# Rolling in the sheepe

## General

Each player is a **random shape**. You can only **roll** through a world full of obstacles.

The map flows to the left, building itself as you go along. (There might be split paths, ways to go down, direction reversals. But keep it simple at first.)

**IDEA:** There are obstacles that *glue* you to them.

**IDEA:** There are obstacles that simply hurt you (so you slow down/reset). But also ones that *split* your shape, like a spike shooting through you.

**IDEA:** When this happens, you simply roll *both of them*. They can be recombined later. But you only win if *all of you* crosses the finish.

**IDEA:** Alaser might also just shoot through at certain moments, slicing *everyone* it hits.

**IDEA:** a way to “lock in sections”. So, players that are further ahead must *wait* for the rest to catch up. (But they might get *bonuses* for arriving there earlier. Or there are things to do while “waiting”. Maybe it’s a *minigame* they need to solve to open the door.)

Why? This ensures players don’t go too far apart, allowing the camera to keep all in frame.

Why? It’s the “Mario kart” way of keeping things competitive.

## Theme

During the game, you can *change* your shape. These upgrades make you look more and more like a *sheep*. (The round, bouncy form of sheep is the “ideal” shape for this game.)

You are all fleeing from a wolf. That’s why you’re running.

## Objective

Be the first to reach the finish line

## Control

Multiple possibilities:

* You roll automatically, press/hold a button to slow down.
* You roll automatically, press a button to reverse direction
* Use *two* buttons: one to roll forward, one to roll backward
* Use the *joystick* to roll/push in a specific direction

# To Do

## New pathing algorithm

**Step 0:** Start with a tilemap *completely filled* with blocks

**Step 1:** Upon “create new room”

* Decide on eraser size. (Any rectangle will do)
* Cut that rectangle out of the map, starting from last known location.
  + Can use a random offset, so we start a bit *higher* or *lower*, not exactly in center.
* Save this rectangle as the “room”
* Save furthest point as new “last known location”

**Step 2:** Upon “delete room”, simply fill the tilemap again (all tiles from original rectangle)

**Step 3**: (*Nice slopes*)

* When doing a new erase, check for *height difference* against last blocks. If so, place a *slope* => *needs more detailing; how do you do this??*

**Step 4:** (*No overlaps*)

* Simply do an AABB rectangle check against all the old rooms
* If no position is possible, change the size of the rectangle.
* If still not possible, place teleport lock here.

## Map creation

**Step 1:** Start in the *center* of the grid, otherwise we run into “out of bounds” issues rather quickly.

**Step 2:** After a certain number of tiles, add a “finish”.

**Step 3:** After a certain distance, add a “lock”.

* Basic Lock: Spawn multiple coins or buttons. After you’ve collected them *all*, the lock opens.

## Player creation

**Step 1:** Dynamically add players.

* Give them their corresponding inputs.
* Place them at the start of the level
* Give their own **color** (used for all parts of them) and make sure sliced parts **copy the player num of their parent**.

**Step 2:** Extra controls:

* Allow “clinging” to walls when rolling.
* Also add air resistance? (So rolling against your momentum will slow you down?)
* And do something special when *both* buttons are pressed?

## 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.

# 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.