# Splitball

## Alt Names

* Cutie Footie
* Multidimensional Soccer
* No Own Goals Allowed <= I like this one as well, or a variation on it
  + Own Goals Not Allowed is better, because “no own” is clunky with two o-s after each other.

## Idea

You’re playing a *very simple* pong/soccer game. You’re just a dot, the ball is just a dot, goals on two sides, small field. 2D, top-down view, simplified graphics.

**However, multiple of these games are happening at the same time.**

So you’re not just moving in one game, you’re simultaneously moving in *all of them*. Which means you might accidentally block your teammate’s shot somewhere else. Or own goal.

“No Own Goals Allowed”: if you score an *own goal*, you immediately lose. (Or it’s just a *huge* penalty, like 10 goals to the opponent.)

The general feel of the game? One where you must always multitask and jump between fields/playstyle, and each field is always unfair (with one side at a disadvantage) by design.

## Objective

Score X goals *across all fields*.

## Input

Each player can *move* their character. That’s all.

## Implementation

Just make each field a *self-contained game*.

* It holds a *score*
* It holds its characters
* It holds its visuals

The direction/location of goals and stuff is *randomized* per field. It’s best if this is *as different as possible*.

Then create a FieldManager => it displays the fields in a sensible configuration. (Probably just put them vertically next to each other. Maybe, when more fields enter, we can just *resize* them to match?)

And create a PlayerManager => with a Player module for each player => which sends its input to *all its characters*.

Also, **AI / Bots** would be a great addition. Would increase player/team count and make solo mode possible.

## Extras

Create a way to reduce the number of characters. (So you’re not present in some of the fields.) => this has both upsides and downsides. You’re not own goaling as much, but you also cannot defend/attack at those other locations.

Dynamically add/remove fields. (On remove, add whatever score it had to the existing fields? Or you just *lose* that score?)

A “max bodies on field” (or “min bodies on field”) powerup/property?

## Potential Pitfalls

**How to prevent players from just playing on *one field* (ignoring all others)?**

The rule “own goal is really bad” and the fact that you can always focus on other/easier fields are *subtle* pushes.

Here’s a more certain method:

* When you score, your character is *removed* from that field. (If this was your last character, it’s simply transferred to another field.)
* When someone scores against a team, one of the players in that team gets an *extra character*. (If you’re already present in all fields, nothing happens.)

Scoring an *own goal* means these cancel each other out: you are removed from that field, then added back.

**How to prevent extremely easy scoring (because one team just isn’t present on a field)?**

The method above kinda solves this. (If you concede a goal, you get an extra character on that field.)

To aid players:

* Provide ways to purposely walk to a different field. (Like powerups or gates you can enter.)
  + Gates at the sides of the field, that literally connect with the field they go to (or point at it) are probably clearest.
  + Don’t allow entering a field where you already are? Or is that actually also fun?
* Provide ways to *defend* a field at which you’re not present. (Through powerups, actions you apply to *other* fields.)

# To Do

**BUGS:**

* When player removed, all interfaces after it should slide down. (From left to right, activate one, deactivate next.)

**EXTRAS:**

* Grow goal if players close, shrink if nobody close.
* Position goal randomly on back edge?

**POWERUPS:**

* Make some basic ones that are just sensible for this type of game.
* Place them randomly. Make them scarce, a big impact when picked up.
  + **IDEA:** Pickup powerups by shooting them with the ball? That’s a bit harder. Less easy to pickup good ones you want, but also less chance of accidental powerup grabbing.
  + I don’t want an extra “pickup powerups” mechanic; how to integrate this with the main game?

**GAMELOOP:**

* **Bots?**
  + Calculate our move for all fields.
  + If these roughly align, average them and do them all. Otherwise, prioritize one field at a time: the one where the *ball* is closest to our goal or the one where *we* are closest to the ball.
  + Move towards a point *behind* the ball (so it always goes away from our goal)
  + (Randomly, decide to move to a *gate*.)
  + (Also try to line up shots so they hit an opponent?)

**Extra:**

* *Color* goal lines appropriately. (Also, I need an impulse too drastic for the effect to work; dampen it less?)
* Reset balls to random positions in the field? How to prevent *insta goals* (just cause you’re close when the ball respawns?)
* *Inside* shadow for the field walls. (Also rounded corners?)
* The goal *placement* can also be randomized.
* Sound effects and some soundtrack.
* Some sort of powerups that
  + Affect the field to the right/left
  + Affect *all* your characters. (So you could pick up a speed boost somewhere else, then become faster *everywhere*.)
  + Reverse goal directions, change goal position, anything to prevent “static goal blocking”
  + Add/remove balls

# Devlog

Welcome to this *really short* devlog for my game “No Own Goals Allowed”.

It’s my first “weekend project”. Previously, I basically worked on my main project(s) all days of the week, but it was just burning me out. So from now on, I’m trying to switch it up during the weekends.

For that, I needed (game) ideas that were short and simple enough that I can make them in 2-3 days. And they’d actually be finished and fully playable.

Here’s the first one: **No Own Goals Allowed.** (Previously called **Splitball** in my notes.)

## What’s the idea?

You’re playing a *very simplified* game of soccer. Two goals on opposing sides. Walk into the ball to hit it. Score in your opponent’s goal.

Simple, right?

But here’s the twist: you’re not playing **one** game … you’re playing **multiple at the same time.**

When you move up (for example), your character moves up in *all* fields.

This seemed like a simple idea that would work wonders, as it’s really hard to keep track of multiple fields. And it’s even harder to not *accidentally* score own goals.

That’s where the second rule comes from: **own goals are 10 points for the opponent.** (Which is, you know, a lot more than 1 point.)

That’s the whole idea! Let’s start.

## Day 1: Basics

I laid down the basic components/systems I needed and powered through them. (I have enough confidence in my game creation skills at this point that I don’t bother testing at this stage. Maybe that’s a mistake, but it does make it *fast*.)

**Field Manager:** Can add/remove fields and position them correctly (centered on the screen). Also has helper functions for *aggregating* data about fields, most notably the *score*.

**Field:** dynamically creates boundaries, but leaves a gap at the top/bottom for a goal. Most work went into making this look acceptable. As fields, goals, etcetera can be *any* size, I couldn’t just pre-create a few beautiful sprites and slap them on there.

I spent quite some time making it obvious which goal was which, as that seemed important :p If the top goal is from team 0, almost *everything* at the top of the field is colored the same way. But not *exactly everything*, as that looked ugly and too “harsh”. The field is still just green.

**Balls:** just a RigidBody with high bounciness. It has code to:

* Cap its velocity (so it never comes to a stop, but also never goes to fast)
* Teleport (e.g. after someone scores, the ball resets)
* Track who was the *last* player to touch it (to figure out if something was an own goal)

**Players:** just a RigidBody again with high bounciness. It has code to:

* Move (*slightly* momentum-based: a small period where it speeds up/slows down before reaching the desired speed)

Erm, yeah, that’s it. Simple idea, right?

## Day 1: Input -> Fields

When created, fields spawn a **ball** and a **player character** for each player.

The ball gets a slight push, but otherwise is completely controlled by physics and its own script.

The players are, obviously, controlled by *you*.

A global “input” script polls input for *all players*, then sends it through to all the fields, who send it through to their player characters.

This is way cheaper than adding a separate input script/module to *every instance of every player*. It’s also more flexible for me, as I can (for example) completely cut off a player with a single line from this script.

One downside? Players can’t randomly walk to a different field. (Which they could before, as the goals are *open*.) But that was never a planned feature anyway.

## Day 1: Checking goals

Surprisingly, checking whether an object has fully crossed a line, and then resetting it, is more annoying than you think.

* Using an Area? You’d need to position it *precisely* so it triggers when the ball is over the line.
* What if a player touches the ball again when it’s inside the goal?
* If the ball immediately resets, it leads to “insta-goals” for the opponent, as *you* still need to walk back after scoring.
* Etcetera.

I chose to:

* Do a basic y-coordinate check: if ball is higher than goal line, someone scored.
* When a ball is reset, it is “frozen” for a second or so.

Sometimes, the ball also, erm, wouldn’t reset. I’m at my wits end on this one and think it’s just a bug from the (altogether quite buggy) physics engine, so I ended up just completely destroying the ball when scored and placing a new one. Not optimal, but who cares.

## Day 1: Extra Rules

After testing this, the idea was *working*, but far from *really fun*.

Fields were a bit crowded on high player counts. And most people, intuitively, still focused on one field only.

I added two simple rules to make the game asymmetrical:

* When you score, that character is *removed* from that field. (Unless it’s the last one you have.)
* When you get scored on, a player on your team that’s *not* on the field yet, is *added*.

In other words, if one team does great on a field, then it’s slowly hindered while the other team is helped. With this system, you *can’t* focus on one field, as there’s a low chance everyone is there.

## Day 1: Visuals

The rest of this day was spent

* Adding loads of visuals, indicators, feedback. (With a game happening on multiple fields, you *really* need this to make sense of it all.)
* Finding bugs/issues/doubts and trying to find ways to fix them.

For example, everything in the game is already animated. (When a field appears, it does a short tween. When a player appears, same thing. When score changes, the text has a flashy/bouncy tween as well.)

Players/balls have a particle trail when walking. There’s a wild explosion on goals (and game over).

I added a simple *Game Over* and *Pause* screen to close the game loop. (Always important to do that quickly: being able to start, play, end, then restart the game. It’s the loop that makes a game, well, *a game.*)

The goals were a bit empty being holes, so I spent way too much time programming a bouncy/wavy line … that reacts to a goal. (So when the ball flies in, the line displaces as if the ball caused it, wobbling for a few seconds afterwards. Looks cool, not sure if it was worth the time though :p)

## Day 1: The Issues

Here are the issues with which I went to bed:

* I need some simple rule for adding/removing fields during the game.
* *Forcefully* moving physics objects (outside the physics simulation) is *bad*. So when a field is added, I shouldn’t have to move other fields to make space/center it … but how?
* I needed a way for players to manually move between fields. (Adds more strategy, fits the theme of the game, helps soften annoying situations where one team is completely wiped from a field.)
* Right now, “goal blocking” is a pretty viable strategy. Just place all bodies you have in front of the goal and done. How to prevent this?
* The game needs just a bit *more* than it currently is.

## Day 2: Solutions to Issues

Sleeping is a good thing.

### Field Changes

**Solution #1:** fields are added/removed *when somebody own goals*. Adds to the theme, it happens enough (in my testing) to make it worth it, but not *so often* that you go crazy from all the fields.

(With the space we have, this means the number of fields is wobbling between 2-4, which I think is a good balance.)

### Add a Camera

**Solution #2:** use the age-old trick of “don’t move the world, move a camera in the opposite direction”. Fields are simply added to the end of the current field list, then the camera is moved to fit them on-screen. (So existing fields are *never* moved.)

### Teleports

**Solution #3:** gates/teleports. But I didn’t want to make them really prominent features of the map, as we’re already out of space, and if players start teleporting too much the game is just chaos.

Instead, there’s one gate in the *left* wall of a field, and one in the *right*. Obviously, the right teleport of a field *brings you* to the left one of the field next to it.

As such, they basically explain themselves and you know exactly where you go. You can also *accidentally* use one, but not so easy you get annoyed.

(Because it’s a one-time teleport, that is something the physics engine can handle if I code it properly.)

### Varying the Fields

**Solution #4:** Of course, the whole idea of the game itself already *softens* the “goal blocking” problem.

Because you can’t position characters individually, it’s really hard to block the goal in *all fields*.

Because own goals carry such a penalty, you *don’t* want to be close to your goal and accidentally hit it last.

But it’s not enough. Here are ways to shake things up, hopefully enough to prevent this …

* Goal size and position are somewhat random.
* Fields can be *rotated* (so that the *other* team is at the top)!
* When you are *close* to your goal … it grows in size. (And when you are *far* it shrinks.)

That last rule is a bit exotic, but it fits well. It also punishes being “too present” on one field, because that’d mean there’s no space to stay away from your goal.

### Powerups

**Solution #5:** well … **powerups!** It’s a game about moving. All you can do is determine *where you are* and *where you are not*.

Those games are perfectly suited for powerups that you grab by being somewhere, or avoid by purposely *not* going there.

I had some doubts about the implementation. Wouldn’t it be *too easy* to accidentally grab one if they triggered so easily?

But in the end, I applied the lessons I learned from previous games. *If you add powerups, they should be …*

* Rare
* So impactful it feels unfair
* Immediately clear on first sight

If they’re not rare, you should’ve just made their effect a permanent/global thing, because being individual powerups is senseless.

If they’re not impactful, players don’t care about picking them up.

If they’re not immediately clear, players will be confused about what happens, or stop playing for ten seconds to figure it out.

To finish it off, one last rule to tie it all together: **picking up a powerup in *one* field activates it for *all your characters* (in all fields).**

(We’re still in the first hour of day 2, I’m just writing down these ideas, then I’ll implement them. Which is where we’re going now!)

## Day 2: Gates & Fields

These were quite quick to implement, but revealed their subtle issues later on (as they always do).

Many bugs came from the fact that “Collisions/Events could still happen while a field was being removed.” (What if a player enters a teleport to a field right when it’s destroying itself? What if someone scores another own goal on a field removing itself, and it comes back?)

Once I realized this, I added a simple variable “busy\_removing”, and when true, everything just stops and calls to events are ignored.

When a field is removed, I also forgot that:

* Its score would also be destroyed. Solution? Keep a “permanent score” counter as well. When a field is removed, add its values to that, so its remembered from now on.
* A player might end up without characters. Solution? Check this, if so, add a random one to a random other field.
* The links between gates wouldn’t work anymore. Solution? Just call for a complete “relink” of all existing fields on removal.