ReadMe.rst

About

Summary

General:
This game consists of a probe controlled by the player, missiles fired by the player (only partially implemented), randomly generated stars, randomly generated asteroids, particle-effects, and a webpage containing, a backstory, instructions, and a control-panel for the graphical settings.
I stopped counting exact hours after around 50, but I invested over 125 hours in this project. For three weeks, about the only thing I did when I wasn't eating, sleeping, *etc* or doing other homework, was working on this assignment. The reason I went overkill on this assignment, was because I didn't realize we had a project1 until the Tuesday before it was due (it was due the next day), and I wanted the overkill to make up for the lateness.

Code:
There are around 2,000 lines of code in this project, between the Javascript itself, the webpage, the css *etc* for that webpage (which I revised substantially throughout this assignment), and the GIMP plugin (which I originally wrote for hackathon3).
I wrote 100% of the code: Javascript, HTML, and LISP. I did not use any tutorials, although I did, of course, have to use Google, Stackexchange, etc to learn how to do certain things. I also used Wikipedia for information regarding stars.
I'm not a math person. The last and highest math class I've taken, precalc, was two years ago, and I did mediocrely. As a result, vectors are not rotated. It took me a week to figure out how to rotate the locations of things. I also had to ask four people for help. Given this, given my tremendous work-ethic (this assignment is pretty much the only thing I've done for the past three weeks), given that I'm not even a Computer Science major, I've been told by the professor to not worry about this.
The physics aren't completely realistic, even ignoring the lack of vector rotation. However, they're a lot more realistic than the original asteroids game! One of my original goals was having much more realistic physics.
I have a "speedhack" option which disables certain computationally intensive behaviors, such as calculating more stars than fit on the screen. These such computations are necessary for rotation to be realistic, as otherwise, stars and rocks are deleted upon leaving the screen, when they should have stayed on it for a time. As a result, with this speedhack on, stars will form a circle when you are spinning rapidly.
I made a lot of optimizations to the code's algorithms, usually by very, very slightly increasing memory usage in order to drastically reduce CPU usage. There is still a lot of room for improvement.
The code is both modularized and not modularized. Generally speaking, only things which are called more than once are modularized. As mentioned in the optimization bit, there is still room for improvement.

Hardware:
The computer which I used to program this -- namely, my laptop -- is very slow, and quite old. The game's brightness was, accordingly, tuned for my laptop. Unfortunately, every other computer out there seems to diplay the game as being much, much darked than does my laptop. So be aware that the darkness was not intentional. However, also be aware that the darkness actually increases realism -- in actual space, you would detect an asteroid not necessarily by what it reflects, but by what it occludes.
The YouTube video was recorded on another computer because of the fact that my laptop can barely run this game on minimal settings. I would say that an Intel i3 is the recommended specs, and that whatever ancient AMD processor I have in my laptop is close to the minimum specs.

Gameplay:
Unlike in the original Asteroids game, the ship stays in the center of the screen, while other things appear to move around it. This allows the creation of a seemingly infinite universe.
The only controls are movement. Missiles are partially implemented.

There is an exploit: If you start out by spinning rapidly in either direction, you will never die. This is because I don't know how to rotate vectors.
Your score increases gradually the longer you survive. The rate at which your score accumulates does not change. Destroying an asteroid with a missile would have been worth 10 points, and launching a missile would have cost you 5 points. Your score is referred to as "research points", since the longer you stay alive, the more data the probe is able to collect. Destroying an asteroid gets you the chance to look at its innards, and so is worth points. Each missile you fire is one more missile you had to pay for before launching the probe, and so is that much less money you had left over to process your research, which means that many fewer research points you can earn. Collecting space-debris increases your score by 1 per particle, as it presents additional data points for your research. If you are playing with reduced particles, your score increased twice as quickly as the baseline; and if you're playing without rock collisions, your score increases four times as quickly as the baseline. This is to allow slower computers to achieve comparable scores.

Missiles:
Missiles are only partially implemented, but the professor told me that I have enough for a 100% as-is, and that I didn't need to write any more code. The assets are there, though! I have a missile-dropping sound-effect (It turns out that if you modulate the amplitude of brown noise, it sounds like a crash; and if you do it over a longer period of time, it sounds like a sliding gate of sorts; and, so, I used that for the audio), and I also have missile art (I based it on the Apache Hellfire missile).

Graphics:
All art was made from scratch by me. I made the asteroid-textures in GIMP, and I hand-drew in a GNU/Linux analogue to Microsoft Paint (Kolourpaint) the probe (the player's ship) and the missile.
The colors of the art were sampled from actual pictures of the things the art represents. No, an RGB color can't be copyrighted. Fun fact: The orange center of the player's ship is sampled from the Orion moonlander.
Rotation of art was done using a GIMP plugin I spent 18 hours writing in LISP.
All other graphics are generated in Javascript.

Audio:
All sounds were generated completely from scratch using only Audacity, and all of them used only Audacity's generated brown-noise as their base. I'm a freaking wizard.
The "music" (ambiance) I'm classing separately from the sounds. It plays in the background while you're playing the game, and simulates the sounds inside a spaceship, complete with radio-chatter. The radio-chatter is actually a conversation between me and my dad about what he did outside that day. I scrambled and distorted it, and threw in a couple samples of him and me saying radio codewords. I generated the beeps using Audacity's sine-wave generator.
Sound does not play when things that are not the ship collide with things that are not the ship. That is because this is in space, and sound cannot travel in a vacuum. In real space, you actually wouldn't even hear the monopropellant (since it's so quiet irl), but I couldn't bring myself to leave it out.

Stars:
The colors, sizes, luminosities, and frequency of appearance for the stars are all scientifically accurate, using information from Wikipedia. The exact link is in the code; but no content was used from Wikipedia. I'm using the same license, anyway; so even if I had used content from Wikipedia, it wouldn't be a legal issue.
The stars have a random amount of parallax, reflected in their luminosity, size, speed, and acceleration relative to the player. This results in a many- layered, essentially 3D parallax effect.
Stars don't noticeably respond to your speed until you're moving at ridiculously fast speeds -- but they do respond!

Particles:
When objects, such as asteroids, the player's drone, and (had I kept working on it) missiles hit particles, the particles go away.
Different kinds of particles take different amounts of time to decay.

The player's drone's nozzle heats up and emits black-box radiation (again, using realistic colors) depending on how long you accelerate forwards. It also cools down when you are not accelerating.
Monopropellant is used for movement in other directions.
The player's probe, upon hitting an asteroid (or a missile, had I finished) explodes, emitting not only debris particles that match its colors, but also monopropellant and liquid fuel.

Asteroids:
Even the asteroids have parallax. This is most noticeable when asteroid collision and textures are turned off in the settings. The longer an asteroid is on the screen, the brighter, faster, and larger it gets. This is a subtle effect, but makes the game continue to feel realistic even when asteroid collision is off. Having the asteroids' characteristics vary per how long they've been on the screen makes it such that asteroids that are supposed to be further in the background never manage to appear on top of asteroids that are further in the foreground. It also allows asteroids that are far in the background to safely pass below asteroids that are far in the foreground without the two touching.
The shapes of the asteroids are randomly generated.
The number of sides of each asteroid is 11. 11 was chosen instead of, *ie*, 10 or 12, because prime numbers result in less identifiably regular asteroids. For example, allowing 12 sides make a Star of David asteroid much, much, much more likely. Prime numbers maximise realisticness. 11 was chosen instead of a smaller number, as smaller numbers allow for a much smaller variety of shapes. 11 was chosen instead of a larger number, as larger numbers create irregular spikeballs that look nothing like asteroids. Insanely high numbers create very laggy circles.
Asteroids have a 1px "shadow" around them which makes it easier to tell which asteroid is on-top when two asteroids cross over each other.

Meta

Author(s):

- Miles B Huff (811296033)

Date Submitted: 2017-02-22

License

Usage

Open ./public_html/csci4070/project1.htm in the latest version of Firefox. Compatibility not tested in other versions or browsers.
As this is just a snapshot of the relevant files from my school website, you can also simply visit `http://cobweb.cs.uga.edu/~huff/csci4070/project1.htm`.

Additional Information

Statement of Academic Honesty

The following code represents my own work. I have neither received nor given inappropriate assistance. I have not copied or modified code from any source other than the course webpage or the course textbook. I recognize that any unauthorized assistance or plagiarism will be handled in accordance with the University of Georgia's Academic Honesty Policy and the policies of this course. I recognize that my work is based on an assignment created by the Department of Computer Science at the University of Georgia. Any publishing or posting of source code for this project is strictly prohibited unless I have written consent from the Department of Computer Science at the University of Georgia.

Grading.rst

Definition of Game Objects

Asteroids, the player, the player's probe, missiles, particles, stars.
And my pixel art is damned good.

Animation of Game Objects (Spritesheet)

The asteroid texture is a moving spritesheet. I generated it using my custom GIMP plugin.
And my art is good.

Interaction between Game Objects

There is collision between:

- Asteroids and other asteroids (toggleable)
- Asteroids and particles
- Asteroids and the player's probe
- Unimplemented: Asteroids and missiles
- The player's probe and asteroids
- Unimplemented: The player's probe and missiles

Keeping Score (or Achieving some Objective / Reaching some Destination)

Acquire points by surviving, collecting particles, and destroying asteroids with missiles (missiles only partially implemented).

Complexity (Fixing Pong is the Baseline)

- Over 2,000 lines of code between the game, the HTML, the CSS, and the GIMP plugin.
- Over 120 hours dedicated to the assignment.
- Particle effects.
- Boatloads of parallax.
- A procedurally generated universe.
- Toggleable graphical settings.
- Realistic, borderline 3D graphics.

This is almost certainly and by far the most complicated project1 of anyone in the class. It's more than safe to say that I nailed the complexity requirement.

Overall Look & Feel

The game's graphical and audio assets were created on and optimized for a very old laptop. As it turns out, the game appears much darker on everyone else's computer, and the thrusters don't sound nearly as cool. But it's still very well-crafted and feels almost professional.
The game runs at 60fps. If it is laggy for you, you likely don't meet the recommended specs (you need to have at least an Intel i3 in order to play this game). There are graphical settings that can be changed during the game if framerate is a problem for you.
If you intend to spin rapidly in circles, the stars in the background will form a circle of their own. You have to disable the speedhacks in order to fix this behaviour. However, this comes at a significant cost to performance. Please see `ReadMe.rst` for more information.

Effectiveness of Personal Twist

Personal twists: (there are more than the below, but these are the main ones)

- **Better physics:** Although vastly improved over the original game, my physics are yet imperfect. I'm not a very good mathematician, but I did my best. See `ReadMe.rst` for more information. The original had friction in space, rocks went through each other, lasers had physical form... Needless to say, it was *quite* unrealistic.
- **Better graphics:** I absolutely nailed this one. My graphics are fantastic. They aren't perfect, however, as the asteroid textures have a tendency to wobble. This is a tradeoff of using floats. Forcing the center coordinates for textured asteroids to be integers helped to mitigate the issue, but as the asteroids' sizes are still floats, this did not completely fix the issue, and this tweak comes at the cost of slightly less accurate asteroid movement. Oh, and pretty much everything is parallaxed (the particles are an exception).
- **Better sounds:** My sounds are much more immersive than the original's, to say the least.
- **Egocentric orientation:** I wanted the ship to be piloted more as it would be in real life: from the orientation of the ship itself. Although this involved some complicated math and although I was not able to rotate vectors, this still paid off quite nicely, and allowed me to create the illusion of a limitless universe, instead of having to rely on border- wrapping like the original.
- **More realistic scenario:** In the original, you're in a trianglular ship in an arbitrary asteroid-field. You are equipped with a laser-cannon, and gain points by shooting asteroids (of which there is a limited supply) and flying saucers. In mine, you're in an orbital station remotely piloting a research probe through the Oort Cloud. You only have one probe. Your probe is realistically designed, and is loosely based off several real-life probes. There is no end to the asteroids, and there is no other life in-sight: just you, and cold, empty space.
- **More everything:** My version of Asteroids has so little in-common with the original that almost everything in it could be considered a personal twist. Even the absence of missiles is almost a personal twist.

Documentation

I write extremely clean and extremely regular code, and include visual spacers and explanatory comments all throughout it. It's even tabulated!
The other documentation required for this project has all been provided.
My gallery's got links to everything I've done in this class, and a link to it is included in `ReadMe.rst`.

Demo/Gallery/YouTube

By the time of the demo (Wednesday), I had a functional asteroid field consisting of randomly generated shapes, sizes, velocities. I showed it to you. I probably lost the sheet, because I'm an idiot.
By the time of the demonstrations (Thursday), I had added parallax to the asteroids, as well as a rudimentary starfield in the background. I showed this to you as well.

I would love to show my work to the class, as I am quite proud of the results, even if I did not accomplish all I set out to.
YouTube link: https://youtu.be/6mNNUjJAHOU?list=PL3981E2447CEF026F
Gallery link: http://cobweb.cs.uga.edu/~huff/csci4070/gallery.htm