# 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

## Basic gameplay

**Step 1:** Make camera follow the players as they move.

**Step 2:** Create a basic “straight blank chunk”. Keep adding it to the end as players come nearer.

**Step 3:** After a certain distance, add a finish. The first to reach it wins.

## Slicing improvements

**Step 0:** Be way more precise with intersect\_shape => create a *rectangle*, the length of the line segment, narrow width, rotated + positioned around angle + avg.

**Step 1:** Don’t allow *really small* slices.

* I add some buffer when doing line intersection checks, which disallows slicing when entry/exit points are really close to vertices.
* **IDEA:** Can I approximate the *area* of the resulting slice? And if it’s too small, don’t do it? (Don’t go through to the “create\_body\_from\_shape” function?)

**Step 2:** Allow concave shapes

* **Current method:** simply re-run shapes until there are no intersections anymore. **This is wrong.** We don’t know the *order* of the intersections. So we might accidentally get two exit points, or two entry points, or whatever.
* **Better method:** 
  + The intersect\_shape automatically returns *triangles* => we don’t want that, we want the full list of shapes inside that body
  + So create a list of parents
  + For each parent, create a list of its triangle shapes
    - Slice each triangle individually
    - Once we have the new list, try to merge triangles. Continue until we can loop the list and *no* triangles get merged
    - Create new bodies from all the remaining arrays

# 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.