Anima: A Singular Journey

CS470 – Capstone Project

By Shawn Aldridge

Spring 2010

Table of Contents

Abstract: Anima is an interactive Java-based game built on the JOGL graphics platform. It follows the path of a budding artificial intelligence from it's first struggles for survival within the CPU to its attempts at world domination. The development of the game will be an exercise in base-level OpenGL programming, as well as a preview of the complexities and pitfalls of commercial-grade game development. This document will initially explain the scope and objectives of the project. It will then go on to detail the design and architecture of the game, followed by the implementation results. Finally I will pinpoint successes and failures within the project, and future directions, both for the project itself and for the author within this industry.

Introduction: This program is a result of my CS470 Capstone Project. The project is designed to test my programming skills in OpenGL while allowing me to stay grounded in my Java experience. I intend to explore the customization of JFrame windows within the Swing environment, adding to my Java GUI Development skills for future projects. I will also explore the differences between student and commercial-grade software, attempting to bridge the gap as much as possible. Finally this will allow me to explore the process of game development, an area of software development that I have considered exploring career options in.

Project Overview: I am the client for this project, it is designed solely to my specifications, with the broad requirement that it be of sufficient complexity to fulfill the requirements as my capstone project. The goal was to produce a near-commercial-quality, if primitive by today's standards, computer game.

The game is implemented in Java, using Swing for the windowing and event-handling and OpenGL through the JOGL interface for the graphics and animation. While I planned on altering the look and feel of Swing I overestimated the need for GUI controls and windowing within this application. All of the buttons I used within the application were easily implemented with OpenGL using screen coordinates, a technique I was not sure would work when I began the project, and much easier than trying to interlace multiple button-filled JPanels within the JFrame and get them to not interfere with the GL Canvas.

I used a modified XP approach for my implementation, including user stories to guide program requirements and architecture, an iterative release structure, and a test-first methodology with refactoring in later stages to clean up program architecture.

Game breakdown:

Anima is Latin for “animating spirit” or “vital force.” The game follows the player's attempt to guide a budding artificial intelligence along the evolutionary journey to self-awareness. It begins within the central components of a computer, with the character, graphical environment, and general strategy of the game becoming progressively more complex as the game goes on. The levels parallel the successively more complex systems within a computer, although only on an implicit level, no actual computer knowledge is required to play the game. The level breakdown is as follows:

Level 1 – Play begins within a CPU register. The player is represented as a simple small sphere, maneuvering left and right along the bottom of the screen. Larger green spheres descend from the top of the screen grouped in double-rows. The player is required to match spheres against the falling rows, adding each double row set in binary fashion, i.e. a single orb if there is one sphere in a column, none if no spheres, and carrying an orb into the next column if there are spheres in each row. Failed additions due to time or improper input take time, allowing the spheres to descend farther. If they reach the bottom of the screen the player loses, and is forced to restart the game. The player must survive for 42 seconds in order to advance to the next level.

Level 2 – Play continues within the CPU, with the player still represented by a small sphere. The playing field is a horizontal rectangle, with grinding registers on the far left side of the screen attempting to chew the player up. Data structures, in the form of rectangular polygons, stream in from the right side of the screen, attempting to push the player into the registers. Other, blue colored diamond-shaped polygons representing crucial data must be gathered up by the player, with the player winning if they collect enough data pieces.

Cutscene 1 – This cutscene begins with an overhead view of the bus structure, with points of data streaming by. It then does a short pan over the scene and zooms in toward the bus lines, following them with successive speed out of the scene. Sidenote: this is easily the ugliest code I have ever written, and I desperately wish I had had time to clean it up.

Level 3 – The third level takes place within main memory. The player is attempting to climb the stack of memory, slaving pages to their control on the way up. As the level continues the player will evolve through various regular polygon forms, becoming increasingly complex as they progress. This was originally intended as another pattern matching level, but attempts at this level have not succeeded. I threw out a trial levels at the end of iteration one and attempted designs again in iteration two but never got very far. Not sure if I want to scrap this level from the final product at the risk of my own

mental continuity or simply try redesigning it again. It is not implemented at the moment.

Level 4 – The playing field is now the computer as a whole. The level appears as a three dimensional maze, which was originally modeled on the bus structure within the computer system, with the player hovering just in front of the screen, navigating their way through the maze, but ended up using circuit-like texture mapping in the final iteration. Once again, I'm thinking about changing this later, but this is the last form I implemented. Once again, the player will evolve through a number of increasingly complex polygons as they achieve their goals. The player needs to perform a series of operations in order to forge itself a logical body for further levels, form yet to be determined.

Cutscene 2 – Following the fourth level I intend to insert a cutscene animation showing the creation and infusion of a newly forged avatar body, but I didn't get to this cutscene due to schedule backups, and nothing beyond this point was attempted.

Level 5 – Player is represented by the aforementioned body. Play expands to a networked environment, most likely containing a major plot twist, which I am still working on.

Final cutscene – I intend to include a victory cutscene should the player finish the game.

Project Requirements:

User Stories:

Windowing and Interface:

•Game display should cover full screen - finished

•Users should be able to create, delete, and access user profiles to save game results – this seemed like a minor detail, and so ended up getting pushed back to later iterations, and as a result never got implemented

•Users should be able to view a player lounge displaying available profiles along with scores and achievements – didn't get this far

•Player should be able to clearly tell the difference between default java swing components and those used for the game, which should provide a comprehensive, immersive feel to the game – technically finished this one – but due to evolution of the project it turned into a relatively minor component.

•Game should display a flash screen while loading, as well as title and credit screens – never seemed necessary

•Player should be able to modify the difficulty level of the game – never got to this point

Level 1:

•Player should be able to move smoothly in two dimensions – jerky animation felt more realistic in the internal environment of a computer, more artificial is more realistic

•Player should be able to clearly identify the task before them after minimal trial and error – seems intuitive to me, some testers have complained

Level 2:

•Polygons should stream smoothly from one side of the screen to the other - done

•Polygons to be collected should be easily identifiable, either through shading of the polygon or glowing - done

Cutscene 1:

•Game should transition smoothly between level and cutscene - done

•Cutscene should be jitter and flicker free - done

Level 3:

•Player should transition smoothly between different representations, while interacting accurately with the environment based on the dimensions of the representation – level scrapped

•Objectives should be easily identifiable through minimal trial and error – level scrapped

Level 4:

•Player should transition smoothly between representations – never implemented

•Environment should appear fully 3d - done

•Texture maps should cause passageways to appear rounded and apply an image to represent sky – never implemented

Cutscene 2:

•Game should transition smoothly between level and cutscene – never implemented

•Cutscene should be fully three dimensional and include texture mapping – never implemented

Level 5:

•Environment should be fully 3d – never implemented

•Player should be represented by a complex avatar – never implemented

Final cutscene:

•Final victory cutscene should be wicked awesome – never implemented

System requirements should be minimal, any user with a sufficiently up to date Java Runtime Environment should be able to run the game. Assuming the graphics don't become more complex than I anticipate, anybody with a system built in the last five years or so should be able to run the game.

System Design

System architecture

The program is implemented in a standard main plus subroutines form with an extra level of abstraction in the form of the GameManager class, which is essentially a shell class designed to control the sequence and timing of when the other classes are active or inactive. The GameManager class initially calls the LevelManager class, which holds the load screen and the moving background for the first two levels. The GameManager successively loads each of the three levels and the cutscene. Helper classes include a Sound class to control sound effects and background music, as well as a mazeNode class to help with the building of the maze in level 4. Event handling is done through the use of inner event listener classes monitoring the various keys needed for controls.

Data structures:

The data structures for this program are quite simplistic, with the exception of Level 4, which builds the maze by means of a disjoint set. Other data structures include a handful of ArrayLists and Linked Lists for storing and sorting objects.

Algorithms

The main GameManager class just instantiates each of the subroutine classes as they are necessary, as each of the subroutines are JFrames and need to be linked to GLEventListeners in order to

operate the animators that iterate through the display methods. The main method pauses during each class and continues when the player either wins or loses.

Each of the subroutines functions through the use of an FPS Animator which strives to hold the frame rate at a steady 60 Frames Per Second. They perform the necessary initialization of variables and then iterate through the display method until a boolean value is activated indicating that the level is finished, win or lose.

Software Development Process:

Coding/Learning Process:

I was hesitant going into this project. I felt confident that I could turn out some sort of project, as I had a little bit of Java game development already under my belt, albeit at a much lower level than this, but I had no idea how quickly I might pick up OpenGL. In the end I would say that the majority of my difficulties came in not knowing the proper architecture for JOGL-based games. The OpenGL was not trivial but fit well within the course of my graphics class, and while I didn't get to the later levels I would still blame that on a lack of proper scheduling and budgeting of time rather than a lack of ability to achieve the goals that I set for myself.

The XP approach worked and it didn't. I liked the iterative approach to development, it made me think about what features needed to be implemented next and forced me to view each iteration of the game as a functional whole, thus requiring me to actually make it functional for each release. That said, the test-first approach failed completely. When I first set down the goals for this project I had not yet written a single OpenGL project, and while they often turned out to be as mathematical as I expected, I did not anticipate the pre-set nature of the required methods within OpenGL. The structure of these methods made it more or less ineffective to try and test each method before implementing it, so once again, the XP approach worked and it didn't.

Testing & Debugging

As previously stated, I abandoned the test-first methodology, and sketched out the algorithm and architecture at the beginning of each iteration. Testing was done by isolating each level and loading it up to verify the graphical behavior visually. Debugging was done most heavily at the end of iterations.

Development Challenges

The main challenges I encountered during development were in implementing sound and in implementing each successive level within the game. The sound class was found on SourceForge by

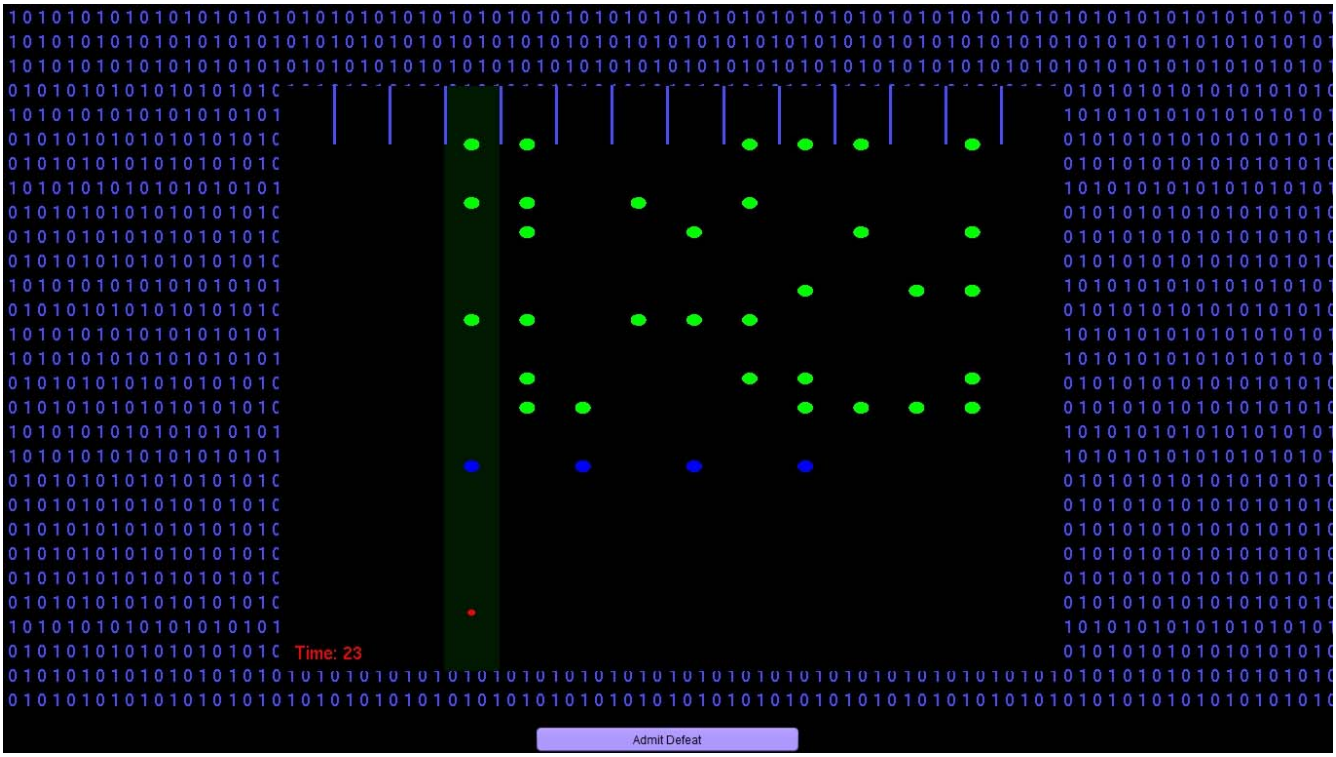
Kyle Aleshire after I unsuccessfully attempted to implement three different sound classes. I'm not sure why this particular task was such a trial for me, but I was completely unable to get a sound class implemented by myself. The other difficulty I encountered was in implementing multiple OpenGL classes within a single program. Each OpenGL class was implemented with an animator, which requires a GLEventListener be registered to the class, unfortunately you can't have more than one of any of these running at the same time. Trying to access more than one GL Context at the same time throws handfuls of errors, halting the system. In order to get around this I created another level of abstraction in the form of the GameManager class. This central class calls each OpenGL class in turn, waits until it signals that it is finished, disposes of the class and frees any resources it might be using, and then creates an instantiation of the next class.

Results & Conclusions

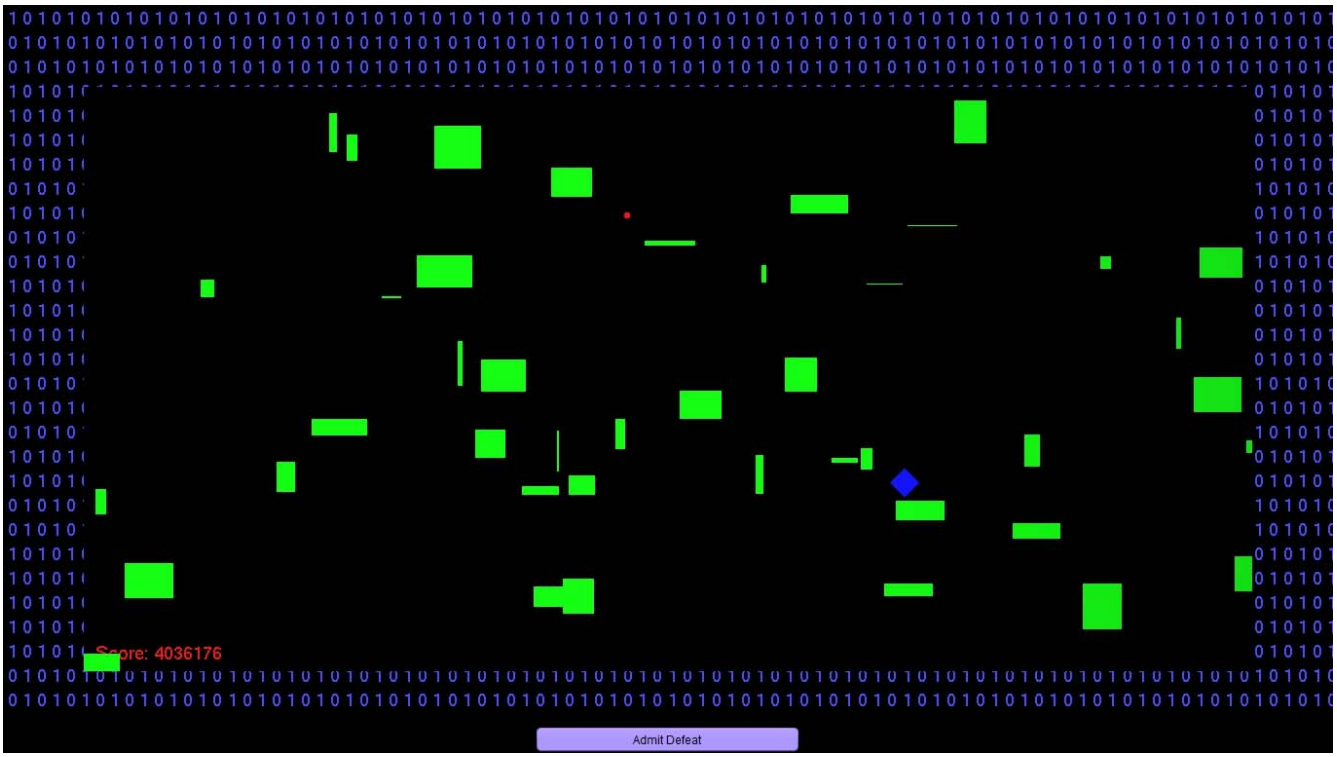
Results

In the end I got through roughly 2.5 iterations. I completely implemented levels 1 and 2 including sound, got a working copy of the first cutscene, albeit without the very end of the cutscene or a soundtrack, and got a working version of level 4 finished, once again without the enemies or objectives. Level 4 will flip textures and colors if you get to the end of the maze, which was originally going to be a sort of surprise difficulty increase, but the only part of that I got implemented was the color/texture flip. Below are screenshots of the 3 levels and the cutscene that I succeeded in implementing.

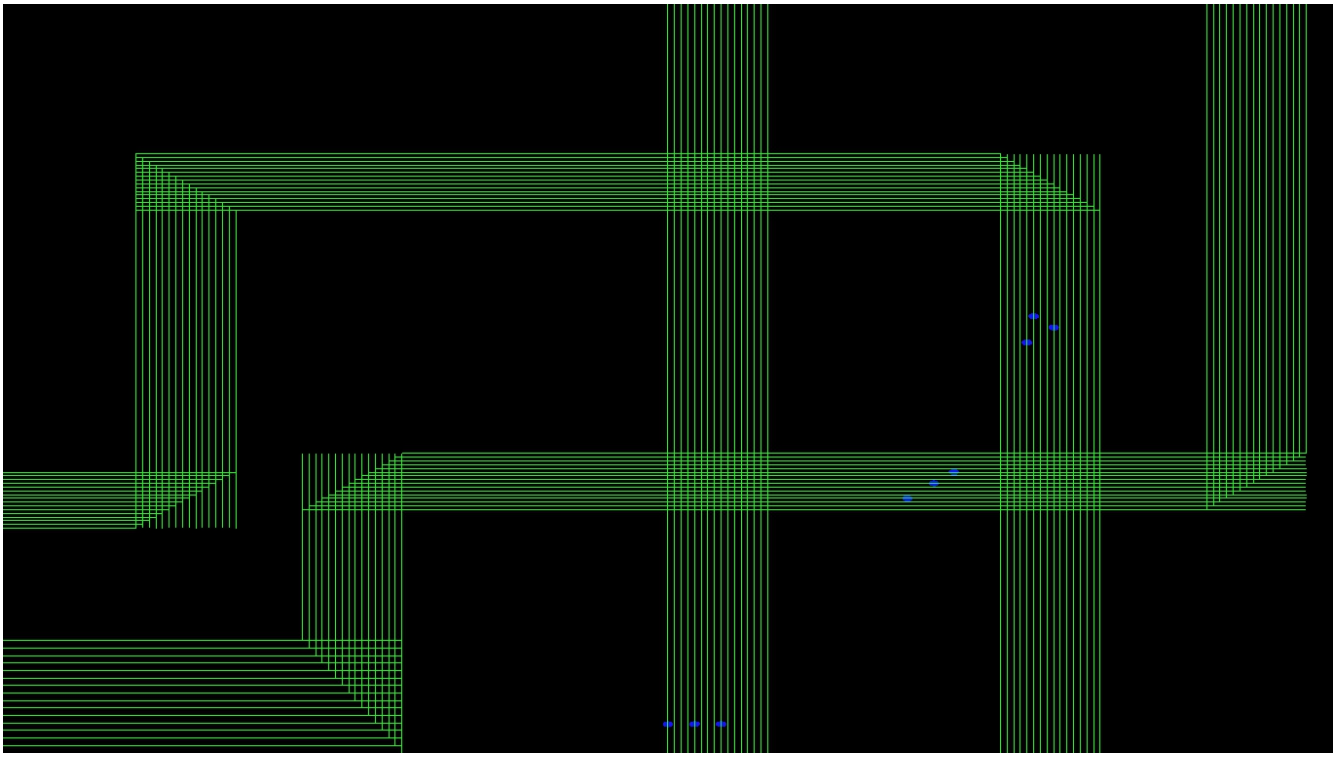
Level 1:



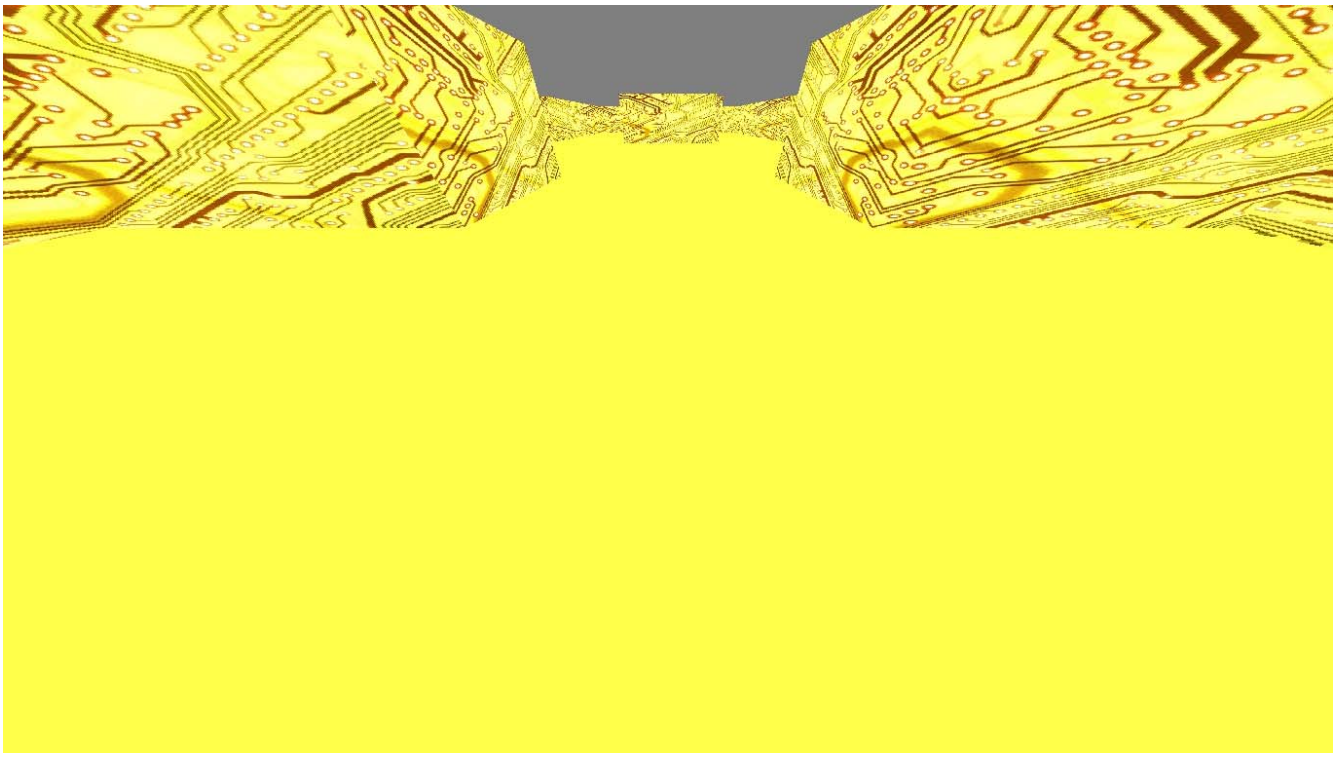
Level 2:



1st Cutscene:



Final texture configuration for level 4:



Conclusions

In the end I would say that this project did exactly what I wanted it to do, it helped me to get a better feel for the challenges particular to game design with a commercial-grade graphics library, at allowed me to explore the differences and similarities between student projects and commercial software, and it gave me a solid taste of application programming with OpenGL. So while I didn't get as far into the game as I had originally planned, I do intend to finish it up over the summer and I did both enjoy the project and learn a great deal during the course of it.

Appendix A: User Manual

Upon starting up the program the user will be greeted with the main control panel. From here the user will see areas for the selection of user profile, difficulty level, game beginning, player's lounge, and an exit button. The only buttons that work are the exit button and the start game button.

Level 1: In the first level the objective is to add the binary patterns falling from the top of the screen. Pressing the left or right arrow keys will move the avatar in the corresponding direction. Pressing the space bar will light up a blue orb in the highlighted column just below the advancing orbs. The player must add the bottom two rows of orbs in a binary fahsion, i.e. an empty column if there are no green orbs in the bottom two rows of this column, an orb in the current column if there is one green orb, and a carry orb in the next column to the left if there are two green orbs. Two orbs plus a carry orb would mean an orb in the current column AND a carry orb in the next column to the left.

Level 2: In the second level the player must intersect the blue diamond shapes without being pushed off of the left side of the screen by the oncoming green rectangles. The player can move in both X and Y directions by means of the arrow keys, and blue diamonds will be automatically picked up when a collision is detected.

Cutscene 1 requires no player input

Level 4: Level 4 has no obstacles, and the only objective at the moment is to reach the end of the maze, which will change the color and texture of the maze to a bright golden circuitboard pattern. The player can move forward by pressing the up arrow and turn left or right with the corresponding arrow keys.

Appendix B: Code Listing

Consult the accompanying zip file, which should include the following files:

GameManager.java

LevelManager.java

Sound.java

Level1.java

Level2.java

Level4.java

Cutscene1.java

mazeNode.java

trial.jpg

trial2.gif

trial3.jpg