I received this email on September 5th, 2024:

> Huge fans of what you and your team have done!...
>
> We have been having a problem in our game with parsing out NPC paths and schedules; how to break apart what they will do on a given day based on the weather and the calendar...But I was wondering if you had a design philosophy that has worked best for you? Do the schedules change dynamically throughout the day or are they static for the calendar and weather conditions? What has worked best when it comes to storing schedule data on the dev side?...
>
> Thank you for reading!
>
> Best,
>
> Ryan

First of all, Ryan, opening with flattery was the right move. Secondly, this response is long overdue, but your question requires a long answer which will be split over a couple posts.

There are two elements at play here:

1. How do NPCs behave within their 2D environment?
2. How do NPCs decide what to do on a given day or on a given time?

Ryan, and I assume others, you are probably interested in this second topic; to get to those answers though, we need to answer how NPC behave *within* a schedule first, because their moment-to-moment capabilities define what their overall schedules can do. To that extent, I want this article to explain how *Fields of Mistria*'s NPC behave, at the most simple and basic level, so that a followup article can explore these more in-depth questions.

My intended audience is, ideally, a designer/programmer who is interested in implementing NPCs in a farming sim, town simulator, or other slice-of-life style game. A little bit of design and programming knowledge will be useful, particularly around pathfinding algorithms, but I hope that I'll explain enough that a newcomer in this space wouldn't be shocked -- in fact, if you just want to learn about all the funny and bad things we did in *Mistria's* development, I hope this article will be entertaining enough.

Okay, that's enough pre-amble -- let's talk about NPC's in *Fields of Mistria*.

How do NPCs Behave

NPCs in farming sims have certain behaviors, which can happen on and off screen:

1. NPCs must animate and make noises in certain positions.
2. NPCs must move between locations and positions.
3. NPCs must be interactable with the player, doing certain animations, opening certain menus, and then resuming their day.

These reduce to a small number of states:

- Animating in place
- Moving between locations

INSERT GIF OF OLRIC WALKING SOMEWHERE INSERT GIF OF OLRIC LIFTING WEIGHTS

Looking at these states, a model NPC for our purposes would be an entity which moves from `point_a` to `point_b`, doing an animation while their position changes (ie, walking), does an animation at this new position `point_b` for awhile, and then moves to `point_c`, which might be the same as `point_a`. Notice that I said **"awhile"**, which leads us to our most important systemic interaction localized to NPCs (as opposed to enemies in action games): the NPCs behavior is driven **primarily** by the time of day, which is just like me in real life.

Our model NPCs then, should be very simple to program, right? We just need to figure out how to move between positions at a particular time (presumably, animating a sprite is handled in your technical setup already, and if it's not, well, you've got bigger problems), and *tutti frutti*, you're good to go. Moving between points, in this case, would mean playing a walking animation and then moving their location at some slow rate every frame, simulating movement.

There is, of course, an escape hatch -- what if they didn't move from `point_a` to `point_b` slowly by walking and instead just teleported directly? *Fields of Mistria* avoids doing this at all costs, but it is an acceptable solution, particularly for *prototyping*. You can probably get pretty far into developing a game with NPCs like this (any farming/town/life sim) by just teleporting the NPCs as you'd like. However, there are three major costs to teleporting NPCs:

- The world will generally feel quite empty. Although some NPCs will be outside, or hanging out near high traffic areas, in these genres, most NPCs tend to spend time indoors in certain locations (e.g. taking care of a shop). In *Fields of Mistria*, the player actually commonly runs into NPCs *while they're traveling*, so if NPCs just teleport, we'll miss them quite a lot.

> In the next post, we'll talk more about macro concerns like schedule design, but to tidy this point up here, most players *rarely* go out of their way to talk to an NPC without some gameplay reason behind it. The most talked to NPC in *Fields of Mistria* is, by a large margin, Celine. Celine, living just north of the player and commonly working or reading outside her home, is in the perfect location for players to run into her. In early alpha versions of the game, the least talked to NPC was Terithia, as she mostly stayed on the Beach. After these tests, we moved her to fish in high traffic areas (such as in the Town, near the Mines) and gave her pathing opportunities which would often have her cross the area of Mistria just south of the Town. Since then, her conversation rate massively increased and she's often in the top 10 most spoken-to NPCs. Conversely, Juniper spends quite a lot of time holed up in her shop (particularly in Spring, which were the earliest made schedules). Players rarely would go into the bathhouse, so players would rarely see her.

- The NPCs will feel "fake", instead of feeling like real people. Ideally, the player should think of NPCs as, effectively, real people who you cannot fully communicate with due to the nature of the game. The way the player interacts with the game is by moving between locations, using tools, and interacting with items -- these methods are precisely what NPCs do in the world of *Mistria* as well -- therefore, we end up establishing to the player that the NPCs are the Player are the *same thing*. These parallels suffuse them with *verisimilitude* (our buzzword of choice). When NPCs instead teleport, which is a thing that the player cannot do (well, barring the actual gameplay teleportation magic in the game), the NPCs cease to be similar to the player, and instead become gameplay objects, or merely bundles of code, images, animations, and writing. Keeping the NPCs feeling like "real" people is essential for the romance aspects of the game; we've taken careful steps to make sure that there is no gameplay difference between who you date, which encourages the player to choose who they want to date or become friends with *based on who they like the most*. If the NPCs don't feel like real people, then players won't be able to make that choice realistically. A bad sign, for example, would be if players say "I chose March because I like his Spring Outfit" -- instead, we want to hear "I chose March because I like his style in Spring". Ultimately, this leads us to an odd definition verisimilitude -- it is not just "realism", but rather the similarity between the NPCs and the Player character. I suspect it works like that because Players know that they are real people, so if NPCs act similarly to their in-game avatar, than those NPCs are also endowed some humanity. This endownment is a requirement for our players to treat those NPCs like real world character.

- Finally, NPCs are visual guides to the world. As they walk around, they show the player were to go. The NPCs also act as landmarks while the player walks along a path. That's why the NPCs almost never are on the *main path* through a given region, but rather just slightly off of the main path. This pattern encourages players to constantly change their plans as they move throughout the world -- maybe the player was expecting to go straight to the mines, but "Terithia is right there fishing! Wouldn't hurt to say hi! Oh, there's some forageables across the river? Maybe I'll just jump over there, oh what's this statue...?" In this way, NPC movement can make players re-think where they want to go.

The rest of this article is primarily concerned with the gameplay problems and benefits of doing real NPC "pathfinding", which is the process of moving them pixel by pixel across the map in some manner, which as I hope I've convinced you, comes with a host of benefits.

> *Fields of Mistria* actually has teleportation for NPCs in it too! When I've been saying "NPC", I've generally been meaning characters, like March, Juniper, or Ryis. However, there is another NPC class we have to deal with, as well -- player animals. (Internally, Gabe Weiner calls them "NPAs", which is delightfully nonsensical). Player animals *do* teleport, particularly when scenes/levels/rooms are loaded and deloaded (more on that later), which neatly solves their problems. In fact, the Player's pet uses teleportation so much that it doesn't even record its *position* within a room -- only the name of the room that it's currently in. Every time the player enters a room that the Pet is in, the Pet figures out what it "should" be doing. When the Pet has a Job ("Dogs with Jobs"), there's only one thing it can be doing (the job, which is an animation, effectively), but when the Pet is at home doing nothing, it'll randomly place itself throughout the room, doing particular animations. This tactic ends up giving the Pet a lot of "independent" feeling, as it seems like every time you see it, it's up to something new!

Pathing Problems

The process of writing out a list of positions (a path) for a character to move is called "pathfinding". There are two main approaches to "pathfinding" in a farming sim: algorithmic and hand-made. I **highly recommend the algorithmic approach.**

Hand-Made Pathfinding

The hand-made approach works by, for every two points in the game which an NPC will walk between, laying out the path which they will take. In the most absurd case, you could define directions for the NPC to take with a sequence of points and lengths in world units:

```
// kind of like very odd directional navigation:
South, 400 pixels
East, 180 pixels,
North, 24 pixel
```

This is, of course, extraordinarily challenging to actually implement and pretty much no one does it like this. Instead, the hand-made approach involves dotting your rooms/levels/scenes (I'm going to call these `rooms` from now on) with *nodes*. Nodes are simply positions with names. Generally, this is done in a room editor, but could also be done just in a text file.

INSERT PHOTO OF MISTRIA TOWN WITH LOTS OF TRELLIS POINTS.

```
{
  "town": {
    "point_a": [128, 1800],
    "square_north": [540, 1800],
    "square_center": [540, 2000],
    "celine_cottage": [506, 124],
    "point_b": [506, 124],
  }
}
```

Then, a given path is notated like so:

```
{
  "point_a_to_point_b": ["town/town_square_center", "town/celine_cottage",
"town/point_b"]
}
```

At runtime, the NPC looks up the location of `"town/square_center" (540, 2000)` and begins to *approach* it. By *"approach"*, I mean the NPC finds the direction to go towards that point and moves a certain amount of pixels towards it, making steady progress. You can also program them to *only* move in cardinal directions, which in these style of games is often a desired behavior.

So if you implement this strategy, you'll start to make errors -- errors are natural and simply happen as a fraction of the work you do. These bugs can be mitigated with automated and human testing, but ultimately, they'll still be made. The errors in these hand-made paths, though, can be *trivial* to make (though easy to detect and test for!).

Imagine that you wrote `"town/square_center"`, but forgot to write `"town/square_north"` first. Instead of walking to this first node, the NPC would walk *straight* to `"town/square_center"`, which possibly could include **walking through walls.** How could this happen?

Well, the real question is *"How could this not happen?" What we're describing is really a list of names with a lookup table to positions -- we haven't discussed collisions, walls, other characters, the *player*, etc.

There's no way for NPCs to *not* walk through buildings -- all you can promise is that they'll go from point to point, hell or high water, and hopefully that path ends up being the correct path.

Speaking of other characters and the player, of course, NPCs will simply walk through each other and through the player. Going all the way back to the teleportation conversation, the NPCs can end up feeling "fake" if they don't have collision. You can't walk through a person in real life, so you also can't walk through March in *Fields of Mistria*.

> This can sometimes lead to bugs, for what it's worth. Right now, in *Fields of Mistria*, if you stand in the back of the gazebo near the mansion in Town, and three NPCs come in and hang out talking here, you can be stuck for hours until they move. We prefer to think of this bug as less of a bug and more of a *fantasy-fulfillment* for some really specific fantasies. You're welcome!

But these NPCs are just moving from point to point. How can they tell if something is in their way? Even worse, what if you're making a game where the player can place furniture *anywhere*? If that furniture has collision for the player, it should have collision for the NPCs (this is how we achieve verisimilitude), so the NPCs should walk *around* the furniture. If they can't walk around the furniture, they should pause and sit there until that furniture is moved, or they should fall into some other sub-routine. A designer might counter though that furniture is not placeable is their game in places where the NPCs will be (indeed, *Fields of Mistria* is like this). However, almost certainly the *player* can go to those places where NPCs might be -- in fact, I suspect even NPCs can go where other NPCs might be! If we want that verisimilitude, we need to handle these cases at least *somewhat*.

> *Fields of Mistria* only allows the player to place furniture in their house or on the Farm. We actually *do* have a few cutscenes on the Farm, but they only happen right in the early game, when players cannot have placed any furniture. The worst offender here, though, is "The Unusual Statue, Part 1", which is a cutscene that takes place on the start of Day 2 -- at this point, it *is* possible for players to get furniture and place it on the farm (though relatively unlikely). As we'll see in this article, we design our NPCs to algorithmically walk between points, but if some collision is already *on the point they're trying to get to*, the game can fail to pathfind them correctly. We tend to choose to crash (fail an assertion) in this case during cutscenes. As a result, we actually "poof" all the furniture within a certain area (ie, we convert them to items) so the NPCs can walk around freely. However, were players to place a Barn, or other complex structure, at this point (which is impossible without cheating), players would likely certainly experience quite a lot of bugs!

Algorithmic Pathfinding

We can do better than hand-made paths by spending computational time to calculate optimal paths at runtime. Not only will this end up being a design win, it also is a production win -- moving an NPC from one point to another is as simple as saying "walk from `point_a` to `point_b`".

I'll elide the details of an actual pathfinding solution -- it's not very interesting. *Fields of Mistria* uses A* pathfinding, and you can get started on that journey here.

INSERT_GIF OF OLRIC WALKING FROM POINT_A TO POINT_B

We learned a lot while implementing pathfinding across *Fields of Mistria*, and here's a few of those conclusions:

First, pathfinding still needs a human touch. Any pathfinding algorithm, by default, is fine with NPCs taking the most "optimal" *distance-based* path, but all humans also generally take paths which are nice to walk on, where they're supposed to walk, and which obey the personal space that we tend to afford other people. Those constraints need to get encoded in the system. These are the "cost" of "edges" in Dijkstra's parlance, but we call them "cost tiles" for simplicity.

INSERT SCREENSHOT OF THE PATHFINDING LAYER IN TOWN.

Even this though isn't sufficient -- NPCs by default in this setup may sometimes end up taking *too close a path* to a collision. For normal humans in reality, we tend to walk in the center of well laid out paths -- even if another path is better, if it's walking directly next to a wall, we might not take that path, all else being equal. So we need to *extrude* from these walls, and other collisions, high cost tiles, so that NPCs actually avoid walking there.

INSERT GIF HERE

The algorithm for generating these cost tiles is rather particular:

1. Transfer over all the level collision data to the word. These would be buildings, fences, or any other *permanent* object, but **not** any player placed furniture (we'll get to that in a moment).
2. "Extrude" collisions -- every tile which borders a collision should be marked as "expensive" in your pathfinding algorithm.
3. Transfer over hand-painted tiles and tile costs. Putting them here at this last stage ensures that it can override any data from collisions or from the extrusion step.

This setup will show a good tile extraction for a given map:

INSERT PATHFINDING DATA AT RUNTIME

But just these changes alone aren't enough to solve a bunch of what we wanted to solve -- NPCs don't actually pathfind around each other, they still walk through the player, and they still walk through Player placed furniture. This failure isn't particularly surprising -- our first step was to only send level geometry, not player or NPC collisions, nor furniture data. The obvious solution, then, is to just *also* record these collisions in the first step of our algorithm above, which *Fields of Mistria* did for awhile, but I do not recommend doing.

First, numerous pathfinding inaccuracies will be caused by caring about pathfinding agents in the cost analysis. Imagine an NPC must pathfind 200 pixels north, and the player is in the middle of the direct route. If we care about the player in the cost analysis, then the NPC will generate a path where they walk *around* the player. This path is, of course, correct, but if the player then *moves* at all, the path would need to be regenerated to continue to be optimal. Regenerating a path continually is very sub-optimal, and particularly with many agents, can actually create frame time issues depending on how quickly your pathfinding is (and how good your heuristic is in A* pathfinding). Moreover, even if we *do* regenerate the path continuously, we have to think about the design -- when someone steps into another person's direct path, unless they're *very* busy, they don't tend to walk around them immediately. Instead, they wait a moment, assuming the other person will move.

For awhile on *Mistria*, we did this regeneration approach, and it made all the NPCs move around the player (and other NPCs) immediately. This approach ended up actually making the NPCs feel cold. Going back to our definition of verisimilitude, NPCs should feel just as interested in the player as the player feels about them. Therefore, the NPCs should never feel *cold* towards the player -- when the player interrupts an NPCs path, they should provide feedback (stopping).

INSERT GIF OF NPCs

Finally, even if we could regenerate paths *and* somehow stop appropriately to provide feedback for players, we still would be breaking verisimilitude because the NPCs would be reasoning about the world *globally*, not *locally*. Roughly speaking, the NPCs should only react to something within about 300 pixels. This, of course, is about the radius of a circle fit to the Camera's frustum!

Meddlers

So we need a way for pathfinding agents to talk to each other. Luckily, because we want this to be hyper local, we don't need fancy algorithms to handle it. Instead, all we really need to do is quickly look around to see if an NPC is about to run into a *Meddler*.

As said before, anyone who pathfinds (NPCs, the Player, and actually Animals) are "pathfinding agents". These agents "claim" a tile -- these are the same tiles which we use to calculate costs. In *Fields of Mistria*, these are eight by eight tiles. An agent claims a tile whenever they enter that tile while pathfinding. The agent which has taken a tile is called a "meddler".

Here's the trick: NPCs look ahead very slightly whenever they're pathfinding to see if there's a meddler on the tile they're about to enter. We look 16 pixels ahead of time in *Fields of Mistria*. If they have, they *stop* and fire off some callback function. For NPCs, they start a 5 real-world second timer, after which they instantly mark the offending meddler's position as a *collision* (as if a building suddenly popped up where that NPC is standing), and then query for a new path. Naturally, this new path will walk around the meddler, though there are edge cases here for narrow corridors, in which case the NPC may not be able to generate a valid path, and they will continue to wait indefinitely. After they query for their path, they will then remove the fake collision they previously generated. In practice, this swap works really well:

INSERT GIF OF NPC PATHFINDING AROUND PLAYER WITH DEBUG OVERLAY ON

This works for NPC and player collision. To extend it to player placed furniture is easy -- when we check for a "Meddler", we should also check for collision data. If we get any furniture collision data, rather than go into a waiting mode, we immediately request a new path using the above algorithm. That allows them to smoothly transition *around* placed furniture like they do around other pathfinding agents.

There is one final problem we haven't addressed though: NPC-on-NPC *moving* collisions (where two NPCs are basically walking into each other and neither is static) could generate infinite waiting games. These waiting games were a funny recreation of something that happens in real life ("After you", "No, after you", "No, after you", ad infinitum) but created traffic pileups in commonly seen areas. We solved this by creating an NPC priority order (which is simply just their alphabetical order) and thus allow Adeline to walk all over March. (Many of ours fans would be delighted to see this action in the game, which they can easily on Monday afternoons as Adeline bosses March out of the way.) Effectively, when Adeline is moving and the meddler is also moving, Adeline will *not* patiently wait for the Meddler and instead will immediately query a new path if the Meddler's priority is lower than Adeline. Adeline (with an "A" name) has the highest priority of any NPC, but as you'd expect, the Player has the truly highest priority. Ultimately, this priority game is what actually makes NPCs wait for the player!

With all of that, we have described *Fields of Mistria's* pathfinding system! There is still room for it grow, though, and we haven't at all covered all our corner cases:

- NPC-on-Player-on-NPC collisions however are mostly a mess, since both NPCs will end up waiting for the player forever, attempt to walk around the player, and then stop as they run into the other NPC. This loop continues forever. Since neither NPC is moving, they don't trigger the failsafe for their own collision. However, these collisions almost never happen in game except when the player is blocking a door (for example, to the Inn before "Friday Nights at the Inn"). Since the NPCs can't get to the door anyway when the player is blocking them, waiting indefinitely is the correct behavior, but there are likely other niche cases were this problem can occur in normal gameplay. Likely, we would need to consider multi-agent pathfinding approaches if it became very common. If *Fields of Mistria* had even more NPCs who regularly were pathfinding around the Inn's bar, for example, we might need those multi-agent algorithms.
- It has been relatively difficult for us to minimize turns for the NPCs. They end up taking *diagonal* or staircase stepped paths more often than we'd like. A staircase stepped path is when, instead of going straight up and straight left, they go diagonally up 8 pixels, to the left 8 pixels, over and over again. If anyone knows of a good way to help us with this, please do reach out! We adjusted heuristics, but getting some "state" into the heuristic felt like a bad decision, and in practice, didn't actually seem to work all that well anyway. In cutscenes, where this matters a lot, we tend to chain pathfinding operations together to reduce the number of staircase steps they'll perform in general.

Multi-Room Pathfinding

Fields of Mistria, like most games, doesn't actually operate within a single giant room -- rather, the game is composed of many rooms. A room is partially a gameplay design, partly an engine optimization similar to culling, and partially a simplification for production, since we can talk about discreet rooms.

There are many ways to handle *pathfinding itself* in a game like *Mistria*. Let's create a base example that we'll work on: we have Eiland, a loveable archaeologist, pathfinding from the Western Ruins, at the very top-left of the world, all the way to the bottom right Eastern Road. He's going from a particular point in the Western Ruins to a particular point in the Eastern Road, so let's call those two points `western_ruins_start` and `eastern_road_end` for simplicity. He'll need to go through several "doorways" on his way. "Doorways" in this parlance are connection points between two rooms. Often, for buildings, these are actual doorways, but for lots of open world areas, they are simply paths which lead offscreen. Eiland will surely need to walk through the `western_ruins_to_narrows` doorway, because the Western Ruins has no other doorways leading to and away from it, but after that, it's unclear where he should walk. He could go south through the Narrows, into Town, and then the Eastern Road, but he could also randomly walk into the Summit, or turn around and go into the Western Ruins, because continuing into the northern part of Town, and then going to the northern part of the Eastern Ruins. Some of these paths are obviously less optimal, but we need to decide what we even *want* multi-room paths to look like. We will call these kinds of pathfinding operations "map pathfinding" when the path goes through multiple rooms.

What is the goal of a pathfinding algorithm in *Fields of Mistria*? We left that question implicit for simpler pathfinding operation, because it has a simple answer, but we should answer it more explicitly: the goal is to minimize "cost" between two locations. "Cost" here is an abstraction combining "time" and "reasonable path". The goal is not *purely* the fastest path for them to walk, since there are paths they can take that would look bizarre (we don't want them to, for example, walk directly next to walls, as we discussed above). In general though, this abstraction is just trying to capture "walk like a normal human would walk in a given area, but go to the correct place, quickly." We have the same goal here too -- move between two locations with the smallest "cost", which is roughly equivalent to "time" barring expensive tiles to walk on.

> We went down many rabbitholes early in development to try to make pathfinding "more interesting" for certain NPCs. For example, Dell, one of the children in the game, would *prefer* paths with a higher cost, reflecting her "off-roading" nature. However, in practice, rather than just walk next to paths, through dirt like the dirt kid she is, like we wanted her to, she mostly just took very circuitious routes that confused us as we watched. Even more damming, though? No one really noticed in playtesting that her behavior was weird. This problem will be explored in the next post in this series, but this problem is about *legibility* of NPC behavior -- players rarely notice an NPC's "weird" ambient behavior, since players are focused on other behavior. Ultimately, it was simpler for everyone if NPCs minimized "cost" rather than some other metric.

We answer our question about Eiland's path, then, with the obvious answer: he should take the quickest path between two points. But we are still stymied here: he has multiple map paths he could choose to go between to get to this ultimate destination. How do we find the best path?

The first idea that might come to mind is to simply do the pathfinding across all of the rooms as one giant mega grid. Each grid *already* does its own pathfinding, so there wouldn't be a huge problem with pathfinding all the way from the top-left corner of the world to the bottom-left corner, in theory. In practice, there are absolutely performance limits -- this would ultimately be a huge amount of data to crunch through with many characters per screen. However, if I was starting from scratch and my pathfinding was written fast enough, I'd probably start there.

We ended up using a "waypoint" system instead, which accelerates our pathfinding dramatically. We break each pathfinding operation up into three parts:

1. First, if the start and end are in the same room, we pathfind between them and we're done. Otherwise, continue onwards.
2. We pathfind from the start to every available door in the room.
3. We pathfind from the destination to every available door in that room (more on this in a second).
4. We pathfind from eeach start_door to each end_door, and see which one does that fastest.

Here's the kicker for steps 3 and 4: we can perform both of these at compile time. Who doesn't love doing things at compile time instead of runtime? That's because, essential for this technique, *Mistria* never adds more doors that NPCs can walk through. If we did add support for that, then we'd need to do stage 3 and 4 at runtime instead of compile time.

So what is this mysterious compile time trickery? It's simple -- we pathfind from every single named position in the game to every single door in their room. We then put all this information into a large hashmap which we then lookup at runtime. Since the player can only place furniture in their house and on the farm, there are only a few times where maps get edited (during Festivals for example) which would impact pathfinding. Those impacts are so small that using the cached data is good enough. These named positions also include every door, so when we're done with this first step, which is ran at compile time, we can also pathfind *between every map door to every other map door*. This process is simpler than it appears, but ultimately we do it with A*, but on the domain of *only doors*, rather than the grid itself. We likely could do this pathfinding more efficiently with something like Floyd–Warshall, but offline, reusing the A* stack was good enough.

We call these door-to-door pathfinding operations "waypoints", because we can go from waypoint to waypoint extremely cheaply. In fact, since we can pathfind from any "named" points to every other door, if an NPC is starting a path *at a named point*, which they do for their schedule options, then pathfinding becomes a simple lookup operation. Moreover, since the output of the algorithm's paths are defined at compile time, we can also inspect those paths offline, visualizing how NPCs will travel per room.

There are a few sticking points here before we wrap up this article:

- First, there *are* pathfinding grid edits which can invalidate the cached pathfinding waypoints. When the room the NPC is pathfinding through isn't loaded (such as when the NPC is walking through Town while the player is on the farm), we just use the cached path. We update out of room NPCs at 10x their walking speed, 1/10th the time. That reduces the pressure of out of room NPCs on our CPU time, at the most minor cost that their movement on the map is a bit choppier. When, however, the player *enters* a room which an NPC is pathfinding through already when there has been a pathfinding grid change, we store the NPC's current progress through their *room-path* (ie, the component of the Map Path that's just within one room) in some variable; we then make a brand new path from their starting door to their end position (either another door, or their final end point), and then move them forward on that path by the amount of distance they had previously traveled. This swap does technically mean if any day-of grid edits would *change* a map path enough, we'd have to scrap this whole plan, but that hasn't come up in practice. If it does come up, then we'll likely just have to do more work on runtime, which wouldn't be a big deal.
- I've been referencing "paths" a lot -- in our case, a "path" is a sequence of coordinates which always differ along *one* axis. The A* implementation we wrote produces a list of 8x8 pixel tile coordinates, which we then deduplicate into that "changing axis" instruction set. NPCs then simply walk *towards* the coordinate, and once they meet it, they walk towards the next on the list. If the list is over, they're either at the end of their path, or they need to go to another room. In the former case, they transition to some animation, and in the latter, they walk out of the room and despawn.
- When path invalidation occurs, by the player or the placement of furniture, generally only the local path need to be recalculated. Very rarely is a path *so* invalidated that an NPC completely re-routes to another door, and even if they should, we want to avoid global reasoning for the NPCs, so instead they'll just fix their local path, even if it's slightly slower.
- We ran into some interesting issues generating these paths offline. Almost every room in *Fields of Mistria* can be accessed via *exactly* one other room. For example, if Balor, after giving the player a big smooch on the Farm, wanted to get to his room, he must go through the Town, then the Inn, and then Balor's room. Theoretically, he could go from the Farm to Hayden's Farm, and then to Town (a circuitous and un-optimal route, but a *valid* route in the sense that he will get to his destination), so we must be more particular -- there are two *doorways* which Balor must go through -- the door between the Town and the Inn, and the door between the Inn and Balor's room. We consider, in almost every case, for these doorways to have "zero" distance between them -- fundamentally, they're a zero-dimension liminal space which it's best to not think too much about. However, there are two exceptions: the Narrows and the Eastern Road both have *two* doors going to Town. This duplication is fine, and the system should work perfectly, except for a quirk about the room -- the distance between the doors in the Eastern Road to the town is different than the distance between the two *identical* doors going to the Eastern Road in the Town. This makes more sense visually:

INSERT COMBINED IMAGE OF THE FARM AND THE EASTERN ROAD

As you can see, the doors basically don't line up! (This same issue would arise with every interior if any interior had *two* doors going into it, since effectively every building in *Mistria* is "bigger on the inside" -- the Inn, for example, is about `500px` wide and tall in its own room, but in the Town, the building only takes up ~`200px` in both axes.) The bugs this made were extremely mysterious -- ultimately, NPCs starting in the top of the Town, who were going to the middle of the Eastern Road, would end up preferring to walk all the way through Town and then go into the Eastern Road, rather than just take the "simpler" path directly to the Eastern Road. That's because the two doorways really weren't zero-width -- the northern one was zero-width, and the bottom one was a teleport several hundred pixels up! The solution to this problem was to make that southern doorway have a cost equivalent to the difference in height between the two rooms, thus making the NPCs correctly judge how far they need to travel, but as I think you can guess, many days of debugging and tracing out info was involved in this fine.

Conclusion

This article has been a deep-dive into the micro gameplay of NPCs, particularly around their pathfinding in *Fields of Mistria*, particularly around how we use pathfinding to enforce a kind of verisimilitude for our NPCs. There's more ground to cover here, however, particularly the macro questions about how schedules are generally decided. I will tackle that in a blog post soon!

These practices took us a long time to develop, and we went through many iterations. If anyone has any questions, feel free to reach out to me via the Contact Me page!