Serious Game on Waging Nonviolent Struggle - Peace and Collaborative Development Network." *Peace and Collaborative Development Network - Building Bridges, Networks and Expertise Across Sectors*. Web. 26 Apr. 2011. <http://www.internationalpeaceandconflict.org/forum/topics/check-out-the-serious-game-on?xg_browser=iphone>.

[5] "Church-Turing Thesis -- from Wolfram MathWorld." *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource*. Web. 20 Apr. 2011. <http://mathworld.wolfram.com/Church-TuringThesis.html>.

[6] "Control Flow." *Wikipedia, the Free Encyclopedia*. Web. 24 Apr. 2011. <http://en.wikipedia.org/wiki/Control_flow>.

[7] "Data Link Layer." *Wikipedia, the Free Encyclopedia*. Web. 29 Apr. 2011. <http://en.wikipedia.org/wiki/Layer_2>.

[8] *Gwap.com - Home*. Web. 16 Aug. 2010. <http://www.gwap.com/gwap/>.

[9] Harteveld, C. 2008. A playful approach to flood defense. *Proceedings of the International Symposium on Flood Defense*, Toronto, Canada, 6-8 May 2008.

[10] *IMVU: Chat, Games & Avatars in 3D. Play, Meet People, Have Fun! Free!* Web. 28 Apr. 2011. <http://imvu.com>.

[11] "Linden Scripting Language." *Wikipedia, the Free Encyclopedia*. Web. 24 Apr. 2011. <http://en.wikipedia.org/wiki/Linden_Scripting_Language>.

[12] "LlListSort." *Second Life Wiki*. Web. 24 Apr. 2011. <http://wiki.secondlife.com/wiki/LlListSort>.

[13] "Luis Von Ahn's Home Page." *SCHOOL OF COMPUTER SCIENCE, Carnegie Mellon*. Web. 16 Aug. 2010. <http://www.cs.cmu.edu/~biglou/>.

[14] "Luis Von Ahn's Research." *SCHOOL OF COMPUTER SCIENCE, Carnegie Mellon*. Web. 16 Aug. 2010. <http://www.cs.cmu.edu/~biglou/research.html>.

[15] "Maya - 3D Animation, Visual Effects & Compositing Software - Autodesk." *Autodesk - 3D Design & Engineering Software for Architecture, Manufacturing, and Entertainment*. Web. 29 Apr. 2011. <http://usa.autodesk.com/maya/>.

[16] "Mono." *LSL Wiki*. Web. 24 Apr. 2011. <http://lslwiki.net/lslwiki/wakka.php?wakka=Mono>.

[17] *OpenGL - The Industry Standard for High Performance Graphics*. Web. 29 Apr. 2011. <http://www.opengl.org/>.

[18] "Serious Game." *Wikipedia, the Free Encyclopedia*. Web. 28 Apr. 2011. <http://en.wikipedia.org/wiki/Serious_game>.

[19] "Serious Games: Darfur Is Dying | DigiActive.org." *DigiActive*. Web. 26 Apr. 2011. <http://www.digiactive.org/2009/08/01/serious-games-darfur-is-dying/>.

[20] *Serious Games Initiative*. Web. 26 Apr. 2011. <http://www.seriousgames.org/index.html>.

[21] "Simulation Game." *Wikipedia, the Free Encyclopedia*. Web. 26 Apr. 2011. <http://en.wikipedia.org/wiki/Simulation_game>.

[22] The ESP Game. Digital image. Web. <http://many.corante.com/images/esp1.jpg>.

[23] The GWAP.com Website. Digital image. Web. <http://designglut.com/images/blog/pomp&clout_gwap.jpg>.

[24] "Top 10 Social Networking Sites by Market Share of Visits | DreamGrow Social Media." *DreamGrow Digital - Social Media Marketing*. Web. 24 Apr. 2011. <http://www.dreamgrow.com/top-10-social-networking-sites-by-market-share-of-visits/#axzz1KUXSFlFp>.

[25] "UFC Personal Trainer Announced for Xbox 360 Kinect, PS3 Move and Wii - Video Games Blogger." *Video Games Blogger - Only the Best Game Guides, Game Walkthroughs, News & Reviews*. Web. 28 Apr. 2011. <http://www.videogamesblogger.com/2011/04/07/ufc-personal-trainer-announced-for-xbox-360-kinect-ps3-move-and-wii.htm>.

[26] *Virtual Worlds, Avatars, Free 3D Chat, Online Meetings - Second Life Official Site*. Web. 28 Apr. 2011. <http://secondlife.com/>.

[27] Von Ahn, Luis. 2005. Human Computation. Ph. D. Thesis – Carnegie Mellon University. CMU-CS-05-193

[28] Von Ahn, Luis and Laura Dabbish. 2008. Designing games with a purpose. *Commun. ACM* 51, 8 (August 2008), 58-67. DOI=10.1145/1378704.1378719 <http://doi.acm.org/10.1145/1378704.1378719>

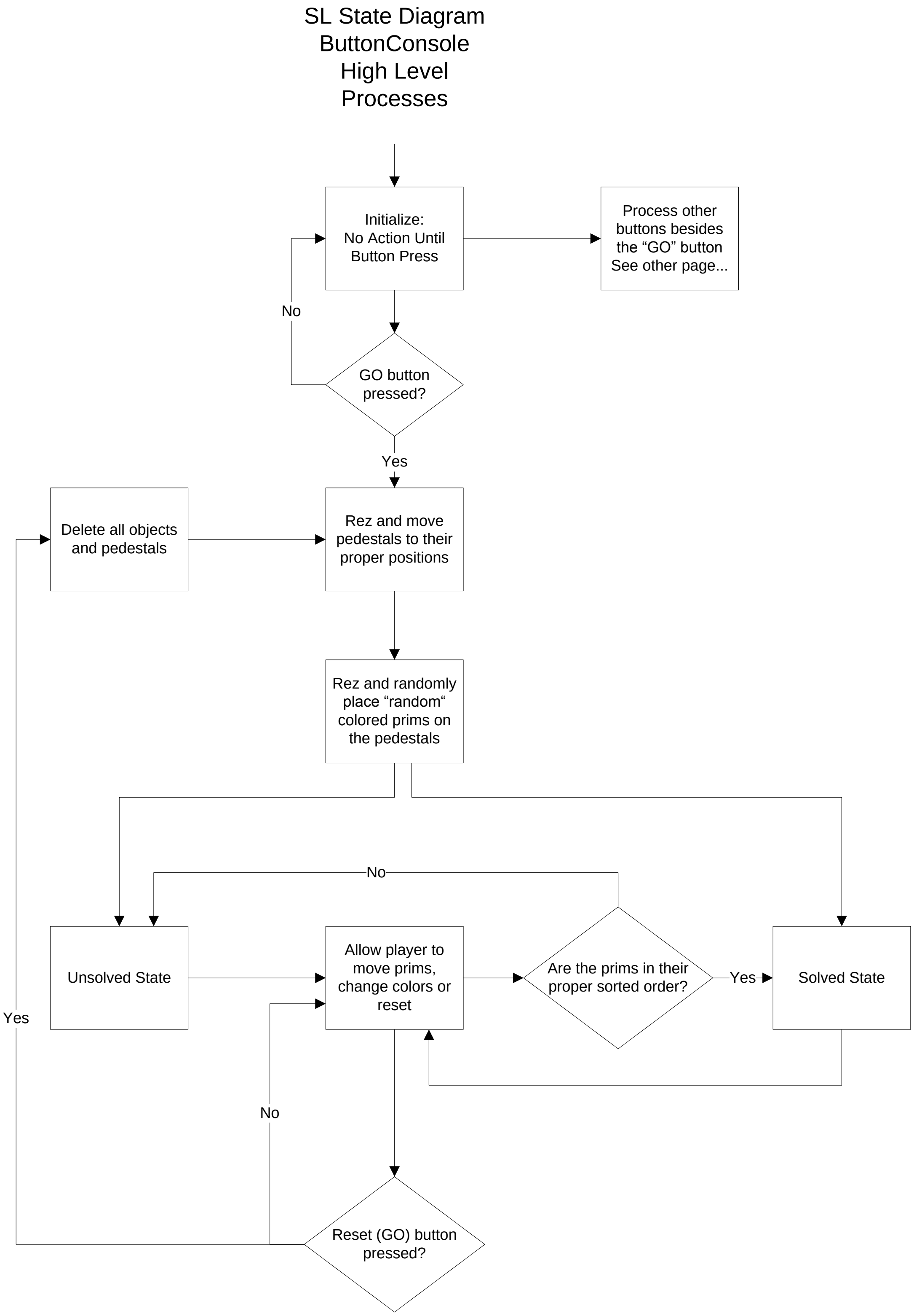
[29] "Welcome to the Second Life Wiki." *Second Life Wiki*. Web. 18 Apr. 2011. <http://wiki.secondlife.com/wiki/>.

[30] World Economic Forum, (2007) Annual Meeting 2007: Shaping the Global Agenda: The Shifting Power Equation, Davos, Switzerland 24-28 January. <http://www.weforum.org>

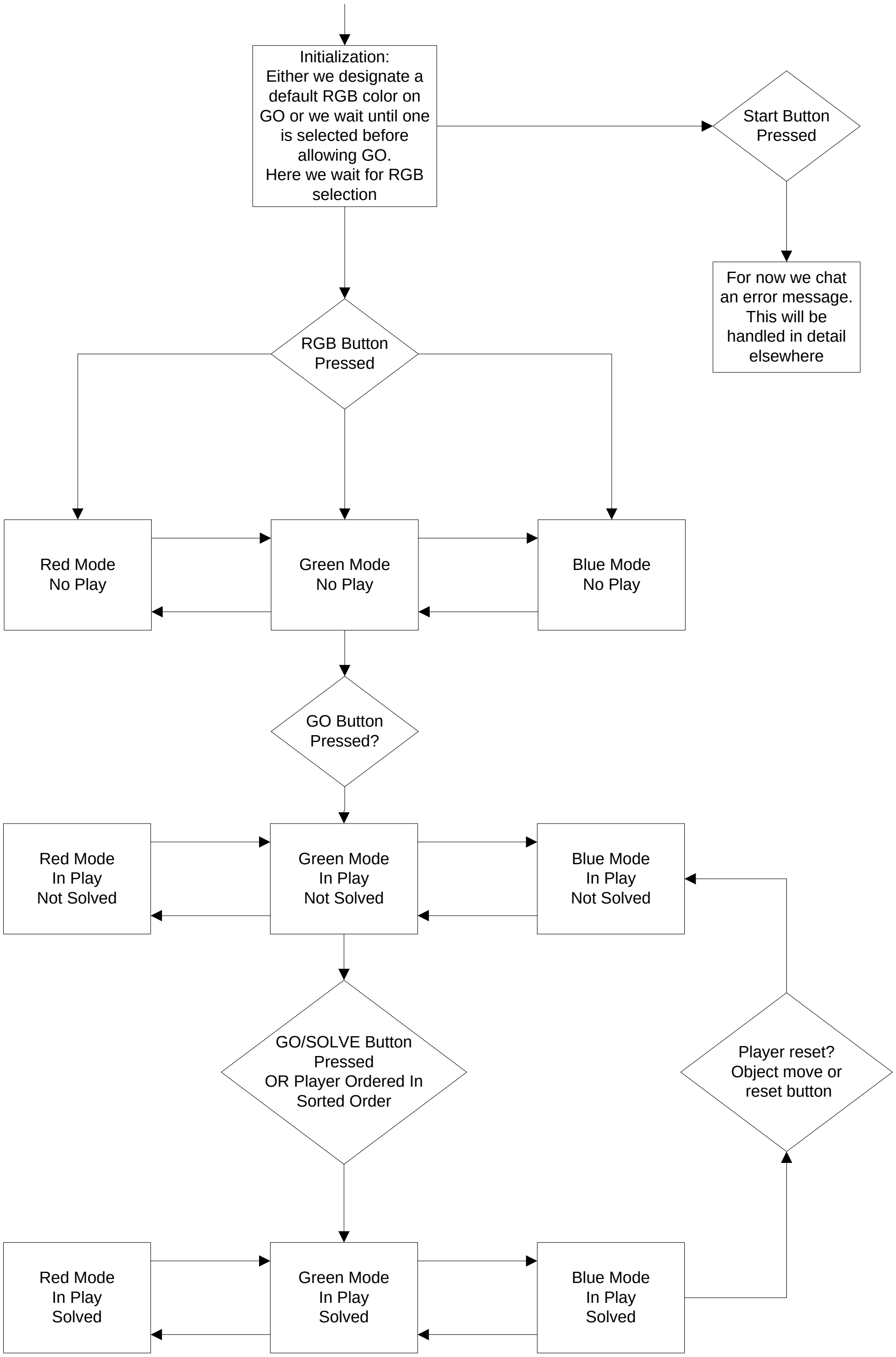
[31] *Www.inivis.com - AC3D - 3D Design Software*. Web. 20 Apr. 2011. <http://www.inivis.com/>.

[32] "XNA Game Studio 4.0 Available for Download! - XNA Game Studio Team Blog - Site Home - MSDN Blogs." *MSDN Blogs - MSDN Blogs*. Web. 29 Apr. 2011. <http://blogs.msdn.com/b/xna/archive/2010/09/16/xna-game-studio-4-0-available-for-download.aspx>.

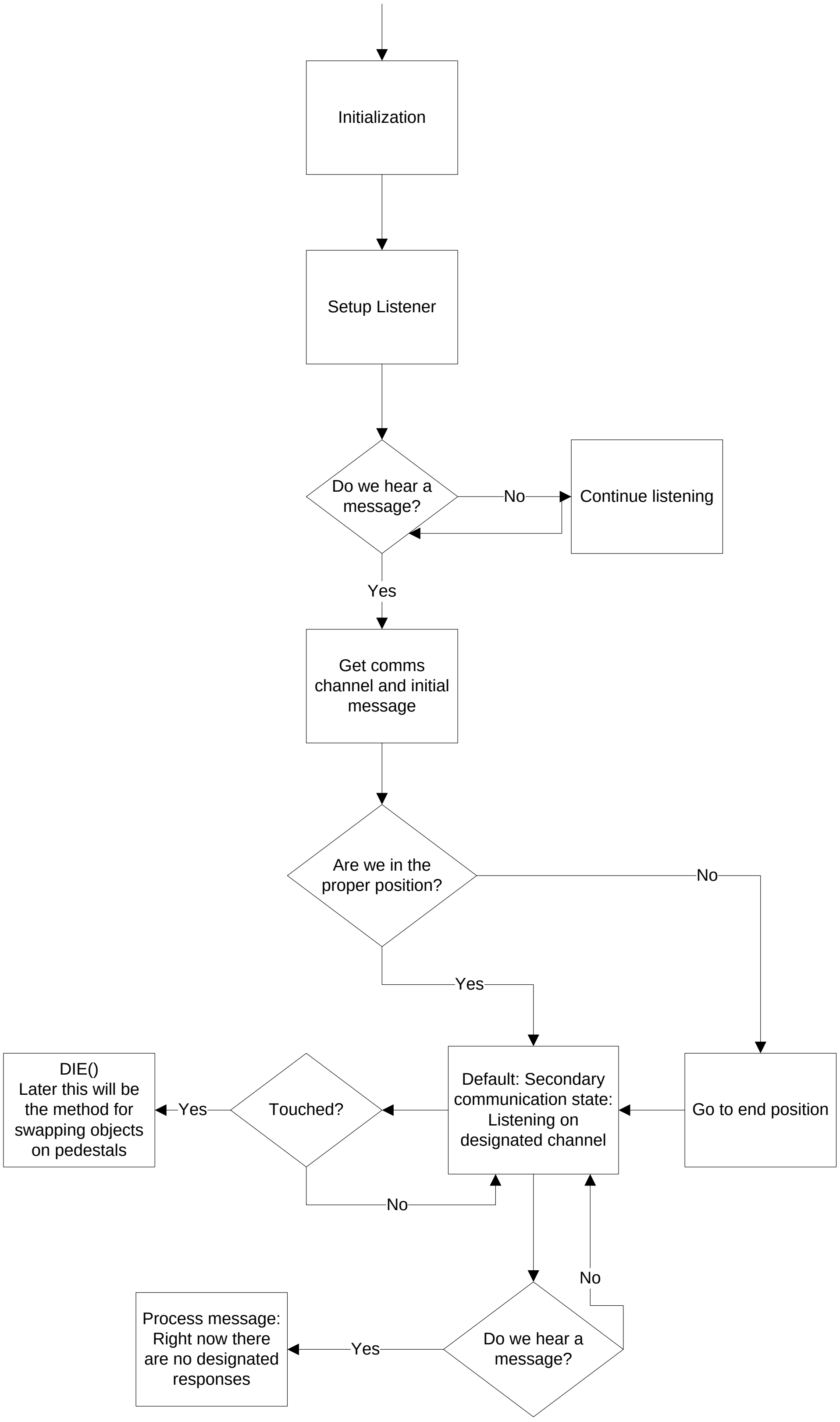
Appendix A. - Mid-project State Diagrams



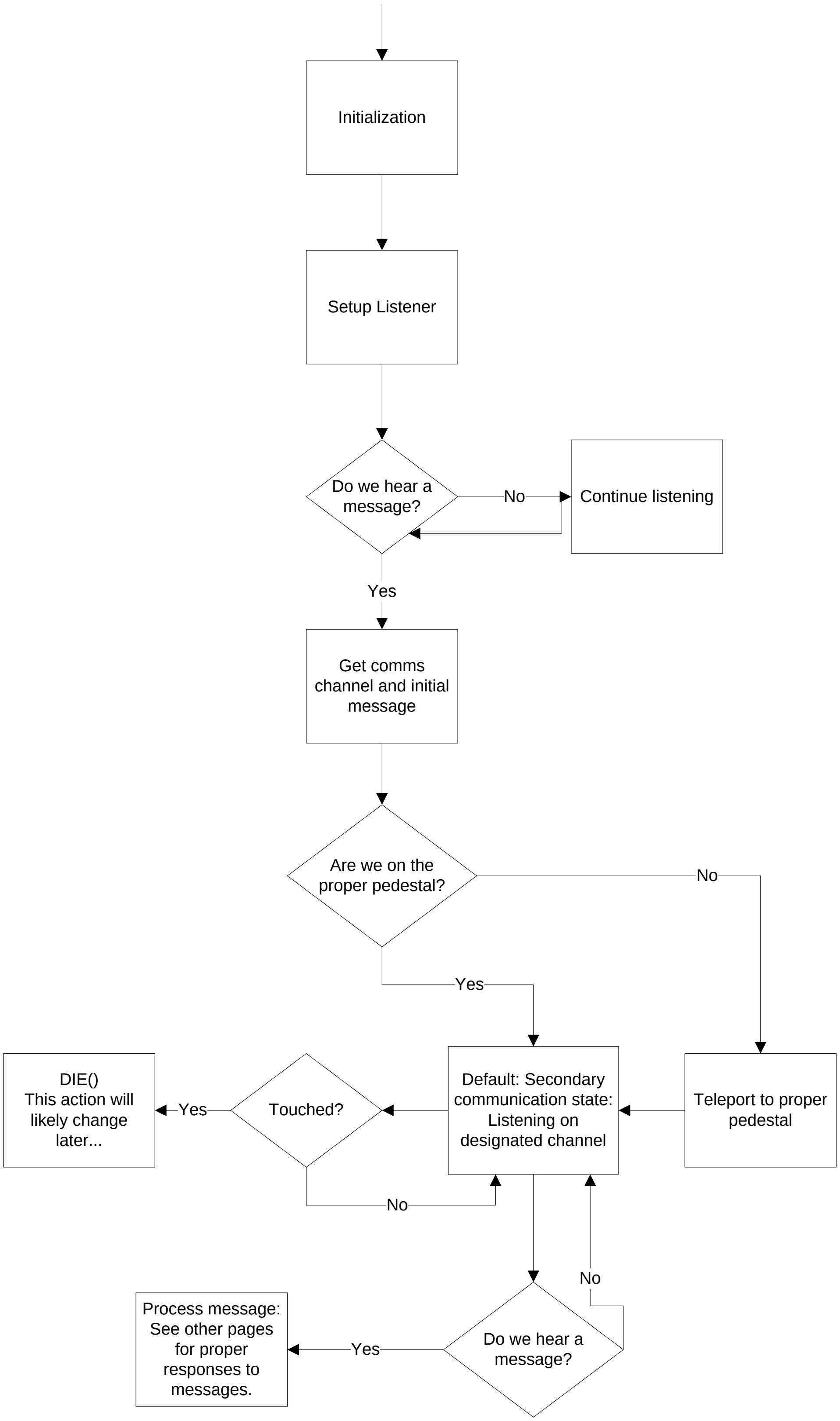
SL State Diagram ButtonConsole Objects and Diagrams


Initialization:
Either we designate a default RGB color on GO or we wait until one is selected before allowing GO.
Here we wait for RGB selection
Start Button Pressed
For now we chat an error message. This will be handled in detail elsewhere
RGB Button Pressed
Red Mode No Play
Green Mode No Play
Blue Mode No Play
GO Button Pressed?
Red Mode In Play Not Solved
Green Mode In Play Not Solved
Blue Mode In Play Not Solved
GO/SOLVE Button Pressed OR Player Ordered In Sorted Order
Player reset? Object move or reset button
Red Mode In Play Solved
Green Mode In Play Solved
Blue Mode In Play Solved

SL State Diagram
Pedestal

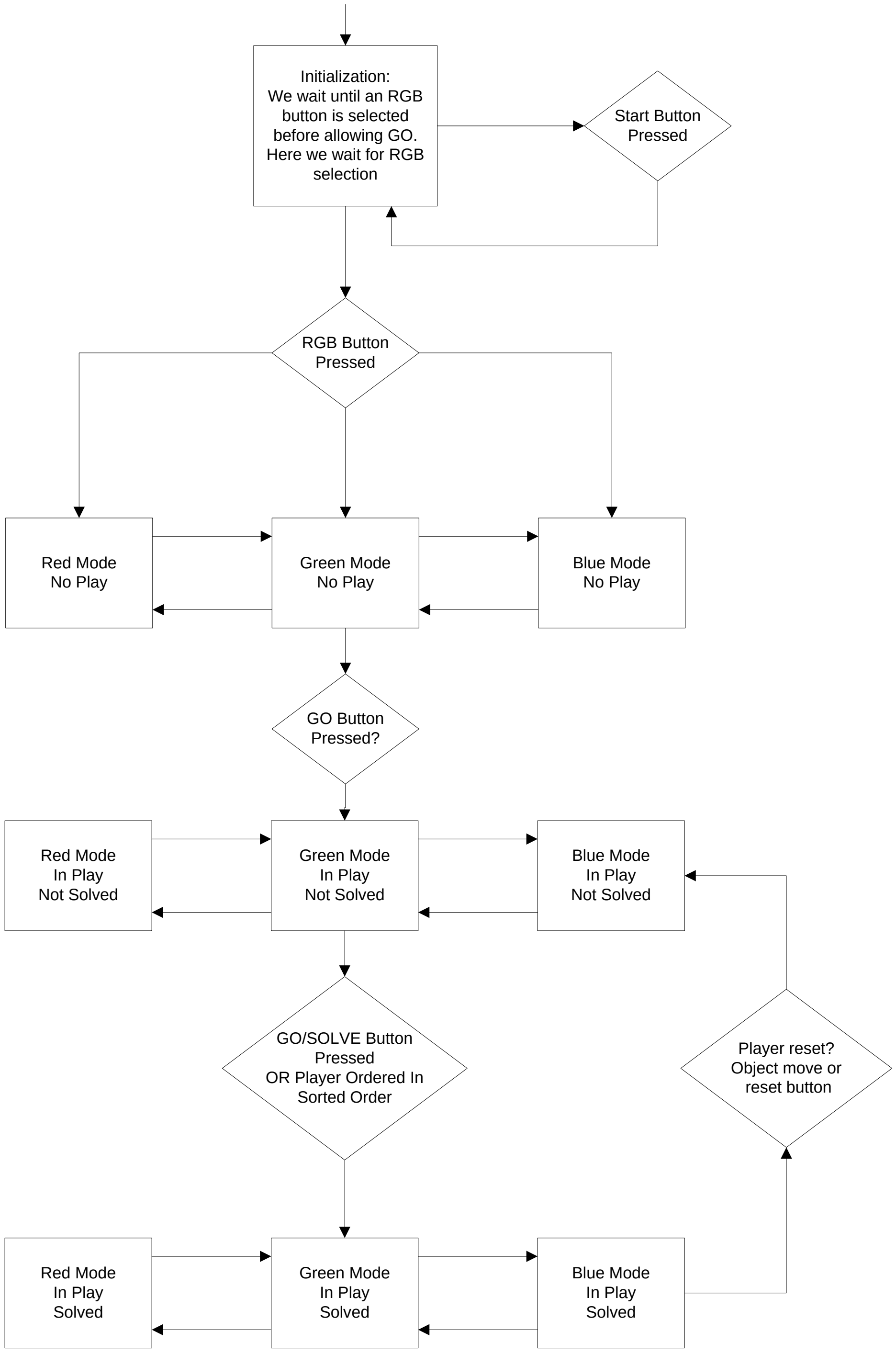

Initialization
Setup Listener
Do we hear a message?
No
Continue listening
Yes
Get comms channel and initial message
Are we in the proper position?
No
Yes
DIE()
Later this will be the method for swapping objects on pedestals
Yes
Touched?
Default: Secondary communication state: Listening on designated channel
Go to end position
No
No
Process message: Right now there are no designated responses
Yes
Do we hear a message?

SL State Diagram
PrimObject

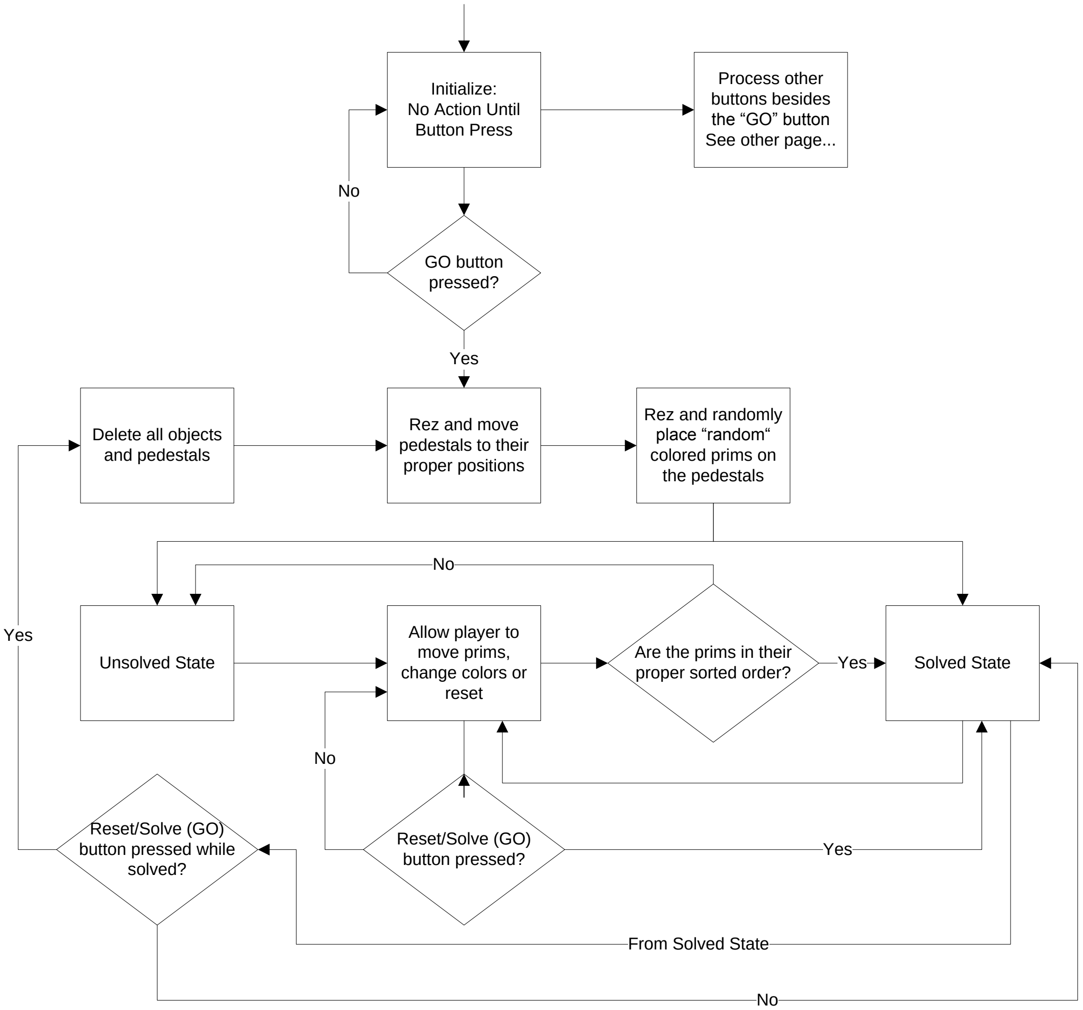

Initialization
Setup Listener
Do we hear a message?
No
Continue listening
Yes
Get comms channel and initial message
Are we on the proper pedestal?
No
Yes
DIE()
This action will likely change later...
Yes
Touched?
Default: Secondary communication state: Listening on designated channel
Teleport to proper pedestal
No
No
Process message: See other pages for proper responses to messages.
Yes
Do we hear a message?

Appendix B. - Final State Diagrams

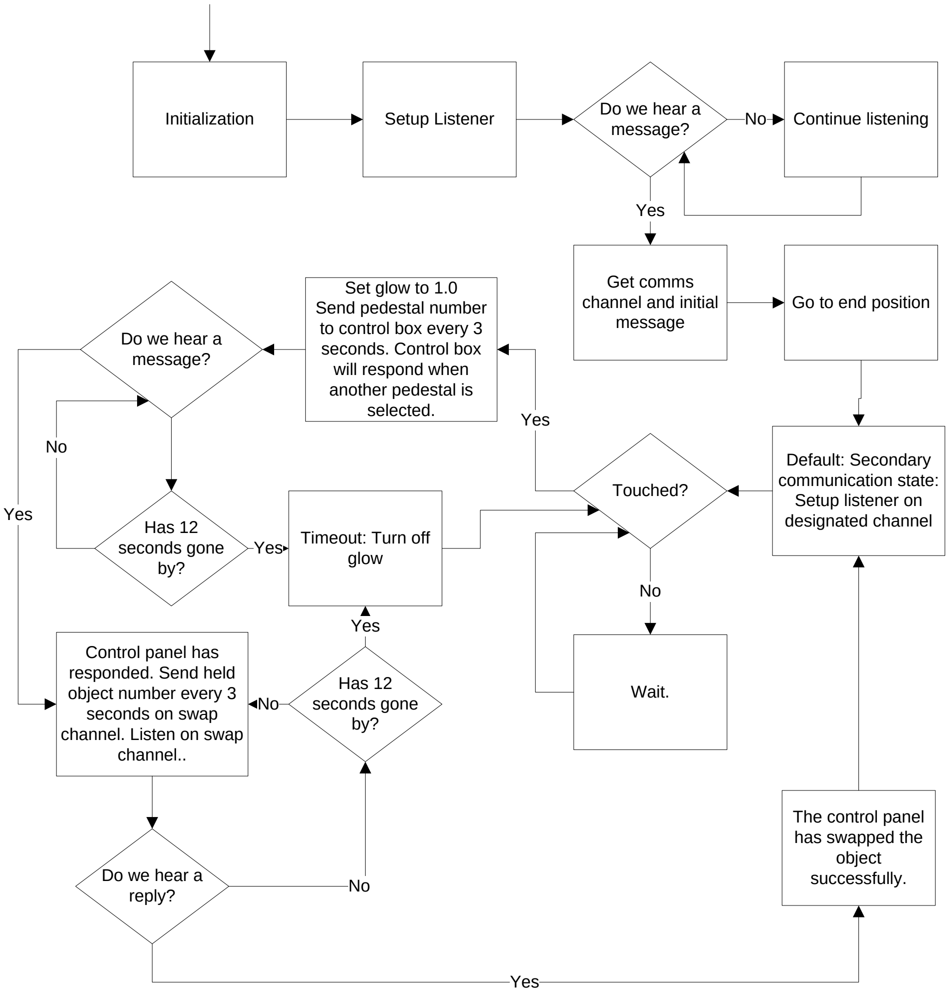
SL State Diagram
ButtonConsole
Objects and
Diagrams



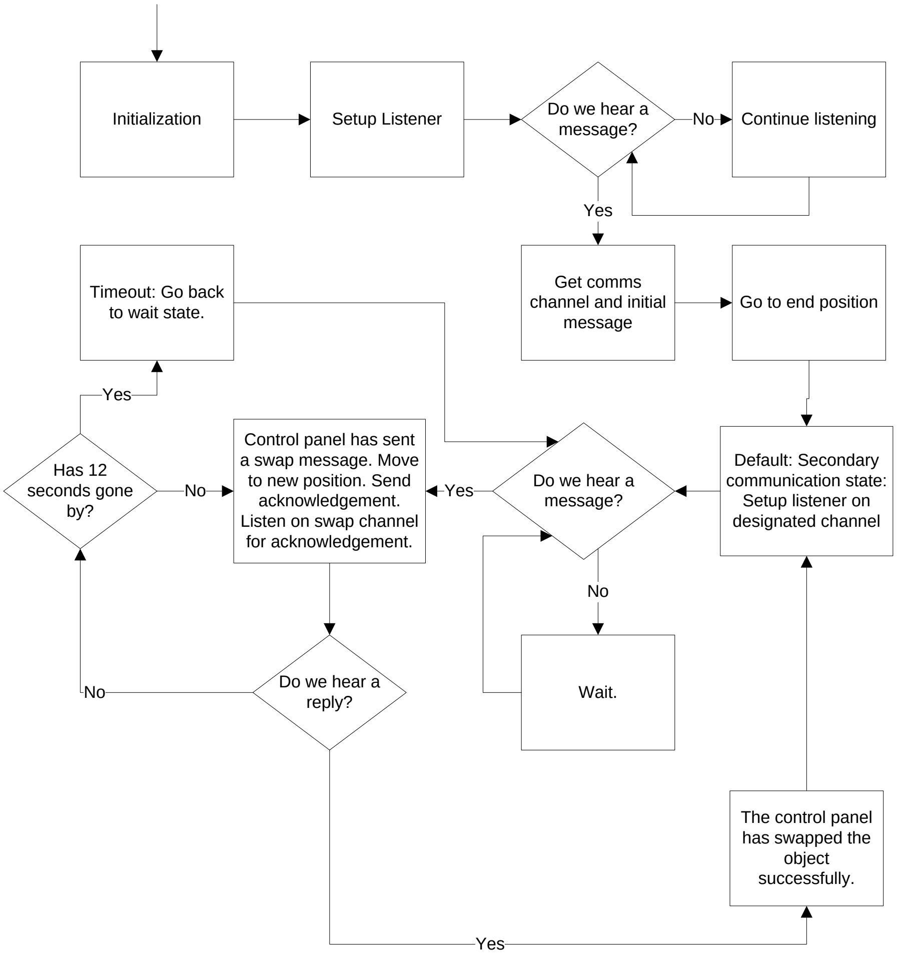
SL State Diagram
ButtonConsole
High Level
Processes



SL State Diagram
Pedestal


Initialization
Setup Listener
Do we hear a message?
No
Continue listening
Yes
Get comms channel and initial message
Go to end position
Set glow to 1.0 Send pedestal number to control box every 3 seconds. Control box will respond when another pedestal is selected.
Do we hear a message?
Yes
No
Default: Secondary communication state: Setup listener on designated channel
Touched?
Yes
Has 12 seconds gone by?
Yes
Timeout: Turn off glow
No
Yes
Control panel has responded. Send held object number every 3 seconds on swap channel. Listen on swap channel..
No
Has 12 seconds gone by?
Wait.
The control panel has swapped the object successfully.
Do we hear a reply?
No
Yes

SL State Diagram
Prim Sort Object


Initialization
Setup Listener
Do we hear a message?
No
Continue listening
Yes
Get comms channel and initial message
Go to end position
Timeout: Go back to wait state.
Yes
Has 12 seconds gone by?
No
Control panel has sent a swap message. Move to new position. Send acknowledgement. Listen on swap channel for acknowledgement.
Yes
Do we hear a message?
Default: Secondary communication state: Setup listener on designated channel
No
Do we hear a reply?
No
Wait.
The control panel has swapped the object successfully.
Yes