Team Hydra: Final Report

CS 467 Winter 2017

Dan Beyer Micah Grossman Joe Valencia

Introduction

By the end of this term, we probably learned more about each other than we might've wished to from a start. The good news is, the term also produced a mildly amusing, albeit charmingly rustic rendition of *Food Truck Wars*, a video game built from the foundations of real time shooter (RTS) video games, past. From the shoulders of titles like *Command & Conquer* and *StarCraft* we aimed to create a nostalgic trip to a walled garden world, void of politics and uncontrollable worries. Our aim was to build a game using modern standard web technologies, relying predominantly on javascript to provide the heavy lifting of creating and managing the game world and objects, as well as initialize the server on which the game is run and provide a means to store saved progress via web cookies.

In *Food Truck Wars*, the goal is domination and control of turf. You choose how you want to enter the market: as team America (slinging burgers and breaking boundaries), Coffee (serving up victories with plenty of pastries), or Fusion (gathering the best dishes from half the globe). How your crew will thrive or falter is up to you. The user must juggle economic management of their food trucks and carts, earning enough money to support a thriving army support of crew to prepare dishes, earn more money and destroy the opposing team's trucks and forces to ensure absolute victory.

Over the course of development, we learned some very hard lessons that came from not properly understanding and planning out underlying game systems that all parties needed to develop on and build from, along with good old fashioned communication issues and showstopping bugs in areas of game development that no one in our group had prior experience in. Nevertheless, we were able to congeal and come together as a well functioning team unit by the final couple of weeks in the project.

The game is provided in its full file directory, and can be initialized with node "server.js"; however, you can directly access the game with us setting up a forever instance of the game at http://flip3.engr.oregonstate.edu:55623/index.html. For best results, it is recommended to engage in the game through a Google Chrome browser, and set it to fullscreen mode. To control the game, Only a mouse is necessary to select and move units, create buildings, units and juggle upgrades for your crew.

Program Description

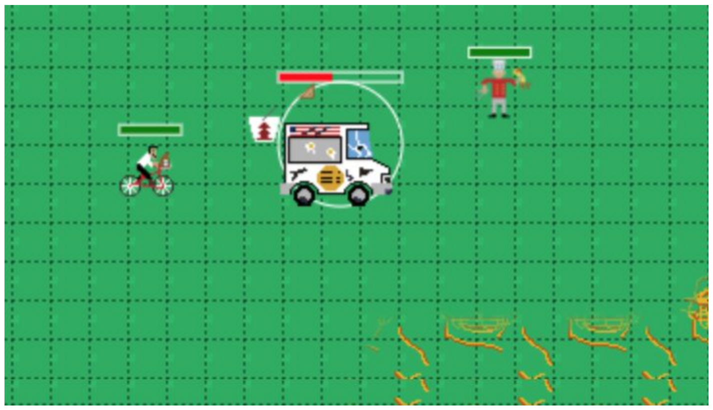
At launch, a user chooses from 3 teams that all have matching unit types, with each team holding particular advantages characteristic to their origins. Team Coffee for instance has a propensity for higher speed values, though all that coffee drinking has lead them to have slightly weaker bones and strength. The fusion team is rather well rounded, with some surprise strengths and weaknesses that are a mystery to be understood and leveraged for victory. Team america, as expected, may be a bit larger, walking around with a bigger hitting stick, but are dragged down by their tendency to be slower.

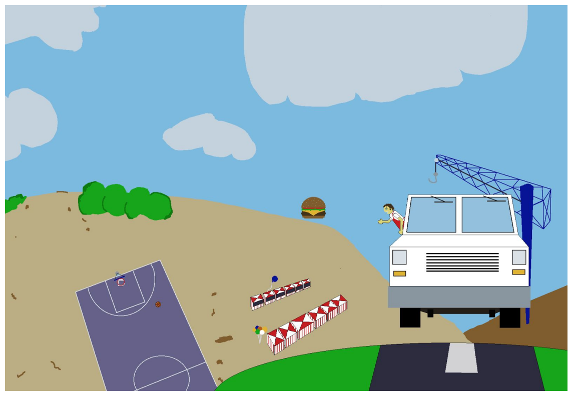
Pristine, damaged, and nearly dead sprites for the various teams

Each team starts with a headquarters, a food truck, and two specialty chefs. The economy is managed by a primary currency, that passively increases in revenue return based on the number of food trucks you have, along with the a multiplier based on the number of workers you've added to each of the food trucks. With the money rolling in from your food truck ventures, you can move your food truck around to help build outposts and buildings that can then generate more advanced units. Each food truck is capped at 12 workers, but each additional food truck is expensive. The user needs to make a decision about whether to prioritize a streamlined economy with a small number of food trucks, or trying to expand quickly to ramp up their primary economy in a dangerous, but potentially fruitful, fashion. From headquarters, the user is able to manage upgrading units, along with increasing the available supply to be used by hiring and building additional workers or vehicles.

To upgrade units, a secondary currency is used to spend on these types of attribute improvements. Secondary currency is generated whenever there is a food truck actively within range of a building that produces units. Each building also has a distance multiplier that increases exponentially with the distance from the team headquarters. The farther you are from home with a more established presence of food trucks and buildings, the greater the acceleration of secondary income, and the faster you'll be able to earn upgrades on attack, defense, speed and range that could prove the decisive factor in your victory. This encourages fast expansion of territory, but it can also encourage users to punish those who become too greedy in their expansion.

To earn a victory, a decisive advantage needs to be inflicted upon the opposing team. Absolute win conditions trigger when either the enemy's headquarters has been eliminated or all the enemy's food trucks have been destroyed. Be careful about how you spend your money and franchise your food empire.





This food truck stands no chance against a delivery boy lobbing takeout boxes and a specialty chef slinging samosas

Stretch yourself too thin and an early loss of units may prove insurmountable. Playing conservatively, you may want to focus on defense and upgrade levels (applied universally to all units or vehicles, depending on the type of upgrade selected from headquarters).

Program Organization

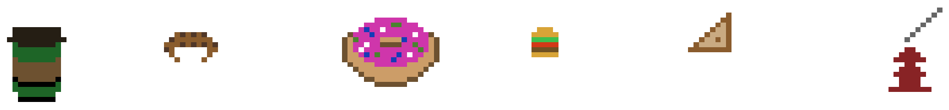
Nearly everything in the project is broken down into component javascript files, along with a heavy reliance on CSS style sheets to skin and theme button and unit types based on team selection choice by the user. At the most object oriented, we have files to manage the attributes and methods for buildings, units, and vehicles, with each type of object referencing one sprite sheet wherein all permutations and similar object visuals are housed. For general engagement with the environment and possible blockers and other units, we have our pathing logic housed in path.js, along with additional functionality in each of the movable objects in their methods for handling commands and move directions.

The main functionality that initializes all components, tracks global variables related to win/loss scenarios, and handles the primary task of rendering and drawing on the HTML canvas to display the game itself comes from our core game.js file. Along with sprite sheets used to draw the various units and objects, we also employed .wav audio files using the Audio object compatible with Chrome for sound effects.

Functionality pertaining to AI and the enemy player is predominantly housed in its own AI.js file, though there are a lot of touch points in various other files that extend the computer's capabilities. In build.js we store the logic and conditions for creating new units or buildings, along with buying and applying various upgrades for your team. Economic progress and the factors that influence it are corralled in an economy.js file, with the layout and starting objects hosted in maps.js file. Besides game.js, our most significant file would arguably be mouse.js, which incorporates all of the logic that allows for unit selection, targeting, drag-select, and everything else that makes up the UX of our game. All of the sidebar and bottom bar theming and possible button display is managed by mouse.js

Upon page load and team selection, game.init() gets the ball rolling that starts a process to load up all game items relating to the map, then continue to draw the map and units on screen until a victory condition is triggered to end the game. Our AI is built to swap between priorities or personalities, per se: general attack and scouting, economic focus, and building focus. On easy difficulty, they shouldn't prove too much of a hassle. With harder difficulties and a stronger emphasis to go with upgrades and attack in numbers, and you'll find yourself with quite a challenge to beat them.

Our projectiles.js is where we house the management of attacks and implement the various possible upgrades and change in stats when two units are fighting. Depending on the level of damage inflicted in the projectile file, the sprite sheet offset may be updated for



If you had to be hit by one of these, I would probably go with the croissant particular objects to reflect their current degree of damage. Similar to Civilization, we implement a defense system that has a defense attribute paired to current hit points. The weaker you are, the less successful your defense will be against pain.

Development Tools

The two most important tools to our group's development proved to be Github and Slack. Slack was absolutely essential to keep communications coordinated and have an easy record to refer back to. The permeability of the medium allowed it to be the singular resource for both team meetings were next steps and expected work loads were divied up to shorthand chat while 2 or more teammates were actively working on the same bug or some chunk of code.

With github, we had grand ideals at the start of how we would implement it and the importance of also incorporating unit tests. Though we strived valiantly to maintain a routine of one new branch for each new feature to build on and then merge into master, mistakes were occasionally made and there were times at which mitigating merge conflicts proved a nuisance. Chrome developer tools with its ability to step through functions was the next most important tool in development. Given our temporal and physical distances, overall the experience was relatively smooth and yielded pretty stable development contributions, save for major bugs that ended up derailing our initial timelines and gameplan. We'll revisit the topic in deviations from the main plan, but suffice to say we ran into issues with AI and pathfinding that were unfamiliar to all three of us and required significant amounts of group huddle to identify workarounds and continue to progress the game forward. Our github master branch, though taking in nearly 400 commits, never got out of hand and never spent more than a couple of hours at any given time being in a buggy and unsatisfactory state. Self monitoring and worry about merging in broken code proved a valuable tool to keep the chaos in some of the branches from spreading.

Additional important tools were text editors (such as Sublime and Atom), along with Tiled to help generate in game maps and set us on our path to using grid coordinates for item placement. Piskel App, Texture Packer and Asperite were all used in the creation of various sprites and art assets in game.

Teammate Contributions

The three members of our team worked together and shared a lot of the load during development as this was a relatively new experience for all of us. While we worked together on many aspects of the project, we did initially divide our work into the categories recommended by the original project description. Micah was our attack and units developer, Joe handled graphics and engine, and Dan took on AI. Using this broad outline, we were able to come up with a division of labor the gave us individual focus but also allowed for much collaboration in all areas of development.

<u>Daniel Beyer:</u>

Dan initially opted to take on the role of AI/Pathfinding development. This was a new role for him so, just like everyone on the team, this role came with a steep learning curve. Pathfinding included researching and selecting a proven pathfinding algorithm (A* algorithm) and finding an implementation that was straightforward and possible to integrate within our framework. Dan also handled the creation of our map using the software program Tiled. Dan created our initial entity loading functions to begin loading our entities onto the game map. He wrote our movement and attack algorithms for our units and vehicles as a precursor to implementing the AI. He then used these movement functions to create the projectile system for projectile movement. Once this was accomplished, he developed movement and attack AI for both hard and easy modes, as well as AI build logic for the computer player.

<u>Joseph Valencia:</u>

Joseph initially opted to work on the Graphics/Engine development. He designed the mechanical framework of the game, along with many of the display elements. Some of display and engine elements include: the game timer, stats/avatar display for units, drawing unit health/selection, displaying messages in a queue for indications to the user, adding sounds and rotating background music, deviating imaged fog generation, the minimap drawing/clicking elements, the saving/loading of the game, game ending conditions/handling the end of game, some of the screen hiding/display choices, CSS choices, mouse selection/drag selection, and economy/supply display. He also managed the majority of the engine/game.js and game object decisions, which was the main branching off point for the rest of the modules. Joseph also contributed to the discussion/design of the overall game mechanics, and he ended up implementing the economics and upgrade mechanics and API for the game. Additionally, Joseph wrote the AI economics subroutine and also fixed bugs in almost every module of the game as they appeared.

<u>Micah Grossman</u>

At the onset, Micah naively thought we had a good grasp on what were going to be the major areas of development to assign time to. By the 5th week, it became clear that our understanding of the “what” and “how” of building a RTS game like this were way out of wack with reality, and time usage shifted accordingly. Initially Micah was to focus on attack and units. The initial objects, stats, and graphical representations for all units, buildings and vehicles were generated from Micah. Early and towards the midpoint of development, Micah fell in line with Joe and Dan to group trouble shoot and persevere through some particularly difficult periods of applying algorithms for identifying pathfinding, getting the game state to a point that in the final 2 weeks we were able to massively crunch and contribute loads of features to catch u and ultimately come close to meeting our initial planned game scope.

Beyond communication management, weekly video report submission, and overall monitoring of progress and grades with the TAs, when Micah wasn’t writing about himself in the third person, he was also primarily focused on getting sidebar integration for upgrades and object purchases with the make() methods he wrote in build.js, a large source of Micah’s contribution to the code base. The initial concept of food and some of the goofier aspects of the game’s humor might be attributed to Micah as well, though the final game product is definitely a shared product of all three of our visions.

Plan Deviations

Our project followed our initial plan fairly closely but with a few noteworthy deviations. One deviation was the removal of scripted events. At the start of our project, we envisioned having scripted events that would occur at random intervals that would change the playing environment by either changing the map terrain or by introducing non-player character units or events that would affect both the human and AI computer player. We found after significant development time that this plan was just not feasible in our time frame available. We ended up removing scripted events altogether. Related to this deviation was our initial plan of having multiple levels and a level select. The levels were to be state fair, school yard, and a construction yard.

As we aimed to bring the game environment to life in the browser, particular bugs related to loading of objects and nailing down pathfinding dragged down group output for consecutive weeks in a row. It became clear that we would not be able to produce a workable product in the time allotted if we were to try to implement multiple levels, different scripted sequences, and fuller support for true animations. We instead chose to use a single large grassland/forest map with obstacles such as houses and hills. By the final days of the project and the rate at which our development velocity increased, it's clear that with one additional week, we could have stretched and lived up to our initial plan, along with a greatly expanded amount of time dedicated solely to bug hunting.

Conclusion

You really don't know what you don't know, until it hits you in the face with game breaking bugs. More than any other particular facet, simply troubleshooting in a shared group responsibility environment proved the most valuable and arguably most educational part of implementing this group capstone project. Our expectations and assumptions were proven wrong again and again over the course of Hydra's development period. Yet through the miscommunications, the forgotten requirements, the misunderstood algorithm implementations we slowly grew a strong skin to the judgments of our teammates, while remaining receptive to the feedback and internalizing the vital different perspectives teammates provided.

This project was a fantastic experience in software development and collaboration for the group members of team Hydra. We are both proud of what we have accomplished and overwhelmed at the work that it took to create this piece of software. We can take away many lessons from the this experience. The importance of communication and true collaboration was absolutely critical to the success of our project. Though the game was built with relatively standard web development languages and frameworks, it still proved more than enough content for us to have only begun to scratch the surface. For at least some of the members, this surely will not be the last game they develop in their lifetime.