# Devlog: Carving Pumpkins & Dwarfing dumplings

Welcome to the devlog for my game **Carving Pumpkins & Dwarfing Dumplings**.

It’s going to be a *really short one*, as this game is basically a spinoff from another game I was making. (It’s **Rolling in the Sheepe**, coming soon.)

In that game, I implemented a system that could *slice any shape (realistically*). So, for example, a player could be a *hexagon*. When I drew a line through that hexagon, it would *split* the shape into two parts. (Which, if you were to glue them together again, would represent the original hexagon.)

It was really cool to figure out *how* to do this. It’s “relatively easy”, though still quite challenging. (Especially when you get to supporting *any* shape, not just the “nice ones” like circles, rectangles, etc.)

However … as the game progressed, the mechanic just didn’t fit anymore. It was more *fun* to split players non-realistically. (Splitting a hexagon would just yield two smaller hexagons.) It fit better with the mechanics, the gameplay, the feeling of the game.

Determined to not let my code go to waste, I decided to create a quick little game that *did* use it!

As Halloween was coming up, it became a silly party game about slicing ( = carving) pumpkins.

**Remark:** I will *not* explain the algorithm for slicing shapes (in 2D) here. It’s quite complex and I discussed my journey of discovery *at length* in the other devlog. This is meant as a short devlog discussing only the interesting bits from *this* game.

## The idea

It’s simple. You can move and you can throw a knife. If the knife goes through another player, you literally slice them in two. The biggest part remains ( = *you* *are the biggest part*), the smaller parts will be lost and out of your control.

Any player who is too tiny, dies and is out of the game. The last one standing wins.

## Step 1: Sometimes you don’t need all the physics

At first, I implemented knives in the “traditional way”:

* I gave them a (narrow, rectangular) body
* When you threw them, I apply an *impulse*
* When they hit something, I decide whether I want to *slice* it ( = hitting another player)
* If not, I simply *bounce off of it* ( = hitting a wall) and let the physics engine do its thing.

This didn’t work.

Why not? Because *slicing something* is completely different from *hitting something*. They are, in most cases, complete opposites.

* To hit something, you need a body with some “area”. Slicing something means cutting it *along a thin, zero-width line*.
* It’s really hard to tell the physics engine to “delay” colliding with something. They’re not built for that, and for a good reason.

As such, the code would work 50% of the time. But the other times, one of these situations would happen:

* The *body* hit something. But when I shoot a line from it, the line *missed* that object. So we clearly hit someone … but still didn’t slice them.
* We sliced someone. But, the *collision* also came through, which means our knife had some random rotation/offset added *before* calculating the slice line. Leading to wildly unpredictable slices.
* Sometimes, if the knife was going fast, both cases would simply fail and nothing happened.

So let’s return to that first remark: **slicing means a zero-width line.** That means … we don’t need the physics body (on knives)!

I removed the body (and its shape). I added some code to handle *velocity* myself. (Simply move according to velocity each frame and dampen it a little.)

Then I added a **raycast** just ahead of the knife. If it hits a player, it shoots a line straight ahead, and slices the player across that line. All bodies that come out of it are saved as “exceptions”. These will *not* be picked up by the raycast from now on.

(Otherwise, it just keeps slicing and slicing every frame, because it will *keep hitting the player* until the knife comes out on the other side.)

If it hits something else, I simply *deflect* the knife. There’s a basic formula for deflecting a force/velocity:

<TO DO: CODE HERE>

With these simple steps, we have a knife that can both *slice* and *collide* (realistically) … without actually having a physics body.

I only use the physics engine for shooting that *raycast* into the world. Otherwise, the knives are completely handled by my own code, which isn’t more than 40 lines.

## Step 2: Throwing and catching

Each knife has an *area* attached to it. (It’s called Area2D in Godot, my game engine. Many others call this a *sensor*.)

If this overlaps with its owner ( = the player that threw the knife), you pick it up again.

Of course, this has one issue: when you throw a knife … it immediately overlaps and you pick it up again! Which means nothing happens.

As such, just after throwing, I “disable” this area for 0.5 seconds. (This has the added benefit that throwing the knife into the wall, and immediately deflecting, will make it go *through* you instead of nothing happening.)

Then I added some simple code to reposition the knives correctly. (On the edge of the player shape, whatever that shape is.)

It uses the **Shoelace algorithm** to estimate the area of the player. We know that, in a perfect circle, Area = pi \* r^2. We can reverse that to get an *estimate* on the player radius, which would be r = square root(Area / pi).

The knives are placed this distance away from the player, a bit offset from each other. This way, they stick out nicely, whatever your shape.

At first, I “repositioned” the knives to always be in front of you. (Which is logical, as that’s the direction you’re facing, and that’s from where people usually hold/throw knives :p)

But I soon discovered this had issues and that there was a better idea: just keep the knifes *wherever you caught them.* If you catch your knife with your back … well, guess you’ll have to put some extra effort in aiming it later.

## Step 3: Cleaning up the mess

Realistically slicing everything has one downside: you can end up with loads of ugly, tiny shapes floating around.

That’s why there’s a minimum area. If a shape falls below this – again, this is estimated using the Shoelace algorithm – it’s simply deleted immediately.

Similarly, the parts that fly off a player have some damping attached to them, so they don’t just endlessly float around. After a few seconds, they will have stopped rotating and moving, and just lie on the arena as a sort of “evidence of what happened”.

TO DO: Image??

## Step 4: Making a first level

Now we need these things for a first level:

* The core game loop. (Know when it’s game over, do something then.)
* An arena in which to play. (Some obstacles, a background, etc.)
* Some powerups would be nice. Something basic like “you get an extra knife”.

Let’s do it!