# TO Do (Sheepe)

## Coding improvements

* Register the content of each room (which players are inside) at all times. (Can also be used by the teleporter directly.)
* Similarly, keep track of our current room *on the player itself.* Access that at all times. If our current room is “undefined”, simply don’t change it (we’re probably *just* outside, on a slope or something.)
* Make the *slope algorithm* much more efficient => convert vectors to integers at start, use simple “integer in list” check
* Can we split up the huge “map” and “room” modules now?

## Tutorial

* Allow placing “tutorial images” anytime during a level, and then activating the rule we introduced.
* Create a campaign overview screen + make “next level” button actually go to next level.

## Gameplay

**Step 0:** I think players should be a *little* smaller? So gaps are easier to fit into. And I could reduce their speed without making it feel sluggish.

**Step 1:** I should use the fact that players can be **any shape** and that this can change more.

* Idea: Your **size** (or “mass”) plays a huge role. (You’re faster when you’re bigger?)
* Idea: Your **number of parts** plays a huge role. (Gates you can only pass through if you have *fewer than* or *more than* the indicated number of parts?)
* Idea: Maybe there are specific “gates” with weirdly shaped gaps. (Like that TV programme where you had to stand in a certain pose while a shape came towards you.) You will have to find one that *you* fit through.
* Idea: There are sections/powerups that *reset* you to a specific (predefined) shape. **Or just your original shape.**

## Bugs

**BUG:** Sometimes it counts collecting a coin as collecting *two coins*. (Sometimes even three???)

**BUG:** Now it doesn’t remove the last *edge* to previous connections anymore in tight spaces!

* What did I change? “Delete edges inside”, “Remove references to room”

In general, the system of **“try to place rooms in tight spaces”** isn’t great. It leads to errors + ugly edges.

**BUG:** What if the number of total players bodies exceeds 20? Then the “fog of war” shader can’t handle that. Two solutions:

* Either never allow this. (Don’t slice if it would get the total body count too high. Or remove to make space.)
* Or *sort* bodies based on their room, and just *ignore* lights on the ones furthest behind.
* Third solution => don’t give PLAYERS lights, add them to ROOMs? (Or give players a light when there is *none nearby.* Only add lights to large rooms.)

## Movement Improvements

**Step 1:** Add something to “help” flatter shapes roll. Or should I keep it as-is and just expect players to be strategic and *stop rolling* if their shape is bad?

* There should be clear situations in which a *round shape* is good, and other clear situations in which a *flat shape* is good.
* **GREAT IDEA:** There are buttons on which you must *stand for a few seconds* to activate them. If you roll well … it’s hard to stay on it. If you’re flat, it’s very easy.
* **IDEA/THOUGHT:** Flat shapes are better when glueing parts back together, aren’t they? It will fit more nicely.

## Map Improvements

Allow for a more controlled variation. “Very big rooms” followed by a series of “very small rooms”.

(Which would allow the very big rooms to actually feel like a *room*.)

Here’s an idea:

* When determining size, look *back* over the cur\_path
* If the average over the last X rooms is very *low*, place a HUGE ROOM.
* If it’s very *high*, place smaller rooms.
* Mark those huge rooms for “special use”

**Other improvements:**

* Fix the bugs, obviously.
* The further we get into generation, the *smaller* displacement becomes
  + Same as the others => calculate ratio from num\_tries, multiply displacement by that, then round it
* Don’t go down more than a few times in a row => make this a *setting* I can set globally.
  + Count how often we went down in the past X tiles (walk back cur\_path)
  + Check against global parameters
* Encourage it to go away from edges.
  + If “dist\_to\_bounds” close to 0 …
  + Make “get dir to edge” function, which returns the *index*.
  + Ignore any rooms with that *dir index*.
* Terrain painting =>
  + **Don’t** allow two of the same “category” after each other. (Because their color is the same, you might miss the fact that it changes.)
  + **Do** encourage using the same terrain multiple times in a row.
* Terrain Painting
  + Now it also paints 1 outside its own terrain. Works nicely … mostly. Smooth it out
* Place “reverse\_gravity” terrain on things going up. Never place it in rooms going down.
* Place “no\_gravity” only in rooms with lots of *slopes*.

Once this is done, I think we can do another “cleanup pass” on the algorithm, as it’s become too messy.

## Slicing improvements

**Step 1:** Be way more precise with intersect\_shape => create a *rectangle*, the length of the line segment, narrow width, rotated + positioned around angle + avg.

**Step 2:** The clinging force/jump force should be *proportional* to player size. (Otherwise small pieces get stuck, and large pieces cannot cling.)

# Done

## Basic Bodies

**Step 1:** Generate a random polygon

* <https://stackoverflow.com/questions/8997099/algorithm-to-generate-random-2d-polygon> => basically, create a circle, but allow each point to vary in radius/angle
* <https://stackoverflow.com/questions/59287928/algorithm-to-create-a-polygon-from-points> => draw a point cloud first, order by angle, then draw through it

**Step 2:** Calculate its centroid. Place a smiley face there. Then center the polygon around it.

**Step 3:** Turn it into a physics body + draw it each frame.

**Step 4:** When given input, roll in a certain direction. (Check if this actually works for movement.)

## Body slicing

**Step 1:** Write the slicing algorithm I scribbled on paper.

* <https://stackoverflow.com/questions/563198/how-do-you-detect-where-two-line-segments-intersect> => detect intersection point of two lines
* The rest of the algorithm is just:
  + Loop through shape.
  + Detect first intersection point. Add it to the shape. (Between the start/end vertices of the edge it intersects.)
  + Continue until second intersection point. Add it to the shape.
  + Now *extract* the part between the two points: shape 2. *Remove* the part you extracted from the original shape: shape 1.
  + Now recreate the *bodies* + *draw/move scripts* for each.

**Step 2:** Allow testing by drawing with the mouse. (Or clicking twice. Or pressing a key and testing a predefined line.)

**Step 3:** If successful, allow applying dynamically.

# Discarded

The old idea with “placing precreated rooms”

## Rooms & Routes

**Issue 1:** How do we allow *rotating* rooms?

* Translate everything to anchor center
* Rotate the thing
* Translate everything back => DOESN’T WORK, because the “position” property is still local, so translating back would just *follow the new orientation*
* Now recalculate opening values

**Issue 2:** What if a single side has *multiple* openings?

* We should be able to match any of them
* But *not* necessarily close the others when filling gaps

**Issue 3:** Now we have ugly *double walls* between rooms.