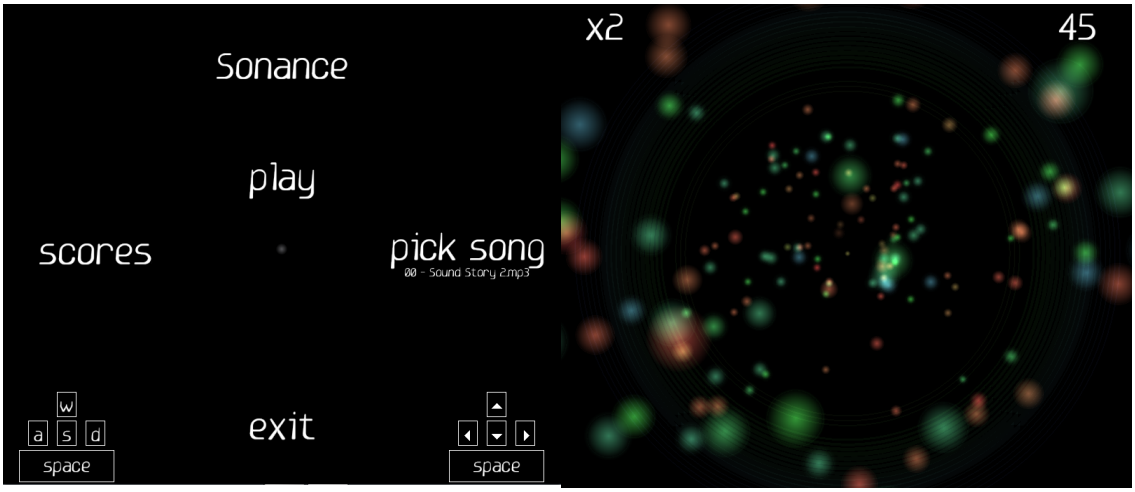
Sonance

4/24/2012



Team
Fenil Sevak (Lead Designer, Music, Engine, Menu flow) f.sev16@gmail.com
Christian Dugenio (Art, Graphics, Engine, End Game) christiandugenio@gmail.com
Dennis McGrogan (Art, Engineering, Menu as game) dmcgroga@gmail.com
Justin Rokisky (Gameplay programming, Engine, Song Chooser) jrokisky831@gmail.com

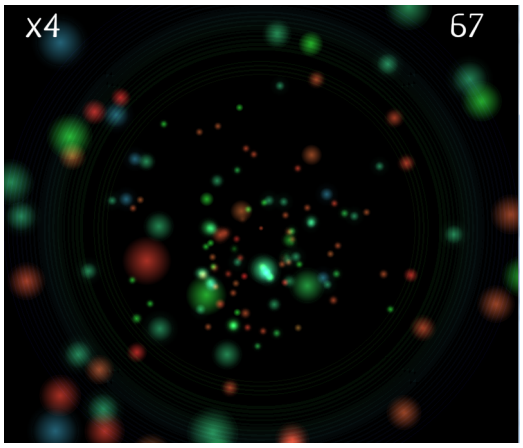
Overview
Sonance is a way to experience music; a two dimensional game built around sound. The song selected by the player is translated into a visual experience unlike traditional games. The game itself is minimalistic, focused on acquiring a high score. This simple premise, combined with the appealing visuals and a few twists provides for some interesting gameplay. Included with the final game are two sample tracks that set the theme of the game, in addition to the player being able to use their own music.

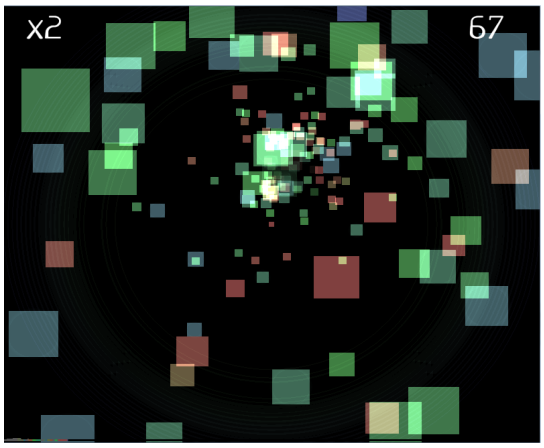
Technology
Sonance was developed in Java using Lightweight Java Game Library (LWJGL) with Minim as the sound library. In addition, Slick was used to aid in the display of text. This was all based on the prototype which used Processing and Minim. Asana was used for task management, while Git via Bitbucket was used for source control. The music that comes with the game was created in FLStudio. Communication was facilitated by Skype, Group.Me, and frequent meetings.

5 things that went right

Art Style
We very quickly converged on an art style for the game. Since the prototype itself was minimalistic we were able to find a minimalistic design that worked well for the player model. The basic idea we wanted to achieve was that the colors and shapes of the enemy would flow well with the feel of the game. By using "glowing" enemies, we were able to capture the feel of

ambient electronic music (such as Deadmau5's "Strobe"). We tried to take a lot of factors into account when designing the player models: the ability to distinguish between enemies you can or can't eat, showing the player how well they are doing by brightening or graying out enemies, and in the case of the square model we took into account ideas from cognitive studies that show how objects can appear to grow larger (in relation to other objects) even when those objects actually remain the same size. (see http://en.wikipedia.org/wiki/Ebbinghaus_illusion for why the squares appear to get larger even though they only ever shrink or stay the same size).





Java / platform compatibility
We decided on Java early for our project not only to make future multi-platform coding easier, but also to make coding in general easier, since Java was a language that we were all familiar with. Since the prototype was created using Processing and the Minim FFT library, the Java compatibility of Minim helped make the choice even easier. This let us focus on designing the game rather than having to relearn a sound library or a new language. The only real “hurdle” we faced when working with Java was upgrading all computers to use JDK 1.7 in order to use the HighScore.java class.

Teamwork
Our group met twice a week regularly, with additional meetings as needed and this was all supplementary to the small meetings after the class itself. In addition, through Group.me (a group texting service) and Skype, we were always in contact with each other; always communicating new ideas and getting feedback from each other. The greatest part of the group meetings is that they allowed for quick fixing and a second eye to look at code to help fix errors, this is something that could not have been achieved as fast over the internet. The in-person meetings proved to be some of our most productive times due to this instant feedback and criticisms.
Once we got into the flow of working together we found it easier to modify and improve each other’s code without conflict. For example, in the last week we had to update our code to use textures to draw all the models in the game instead of generating them procedurally. The basics of texturing and displaying the model were implemented by the artist and were soon after refined for better scaling by the graphics programmer.

Stuck to prototype – main game feel
Largely, the prototype remained in the final game as the basis for it all. There were some changes, like the addition of color, but were all planned when the prototype was presented. There were only minor changes to the actual gameplay such as instead of dodging everything

to dodging only certain colors. This allowed for the game to come together rather quickly, and allowed us to shift our focus to various aspects of the game and tweak them more critically. In addition this freed up time towards the end of the project for experimentation and the vast amount of standardizing that we had to do to compensate for the differences over various computer specifications.

Menu as a part of the game
The seamless integration of the menu into the game allows the user to experience the game even before starting their first song. We use the same physics as in the game, making the menu playable in the same way that the game is played. In addition, we added a few features to the menu that will signal to the player the actions they can take. By adding in the “WASD” and arrow keys to the bottom left and right respectively we are hinting to the user that those are the controls to the game; when any of those keys are pressed they become highlighted. Beyond just the transitioning from the menu into the game, the transitions between the different screens of the menu helps convey the feel of the game.

5 things that went wrong

Experimentation
In the early stages of development the group didn’t have a clear cut goal for the game which led to tons of experimenting, visually. Each week we would try a bunch of ideas, see which ones would fit the best, and toss the rest. Various experiments such as the color-blind “fix”, gridded backgrounds, and various visualizers were destined to be tossed out. Until the last two weeks or so, everything began to converge, which possibly could have been done at the beginning of the development and saved room for modifying the FFT and enemy generation and other game system tweaks and improvements. Also we focused the experiments based on the prototype design, that is we didn’t experiment enough with visualizations as a whole (i.e. themes) until the last two weeks or so. Looking back on the project, we should have focused more on large thematic changes in the early stages of development and left smaller tweaks, such as the specific color palette being used, for the later stages.

FFT (Fast Fourier Transform)/ Determinism
FFT was something that we wanted to nail down in the first week or so as the prototypes usage of the FFT analysis was decent enough. Instead perfecting the FFT took nearly 4 weeks and a bunch of iterations to make it independent of the frame rate in addition to improving accuracy. On top of that, many iterations were made attempting to make the FFT 100% deterministic. We seem to have gotten close to making the game completely deterministic based on the song, but due to some issues likely associated with the way we process the song, we haven’t fully satisfied goals we set out to reach.

Playing off the sound card
Originally the game was envisioned to be played with both selecting a mp3 and with streaming music. For this reason we based the entire FFT analysis on being able to work in real time. This prevented various ways that we could have improved or simplified the orb generation through the music. On top of this we never did get to implementing this ability to play off of the sound card; this was due to the amount of other work that we did, assuming this would be a rather simple thing to add later. Later it was found that this would take much more work than required as that sort of access is not given in java, thus requiring the use of JNI. At the end we just scrapped this for the sake of this project, if we were to work on this in the future this may

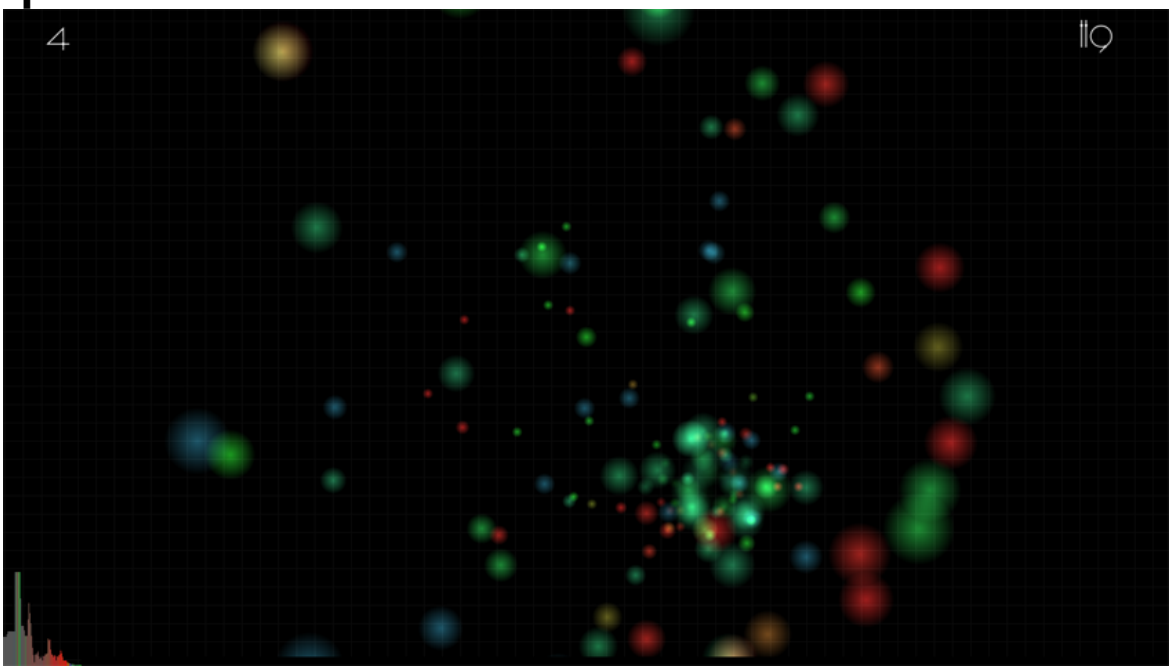
something to be added.

Optimization
In the later stages of development we found that it was possible for the frame rate to drop significantly dependent on a computer's hardware. While the game was able to run at 30 frames per second on most PCs (ranging from single-core PCs from 2005 to quad-core gaming PCs) a limited number of PCs weren't able to run our game at an acceptable frame rate. We were able to fix the issue before the project was completed, but it highlighted our group's need to brush up on the graphics pipeline as well as graphics hardware. Instead of using the hackish approach of drawing each enemy pixel-by-pixel we now use textures to draw all enemies with the same model and just vary their color.

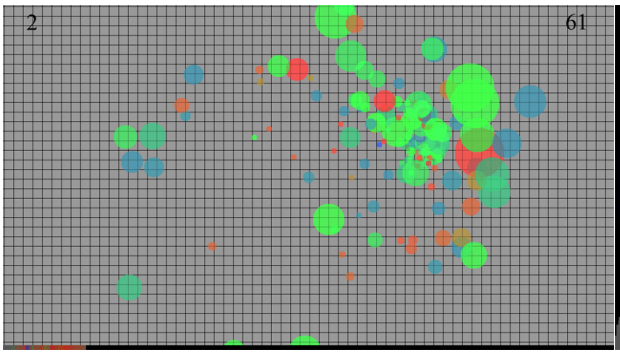
Engineering
While we kept the model-view-control (MVC) structure in mind our code started out a bit rough. Early in the process the code was thrown mostly into the main class and left to be separated by whoever would take on the extra task of code refactoring. We had many conflicts early on, as many of the classes were dependent on one another, and it wasn't until the later stages of production that we were able to really refactor the code well enough to see the full benefits of the MVC structure. In retrospect, even though it would have taken more time in the early stages to have developed with MVC in mind it would have made it easier to work on various aspects of the code without worry of conflicts.

Conclusion
Developing this project from the concept to the final game was an learning experience for us as a team. We are satisfied with how far the game has progressed from the monochromatic version that the prototype once was; but there is still room for improvement. There are a number of things we can expand upon if the development of this game is to continue, but the game itself is complete and we believe it to be a success.

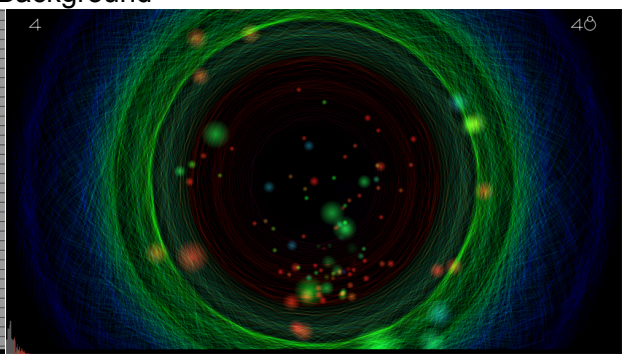
Experiments



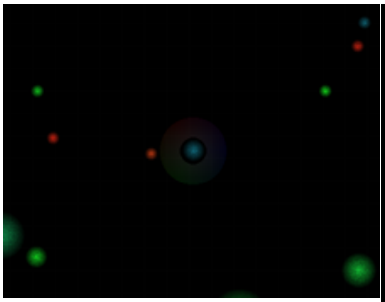
Gridded Background



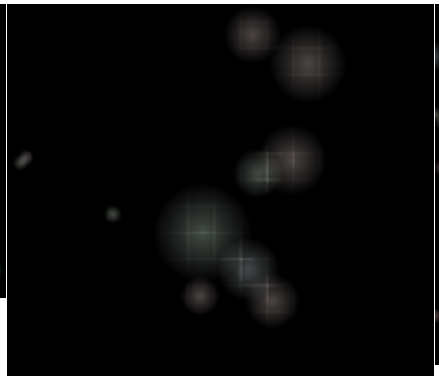
Solid Theme



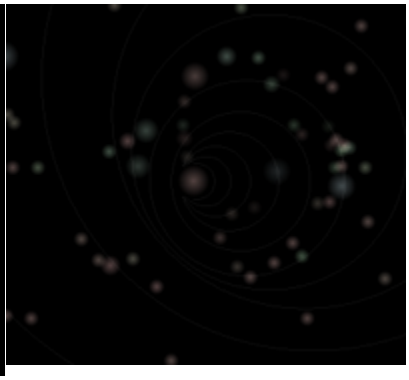
Intense Visualizer



Hue Wheel



Gridded Highlighting on Edible Enemies



Tunnel Background