# 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.)

## 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, with any of your parts.

## Control

* Button 1: roll right
* Button 2: roll left
* Both: jump
* (Nothing = well, nothing)

## Minor Rules

Catchup mechanics:

* Any player that’s too far behind (either in terms of rooms *or* concrete distance), is teleport to the next best player. But gets a time penalty.
* Any player that does nothing for 10 seconds, is teleported forward a single room. And gets a time penalty.
* (Any player not on a teleport yet, still goes with the group, but gets a time penalty.)

## Terrain types

### Essentials

* **Finish:** touch it to win
* **CoinLock** =>backdrop for coin lock
* **Teleporter** => backdrop for teleporter

### gravity

* **Reverse Gravity**
  + Use this more often on parts *going upwards*
* **No Gravity**
* **(Collision-normal-based-jumps?)**

### physics material

* **No Friction => Ice**
* **Bouncy**
* **Spiderman** => you cling (strongly) to all walls around you
  + Or make this the default and add sections where you *can’t* cling.

### speed

* **Speed Boost**
* **Speed Slowdown**
* **(Speed reset? Slowmo?)**

### Slicing/shapes

* **Spikes** => hitting anyone else will *slice* them
* **Glue** => touching an old part of yours will glue it back to you
* **Grower =>** instead of rounding (when rolling) and deforming (in the air), you grow and shrink (respectively)
* **BodyLimit =>** can only contain as many bodies as there are players, any more are repelled

### Coins

* **Invincibility:** if you have more than X coins, you’re invincible => **Test**
* **Rounder:** if you have more than X coins, you immediately become round => **Test**
* **Halver:** your number of coins is halved, every time you enter. => **Test**
* **Slower:** the fewer coins you have, the slower you move => **Test**
* **Bomb:** if you hit a cell, it’s destroyed, and you get a coin => **Test** (to make sure it doesn’t trigger on outside cells, *and* to see if it needs a “minimum speed” limit)

### Misc

* **Reversed Controls**
* **Ghost =>** players can pass *through* each other (and obstacles within rooms
* **No Wolf =>** the wolf is disabled here

**TOO CONVOLUTED:** A **terrain** where you invest X coins. When you get out, you receive your investment + a bonus. (Based on how long you were in there? Based on how many other players are there?

**NOT SURE WHY:** Some terrain, or mode, or whatever that allows you to **destroy blocks** in the tilemap. (Maybe everyone becomes a bomb in that section.) => Wouldn’t work for *all blocks*. Would work fine for *inner blocks*. But if they just disappear when touched, what’s the point?

Unless you get *coins* for this! For each block destroyed, you get a coin.

## Lock types

These should always be **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).

* **Coin:** Regularly spawns new coins. (Within min/max bounds.) When touched by player, its collected. Keeps counter in background. When counter above X, the lock opens.
  + **CoinShop.** If you have enough coins, you can move through the lock immediately, *but must pay them*. (Number lowers with each visit?)
* **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.)
* **Mass.** Requires X *bodies* to be inside the lock. (Spawns coins for leading player? Regularly adds spikes to ensure slicing?)
  + **Holes.** Holes are punched in the background. When you move over them, you “fill” them (by painting over them.) => *yes, this would require another mask like the general tilemap one*
  + **Painting.** Same as holes, but now you just need to paint 99% of the whole lock.
* **Buttons.** Buttons appear. Press X of them to unlock.
  + **ButtonTimed:** must standon these *for a few seconds* to activate them.
  + **ButtonOrder.** Buttons appear (all at once). Press them *in order*.
  + **ButtonSimultaneous.** Buttons appear (at most #players – 2). These must be pressed simultaneously.
* **Sacrifice.** Someone must slice themselves to open the door. (This room must have a laser/spikes.)
  + **Coin.** Someone must pay loads of coins to open the door.
* **Shop.** Spawns X special items (to buy). Each player can buy only one. Once they’re all bought, the lock opens.
  + (These are good things, like: *time bonus*, *round shape*, *invincibility*, etc.)
* **Gates.** Not sure if this should be *lock* type??
  + **Fast Gate:** Opens/closes at random intervals, very quickly. You can only pass through when it’s open, obviously. (These intervals *lower* over time, increasing the probability of getting through.)

The idea behind locks:

* Players that are further ahead must *wait* for the rest to catch up, bringing people together.
* The trailing players *don’t* have to do the minigame or challenge anymore; they can just power through.
* But it’s a fun minigame *and* it yields bonuses to be there first and play it.

Keep these ideas in mind on all locks!

## Powerups/Obstacles/Elements

These are all the same. Within any room, it can place *something*. This can be an obstacle to navigate around, something to grab, etc.

**IDEA:** An obstacle that breaks apart/is destroyed if you hit it (with enough force). Would have to be a “tile inside”, not the outer tiles!

**IDEA:** Sometimes it would be nice to place extra *edges* between rooms. These top players from “flying through” and can be a nice gate/obstacle/variation.

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

**IN GENERAL,** many of the terrain types could also be reduced to a single item. (Spikes = spike terrain, Ghost = ghost powerup, Ice = ice walls, etc.)

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

**IDEA:** A delayed laser => shoots a bullet/projectile that *moves in a straight line* (until it’s at the other side).

This might even be a good thing (like a *coin*).

**IDEA:** A powerup that makes you a ghost *for a certain period of time*.

**IDEA:** A powerup that makes you invincible.

**IDEA:** A powerup that instantly teleports you to the next room. (Or to the next best player. Or better yet, the leading player.)

**IDEA:** An “immediate circle” or “immediate deform” powerup.

**IDEA:** A block (inside rooms) you can break by *hitting it with speed?* (And breaking might get you something?)

**IDEA:** Gates you can only pass through if you have *fewer than* or *more than* the indicated number of parts?

**IDEA:** A gate that varies in size (bigger, smaller, bigger, smaller).

**IDEA:** A powerup that *resets* you to a specific (predefined) shape, or just your original shape.

**IDEA (more physics fun, if I want):** rolling against something, *also* creates a force on that other object. So, I can create doors/panels that you can slide open/closed by rolling against them.

## 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).

Anything else is taught in a campaign-based system. Each level has several images in the background explaining stuff. (**Could even be halfway!** Just “unlock” a terrain only after seeing the image.)

## Predefined shape list

* Circle
* Square
* Triangle
* Pentagon
* Hexagon
* Parallelogram
* “L”-shape
* StarPenta
* StarHexa
* Trapezium
* Crown
* Cross
* Heart
* Drop
* Arrow
* Diamond
* Crescent ( = half moon/crescent moon shape)
* Trefoil ( = “klavertjedrie”)
* Quatrefoil ( = “klavertjevier”)

## 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.)

## Painting the tilemap (Documentation)

To make this possible, we need two things:

1. A texture containing **the paint**
2. A texture containing **the tilemap** (the shapes of filled tiles)

**First one:**

* Create an Image
* Whenever a player hits something, paint a circle (of random size, in its own color) at the location of the hit, in this Image
* Every frame, convert the Image to an ImageTexture and hand it to a sprite
* Add a shader to the sprite.

**Second one:**

* A copy of the tilemap exists which is updated anytime the “real” tilemap is updated
* This is inside a *viewport*, which is the same size as the world/level/tilemap itself. (So it sees *all of it* at all times.
* The ViewportTexture (from this viewport) is sent to the shader on the sprite.
* The shader simply shows the paint but *masked* based on the tilemap shape. This way, it only shows up on actual tiles, not in empty space.

## Manipulating shapes (Documentation)

**Type 1 (simplest):** When creating a new body,

* Simply put a list of its points into a new ConvexPolygon2D shape. (shape.points = point\_list)
* Reminder: these are PoolVector2DArrays. They are more efficient. They must be cast to arrays if you want to use them like that (Array(list)). They are passed by *value*, not *reference*.

**Type 2 (medium):** When slicing a body,

* Make all shapes global.
* Slice them (using my own simple algorithm).
* Group the remaining shapes into connected areas.
* Reposition these shapes to be around their *centroid*. (Making them local coordinates, which we need, and already perfectly around center of mass.)
* Delete the old body.
* Create new bodies from each shape *group* (by repeating type 1 multiple times).

**Type 3 (medium):** When *growing*/*shrinking* a body,

* Make all its points global.
* Offset to be around the body’s *center*. (Needed for scaling, also makes it local, which is nice.)
* Now multiply by shrink/grow factor.
* Now *inversely rotate* the points. (Because the body is rotated at this point, simply adding the shapes back would put them back in the wrong rotation. So, simply offset the body rotation by going against it.)
* Delete the old *shapes*.
* Repeat type 1 until all shapes are back inside the new body.

**Type 4 (hard):** When making a body *more round* or *more malformed*

* Make all its points global
* Offset to be around the body’s center. (This would also be the center of the circle *if the body were a perfect circle*. So we need it for making the shape more round.)
* Approximate the original body’s radius.
* Move each point to be *more* like the perfect point (that would lie along a perfect circle) with the same angle. In essence, we simply shrink/grow the radius to even things out.
* This changes the center. Calculate average centroid and offset points.
* Then *inversely rotate* them (see type 3).
* Delete the old *shapes*.
* Repeat type 1 until all shapes are back inside the new body.

**Note:** so, when slicing bodies, we create *new ones*. As such, their rotation/scale/position is default, and we don’t need to be afraid of it. Just add the shapes as they are.

But when growing or rounding *existing bodies*, no new ones are created, so we must take the properties (mostly rotation) of that original body into account. Before adding back the shapes.

(Knowing this, it’s obvious what happens then. Although the final shape is correct, you see the whole body *rotating* every time they are updated.)