# 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

## Terrain types

* **Finish:** touch it to win
* **CoinLock** =>backdrop for coin lock
* **Reverse Gravity**
  + Use this more often on parts *going upwards*
* **No Gravity**
* **No Friction**
* **Bouncy**
* **Speed Boost**
* **(Speed slowdown? Speed reset? Slowmo?)**
* **Glue** => touching an old part of yours will glue it back to you
* **Spiderman** => you cling (strongly) to all walls around you
  + Or make this the default and add sections where you *can’t* cling.

**IDEA:** Some terrain, or mode, or whatever that allows you to **destroy blocks** in the tilemap. (Maybe everyone becomes a bomb in that section.)

## Room types

**Make these modules.** Instead of putting them *inside the room script*, just spawn an extra node with its own script. Once fulfilled, it just sends a signal (to its parent room) that it should release its lock).

* **CoinLock:** Regularly spawns new coins. (Within min/max bounds.) When touched by player, its collected. Keeps counter in background. When counter above 10, the lock opens.
* **Teleporter.** Keeps a timer. When it runs out, *or all players have at least one body here*, it teleports you. (Old map is destroyed, new one started somewhere else.)

## Tutorial

Only the specific buttons are taught per player, with a prompt.

Everything else is shown as images in the background of the map. (Like a terrain paint.)

Something like this:

* Show prompt above players for “ROLL RIGHT”
* A bit later show “ROLL LEFT”
* Then, *in the background of the map itself*, show “press/release both at the same time to JUMP”
* When the first (coin) lock appears, show *in the background* “collect coins to unlock the next part”
* When the first teleporter appears, show *in the background* “once all players arrive, you *teleport* to a new part”

For this to work, we need to force a large room at those spots (so we have space for the image).

## Collision layers

* 1 = all
* 2 = terrain
* 3 = players
* 4 = ??

## Rules for coding

Everything is done via **modules**. No script should do everything at once. Every functionality is a unique script, attached to a parent.

This also means that **when an object is passed around, it’s always the parent.** (Which usually does not have a script itself. But any modules can be accessed with a simple get\_node(<modulename>) call.)

# To Do

## Gameplay

**Step 0:** Find a **clear, intuitive relation between “size / number of parts” and success**.

* Do you need *all your parts* to finish?
* Or do you simply need *a certain total “mass”*

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

* Maybe your *size* plays a huge role.
* 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.

## Gameplay (Essentials)

**Step 0:** When a *body* enters the finish …

* Register it =>
* Then check if *all* bodies from a certain player have finished
* If so, make it definitive => save their rank, show it

**Step 1:** Once *everyone* has finished, show an overview of the ranks + times on screen.

* Don’t go to a separate screen; just show in a neat widget to the side.
* Allow restarting. (Or going back.)

## Pathing improvements

**Step 1: Prefer going in the same direction as you are**.

* For example, if we’re building rooms to the right, prefer adding another one to the right.
* Make going up/down much less likely
* And completely reversing our direction very rare. (=> Currently it’s *impossible*, and that’s fine?)

**Step 2:** Annoying situations!

**Situation #0:** Don’t allow the **very last body** of a player to be removed. If this is about to happen, simply don’t allow their body to be *sliced* in the first place?

**Situation #1:** Two triangle slopes right after each other => often there’s no space to squeeze between there.

* If this happens … place a “cutter” before it? (So players have a *chance* to reduce their size?)

**Situation #2:** Growing a teleporter, and just plopping it down (+ erasing all around it) … looks ugly. Additionally, sometimes it happens *way too often* and *way too quickly.* How to ensure paths have a bigger chance of survival?

* Create *edges* on the outline. But this time, keep them open when there is *any* free tile on the other side. (Not just in the last room, but all rooms.)
* Be *smarter* about placing rooms when space is tight.
  + Add **backtracking** => instead of trying to place something at the end, place it at a few rooms before that point.
  + The “backtrack limit” should be roughly the same as the “front room buffer”, to ensure the leading player has no disadvantage from this
* **Rewrite the room creation code to be cleaner and more modular.**

## Player creation

**Step 1:** Give players a **color** (used for all parts of them) and make sure sliced parts **copy the player num (and other properties) of their parent**.

**Step 2:** Start players with **predefined shapes**. (Probably looks cleaner and is more functional.)

* Circle
* Square
* Triangle
* “L”-shape
* Pentagon / Hexagon
* Parallelogram
* Star shape
* “klavertjevier”

## Movement Improvements

**Step 1:** Limit influence of air resistance. (Perhaps add a force that slows you down *when in the air*, towards a maximum velocity. So you can still go *really fast* if you gain speed, but moving in the air will always be limited.)

**Step 2:** Might need some extra *safeguards* to ensure you can always move somewhere successfully.

* Stronger jumping?
* Getting bitten by the wolf behind you, somehow *resets* you or gets you *unstuck*.

## 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.