Written Component

http://users.aber.ac.uk/cba1/iwp/

Executive Summary

Game Description

The player controls a fish by moving the mouse or using the alternate keyboard controls, trying to avoid larger fish (predators) and trying to eat smaller fish and plankton (prey).

The game is played in 30-second levels, with fish being spawned incrementally as the player advances through the levels, making the game increasingly challenging.

Larger fish will actively chase the player- if they catch them, the game ends. Fish that are smaller than the player will flee if they come within a certain range. All fish have randomly generated sizes, colours, speeds and ranges (i.e. the distance from which they can spot the player).

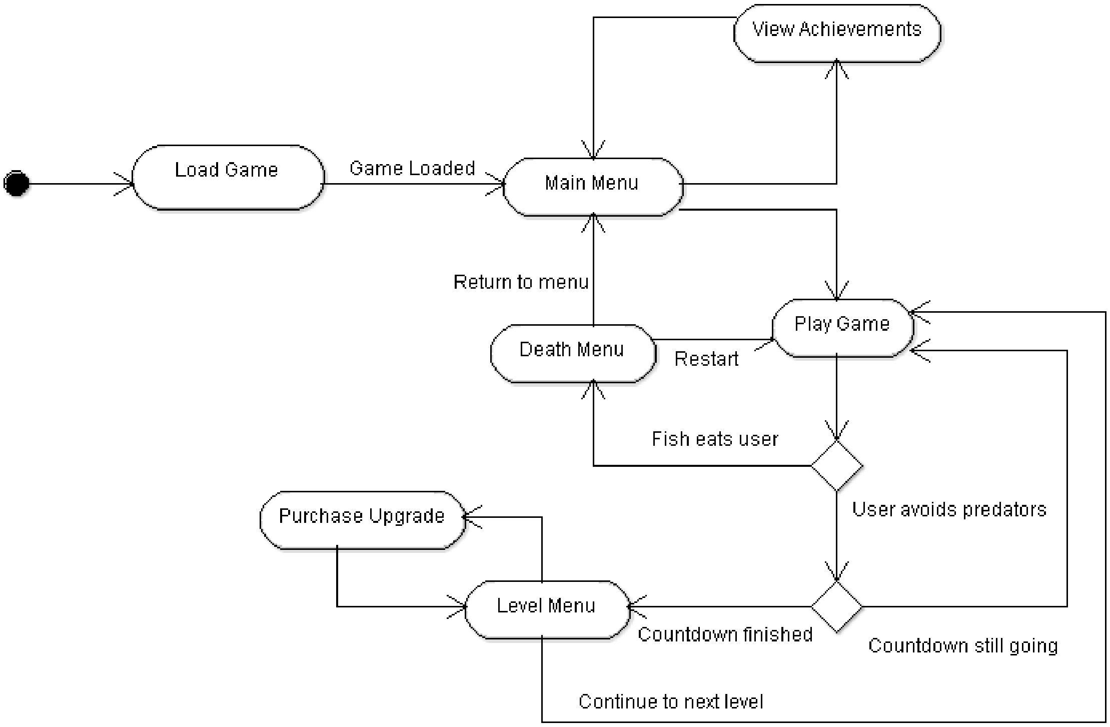
Eating fish and plankton earns the player some XP (experience)- the larger the fish they eat, the more XP gained. Between levels, XP can be spent on evolutionary upgrades including growing and camouflage.

The player's final score is the total XP earned (including what was spent on upgrades). In addition to a high score, players will strive to unlock “achievements” through skilled (and occasionally unskilled!) gameplay.

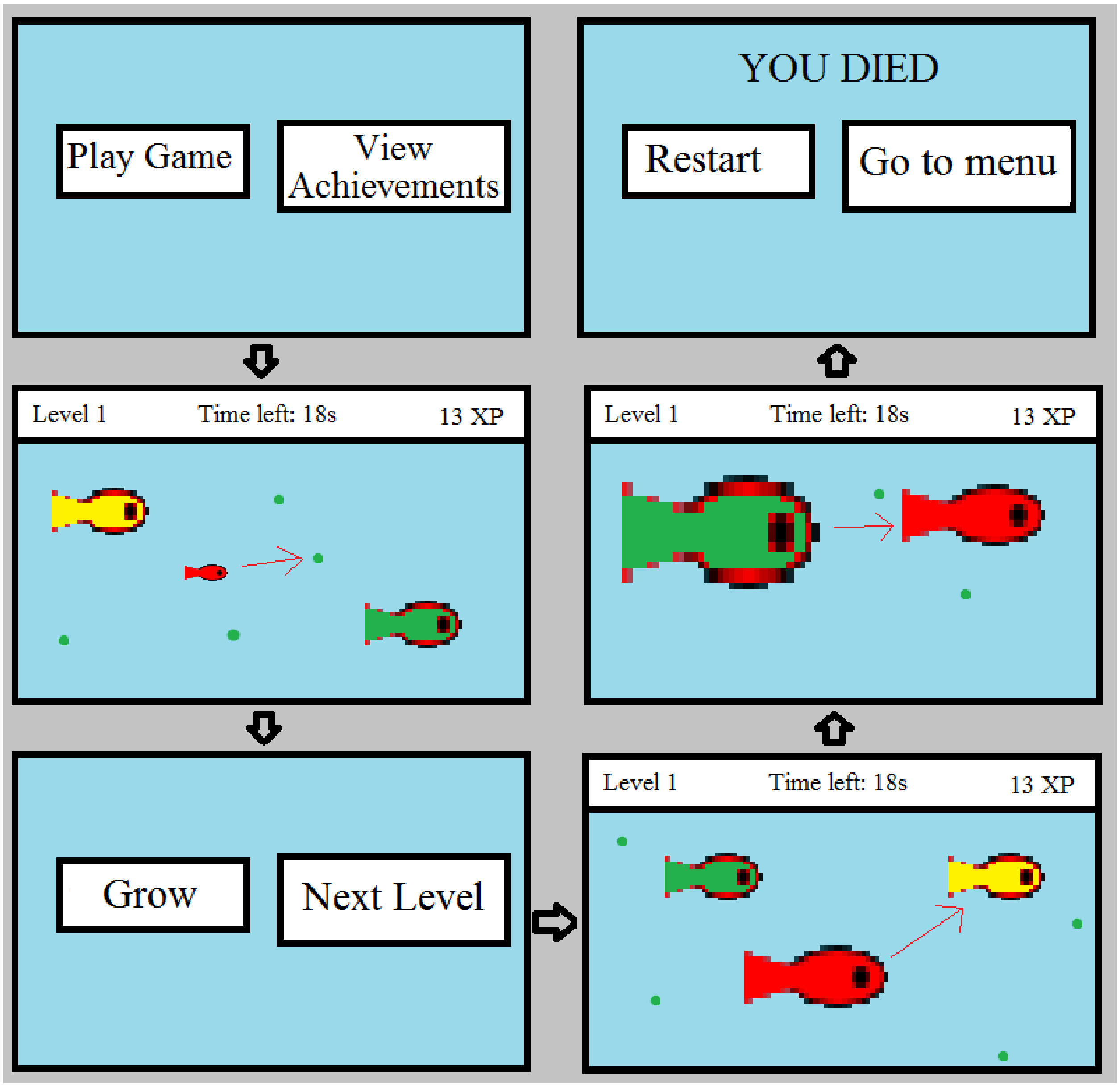
Mouse control is fluid, effortless and highly accurate. The game mechanics were built with mouse control in mind and admittedly do not translate well to the alternate keyboard controls, though this has been implemented and the player can move by using the W, A, S, D keys.

Players interact with menus by clicking on buttons on the canvas. During gameplay, clicking on the water leaves a poisonous toxin in the area clicked, but only if the player has bought the poison ability. Fish that swim into toxins die instantly, at which point the player can eat them.

Activity Diagram



Storyboard



Frame 1: main menu.

Frame 2: a typical game screen. The small red fish (the player) is heading towards plankton.

Frame 3: level screen, reached at end of countdown, gives player chance to purchase upgrades.

Frame 4: player has grown and can eat smaller fish.

Frame 5: a larger fish goes to eat the player.

Frame 6: the game ends.

Technical Overview

I’ve used two JavaScript APIs; RequireJS for asynchronous downloading of the game files, and Mousetrap for handling keyboard input. Everything else is written in object-oriented JavaScript; a powerful software architectural pattern which is perfect for modelling real world objects and facilitates the idea that the program contains objects interacting with one another, rather than a list of functions and computer instructions.

I avoid code duplication through the use of inheritance. For example, Creature.js defines a creature object with attributes and functions that are shared across all types of creature. I also use the Builder design pattern- in Menu.js, for example, the mouseMoved() function calls the mouseMovedActions() function; an empty function which has no consequence on the program's operations. However, classes inheriting from Menu.js, e.g. LevelMenu.js, can override the mouseMovedActions() function if they want the program to do something when the mouse moves (*in addition* to what is defined in the mouseMoved() function). This means that I can “plug in” extra functionality for specific classes without having to redefine the entire function.

I could have used any of the following alternative technologies:

- Flash
- Silverlight

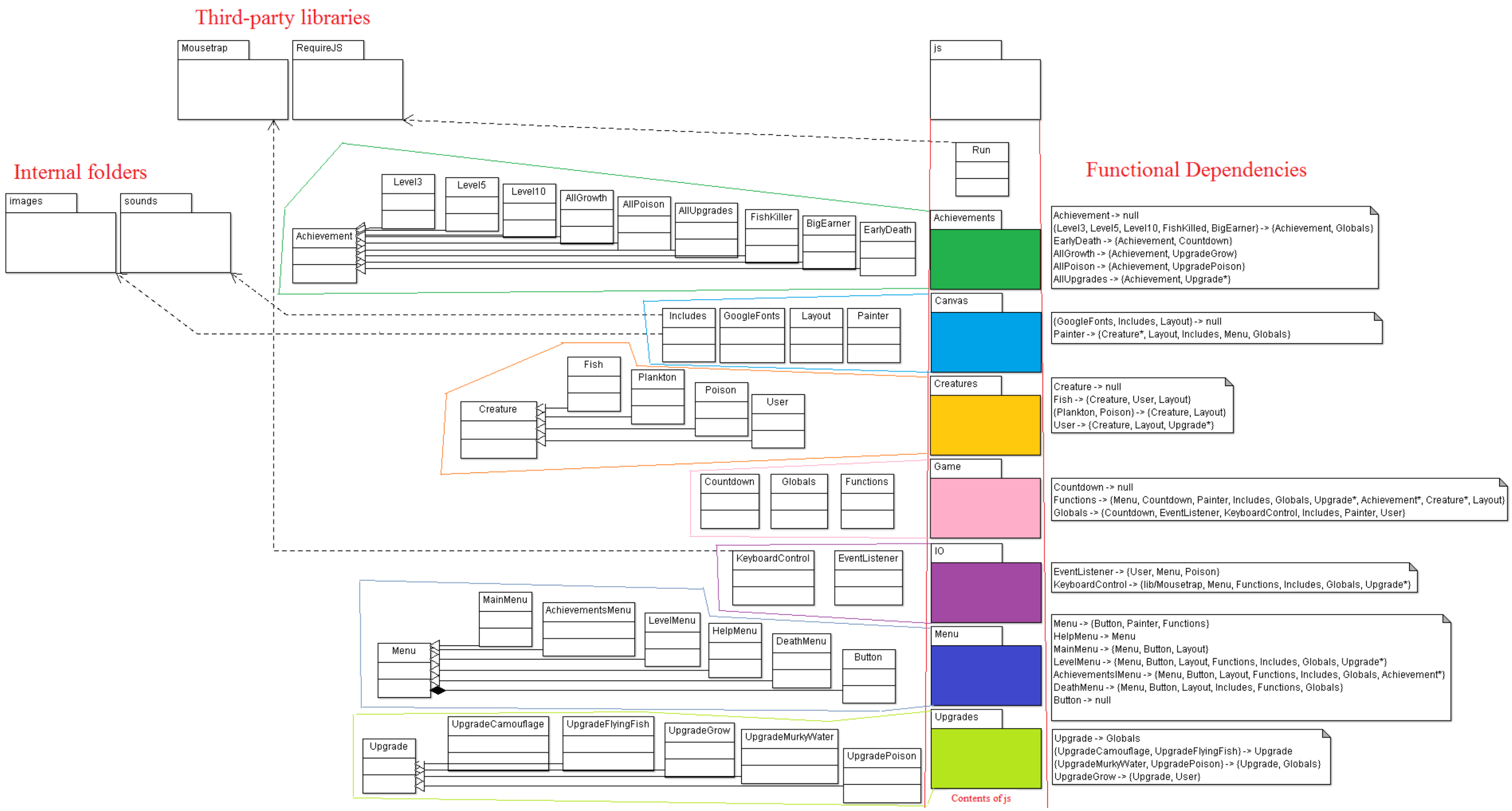
These require proprietary software; they aren't open-source, which is a disadvantage. However, my biggest gripe with these technologies is that the user must install additional plugins in order to be able to play the game. This is inconvenient and in some cases, impossible. There may not be a version of the plugin for the user's OS or browser, and users in certain countries might have licensing issues.

Finally, it's not wise to rely on another company's platform for your own franchises. Even if the game has commercial promise and starts to become successful, this can be immediately tarnished with a quick change to the platform company's terms and conditions.

Well constructed HTML5 and JavaScript aims to be free, widely accessible to developers, and most importantly, cross-platform. This is why I would choose to use these technologies, even if it wasn't a requirement of the assignment.

The website hosting the game is written in PHP solely for its ability to include files (header and footer) so that the menu does not need to be copied and pasted on every single page. There are no server-side elements to the game itself.

The below diagram gives an idea of the scope of the game and shows which files are dependent on which; this is very significant for deciding the file download order in Run.js.



Functional Dependencies

Achievement -> null
{Level3, Level5, Level10, FishKilled, BigEarner} -> {Achievement, Globals}
EarlyDeath -> {Achievement, Countdown}
AllGrowth -> {Achievement, UpgradeGrow}
AllPoison -> {Achievement, UpgradePoison}
AllUpgrades -> {Achievement, Upgrade*}

{GoogleFonts, Includes, Layout} -> null
Painter -> {Creature*, Layout, Includes, Menu, Globals}

Creature -> null
Fish -> {Creature, User, Layout}
{Plankton, Poison} -> {Creature, Layout}
User -> {Creature, Layout, Upgrade*}

Countdown -> null
Functions -> {Menu, Countdown, Painter, Includes, Globals, Upgrade*, Achievement*, Creature*, Layout}
Globals -> {Countdown, EventListener, KeyboardControl, Includes, Painter, User}

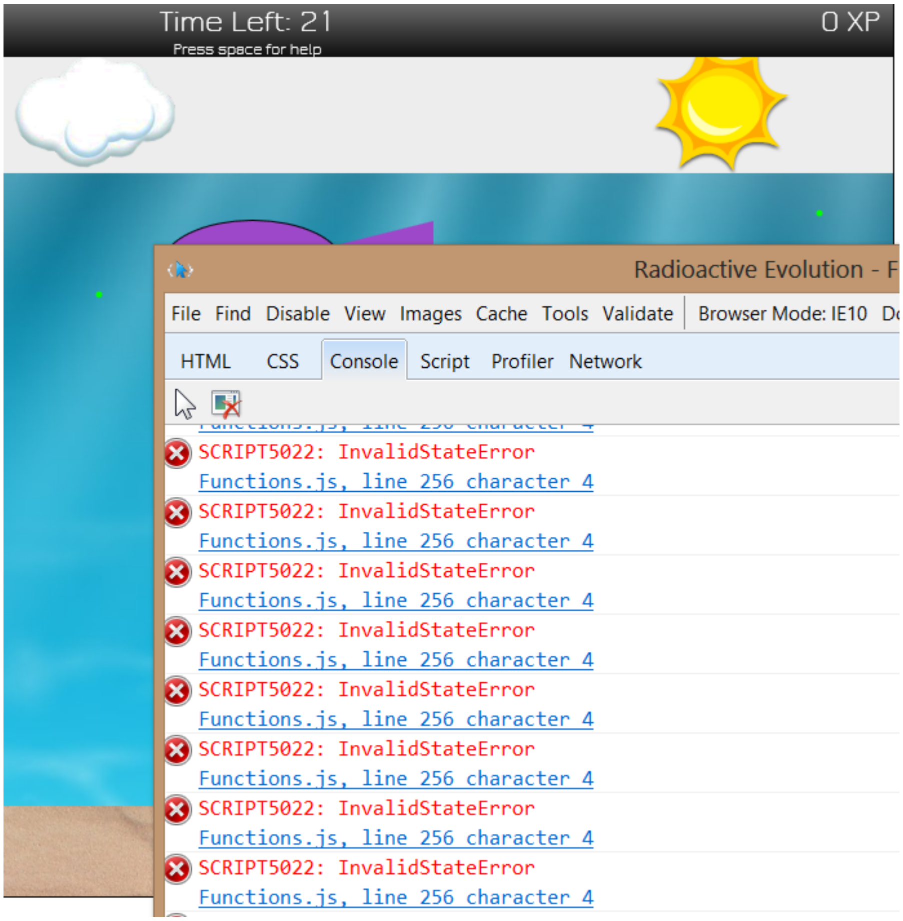
EventListener -> {User, Menu, Poison}
KeyboardControl -> {lib/Mousetrap, Menu, Functions, Includes, Globals, Upgrade*}

Menu -> {Button, Painter, Functions}
HelpMenu -> Menu
MainMenu -> {Menu, Button, Layout}
LevelMenu -> {Menu, Button, Layout, Functions, Includes, Globals, Upgrade*}
AchievementsIMenu -> {Menu, Button, Layout, Functions, Includes, Globals, Achievement*}
DeathMenu -> {Menu, Button, Layout, Includes, Functions, Globals}
Button -> null

Upgrade -> Globals
{UpgradeCamouflage, UpgradeFlyingFish} -> Upgrade
{UpgradeMurkyWater, UpgradePoison} -> {Upgrade, Globals}
UpgradeGrow -> {Upgrade, User}

Software Testing

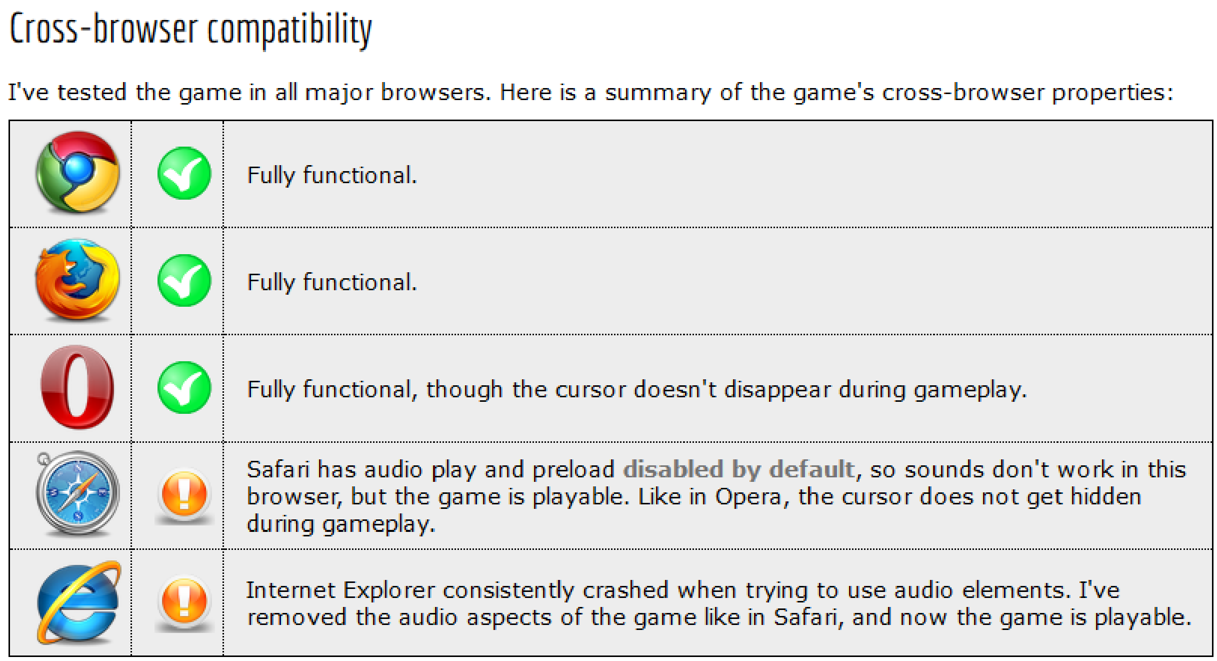
The biggest problems came from including audio. Safari disables preload and autoplay for the benefit of its customers who may be on a cellular network and are charged for their data usage. Internet Explorer ran into similar problems. My attempts to play my game in IE normally (with audio) led to these messages in the console:


Time Left: 21
Press space for help
0 XP
Radioactive Evolution - F
File Find Disable View Images Cache Tools Validate Browser Mode: IE10 D
HTML CSS Console Script Profiler Network

```
SCRIPT5022: InvalidStateError
Functions.js, line 256 character 4
SCRIPT5022: InvalidStateError
Functions.js, line 256 character 4
SCRIPT5022: InvalidStateError
Functions.js, line 256 character 4
SCRIPT5022: InvalidStateError
Functions.js, line 256 character 4
SCRIPT5022: InvalidStateError
Functions.js, line 256 character 4
SCRIPT5022: InvalidStateError
Functions.js, line 256 character 4
SCRIPT5022: InvalidStateError
Functions.js, line 256 character 4
SCRIPT5022: InvalidStateError
Functions.js, line 256 character 4
```

The line in question attempts to alter the currentTime property of the audio element. It appears that the audio element is never loaded, presumably for similar reasons to Apple.

Other than the audio, Radioactive Evolution is more or less cross platform. See this screenshot taken from the website:

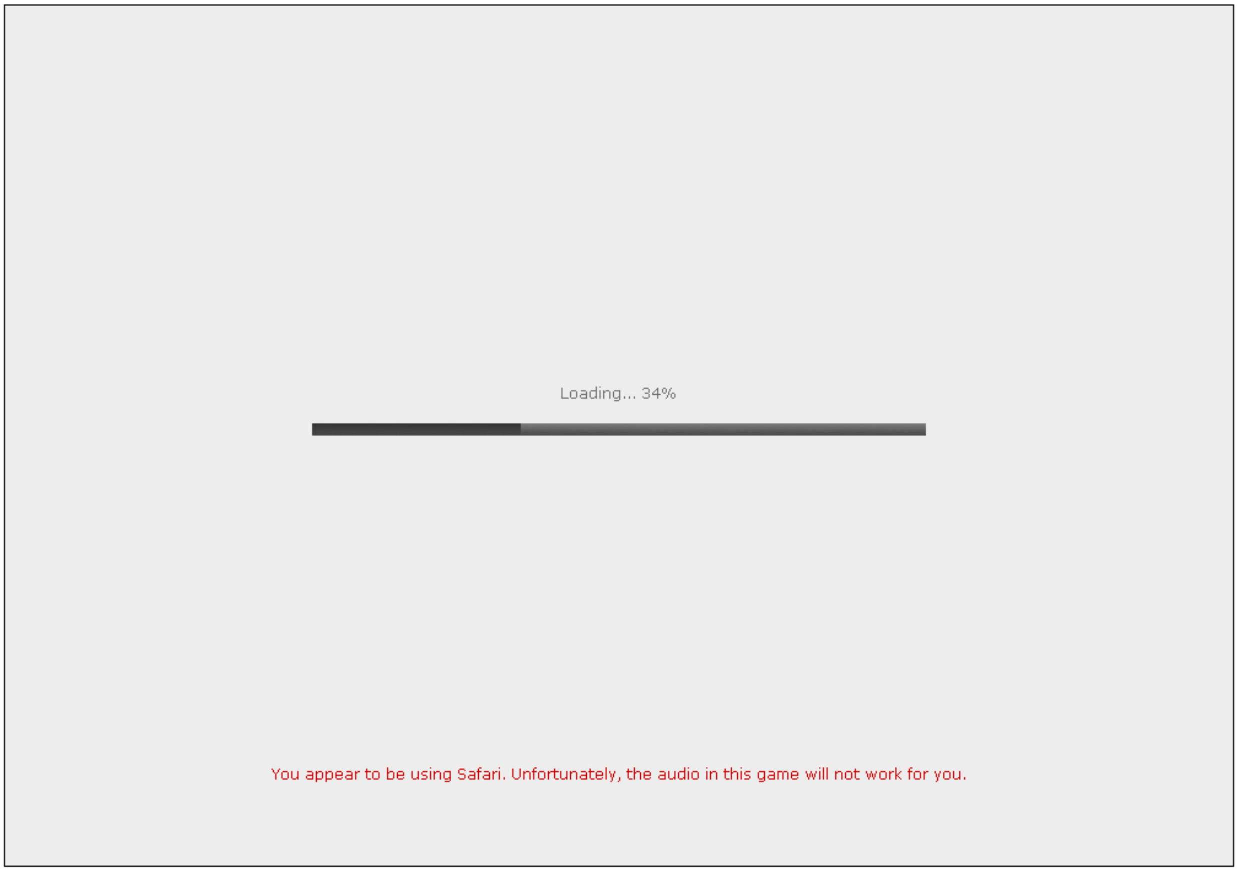


Cross-browser compatibility

I've tested the game in all major browsers. Here is a summary of the game's cross-browser properties:

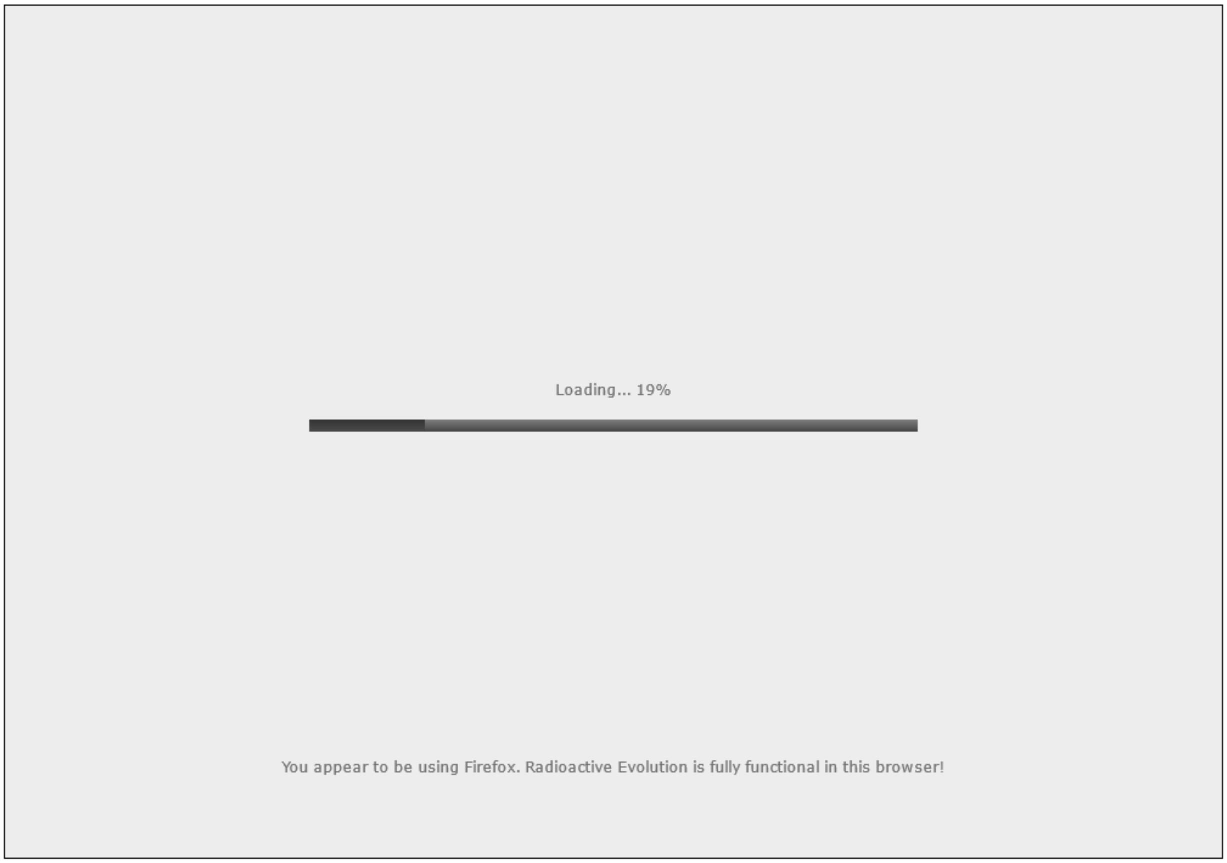
	✓	Fully functional.
	✓	Fully functional.
	✓	Fully functional, though the cursor doesn't disappear during gameplay.
	!	Safari has audio play and preload **disabled by default**, so sounds don't work in this browser, but the game is playable. Like in Opera, the cursor does not get hidden during gameplay.
	!	Internet Explorer consistently crashed when trying to use audio elements. I've removed the audio aspects of the game like in Safari, and now the game is playable.

The game detects the user's browser in Run.js and warns the user with an informative message if their browser is expected to have difficulties.



Loading... 34%

You appear to be using Safari. Unfortunately, the audio in this game will not work for you.



Loading... 19%

You appear to be using Firefox. Radioactive Evolution is fully functional in this browser!

Reflections and future work

I've thought of a few improvements and tweaks I'd like to introduce in future versions of the game- a full list can be found in the game's README file, or here on the website. The highest priority would be to get audio working on IE and Safari. More exciting developments might be the introduction of more challenging enemies as levels progress, for example crabs which crawl along at sand level, or jellyfish with far-reaching tentacles.

Radioactive Evolution is now open source, made clear on my website which offers documentation and a download link for the zipped software. I doubt that it has any commercial promise- the number of free online games (HTML5, Flash or otherwise) is astonishing. Even if it began to make money, an enthusiast might create a replica under a different name and the monetary value of my game would become negligible. My game is good, but I don't think that this platform is commercially viable for a client-side game, so I am happy for it to be put to use in the open source community.