# Rolling in the Sheepe

## General

Each player is a **random shape**. You can only **roll** through a world full of obstacles. The map builds itself as you go along, going in any direction.

## 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)
* **ReverseRounding =>** air = become rounder, roll = become malformed
* **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
* **Rounder:** if you have more than X coins, you immediately become round
* **Halver:** your number of coins is halved, every time you enter
* **Slower:** the fewer coins you have, the slower you move
* **Bomb:** if you hit a cell, it’s destroyed, and you get a coin

### 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?

## 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.

**Gate (variation):** If you have enough coins, you can move through the lock immediately, *but must pay them*. (Number lowers with each visit?) => This edge stays up for everyone, as opposed to the “coin sacrifice”

### 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.

### Sacrifice

Someone must slice themselves to open the door. (This room must have a laser/spikes.)

**Coin (variation):** Someone must pay loads of coins to open the door.

### Buttons

Buttons appear. Press X of them to unlock.

**Timed (variation):** must standon these *for a few seconds* to activate them.

**Order (variation):** Buttons appear (all at once). Press them *in order*. => these have identical colors and design, which is *different* from a regular button

**Simultaneous (variation):** Buttons appear (at most #players – 2). These must be pressed simultaneously. => these all have identical color and design, which is *different* from a regular button

### Gates

**Slots (UNTESTED):** one of the gates disappears ( = becomes faded out, collision removed) for a while, then comes back and fades another, etc. => Over time, they stay open longer. Or more of them stay open.

**Fast Gate (UNTESTED):** 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.)

### Float Lock

Stay afloat (no touching something) for X seconds. Accumulates for all players, doesn’t reset when you touch, so it’s not that hard.

### Painter

Paint 100% (roughly) of the area by moving across it.

**Erase:** the area is pre-painted, you just need to erase it. (Same basic idea, just a nice visual variation, and feels satisfying.)

**Holes (variation):** black holes are punched into the background. Remove them all (by painting over them). This is more *precise* (higher resolution, really need to remove 100%) than the others.

### Shop (TO DO)

**ALL OF THIS IS WAY TOO COMPLICATED. Find a way to completely integrate it with the other systems in the game, or leave it for another time.**

Touch the <PURCHASE ICON> to buy the item on display => this can be a *terrain* or an *element*.

* This means you get *whatever effect it normally has* for a limited time?
* Or are these another unique element, with their own tutorial? Feels like too much. Unless I keep it *real* simple … (2-3 word explanation) … and whenever possible use the same icon/color as an existing terrain that does the same.

To unlock: buy X items.

**Problem:** what if players don’t have enough coins? => The price slowly lowers and lowers.

Remarks:

* These “special items” use the same icons as terrain already in the game. Still, they get their *own* little tutorial above them, as well as their price tag.
* This room is *gigantic* to accommodate the icon + tutorial for it + enough space to choose whether players want it or not
* **Optional:** effects are active until the next shop? Or they just wear off automatically after a time?
* **Optional:** players can only buy *one*?
* **Optional:** the price lowers over time, until it switches to something new when price is 0?

Item ideas:

* All your bodies are destroyed (except the foremost)
* Grow bigger
* Grow smaller
* Permanent speed boost
* Permanent speed buff
* Permanent wolf

### Lock “label” improvements:

* Place the “label” enough to the inside that it fits nicely in the room
* Also, if possible, prefer labels on solid tiles. (Not empty space where players might go through or something important might happen.)

## Elements

Within any room, it can place *tiles* to fill the space. A tile can have one element attached to a side. (Which can be an obstacle, a powerup, a special item, whatever.)

### Spikes

Hit it (dead on, not just a roll brushing past) and you’re split in two.

### Buttons (Unpickable)

Regular, Timed, Order, Simultaneous => never appear on their own (*unpickable*), used in their locks

### Trampoline

When hit, gives an enormous boost opposite to gravity direction

### Speedroll/Slowroll

When hit, accelerates your roll immensely, or slows it down.

### Ghost

While touching it, you are a ghost

### Shield

While touching it, you are invincible

### Breakable

If hit with speed, it breaks. (Must be a “tile inside”.) Might give you a coin as reward?

### Shape

**Shape Reset:** resets you to your starting shape

**Change Shape:** shows a specific shape; resets you to that

**Rounder:** very quickly rounds your shape + grows

**Sharper:** very quickly deforms your shape + shrinks

### Cannon

Shoots bullets across the room. When they hit you, you’re bumped back (and sliced?)

### Laser

Cuts through the room as far as it can, in a straight line. But turns on/off on a timer.

### Ice

Same as ice terrain.

### Spiderman

Same as spiderman terrain

### Glue

The “glue” property stays active when near. (So bring two bodies here to glue them back together.)

### Coin

Land on this cell to get the coin inside. Very rare.

### Freeze

Freezes you until some other body bumps into you

### Time Bonus/Penalty

Gives a time bonus/penalty

### Fast Forward/Backward

Teleports you to the *leading player*. (Or the *trailing player*, if backward.)

### Platforming

Clear out a (big room). Then just place platforming stuff inside:

* Small platforms (at weird angles)
* Moving platforms
* Ramps
* Moving walls/gates/obstacles

### Inner Gates

Basically, anything that can be placed *inside a room*, to prevent players with momentum/skill from just *blasting* through all these rooms.

These have to be “temporary” or “bypassable”, otherwise they’re just a terrible version of a *lock*.

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

**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 (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 = edges

## 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.)