# Contents



Github: https://github.com/JacobShirley95/truck-physics-game

# Game Summary

The game I have made is a side-scrolling, physics-based truck driving game, which goes by the name of Truck Physics Madness for lack of a better name. It uses the Phaser 3 engine coupled with the MatterJS physics engine to create a fun 2d game experience.

## Objectives

The main objective of the game is to earn as much money as possible. This is achieved by carrying crates of different values on the back of a truck, from the start of the map to the end. Along the way, the player will have to control the speed of the truck as it goes over obstacles so as to avoid crates falling out the truck. The money earned at the end of each level is calculated by multiplying the number of each type of crate by their value, plus a time bonus.

## Rules

The only rule of the game is to keep the crates in the truck until the end of the map and not flip the truck.

## Gameplay

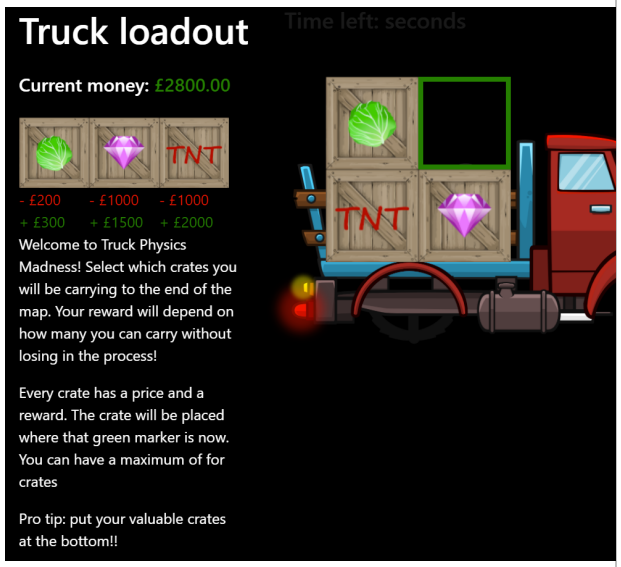
The game starts on a loadout screen. Once the player enters this screen, they will be able to select crates of different values to load onto the truck. The player's money is also displayed on this screen and decreases when a crate is selected. A green marker is placed on the image of a truck to show where the crate will be placed. A maximum of 4 crates can be added currently. Below each crate two values are displayed: the cost of the crate, and the reward for getting the crate to the end of the level. When a crate is clicked, the player's money is reduced by the cost of the crate. If the player's money is not enough to buy a crate, the crate cannot be clicked.

Once the player is ready, they click the "Start game" button. The game is a side scroller; pressing the right arrow key moves the truck forward, pressing the left one slows the truck down or reverses it. The background has a parallax effect to make parts of the background look closer or further away. The player is trying to keep the crates of the truck inside the truck until the end of the level, while trying not to drive too fast over obstacles. When the player reaches the end of the level, another screen will pop up showing the player's rewards for completing the level. The player's new money will be a calculation of the reward for each crate multiplied by the number of the type of crate. There will also be a reward for the speed at which the level is completed. Once finished, there will be a button to proceed to the next level. There are currently a total of a three levels, each with different obstacles and different scenery. If all levels are complete, they can enter the highscores page where the last 10 highscores from their games are displayed.

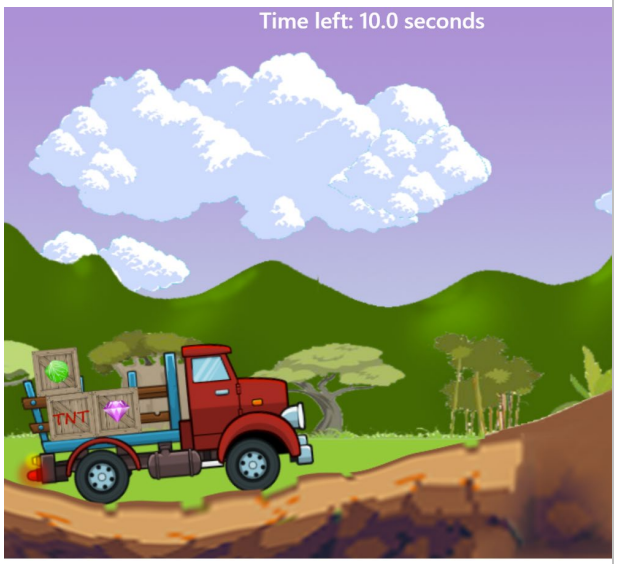
Some other small features of the game: when a TNT crate collides with the ground, an explosion is caused, which adds force the truck away from the explosion. There are some fade in effects on the user interface to make it more interesting. Background layers can also move independently of the rest of the map. This is used by cloud layers, for instance.

# Storyboard

**The main truck loadout page**
The screen where crates can be placed onto the truck




**Map 01**
The first map




**Finish screen**
This is the screen which appears after a level is completed

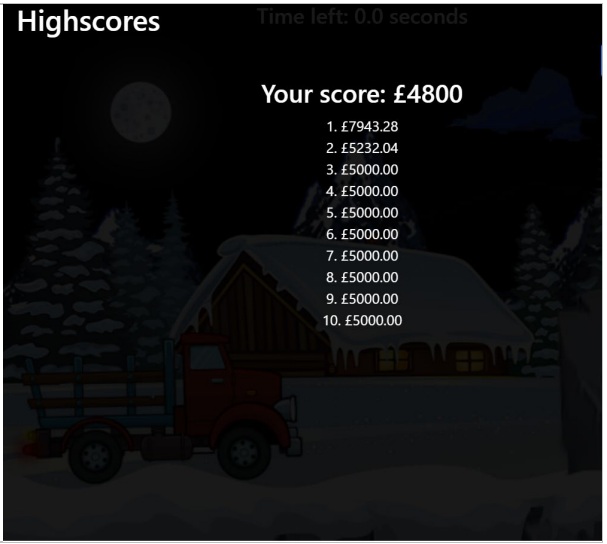

**Map 02**
The second map

**Map 03**
The third map

**Highscores Screen**
The highscores screen appears when all levels are complete


Time left: 33.4 seconds


Time left: 0.0 seconds


Highscores
Time left: 0.0 seconds
Your score: £4800
1. £7943.28
2. £5232.04
3. £5000.00
4. £5000.00
5. £5000.00
6. £5000.00
7. £5000.00
8. £5000.00
9. £5000.00
10. £5000.00

## Implementation specification

Since this is a physics game, I used the MatterJS physics engine that Phaser 3 integrates with to perform physics calculations on the truck and crates. **To run the game, execute "npm start" from the root directory of the game, and navigate to http://localhost in a browser.**

All the code is located in the "src/" folder. Each module uses the standard Node CommonJS format. The code is bundled use webpack, whose configuration file is located in the root of the project in a file called "webpack.config.js". Build and run scripts are located in the "package.json" file under the "scripts" property.

The structure of the project is as follows. I attempted to keep related code together in individual modules. For example, most of crate handling code is kept in the file "crates.js". In this file crate assets are preloaded and created, and collisions with the ground are handled, which can involve effects (explosives crate). I used a similar class format (the same class functions) for each of these files to the "Phaser.Scene" class. These classes have methods called "preload", "create" and "update", without actually being scenes. This made logical sense to me as each of these methods can be called in the same function of the scene. The "maps", "truck" and "effects" modules work the same way.

The "maps" module sets up Phaser to load the required assets for a map and handles the parallax effect. It knows where to load these assets as a JSON file with asset paths for each map is passed to it. The JSON file is stored in each map's folder and gives a list of background images, which contain how fast they should move relative to the camera (parallax), and foreground images, as well as how long each level should take in seconds (players receive a bonus at the end).

The "truck" module loads all the truck assets and creates a physics object of the truck. The wheels are connected to the chassis with spring constraints that are part of MatterJS. The code for applying angular rotation to the wheels is kept in the main script file, which ties everything together.

The "effects" file simply loads and spawns affects that can be used with other game world objects. Right now there is only the explode effect but others could be added in future. The explode effect takes as a parameter some objects that will be affected by it. These are looped through and the explosion force is calculated by scaling the vector between the exploding object and the object affected by it, by an exploding force constant divided by the distance between them in pixels which are divided by 100. This looks like: f = direction vector * (explosion force / (distance / 100)).

For the user interface I used HTML and CSS which are overlaid on the game canvas. These are controlled by the main script via the "ui" module, which contains code handling crate selection, the finish sequence and highscores sequence. The ui module uses callbacks to the main script to handle events like starting the next map or restarting the game.

There is a module called "parseVerticesFix" which I had to write in order to get the map loading functionality working. I used a piece of software called Physics Editor to create the vertices for each sprite, but they would not load correctly. I fixed this by rewriting the import code and updating Phaser's reference to the function that handles it (parseVertices). I used the code **Bodies.fromVertices(pos.x, pos.y, verts, options)**, as this worked better than the original import method.

The "script" file ties everything together. It loads all the map json files and creates scenes by passing them to an instance of a map loader. There is also a global game state object created which contains

all the data of the current state of the game (current map, time left, crates loaded, etc). This is passed to the various modules such as ui which modify certain properties in it such as “crates”.

The last thing to note about the structure is that everything that will be loaded by the browser goes into the “public/” folder. This folder is exposed by the “server” script, which creates an express JS HTTP server.

# Research

I did not do a whole lot of research for this game. I had some experience with MatterJS before so it was a logical choice to use it integrated with Phaser. Most of the research I did was on how to use Phaser 3, as its documentation is not particularly good. I used a lot of stackoverflow.com answers to work out what to do.

I did have to research how to make an explosion effect from a physics point of view. I derived a simplified formula using the laws of mechanics and worked out that the explosion force would be inversely proportional to the distance from the exploding object. I had to make it per 100 pixels as, if the force is divided by the distance in pixels, it ends up being very small; dividing the distance by 100 solved this problem. The actual animation of the explosion is a single image composed of 128px by 128px frames, which is loaded into Phaser as a spritesheet. This is had to research how to do as I did not know that animations could be composed in a single image and then split up into smaller frames.

Another effect I wanted to implement was the parallax effect of the background. This required research for how to achieve this using Phaser. I discovered that it is done through tile sprites and setting the “tilePositionX” on each update to the camera’s x scroll position, multiplying it by a parallax constant to adjust the speed at which it moves.

I decided to use HTML/CSS for the user interfaces since this is a web-based game and these technologies are very mature. Phaser 3 does have its own UI functionality, but I believe that HTML/CSS offers more flexibility, and I knew both technologies better, increasing development speed.

# Critical review

## Design and implementation

There are three reasons why the design and implementation of the application are good. The first is the way that modules for specific functionality are created is easily extensible. For example, if I wanted to add more effects to the game, I could put them in the effects module and use them in other parts of the code, just like the crates module does.

Second, the code is modularised and self-contained using the CommonJS format used in Node applications: there are no references to global objects inside a module, everything is passed as a parameter to its functions. This means that the coder can be sure there won’t be unexpected bugs due to global variables having unexpected values stored in them.

Finally, the organisation of the code and assets is logical and easy to update. For instance, when I want to update a map, I can update it in photoshop and export it as a png, and then reimport it into Physics Editor and use the magic wand tool to automatically create the physics shape around it. In this way it is also very easy to change the friction/density/restitution/etc constants, as you can change them in Physics Editor.

### Improvements

There are three areas where the game could be improved. Firstly, I would add more content to the game. There are only 3 maps currently so if I had more time I would like to create more maps, or potentially create a map-maker for players to create their own. I would also like to balance the existing maps more so that they have just the right level of difficulty for the order in which they are played. I would also add more crate types and different trucks, as well as a lot more types of obstacles.

The next area I would improve is user customization: ideally there needs to be an upgrade system in the game to make it more interesting, and have a use for the money earned. Potentially upgrades could include better suspensions, more positions for crates to be place in, better engine performance (faster truck), bigger wheels, and others.

Finally I would like to fix some of the existing problems in the game. There is a bug that occurs when the crate collides with the ground, but does not register as a ground collision meaning that when the truck gets to the end of the map, that crates gets added to the final score. The finish line on map one also only has one post as I could not work out a way to line up the back post as part of the background.

## Assets

### Truck

The truck assets were taken from a paid site called Game Developer Studio (https://www.gamedeveloperstudio.com/graphics/viewgraphic.php?item=1h5i465d8p3q4v6n41). They cost $3.25 and included a set of 3 trucks, each having different coloured versions, and included wheels for each truck. I wanted an old, unreliable-looking truck so these suited those needs perfectly.

### Crates

For the crates I wanted an old fashioned looking crate that I could modify. The base crate asset was taken from https://opengameart.org/content/2d-wooden-box. I found images of different items to put on them. The diamond I found at https://opengameart.org/content/diamond. The cabbage was taken from https://webstockreview.net/explore/cabbage-clipart-cartoon/.

### Maps

The boulder and log assets used in each map I took from https://www.freepik.com/free-vector/stones-cartoon_997676.htm and https://www.freepik.com/free-vector/wood-slices-different-types-trees_922502.htm#page=1&query=wood%20slices%20different%20trees&position=0 respectively.

The rest of the maps I used these background assets to create the foreground, doing a lot of this work in Adobe Photoshop, and loading the above log and boulder assets into some of them for the obstacles.

#### Map 01

The background assets for this map were taken from https://openpixelproject.itch.io/opp2017jungle.

The finish post was taken from http://clipart-library.com/finish-line-cliparts.html.

### Map 02

The background assets for this map were taken from https://raventale.itch.io/parallax-background.

### Map 03

The moon was taken from https://www.shutterstock.com/image-vector/vector-illustration-full-moon-isolated-on-720891703.

The background with the house was taken from https://www.hiclipart.com/free-transparent-background-png-clipart-idnxb.

Other background assets were taken from https://vnitti.itch.io/glacial-mountains-parallax-background and https://vnitti.itch.io/glacial-mountains-parallax-background.

## References

Explosion tutorial - https://www.physicsclassroom.com/class/momentum/Lesson-2/Momentum-Conservation-in-Explosions

Parallax effect in Phaser 3 - https://www.youtube.com/watch?v=pknZUn82x2U