The Peerful Project

**What’s the idea?**

* Local multiplayer games (any number of players, perhaps even solo)
* Start the game on a computer (or some big screen) …
* … connect and play completely via your smartphone!
* The computer is the host (and the ultimate boss when it comes to game decisions and game logic), smartphones literally function like if you had plugged a joypad into the computer.

**What’s the technology?**

* The game uses standard website code (HTML, CSS, JavaScript) for the mobile interface and most of the computer setup.
* It uses **Phaser.io** (version 3) for the actual game and drawing stuff.
* It uses Peer-to-Peer connections (using **simple-peer** from feross) for *all* signals. Especially if you’re in the same room (on the same Wi-Fi network), this is so fast that you won’t notice any delay.
* It uses a default (very simple) **Websocket Server** (made with **Node.js**) for the *signaling*. In other words: to make sure people can easily connect their phone with the computer and do a handshake. (It’s also used to serve the game files themselves, but that’s a relatively simple and trivial task.)

**Why is this fun?**

* It easily supports *any* player count. And, (local) multiplayer is by far the best way to play games.
* You do not need to buy any controllers, or play together on a cramped keyboard, or set up *anything*. Go to the website, click play, boom you’re playing.
* It supports any platform that has been updated to something close to modern standards. In most cases, this means all computers will do, and smartphones that are 5-10 years old should barely make the cut.
* It allows for very quick development and iteration, allowing me to create tiny, innovative, fun prototypes in little time.

**List of games within the Peerful Project:**

* Pizza Peers
* Peerball
* Peers in the Post Office
* Les Miserapes

**Random images in my head:**

* A pirate who, instead of a wooden leg, has a (broken) vacuum cleaner “stang in het Engels” as his leg. (Which he can use as a sort of jetpack, or conversely to attract stuff towards it.)

Phaser stuff

**Matter.js physics:**

* <https://www.html5gamedevs.com/topic/43615-phaser-3-matter-tutorials/>
* <https://itnext.io/modular-game-worlds-in-phaser-3-tilemaps-4-meet-matter-js-abf4dfa65ca1>

Pizza Peers (or “Pizza Place”)

**What’s the idea?**

* You are running a pizza place.
* Run across town to gather ingredients and orders, combine them into delicious pizzas, then deliver them at their doorstep before time runs out.
* But you cannot do it alone … as every player is allergic to some of the ingredients.

You **lose the game** if your reputation reaches 0 ( = five failed orders) or your money dips below 0.

You **win the game** if you manage to survive for 5 minutes.

**Rules of the game:**

* Every player has a personal backpack. It can store 3 items.
* Every player has certain *allergies*.
  + In a solo game, you do not have allergies (otherwise it’d be impossible).
  + With 2-3 players, this means you cannot pick up certain raw ingredients.
  + With 4+ players, this *also* means you cannot pick up any combination that includes your allergies!
* A table is needed for storing, combining and baking (“preparing”) anything. A table can only hold one thing at a time.
* You can only combine ingredients on a table if you have the basis: dough.
* To find out what people want, you must first run to their home and ask them.
  + If you take too long, you get a money and reputation penalty.
* Once you know an order, you must deliver it to that same address in time.
  + If you take too long, you get a money and reputation penalty.

**“Unseen” rules of the game:**

* The number of outstanding orders (which means the buildings *ordering* and *waiting* for their pizza) is equal to the number of players, **plus** one at random. (Sometimes, it can be one more.)

**City generation:**

To Do

**Better workspaces + tables:** Perhaps, instead, use a “rectangle blocking” and “path first” algorithm.

* First, create a *random path* ( = random walk) from A to B. The points A and B are the connections to the outside road. The rest of the path are marked as *unavailable* (so nothing is placed there).
* Now randomly add X rectangles of random size (2x2, 3x4, 1x2, etc.) to this path.
* Ensure we have at least X available spaces and at most Y spaces, then stop.
* Now simply do what we did before: determine the walls, place tables *anywhere*, except on the path.

**Table placement:** write code to randomly place tables in a building.

* Check if they aren’t blocking a path.
* Favor tables near the walls, but center tables are allowed as well. (In fact, those are probably easier to use.)

**Ovens:**

* Add modifier/parameter to tables that turns them into an oven.
  + Create different sprite for oven table? Or just put another sprite *on top* of the table.
* You cannot *combine* ingredients in an oven, you must put in the pizza completely in one go.
* When something is placed on an oven table, the oven switches the heat to the *new ingredient heat*.
* While something is inside the oven, it will slowly ramp up the heat. When you take it out, the oven is “in essence” cooled down to 0 immediately.
* When you take something out of the oven, its heat is maintained by the player. (A separate array that is updated in sync with the actual ingredients.) But, it does decay slightly over time.
  + A burning/steaming animation behind the ingredient indicates its heat. As long as it burns, it’s warm enough.
  + If it became too cold, it stops being animated.
  + The colder it is, the more transparent this sprite becomes?
* When the heat has surpassed a maximum value, the oven turns on fire?
  + As much as I’d like to build in a “fire extinguisher”, it’s a bit much work.
  + Instead, simply turn the pizza sprite above it (that shows what’s inside) into a *blackened* version.
  + However, we must *warn* players when something is starting to overcook. Perhaps we should have some sort of “heat meter” next to sprites inside the oven, which clearly marks temperatures.

**Workspace creation:** do not allow roads to be placed inside the building.

* If a road is accidentally placed inside a building, turn it into … a secret passageway!
* Simply change the sprite and add another passageway somewhere else. If a player touches it, they are teleported.
* (This seems the most elegant solution. Trying to check if a road is inside an irregular shape seems like hell.)

**Code cleanup:** boy, this might be a pain in the ass.

* Restructure code to split into multiple files:
  + One for websocket server only
  + One for peer to peer only
    - The biggest function (‘on data’) could be split into multiple parts, or at least moved to another file
  + One for mobile interface (touch events and the like)
  + One for phaser game
    - Subdivide into scenes and perhaps modules based on functionality?
* Gracefully deal with disconnects and game finishes:
  + When the *computer host* shuts down, inform WebSocket, delete game from server.
  + If peer to peer disconnects, show it on the screen (show the player disconnected). Allow players to reconnect (with the same username).
  + Do *not* allow players to join once the game has already started. Would introduce problems, because game parameters depend on player size.

**Game over screen:**

* Show the result (win/loss)
* Show *why* this happened
* Perhaps some statistics, or simply tell the VIP that they can restart if wanted.

**Optional**: make players allergic to *other things* than ingredients. For example: allergic to heat, allergic to taking orders, etc.

(Use negative numbers for this?)

**Optional:** add actual doors and try to make workspaces look better – less random.

**Optional:** nice particle effects when we receive coins

**Optional:** music (mostly because I’m interested how it impacts loading speed and playback delay)

Peerball

**NO:** **Change it to be a simple 3D game!**

* Use Phaser, but use an additional layer of **Enable3D**
* This uses Three.js and ammo.js for 3D rendering and physics.
* Then I could just create some basic shapes + hinge joints (or other constraints), play with forces, and voila we have our game!

**GREAT IDEA:** The players do not … need to look like humans of course! I can just make them from random shapes (circles, triangles, rectangles, polygons, etc) in a random combination.

Although, for first prototyping/testing, it’s probably best to stick with simple figures.

A variation on **Drunk Soccer** (my Bester game) and **Soccer Physics** (from Kongregate).

**What’s the idea?**

* You play a rudimentary version of soccer: two teams, move around a rectangular pitch, score by getting the ball into the goal
* We look at the pitch from a ¾ top down view. The game is 2D.
* All players are ragdolls. They consist of a few body parts (torso, legs, arms, head), connected with joints that allow them to move freely but also be controlled.
  + They have some form of correcting force, keeping them mostly upward and controllable.
  + They use the **Matter.js** engine within Phaser.
* When you start the game, you receive a set of *four random buttons*. These buttons do completely different things:
  + **Movement:** a button might move you in a certain direction
  + **Kick:** a button might move your legs (or arms?) to kick stuff
  + **Jump:** a button might make you jump, or dash forward, or salto in the air
  + **Modifier:** this button *modifies* one of your other buttons. For example: pressing B1 makes you jump upwards. But if you hold B2 at the same time, you instead *slam* downwards!

Now you simply play the game, testing your controls and learning how to deal with them as you go, and after 3 (or 5) minutes the team with the most points wins!

**BIG QUESTION 2:** Are the joints I need supported?

* They are called **constraints**, not joints.
* However, Phaser 3 *does* use the **joint** name.
* Video on constraints (from Nature of Code): <https://www.youtube.com/watch?v=szztTszPp-8>

**BIG QUESTION:** If the world is ¾ top-down 2D … how do we work with bodies?

Players only need to collide with stuff if it’s actually within their range. (A ball might be in front of their face, but it might actually be high in the air on the other side of the field.)

Can we disable/ignore collisions based on certain constraints?

* Yes, probably: <https://github.com/liabru/matter-js/issues/307>
* (Also: <https://itnext.io/modular-game-worlds-in-phaser-3-tilemaps-5-matter-physics-platformer-d14d1f614557>)
* (Or: <https://www.html5gamedevs.com/topic/38622-phaser-3-matter-collision-events-for-specific-objects/> )
* **On collision start, check if pairs are close (in pseudo-3D), and disable it otherwise.**
* (Otherwise, use collision groups/filters, that change completely based on the current situation in the world. Stuff close together (in “pseudo-3D”) can hit each other, stuff far apart can not.)
* (Otherwise, temporarily turn stuff into a sensor? Nah, doesn’t seem like the right thing to do.)

Peers in the Post Office

Instead of focusing on a complete village/town, go in the other direction: you only focus on the actual *post office*.

Packages are arriving left and right on conveyor belts.

They need to go through the right machines, get the right stamps, be sorted, and be put in the right boxes before their timer runs out (or they land in the bin automatically?)

(Perhaps even allow shooting/throwing packages and letters across the work floor.)

This gives me an **IDEA:** a multiplayer game where people must play both sides of a puzzle.

For example, take delivering stuff. One part might be the people who *prepare* it (they get their own screen/tools/interface), the other part might be the people who *deliver it* (who also get their own stuff).

Les Miserapes

I don’t know.

I like having slingshots in a game, and I had this weird image of players having to stack random furniture as high as possible (like the barricade in *Les Miserables*).

Combine this somehow?