# 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.

## 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.)

## Powerups

### Field-Focused

* Remove everyone
* Remove everyone from the opposing team
* Add a ball
* Make the ball worth 5 points
* Make the ball worth twice as many points
* Slow down the whole field
* Shrink / Grow the goals
* Reverse goal dir (which team at top, which at bottom)
* Make ball small
* Make ball extra bouncy
* Shrink all players on field

### Player-Focused

* Move faster
* Move slower
* Reverse Controls
* Shrink
* (Attract ball towards you?)
* (Can’t own goal?)

Something more to change the physics? (Players get a tiny attracting/repelling force?)

(Only shrink or grow *your* goal with that powerup?)

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

# To Do

**PUBLISH/POLISH:**

* Remove all TO DO/DEBUGGING stuff
* Add screenshots/GIFs
* Upload builds => publish => mention it (Twitter/Pandaqi/Portfolio/NDHU)

### Optional To Do

**POWERUPS:**

* Create the opposite variation for many of them (shrink ⬄ grow, extra bouncy ⬄ less bouncy, etc.)
* More feedback (what they do, when activated, particles, etc.)

**FIELDS:**

* Differentiate movement between fields a bit more => random blocking elements within each field?
  + Maybe this is a *powerup*? (That places obstacles in both its neighboring fields, or only the one it points towards.)
* *Inside* shadow for the field walls. (Also rounded corners?)
* The goal *placement* (on back edge) can also be randomized.
* Grow goal if players close, shrink if nobody close.
  + Or just grow/shrink goals after each goal
  + Or check which team has *more* players here (on a fixed timer)

# 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.

(General rule: it’s often *way easier* to just recalculate/redo/update everything when any change happens. Way less error prone, more flexible and robust in the long run. Yes, don’t do this when the update is *very expensive*, but if that’s the case, the algorithm you’re writing is probably at the core of your game and you should take more care anyway.)

## Day 2: Input Screen

I quickly decided that an in-game tutorial with several images popping-up just wasn’t a good idea.

If I wanted to explain all the rules, it’d be too much information, not in the spirit of the game. But not explaining *anything* also isn’t great.

So, on the main screen, there’s a big image on the left with just 4 short lines and example images, explaining the core:

* Hit the ball
* Score goals
* Don’t score own goals
* We play until X points

That’s really all you need to know to start playing. The rest shows itself during gameplay.

On the right, we have the login screens. To make my 2-day deadline, I decided to keep it *very simple*. All you can do is:

* Add a new player/device
* Ready yourself ( => once everyone is ready, the game starts)
* Remove yourself

But interactive menus are *awesome*, so this is one as well. When logged in, you get a player character (confined to your box). The keys for movement are shown in the background and you can move around. In fact, to ready/remove you must *move yourself* to that area of the login screen.

This makes the menu much more lively, fun, unique. But it also *teaches the buttons*!

Great. (I had some trouble with removing players not updating the background device dictionary correctly, but that turned out to be a bug with my general library. I had to “port” it to Godot 4, from Godot 3, and a few mistakes slipped in.)

This means the game is **fully playable.** You can start, login, learn the buttons/objective, play the whole game, and someone wins in the end. (I could publish it like this, if I wanted.)

## Day 2: Powerups

I did a playtest *for a different game* (my main project) around this time. So, no progress for a few hours here, but lots of insight on how to improve that other project.

Anyway, it’s evening now. Let’s try to pump out the powerup system and a few good ones.

Very quickly, I realized that powerups across the field just weren’t a good idea. It was *so easy* to grab one … and even easier to accidentally grab one and don’t know what the hell just happened, because you weren’t looking at that field

**Solution 1:** Make them quite small and place them in the *corners* => only a weak fix

**Solution 2:** Powerups are yours by hitting them with the *ball* => harder to do, encourages looking for who last touched a ball, but still not a complete fix.

**Solution 3:** **powerups spawn between teleports.** In other words, when you teleport from field A to field B, you get the powerup showing between them.

Very thematic, encourages/discourages teleporting at different moments, harder to do accidentally, and more visible. (There’s nothing else showing in that space.)

That’s what I went with.

Sounds easier than it is to implement. (Because we have to find pairs of gates that are *not* already being used, which are *not* at the extreme of the map, and remove them properly when their fields are removed, as well as reposition the powerup properly when the gates reposition.) But I got there in the end.

Which powerups sound fun?

I saw several categories:

* Those that allow defending/attacking in the field to your right or to your left. In other words, do something about a field *where you are not*. (I can just draw an arrow on those powerups and flip it when needed.)
* Those that mess with your movement and ball hits (as that is all you have, and it transfers well to *all* your characters)
* Those that give *variation* to prevent many issues I mentioned. (Camping on your goal line, focusing on one field only, etcetera.)

I was running out of time, though, and I realized I actually didn’t have that many *great* ideas for this. So I kept the list small.

Eventually, my creative juices were just depleted, so I went to bed with some work on this left to go.

## Day 3: Finishing touches

The next day, I woke up and immediately drew all the icons and finished the missing powerups.

Then I powered through a list of small issues or improvements, which I thought were important enough to work on. (For example: ramps in the corners to prevent balls getting stuck there.)

Which left me with some polishing left to do:

* Some sound effects and something resembling a soundtrack
* Some animations/tweens/particles where they were missing (mostly the newer stuff I implemented)

This was quite some hours of work. Which taught me my next weekend project should be even simpler and shorter than this one.

## Day 3: Bots?

And lastly a question: do I add *bots* to this game? (Limiting the game to 2+ players only is always a great way to ensure almost nobody will or can try it.)

It’s not really trivial to add bots, but also not the hardest thing in the world.

So I pulled the trigger and did it, as the final task for this project.

* In the main menu, you can turn on “bots”.
* All players are in the same team. At game start, as many unique “bots” are started as there are players.
* Each bot checks for the closest ball and tries to get *behind* it. (Between the ball’s trajectory and their goal.)
* When spread over multiple fields, the bot focuses on the closest ball first.

Not great, but it’s something.

## Day 3: Mustn’t forget

Marketing. Of course, I’m not talking about *really marketing* this game, but I also can’t just upload it without a piece of text or a few screenshots.

So another half hour was spent getting those screenshots, writing that text, making a logo that’s somewhat decent.

And with that, we could finally, definitively, upload this game and call it done.

## Conclusion

That’s it. My first “weekend project”. A very simple game about playing soccer on multiple fields simultaneously.

Still took too long and became too complex in certain parts. But all in all, especially for *me* and my bad habit of overscoping, it was a great training in keeping something small and finishing it in a really short time.

It was also a nice diversion from my regular, bigger projects. And another idea that’s been bugging the back of my mind for weeks that I can now put to rest with a satisfied heart.

(Forgot to mention: I also tried a different *structure* for organizing projects in this one. Not sure if I like it better, need more projects to determine that. But it was, again, something different and something educational.)

The end result is … fine, I guess. But that’s what I always say about my own work :p

Until the next devlog,

Pandaqi

## An update

Before releasing, I actually *did* come back and fix some issues. It seemed to important not to do.

The biggest ones:

* **Overcrowding:** if multiple players just gang up on you or the ball, they get total control, and there’s nothing you can do.
* **Corners:** same thing. Ball is in a corner? Someone else is there as well? It’s basically stuck and won’t get out.

How to solve this?

**Solution 1:** We just make the ball much bigger and the players much smaller. A player can’t stop a ball that’s 10 times its size. It’s harder to provide a brick wall if it’s really thin.

**Solution 2:** Add a small “stun” when players collide.

**Solution 3:** Make corners much bouncier, so players/balls fly away from it anyhow.

**Solution 4:** And then allow the ball to have more influence on players, pushing them out the way. (Now players are basically much heavier and in control.)

**Solution 5:** Add random obstacles *inside* the fields to prevent players from moving just anywhere, or moving like a gang/group.

In general, I already wanted to increase the size of the field *relative to the ball/players* by quite a bit. Currently it was a bit cramped and left too little options.

So these changes should go hand in hand.

In the end, I decided to simplify/streamline these ideas (as I noticed some extra problems with them), with the simple rule: **when you hit the ball/another player, you bounce away from it.**

This means you simply *can’t* overcrowd, or just walk the ball into the goal, as you’re being bounced around the place. By tweaking the physics a bit, I was able to make this feel nice, and balance *chaos* with *strategy/control*.

Lastly, I added some “quality of life” features:

* Particles when you grab a powerup (or a powerup someone grabbed affects you)
* Text feedback with a hint as to what the powerup does
* A graphic with an overview of powerups. (Not in-game, but useful for myself and in the marketing.)
* More powerups to balance it all.
* Instead of making corners “super bouncy”, I simply gave them a timer that pushes away everything near it once a second. When that happens, it plays a nice animation of a growing circle, somewhat “explaining” to players what happens there.

Also, because I opened the project in a *newer* version of the software, it unexpectedly broke some stuff. So I had to waste some time figuring out exactly what broke and *why*. Among those were *bots*: suddenly they walked away from the ball instead of towards it :p

But then I decided it was *really* done. I tested it with some people, used that footage for trailer/screenshots, and this project is done!