# To Do (Sheepe)

## Teaching

Make the menu **interactive** => everyone is a ball rolling around, two rooms to choose from, once everyone inside one of them it starts that. (Some instructions above and below, plus buttons for *settings* or *quitting* or something.)

Make the **tutorial** course. (Which are partially player-focused, and partially global images??)

Find some way to *also* teach the teleporter (in time) before it appears. (Or just teach it in the background of the teleporter itself? The explanation is quite short …)

## Locks

* **Painter**
* **Shop (with modification?)**

## Solo Mode

**How does this work?**

* The map is slowly dissolving behind you
* If it reaches you, you lose
* If it swallows one of your parts, you get a penalty.
* Just try to reach the finish.

## Gameplay Improvements

Making coins useful:

* If you have coins, the wolf does not *split* you, but rather takes a coin from you. (Both makes the wolf more interesting to play actively, and gives anyone a use for coins.)
* Ensure at least one *terrain* or *lock* is included that does something with coins.

Making body splitting useful:

* Any body of yours *that does not finish* results in a time penalty. (The game ends once every player has *one* body over the finish line.)
* Older bodies of yours can still activate stuff, so they can still be useful (or very annoying) if you keep track of this. (For this to work, we need one other rule: *when one of your bodies does something or gets something, this is copied to ALL your bodies*.)
* Add many ways to get back together again
* Ensure at least one *terrain/lock/item* helps you retrieve or use your old bodies.

## Coding improvements

* The system for saving gates/edges is a bit sketchy.
  + Now I *only* add a “my\_room” and “general\_parameter” on edges created via a lock.
  + (Additionally, gates are saved and assumed to belong to one room only, but that’s fine, as gates are always apart and one-way anyway.)
  + Can I generalize this code? Would require passing some more stuff through the function, maybe as *params* object?

## Performance things

* Only do “update\_bitmask” once, when *everything* is done.
* Remove *modules* from player bodies that don’t need them. (For example, *tutorial* module only works on first body of player.)
* We’re looping through full rectangles awfully many times. Instead:
  + Combine these loops to calculate multiple things at once.
  + Or save all positions *in a list* and just iterate that list. (Instead of positions, we might also save a reference to the cell itself. Is even faster access.)

## Coins

Bigger question: what exactly are coins good for? (It feels a bit tacked on at this point.)

* Coins protect you from the wolf. (They just take a coin, instead of biting you.)
* Some locks require them *or* make your life easier with coins.
* Some terrains use them.
* Some elements should also require payment? (Make sure it doesn’t become overly complicated!)

**TO DO** => Mark certain items as “coin” => show coin interface when *near* them

* Use the physics for this? Or just do a loop through the 3x3 area around us in the grid?

## Big Question

What’s the point in *slicing bodies*? If only the first one needs to finish?

* Make you smaller => which might or might not be good.
* Individual bodies have fewer coins => which means fewer possibilities.
* **The more bodies of yours that finish, the more *time bonus* you get?**
* Stray bodies might activate something you don’t want. Or be an easy target for a wolf?

## Bugs

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

**BUG:** Sometimes item (spikes) not removed after hitting it? (Probably has something to do with being unable to find the item in that location, there was something wrong with that, wasn’t there?) => **might be fixed**

**BUG (?):** When glueing objects back together, it sometimes crashes? Because the room.players\_inside array has a few null entries. (Which would be caused by a body being killed but NOT removing itself from that array first.) => **might be fixed**

## Map Improvements

**FILL ROOM Algorithm:** Add a variation where we’re allowed to place tiles *against the walls*, but *not in the center*. (By default, we only place away from walls, in the center.)

## Polishing

* Play with generation parameters => I feel big rooms should be *slightly* less filled (or have more varied filling), maps should *flow* a bit more (with slopes, rooms that are not *too* different in size/displacement)
  + “Preferred” displacement would be something that does NOT create a bump in the line. So either it stays flat at the ceiling, or it stays flat/falls down on the ground.
* Whenever I do something to a jump normal, show a tiny line for that. (Similar to debug, but prettier.)
* Add “bouncy” tween to the *shaper* node when jumping or hitting stuff.
* Give an indication when someone is holding *both buttons* => perhaps show a different color or overlay when *floating*. (Add wings at the side?)
* Give feedback
  + Especially when getting a time penalty or getting/paying coins.
  + But also enable option to give *textual feedback* when someone first enters a terrain, which gives a hint *how* this terrain works.
* Add different control scheme for controllers: joystick to roll left/right, any button to jump/float.
  + (Make this default? Or can players configure it themselves?)

## Optional & Questions

* BUG: Buttons can be placed outside the actual room => make “tiles\_inside” tighter to the inside on locks ( = only use the shrunk rect) **IMPLEMENTED** => does this work and is this a good idea?

# Done

## Annoyances

**ANNOYANCE:** When you jump with your head against the ceiling, your *rotating* movement actually pushes you in the wrong direction. Which is just … annoying? (Yes, you can learn it, and use it for stuff, but … not great.)

* Solution #0: Make ceilings frictionless => can’t do it, as they’re part of the tilemap, which has *one* physics material.
* Solution #1: Always cling to ceilings => possible (check if cling vector is opposite to gravity vector)
* Solution #2: Make jumping less powerful
* Solution #3: *Hold* both buttons to *float* or *steady yourself*. (So when you hold both, your Y-velocity becomes 0. But your X-velocity continues.)

## 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.