Sprint 3 -- 10 day Duration (Ends 7th of April. 2017) 60 hours of work

Basic Design

The active design is another attempt at an implementation of Paper Prototype 2.2, trying to finish off the work of the two sprints that came before it. Nothing is substantially different about its parameters.

Risks and Mitigations

1. Partially complete sections of the code might not result in player-interactable sets of behaviour.
   a) I'll divide the work I have into a set of independent mechanical features, behaviours that I think represent a meaningful piece of the actual gameplay and don't rely on anything that hasn't already been created to function properly. I will then focus on finishing the entirety of one of these independent sets before moving onto another one. That way, once I've completed a set, I'll have some amount of presentable behaviour, which will make partial successes still produce something of a viable prototype.
2. I might not have a working prototype by the end of the sprint.
   a) I'm going to try and intentionally lowball the amount of work I'm capable of doing, focusing entirely on finishing the tasks defined in Sprint 2, and those that follow immediately from it. I will endeavor to define more of the non-critical behaviour (the implementation of unit tests may be one example) as stretch goals, so to speak, rather than critical functionality.
3. My existing suite of unit tests might not reflect the required behaviours of the module they're testing, especially when it comes to things facing the underlying engine (transform manipulation, parenting, etc).
   a) I think the best way of addressing this problem is to build unit tests that correct for bugs or missing behaviour as they're encountered. Rather than trying to pre-imagine everything I might need a module to do, I should keep a living list of tests that's in a continual state of being defined and implemented. Upon discovering a missing behaviour, I'll define a descriptive name in the appropriate testing class but not implement it, saving that task for a refactor period on that part of the codebase, or for a more dedicated task.
4. There might be some hidden amount of complexity that I hadn't considered that will be necessary to deploy some important part of the prototype.
   a) I'll try and lowball the amount of work I've set myself up to do, maybe undercutting myself by as much as 20 hours. At the very least, I'll do what I can to define a small number of critical tasks and have sensible task priority, so that I have more spare time to address unforeseen consequences.
5. I might spend more time trying to prove an implementation with unit tests than those proofs are worth.
   a) To resolve this issue, I'll try a new way of doing unit tests. I still like setting out the tests when designing a new module, as that lets me reason about what the module needs to do before I begin coding it, and makes YNGNI easier to follow (as all I need to do is greenlight the tests). However, I'll try and implement only the tests that I think I need. When I come up with new unit tests, I will not try to immediately implement them, instead saving those implementations for when I think I can afford to perform them.
6. I might spend more hours senselessly fighting Unity's serialization nightmares, eating up productive hours.
   a) This will be a recurring issue as long as I use this engine, I think. So far, the best way I've developed of circumventing this shortcoming is to define pretty much all logical units as

MonoBehaviours and ignore the serialization issue altogether. That does create issues for modules that use dictionaries, but it's about the best technique I've got at the moment. Resolving this issue is going to require quick identification of what exactly needs to be serialized and a committment to using the MonoBehaviour solution as quickly and completely as I can.

7. I might poorly prioritize my work, spending too much time on tasks that aren't strictly critical and not enough on those that are.
   a) I'm going to make sure not just to enumerate tasks, but to prioritize them. I'm fairly certain there are schemas for this sort of task prioritization, and if not I can easily come up with my own. Something along the lines of Critical, Important, and Desirable. Once I've prioritized my tasks in that manner, I will focus on executing the critical tasks first, then the important ones, then the desirable ones. By carefully organizing the Sprint Backlog in these terms, I should be able to address the important issues first.
8. The code I've built for handling the construction of various game entities during runtime might not extend well to constructing them at design time.
   a) A lot of this is a serialization issue, or a matter of fiddling around with Editor scripts. It's also not a critical task at the moment. I shouldn't focus on building editor tools (beyond what's absolutely necessary) until I've got the prototype running and have decided that it's a decent enough base to perform further iterations on it.
9. I might not have good coverage with my various tests, and that lack of coverage might cause me to forget necessary behaviour before I manage to cover it.
   a) Again, the best way of addressing this, I think, is to populate the requirements as I go. If bugs appear, I can create unit tests that address those bugs, or at least define their names. I think it's a fool's errand to go back through all of my tests, trying to fill in the testing gaps when I might not have a good idea of what those gaps are.
10. I might not have a good idea of what's critical, what's important, what's useful, and what's merely nice to have.
    a) I will devote a considerable amount of time in my preparation to enumerating all the tasks I have left and deciding how important they are. On the chopping block are things like Highway Upgraders, editor scripts, map persistence (which is barely even a desirable trait at this tmie) and missing unit tests.
11. UIControl might become a tangled mess of nonsense that's hard to debug, and separating it out into components might be more trouble than it's worth.
    a) I can always perform halfway functions, dividing methods like PushPointerClick into separate handling methods. That makes things easier to refactor more completely later down the line, using things like Chain of Responsibility. But that is not a priority at this time.
12. Some of the behaviour that's less well-specified might be fiddly and irritating to test and debug.

Sprint Backlog

To Do

Critical

1.

Important

1.

Desirable (8 hours remaining)

1. Extend ConstructionZone creation into the editor.
    a) Estimated 5 hours to complete.
2. Extend Highway construction into the editor.
    a) Estimated 3 hours to complete.

In Progress

1.

Completed

1. [Critical] Get the highway priority and permission UI to properly display and modify a highway's state.
   a) Estimated 2 hours to complete.
   b) Took ~2 hours and 40 minutes to complete.
2. [Critical] Debug ResourceDepot creation and state.
   a) Estimated 2 hours to complete.
   b) Took ~ 40 minutes to complete.
3. [Critical] Finish debugging highway pulling and pushing.
   a) Estmiated 3 hours to complete.
   b) Took ~ 1 1/2 hours to complete.
4. [Critical] Change the ResourceTypes, ComplexityDefinitions, and ComplexityLadders to conform with the parameters of Paper Prototype 2.2.
   a) Estimated 30 minutes to complete.
   b) Took ~ 11 minutes to complete, but necessarily could not implement a set of desirable behaviour (Empty Land).
5. Construct and integrate a SocietySummaryDisplay module.
   a) Design the UI proper.
      - Estimated 1 hour to complete.
      - Took ~30 minutes to complete
   b) Design the code to control the UI.
      - Estimated 30 minutes to complete.
      - Took ~1 hour to complete, though bled into subtask C.
   c) Integrate the necessary eventing code into Society and UIControl.
      - Estimated 1 hour to complete.
      - Took ~ 30 minutes to complete, though bled into subtask B.
6. [Important] Resolve the position and scaling bugs that occur when ResourceBlobs move through BlobSites and BlobHighways.
   a) Fix the scaling issues so that ResourceBlobs never change size.
      - Estimated 30 minutes to complete.
      - Was completed incidentally while resolving other tasks.
   b) Fix the in-Highway position bugs so that ResourceBlobs always travel down the tubes they are contained by.
      - Estimated 1 hour to complete.
      - Took ~1 hour to complete, though it overlapped with unanticipated tasks.

c) Return BlobAlignment strategies to BlobSites, so BlobSite contents are distributed in space.
   - Estimated 1 hour to complete.
   - Took ~30 minutes.

d) There was a decent amount of unanticipated debugging work that weaved its way through most subtasks but was otherwise unrelated to them.

7. [Critical] Debug Society production cycles, consumption cycles, ascent conditions, and descent conditions until they conform as close as they can to Paper Prototype 2.2.
   a) Estimated 2 hours to complete.
   b) Took ~2 1/2 to 3 hours to complete.
8. [Critical] Permit manual changes of a society's position on the complexity ladder in the editor.
   a) Estimated 2 hours to complete.
   b) Took ~10 minutes to complete.
9. [Important] Debug the creation and destruction of ConstructionZones.
   a) Get the ConstructionPanel UI to correctly construct ResourceDepot ConstructionZones.
      - Estimated 30 minutes to complete.
   b) Get the ConstructionZoneSummaryDisplay to properly destroy its ConstructionZone when requested.
      - Estimated 30 minutes to complete.
   c) Debug the creation of ResourceDepots via ConstructionZones.
      - Estimated 30 minutes to complete.
   d) Whole task took ~40 minutes to complete, with no clear separation between the subtasks.
10. [Important] Establish the logic necessary to handle Empty Land conversion.
    a) Estimated 5 hours to complete.
    b) Took ~ 2 1/2 to 3 hours to complete.
11. Integrate HighwayUpgrading into the game proper.
    a) Estimated 2 1/2 hours to complete.
    b) Took 2 1/2 hours to complete.
12. [Important] Repair, augment, and greenlight all of the broken unit tests.
    a) Estimated 8 hours to complete.
    b) Took 6 1/2 hours to complete.
13. Change ConstructionZone creation so that it displays costs in the ConstructionPanel and resources left to acquire in the ConstructionZoneSummaryDisplay.
    a) Estimated 2 hours to complete.
    b) Took 1 1/2 hours to complete.

14. Debug the issues that emerged from greenlighting all of the unit tests.
    a) Estimated 30 minutes to 1 hour to complete.
    b) Took ~30 minutes to complete (though there may be incipient bugs awaiting me).
15. Improve the BlobHighwaySummaryDisplay to disambigulate which BlobHighway is selected and which endpoint is which.
    a) Estimated 1 hour to complete.
    b) Took 1 hour and 10 minutes to complete.
16. Improve the ConstructionPanelUI to properly disambigulate which MapNode is being selected.
    a) Estimated 30 minutes to complete.
17. Improve the ConstructionZoneSummaryDisplay to properly disambigulate which ConstructionZone is being selected.
    a) Estimated 30 minutes to complete.

Abandoned

Sprint Review

I finally managed to create a full implementation of Digital Prototype 2.2. There are no evident bugs in any of the necessary mechanical behaviour, though the user experience is marginal to say the least. I ended the sprint with only two desirable tasks in the backlog, both of which I ultimately considered to be so tangential to the prototype that I did not assign any time to them, essentially leaving them for a future version of the prototype.

Even with somewhat limited playtesting, I'm beginning to uncover shortcomings and unexpected consequences of my chosen design. Particularly, I'm finding the logic of city creation to be fairly complex and hard to reason about. It also seems like the permission and priority system upon the highways isn't expressive enough. I've encountered some desire to have a high-priority highway that only provides resources for a limited period of time. I've also found that it's quite easy to accidentally ascend villages into towns, which can cause serious disruption.

I consider this implementation to be thorough enough to be a good base for future iteration. I already have ideas for changes and a fairly solid understanding of how I'd end up implementing them.

Sprint Retrospective

What went well?

- I managed to deploy a complete and playable product that meets the requirements of Paper Prototype 2.2. There are some issues with the UI, but the core functionality is all there.
- I successfully prioritized tasks and used those prioritizations to organize my work effectively.
- I focused on individual units of player interactivity rather than the full task, which allowed me to incrementally deploy interactable segments of the prototype and avoid the pitfalls of Sprints 1 and 2.
- I ended the sprint with a fairly complete set of regression tests, all of which were greenlit, which means that I'm continuing to maintain my testing apparatus and continuing to check for errors as a result of implementation changes.
- I wasted very little time pursuing unnecessary ventures, falling down rabbit holes within which were no meaningful results. Nearly the whole time I was programming, I was applied to a specific, important task and making progress in that task.
- I was able to, through careful analysis and application of my debugging tools, resolve several fairly challenging bugs in a reasonable amount of time. And I managed to do this without getting frustrated or disheartened.
- I fought Unity very little this sprint, instead using its logic and some of its features to create quick solutions to certain problems. Of notable accomplishment was how I used the EventSystem's select event to intelligently open and close various summary displays and panels in a way that interacted well with buttons. I could've designed a complex nest of logic complete with raycasts to detect when the mouse was or was not over a proper panel, but instead I used existing tools to come up with a reasonably robust solution that was not over-engineererd for the task at hand.
- I managed to make several important changes to the types and ways of information tranfer

between game logic and UI, but without compromising the strong decoupling between them. Despite the sometimes substantial changes I had to make, the UI and the game logic still act as completely independent layers that interact only through the Core modules UIControl and SimulationControl. Even more heartening, this division proved to be a very effective one, and it was not challenging to maintain the disconnect I'd established.

What could be improved?

- Though I logged a full 6 hours of work everyday, not every one of those hours was programming work. In fact, I very often opted to call programming work closed for the day as tasks were completed, feeling that I had plenty of time to finish the work arrayed before me. This ended up being quite true, as the sprint was successful, but it's a manifestation of a bad habit, of a mild form of procastination that in a more challenging context would've proven very dangerous. I need to build my work discipline so that I devote fully to at least 6 hour work days, even if I'm fairly certain I underbugeted time. It provides me no benefit and is of great risk if unintended circumstances come up. I might as well do the full six hours, and then finish all of the sprint tasks early and have nothing to do.
- I stepped away from TDD a bit, letting my regression tests redlight for considerable periods of time and building implementations without tests to guide them. I'm not sure if that is a problem for prototype-level code, as it is supposed to be somewhat ad hoc, or if retroactive unit tests are acceptable. It certainly would've been frustrating to pre-test for the problems I was trying to solve, but those problems might've arisen from poor TDD practices in the first place. I'll need to keep an eye on this development and think harder about what purpose tests serve and when.
- I don't know how much of my codebase is covered by tests. There are certain modules, like ResourceBlob, that I know have no tests associated with them, and I feel like there are several gaps in other modules, but I don't know for sure. Taking inventory of all my tests seems like a very unpleasant experience, but a critical one if I'm to make use of them. I need to figure out how to organize and structure my tests, so I can know more easily whether or not my tests cover the codebase sufficiently.
- This sprint was probably underbudgeted. Though it's hard for me to say how many additional tasks I could've added for this sprint, I need to make sure not to underestimate the amount of work I can do, and not to severely underbudget sprints in the future. Ideally every sprint should entail about 60 hours of productive work, and while the deadline shouldn't feel oppressive, it also shouldn't feel so loose that I can afford to cut work short some days.
- There is a fair amount of repetitive code in the UI that needs to be collated into a single class, but which I decided not to for reasons of haste and priorities. Collecting that code will most certainly be a task on the sprint backlog for Sprint 4.
- I still haven't playtested the game with other people. Given its current functionality, I've no excuse not to do so. I need to get better at testing early work upon other people, so that I can see what problems are emerging that I'd never even considered, and to make sure that it's doing what I think it's doing, which it almost certainly is not.