Netspeak Games Coding Test

Progress note

Toolset

- Unreal Engine version: 4.20.x
    - Starting template: Topdown.
- Visual Studio 2017, with VAX.
- Git Bash

Daily notes

Thursday, 20th June 2019

- Morning: Got the doc from Callum.
- 15.00: Had some free time at work, decided to setup the project on GitHub private. My approach was starting from Item 1 (Farming) with single player mode to get myself familiar with UE4 - from what I'd read earlier about the UE4's multiplayer architecture, it should be easy to switch from single to multi.
- 15.15: Things got set up nicely. Time to commit.
- 15.40: Got another bunch of free time at work. I'm gonna get rid of mouse-based navigation and switch to arrow key-based navigation. Assuming currently I'll only let gamers interact with the map with keyboard. Mouse interaction, such as with cell states, will be in future work category.
- 21.00: Finished dinner.
    - Next 2 hours plan:
        - Design for bullet #1: *Programatically generate a grid of slots and have slot states* and #2 *Have each slot be able to transition from sensible farm based states*.
        - Make a cube mesh to mark the "selected" slot for the character.
    - Design decision:
        - Obviously, we can save a lot of space by not storing empty cells.
        - State machine fits quite well with the situation: I can control the flows between states easily and clearly. Gotta read about UE4's approach on interfaces though. I plan to use something like an ISlot C++ interface with generic functions like get appearance info or switch to another state. Specific classes can derive from it, ideally with blueprint so that designers can change some details, such as state transition.
    - Player movement: I'll have a cube mesh to mark the player's selected slot. Should be easy, but I'll do it in C++ as it involves some vector athrimetic. I'm not comfortable doing so with BP.
        - I tried to have a scene component attached to the player actor, but it'd be easier to have a separate actor just follow the player actor. This would help in multiplayer as well, because this is trivial and can easily be interpolated from player's position - no need to replicate at all.
        - Not sure why when I use this code, the player stops moving. Will investigate tomorrow: `if (Target) SetActorLocation(Target->GetActorLocation());`
    - *Side notes*: Currently I'm grouping source code files into task folder, eg Farming, Moving…, because that's the way I work in Unity. If this is not a recommended folder structure, please make a comment and I appreciate that :)

Friday, 21st June 2019

- 14.30: Had an hour of free time at work after the lunch break. Decided to continue working on the marker mechanics. The bug turned out to be the physics settings blocks player's movement. Disable collision of marker actor in C++ constructor then things are fine.
    - Got a moment of confusion in debugging because instructions keep running around (*randomly*?), not sequentially as I expected. Local variable values are also messed up => That was caused by binary optimization, switched to *DebugGame Editor* instead of *Development Editor* makes the debugger works as expected. Thank you StackOverflow!
    - The basic idea for the mechanics:
        - I use player forward direction to decide which slot to put the marker.
        - A static const array in the cpp file is used to store allowed direction (currently just Forward/Backward and Left/Right, but we can easily add other directions if the situation needs, while still being by the vector athrimetic).
        - Dot product operation is used to see which direction is "closest" to where the player is heading at right now.
        - Things still don't work correctly with negative direction cases, gonna check them out this evening.
- 18.00: Finished work but staying at the office for a while. Gotta fix that marker placement first.
- 18.15: OK there was a flaw in my conversion back and forth between world position to integer-based slot coordinate. Fixed it.
- 18.25: commited the result. Now is the time to take it another step further: cell interaction. Every state of a slot can be switched to *only* one another - except from "soil" back to "empty". Since a soil slot has no expense maintaining it, I assume that a slot, once is soiled, cannot be "unsoiled" (I don't even know if this word exist, I just made that up). So the ISlot interface should have one function to react to player input (only one key will do). To make things clear for gamers, when player approach the slot, there should be a GUI element displayed to announce about what happens if he/she presses the interact button. Imagine the similar situation in Assassins' Creed, when the main character step on a bag of coin, there would be a GUI element saying things like "(Y) Pick up". So that means I have two tasks ahead, hopefully I can finish this within 1 hour before leaving the office and heading home.
    - Continue setting up the ISlot interface with switching and interacting function. An ISlot variable, which should be editable in BP, is required as well. Designers can change this to any next state they want, without messing up any logical flows. Normally when I work with Unity, game designers are not good at thinking in logical order, so I prefer to give him limited choices that they can select, compared to let them modify the flow directly (even in BP).
    - Setup a GUI element for showing the purpose of the next action.
- 20.00: The interface approach turned out to be useless. I switched into a base UObject C++ class, called SlotHandler (but it should be SlotStateHandler), with a BP-exposed variable to select next state. This class can contain other information such as state color and GUI message. I mean to use it as a configurable data container for a SlotActor, which would contain a pointer to SlotHandler, indicating its current states and process to next state based on time or input. I'll take time to think about this tomorrow.

Saturday, 22nd June 2019

- 10.00: Today I'm gonna finish the interaction part of they player with the world. Yesterday I left off with a SlotHander UObject, so that would be where I'm gonna pick up today.
- 10.40: The player needs to know whether the slot he is facing has any actor on it, so I move the position/coordinate conversion to a utility class.
- 13.40: Took me more than two hours to make the interaction works as intended

    - A bug in UE caused newly added UPROPERTYs has blank detail panels - I thought that was my fault and spent my time tweaking UPROPERTY's options.
    - UE also has many different functions for purposes with some slight difference. I kept switching back and forth between Development and Debug mode to see why when I try to change the material of the slot, UE crashes. It seems there is a difference between `Mesh->GetMaterial()` and `Mesh->CreateAndSetMaterialInstanceDynamic`. I had even ignored the latter, with the assumption that when I have a material correctly assigned, why do I need to create another one?
    - Anyways, now the character can run around, with a marker indicating the closest slot in front of him, and can soil an empty slot. Next work would be allowing him to change the slot from soil to other states, which should be easy with current code base. Also, time to play with some HUD stuff, so that player can have an indicator on the slot telling him if he presses Enter, which would happen.
    - But I'm gonna take a short break.
- 17.00: Back from break.
- 18.30: Took me more than an hour to figure out how to make a UI widget, and how to make it follow the actor in world space. Luckily on math involved, UE has a few handy functions for that. Next I'd make the widget indicates next state of any slot, event empty ones. Off the top of my head it seems I should make another helper/utilities function for this, because I'd already have the same thing when player hits Enter key to spawn new cell/switch to next state.
- 19.20: Done with the widget's content. Now it would display what the gamer should expect when pressing Enter.
- 19.50: Done adding remaining states and experiment switching between them. Next step would be add code for handling time-based transition (plant to harvest ready), and starting to tackle multiplayer.

Sunday, 23rd June 2019

- 9.30: Get up and start working. Turned on Dedicated server.
- 12.00: Game crashed when just fire Play button. The call stack and debugger showed that there is something wrong with a reference somewhere, turned out Game Mode object cannot be accessed on. A couple of minutes wandering around on Google leads me to the great PDF called UE4 Networking Compendium. Great, so I have to switch SlotSize config variable to a Game State class. Really got confused though. When my game state derives from AGameState, the character's BeginPlay event isn't even fired. Took me near one hour to figure out I should have it derives from AGameStateBase. But then, the SlotSize value is not deterministic every time, which leads to an inconsistent game result. One time it'd be zero, another time it was 27 and another try gives me the result of 84!!?!! Not sure if I've done anything wrong with the setup. Gonna take a rest and comeback later.
- 14.00: Got back from break. Gotta tackle the bug.
- 15.00: Looks like that I found the solution: From the hint that in World Outliner, there seems to be something wrong with the initialization process. I spot that in the custom game mode there is also some code for setting default Pawn class, so I think it can be done the same with Game State class. Increased player number to two, and things seems to be working fine except those things:
    - Non empty slots don't get replicated.
    - Some runtime error on slot marker blueprint.
    - There are two HUD elements on each client, where should be only one for the local player.
- 20.30: Solved most of the networking problems
    - Spawning Slot Actors on server and replicating them to clients
    - Use RPCs for spawning actor (aka Empty->Non Empty slot) and transition to next state.
    - However, UObject replication is frustrating. I've been searching around for hours, asking on dev Discord, but can't find out what's going wrong. I'm gonna take a rest now and come back to solve it tomorrow with a fresh mind.

Tuesday, 25th June 2019

- 10.00: OK, so UObject might or might not work with replication. I reckon something with UE's garbage collector got into the way so that its value gets reset into NULL, so I might benefit from value-type variables. Given that I only need a FText and a FColor to render slot's information in client side, I don't need to replicate the whole UObject at all. Let's put those UObject work to the server, which is safer, and just replicate visual-related variables. So that seems to work well. The only problems remaining are:
  - The initial color of a slot is not correct, even though the debugger showing the color variable get replicated *after* server's BeginPlay().
  - The server needs to handle time-based state as well, for transition from Seed to Plant, before the player can harvest it.
- 20.00: Back from work and just finished dinner. Let's put this to the end. Solve the initial color bug first.
- 20.20: Fixed it. The last item on the list is time-based states.
- 21.00: Done with time-based states. The current approach would be the server take care about the actual timer of the state - this, in theory, can prevent cheating. Clients also have reference values, so that players can know how much time is he away from harvesting a slot.

Closing thoughts

- Switching from Unity to Unreal architecture is quite hard, and it took me longer than expected to get used to the new approach to problems.
- I really like the built-in authorative server support in UE4 - which took me a huge effort to achieve with Photon and Unity (an incomplete and buggy version, it's shamed to admit, but truths are truths).
- My college days just come flooding back to me when working with C++. I've had a lot of fun debugging crashes.
- This is a list of my assumptions when working in this item:
  - Gamers interact with the map with keyboard only.
  - A slot, once is soiled, cannot be "unsoiled".
  - In multiplayer mode, players should be able to observe the farms of other players in realtime, which includes how many tiles of each state does his friend has, and how long does it take him to wait to loot his friend's plant (or help him to harvest).
- Total time taken in this test: roughly 22-24 hours: two weekends, eight hours each and three weekday evenings, two to three hours each.
- And last but not least, I find working with UE4 very interesting and eye-opening. Hopefully, I can come to work with you guys and learn more about how to get the best out of this engine!