Modder Primer for 1.6

General

- Unity has been updated from 2019.4.30 -> 2022.3.35. We haven't noticed many issues besides a couple of minor UI bugs to do with input field focus.

Variable Tick Rate (VTR)

- This is a new system which allows **`Things`** in the game to easily run logic using a delta based paradigm, rather than running all their logic every tick. **`Things`** which aren't spawned, or are off map, or when the camera is zoomed out will reduce their tickrate from 60Hz down to 4Hz (1 tick every 15 ticks)

- **`Job`**, **`ThingComp`**, **`Thing`**, etc. all have new methods to hook into the VTR system, usually with the signature **`TickInterval(int delta)`**. You can use these methods to add code which runs on a delta basis. Delta refers to the number of ticks that have elapsed since the method was last called.

- You can override the default minimum and maximum update rates via the virtual properties **`MinTickIntervalRate`** and **`MaxTickIntervalRate`**, or override **`UpdateRateTicks`** to use custom logic to determine the current tickrate for something.

- Animals for example are an exception which use a constant fixed tick delta of 15 ticks, so their health, needs, and so on only update every 15 ticks, and all time based values are multiplied by the delta, in this case 15.

- In addition to the **`TickInterval`** method, **`Things`** still have a **`Tick`** method which runs every tick. Due to RimWorld's nature a lot of code is just required to run every tick (e.g. path following) so the two systems have been designed to be able to coexist. Where possible, however, consider changing logic to be delta-based, as this is better for performance.

Game setup

- Game setup has been broken out into **`GameSetupStepDef`**s, these are game wide and general setup steps that aren't attached to any specific planet layer.

- Note that "game setup" here refers largely to the process of ensuring a World is ready for use by other systems, both when creating a new game and when loading one from a save file.

Float menu changes

- The enormous methods in **`FloatMenuMakerMap`** have been refactored out into **`FloatMenuOptionProviders`**. These are each wrapped in a try catch so individual options will no longer break the entire float menu.

- You can override several properties to determine when the float menu should appear, as well as whether the target is valid, and several overridable convenience methods to get options for single or multiple things/pawns.

Planet layers

- Planet layers are a new system which allows the world map to have multiple layers of tiled locations. The planet as it previously existed is now a new "surface" planet layer. In Odyssey, orbit is another new layer which can coexist with the surface.

- Planet layers are broken down into two main parts: their data, which is defined in a **`PlanetLayerDef`**, and their behaviour, which exists in a **`PlanetLayer`** instance.

    - The **`PlanetLayerDef`** defines generic information about a layer, such as its **`PlanetLayer`** type, gen steps, **`Tile`** type, ability to form caravans, etc.

    - Additionally, you define **`WorldDrawLayers`**, **`WorldGenSteps`**, and **`WorldTabs`** here. These are responsible for generating the layer's content, drawing the layer's components, and the tabs available on that layer.

    - The **`PlanetLayer`** contains all the data and logic required to generate and query the planet layer at runtime.

- Multiple planet layers of the same type can exist at once. They can also be added and removed during gameplay. There is an expectation that at least one surface layer will exist during game creation - this surface layer is considered the "root surface". Any

existing mod code which queries the world grid will use this layer automatically.

- While using the integer-based **`tileId`** is still possible to query the world grid, we highly recommend moving over to the new **`PlanetTile`** data structure. In most cases simply changing the type from an **`int`** is all that's required to support planet layers in your code.

- Planet layers can be created with the scenario part **`ScenPart_PlanetLayerFixed`**, or via scripts at either game setup or running gameplay using either a **`PlanetLayerSettingsDef`** or by passing values explicitly.

    - The **`PlanetLayerSettingsDef`** allows you to define data in xml, and can be reused to simplify creating planet layers. The scenario parts require you to select an existing settings def, this is to prevent too easily breaking the game via the scenario editor.

    - Via script you can add new layers by calling **`WorldGrid.RegisterPlanetLayer`** directly and explicitly setting the **`position`**, **`radius`**, **`viewAngle`**, etc.

- Many def classes, such as **`MentalBreakDef`**s for example, now have fields like **`layerWhitelist`** or **`layerBlacklist`** to be able to configure which planet layers they can interact with.

- As an example, we've included a moon def which will be included in Odyssey's Planet Layers XML file. We don't guarantee the def will work out of the box - but it can be used for reference purposes when Odyssey is available.

Terrain grid changes

- Two new layers of terrain have been added: foundations and temporary terrain. These should both work without Odyssey enabled.

    - Foundations include bridges in vanilla. They sit between the top grid and under grid, which allows for things like floors to be placed on bridges.

    - Temporary terrain is not used by vanilla. These sit on top of the top grid. Temp terrain doesn't have any logic performed on it by default, you will need to do so using something like **`SteadyEnvironmentEffects`**, a map component, or an ethereal thing.

- Terrain can now be a light source or a heat pusher through **`glowRadius`** and **`heatPerTick`** respectively.

- Terrain can create flecks using **`throwFleckChance`** and **`fleckData`**

Map generation

- UsedRects is a new system we have added to ensure that important map features don't overlap one another, you can access the list using this line during map gen: **`MapGenerator.GetOrGenerateVar<List<CellRect>>(MapGenerator.UsedRectsName);`**
    - There is a debug setting to draw used rects on the map after generation

- Gen step order has been revisited to be more consistent and structured, and several steps have been moved around. The general order you should follow is:

10-100	Underlying grids (elevation/fertility/caves)
200-300	Base natural terrain gen (rocks, terrain etc.)
400-500	Critical large structures (quest locations, settlements, etc.) These should add to UsedRects
600	Reserve gravship area
700-800	Non-critical large structures (ruins, etc.) These must take into account UsedRects to avoid overlapping with the critical structures/gravship area
850	Player start spot This now chooses a buildable location in the largest contiguous open area (unless we're generating via gravship in which case the gravship location is used)
900-1200	Non-critical gen (geysers, plants, animals, etc)
1500	Fog
1600+	Important buildings/features that need to be placed in unfogged areas (monolith, exostrider etc.) Important to note that if you remove or add any edifices here you will need to update the fog grid manually.
1700	Place gravship marker

- Several aspects of map generation have been moved into tile mutators, which can be considered “features” that are specific to a world tile. For example, rivers, caves, mountains, coasts are all mutators now.
  Instead of just being new **`GenSteps`**, mutators can run at several points across map generation. This allows for shared state across several gen steps - for example the coast mutator reduces elevation in **`GeneratePostElevationFertility`**, but actually affects the terrain in **`GeneratePostTerrain`,** and is able to share the noise across both of these gen steps.

    - Mutators are added to tiles in **`WorldGenStep_Mutators`** (and in Odyssey from Landmarks). **`TileMutatorDef`** has fields to restrict what tiles they can spawn on, as well as ways to prioritise/override/exclude other mutators.

    - Mutator workers are able to tick on maps post generation, but since they are just worker classes they can’t store any state. If you need to store state you will need to create a **`MapComponent`**.

    - Landmarks are bundles of mutators with icons/namers attached. You can mark mutators as “required” by the landmark, and the landmark will ensure it can at least spawn those mutators before determining if it can spawn on the tile.

- We’ve added new debug tools to step through map generation step by gen step. These are “Regenerate Current Map Stepped” and “Do Next Gen Step”

- Several new functions have been added to MapGenUtility for finding clear rects (specifically rects without rocks) on the map. **`GetClearRects`, `TryGetRandomClearRect`, `TryGetLargestClearRect`, `TryGetClosestClearRectTo`**

- Biomes can now define several terrain types to be used by the map generator when generating beaches, water, etc.
    - coastalBeachTerrain
    - lakeBeachTerrain
    - riverbankTerrain
    - mudTerrain
    - gravelTerrain
    - waterShallowTerrain
    - waterDeepTerrain
    - oceanShallowTerrain
    - oceanDeepTerrain
    - waterMovingShallowTerrain

    - waterMovingChestDeepTerrain

Structure layouts

- Structure layouts from Ancient Complexes have been significantly improved, including new utilities for creating structures and rooms, and filling them with various content such as prefabs.

- Fundamentally, structures are a collection of rooms made up of rects. Structures themselves can define structure-wide data which can be used in all rooms.

- Rooms define their details in a **`LayoutRoomDef`**, which contains all the metadata required to generate a room. A single Room (i.e. enclosed indoor space) can contain multiple layout room defs if this is configured to be allowed.

    - While you can create a custom worker type deriving from **`RoomContentsWorker`** to control a room's generation, it's generally better to break your code down into new **`RoomPartDef`**s, which are small blocks of code which can be reused across multiple room defs (e.g. spawning a mech threat, placing a turret in the corner of a room, etc.)

    - Additionally, the base **`RoomContentsWorker`** contains lots of logic to act on layout room metadata to fill the contents of the room easily. This allows you to define most or all of a room's details entirely in XML, such as wall attachments, prefabs, scattered items, flooring, etc.

Prefabs

- Prefabs are a new system which lets you create a collection of things in-game, then export the data to XML for the purposes of spawning those prefabs during gameplay.

- To create a prefab, use the debug action **`Generation/Create Prefab`** and the data will be copied to your clipboard. You can then paste this directly into an XML file.

    - In addition to the generated data, you can set extra values in XML to change how the prefab will spawn, such as part spawn chance, HP, valid rotations, etc.

    - By default the tool will only copy buildings, you can override this by changing the

debug generation setting **`prefabCopyAllThings`**.

    - By default the tool will also not copy terrain, you can also override this with the debug generation setting **`prefabCopyTerrain`**.

    - Creating a prefab will also store it in a buffer which can be placed via the **`Generation/Create Prefab/Buffer`** action.

- While we don't utilize it in-game, prefabs can be optionally spawned as blueprints. Currently this exists for modders to take advantage of.

Pathfinding refactor

- The pathfinder was completely rewritten; it is now batched, multithreaded, and using low level optimised bytecode via Unity's jobs/burst technologies.

- While it is possible to query a path immediately/synchronously, we highly recommend against it, especially in hot paths. Where possible, you should instead submit a **`PathRequest`**, which will return the path result once it has been computed. In most situations the result should be ready after at most one tick.

- Map data is now batched together, so pawns which have similar pathfinding parameters can reuse the same computed data. Likewise, map data will be reused for as long as the map state hasn't changed, reducing required work significantly.

    - To notify the map that a **`Thing`** has some additional cost-addition, you can implement the new **`IPathFindCostProvider`** interface.

- As a result of these optimizations, we have *significantly* reduced the impact of the heuristic factor, paths will now almost always be perfectly optimal.

Lighting grid refactor

- The **`GlowGrid`** system was completely rewritten in a similar manner to the pathfinder. It is now entirely multithreaded, and uses the same low level optimised bytecode jobs/burst technologies.

- As a result, the number of glowers which it is feasible to have on a map has increased greatly, to the point that in the vast majority of situations, they are no longer a

performance concern.

Mutant refactor

- Due to the nature of Anomaly's development, shamblers and ghouls were reworked several times, and by the time release came around, the mutant system wasn't in a super extendable state, and we couldn't risk a rewrite without causing many bugs close to release.

- For 1.6 we've rewritten how mutants work. A mutant with a **`MutantDef`** that contains default values will now behave exactly like a colonist; all behaviour disabling is opt in. We've removed most **`IsMutant`** checks, and added fields to **`MutantDef`** to determine whether to disable the behaviour. This means that it will be much easier to create custom mutants.

- In essence, a mutant is simply a colonist which can have most kinds of colonist behaviour disabled through the def. There is also some addable behaviour - you can add abilities/tools/verbs to the mutant, as well as custom graphics.

- The one gotcha is that due to the huge amount of **`IsMutant`** checks in 1.5, it wasn't feasible to add a field for every single minor tweak. These are now handled via an **`IsSubhuman`** check instead, which can essentially be considered a catchall for whether the pawn should be treated as a colonist by the game. Since **`IsMutant`** and **`IsSubhuman`** are now distinct, the mutant system can now be used to create mutants which are still considered colonists by the game.

Designator draw styles

- Draw styles are a new addition which allow for shapes other than a rectangle for the purposes of designating an area. They include the existing filled rectangle, ovals/circles, lines, etc.

- To add a new drawstyle create a new **`DrawStyleDef`**, here you can define your worker which will update the target buffer with what cells should be selected.

    - Draw styles by default allow the user to draw a box, however you can override the property **`SingleCell`** to change the behaviour of the dragger to instead select a single cell. This would be useful if you wanted to add a click fill-area

draw style.

- Draw style categories define what draw styles are used for certain actions, designators define which category they will use (i.e. **`Default2D`**, **`Fill2D`**, **`Default1D`**, ect.)

    - You will notice we have lots of Draw Style Categories which don't define any data and just inherit from a base style, these exist so that when a player selects a designator with the same category they will automatically select the previous style they were using for that category, if they have the setting "Remember draw styles" enabled

Asset bundles

- Bundles now respect load folders the same as other assets do.

- Bundles can have platform-specific suffixes, **`_win`**, **`_mac`**, or **`_linux`**, at the end of their filename, which will cause them to only be loaded if the game is running on that platform.

    - Bundles without a suffix are loaded on all platforms.

    - This is helpful for custom shaders, because compiled shaders are platform-dependent.

- Asset paths in bundles can now have the form **`assets/data/[packageid]/...`** instead of **`assets/data/[foldername]/...`** so that the same bundle works regardless of if the mod was installed via the workshop or standalone distribution.

    - The old format will still work, for backwards compatibility.

- Shaders can now be loaded from bundles by the game automatically. They must be in a **`Materials`** subfolder of your bundle's main mod folder (see above), and the path relative to that **`Materials`** folder is the path which must be provided in **`ShaderTypeDef`**s (or directly in calls to **`LoadShader`**).

- If you are building your own asset bundles for your mod, you should consider using **`BuildAssetBundleOptions.ChunkBasedCompression`** in your build script. This will reduce load times, at the cost of a modest increase in file size.

    - Without chunk-based compression, the default is LZMA, which, while slightly more efficient than chunk-based LZ4 compression, is slow to decompress and so

unity must decompress the entire bundle into memory ahead of assets being used, which is done during the game load (before the main menu is shown). This increases both load times and memory usage.

- As a reminder, for assets in bundles to be found by the game automatically, they must have one of a fixed list of file extensions:
    - For shaders, the only acceptable extension is .shader
    - For audio clips, .wav, .mp3, and .ogg are allowed.
    - For textures, .png, .psd, .jpg, and .jpeg are allowed.

- Please note: Due to Unity limitations, mod asset bundles must have a unique file name at the time they are built. Renaming the file afterwards **will not work**. Attempting to load two or more bundles (even from two or more different mods) which were built with the same name will result in all bundles after the first failing to load “because another AssetBundle with the same files is already loaded”.

Miscellaneous

- Pawns now cache which lord they are a part of, making **`GetLord`** almost free.

- Improved performance of **`ThingWithComps`** - specifically, getting a comp (or checking if it is present, etc) should be more performant if the comp is present, or if it is not present and the comp class is sealed, or for things with no comps.
    - In the case that the requested comp is not present and the comp type is not sealed (but there are other comps on the Thing), or a comp of the requested type is not directly present, but a comp of a derived type is, the performance remains approximately the same as it was in 1.5. Ideally these cases will be avoided in hot paths.

- By default, when a Thing or WorldObject which implements **`IThingHolder`** ticks, it will now also automatically tick its contents. This behavior can be disabled by implementing **`IThingHolderTickable`** and returning false from **`ShouldTickContents`**.
    - Returning false will change the behavior back to how it was in 1.5.

- The Pawn Render system will automatically load required graphics when playing animations, removing the need to add code for dynamic graphics.

- Added **`FastTileFinder`**, which maintains a cache of basic information about tiles on the World Map, and allows for near-instant querying to locate a planet tile matching a set of configurable parameters.

- Weather can now limit max range on all ranged abilities. This includes ranged weapons, psycasts, other verb-based abilities, and royal aid. It also includes non-pawn casters, like turrets.
    - Note that if your mod has a custom implementation of **`Verb#EffectiveRange`**, you may need to make changes to ensure this max range limit is applied correctly. Calls to **`VerbProperties.AdjustedRange(Verb, Thing)`** will automatically apply the cap from the provided **`Thing`**'s Map.
    - If you can't use the above method, or need to apply the cap to something other than a **`Verb`**, then you will need to manually apply **`WeatherManager#CurWeatherMaxRangeCap`**
        - This is -1 if there is no cap.

- Mechanoid overlay coloring was abstracted into **`Faction.AlliganceColor`**, so mask coloring by faction isn't mechanoid specific.

- Once the weather on a Map changes and finishes lerping to the new weather, the **`WeatherManager`** will no longer call **`WeatherTick`** on the old weather. In 1.5, **`lastWeather`** would continue to receive ticks with a **`lerpFactor`** of 0 until the weather changed for a second time (and so **`lastWeather`** was replaced). In 1.6, once that reaches 0, the tick method will no longer be called.

- Animals can cross-breed with other species, if configured in XML. Currently this is only used in one situation in Odyssey.
    - E.g.

```
<canCrossBreedWith>
 <li>Thrumbo</li>
</canCrossBreedWith>
```

- MayRequire will now work without errors if there's a ParentName from the MayRequire'd mod.

- LoadFolders now support an **`IfModActiveAll`** attribute which functions like **`IfModActive`** except requiring all of the provided mods (comma-separated) instead of any of them (which **`IfModActive`** already does)

- Animals now require **`eggUnfertilizedDef`** even if they are incapable of laying unfertilized eggs (e.g. due to the 50% egg progress limit before they're fertilized), because fertilized eggs turn into unfertilized ones when they are ruined.

- DamageDef now has a **`ignoreShields`** field which can be set to make the damage bypass shield belts.

- You can be notified when a thing is being moved between maps on the Gravship via the new **`PreSwapMap`** and **`PostSwapMap`** callbacks on **`Thing`**

- Top-level XML elements in Def files which do not correspond with any type the game can find, or which don't derive from Def, will now be logged as errors during game startup - capped at one error per type name.
    - So there is no longer any need to use a different def type and override with **`Class="MyCustomDef"`** in order to be notified about types which are missing.