OpenRA: Mods, Actors, Traits

References

[1] OpenRA wiki modding guide:
https://github.com/OpenRA/OpenRA/wiki/Modding-Guide
[2] OpenRA Book:
https://www.openra.net/book/glossary.html
[3] Delft Students On Software Architecture - OpenRA:
https://delftswa.github.io/chapters/openra/

Scope

What is this about?

- Overview of the concept
- How to map to bevy?
- Next step

What is this NOT about?

- Detailed implementation (how mods/traits are imported)

Mods

- “Everything is a mod (including RA - which is loaded by default).” [1]

```
./mods
├── all
├── cnc
├── common
├── d2k
├── modcontent
├── ra
└── ts
```

Mods

"The only file which is ***absolutely required*** for a mod is `mod.yaml`" [1]

- `rules` contains MiniYaml files describing how to assemble actors (units/buildings/etc)
- `maps` contains maps.
- `tilesets` contains MiniYaml files describing the various tilesets -- temperate, snow, etc.
- `chrome` contains MiniYaml files describing the UI chrome
- `uibits` contains various textures used by the chrome
- `bits` contains various loose in-game assets -- SHPs, etc.

```
./mods/ra
├── audio
├── bits
├── chrome
├── chrome.yaml
├── cursors.yaml
├── hotkeys.yaml
├── icon-2x.png
├── icon-3x.png
├── icon.png
├── installer
├── maps
├── metrics.yaml
├── missions.yaml
├── mod.yaml
├── rules
├── sequences
├── tilesets
├── uibits
├── weapons
└── ZoodRangmah.ttf
```

Actors

What are actors?

- “An actor is the entity part of the *entity-component-system*.” [2]
- “All units/structures/most things in the map are Actors. Actors contain a collection of traits.” [1]

Defined in `.yaml` files in `rules` directory

Actor Example

```
# mods/ra/rules/infantry.yaml
DOG:
  Inherits: ^Soldier
  # Some Traits ...
  AttackLeap:
    Voice: Attack
    PauseOnCondition: attacking || attack-cooldown
  # ...
```

“`Inherits` technically isn't a trait, it is a MiniYaml mechanism that is explained in the chapter 2 link above.” [2]

Loading Trait for Actor

```
// OpenRA.Game/GameRules/ActorInfo.cs
namespace OpenRA {
  public class ActorInfo {
    public ActorInfo(ObjectCreator creator, string name, MiniYaml node) {
      // ...
    }

    static TraitInfo LoadTraitInfo(ObjectCreator creator,
      string traitName, MiniYaml my)
    {
      // ...
    }
  }
}
```

Traits

- “Traits consist of an info class and a class that does stuff.” [1]
- “Technically a *trait info* is the *component* part of the *entity-component-system* architecture.” [2]
- “Traits consist of an info class and a class that does stuff.” [1]

BUT there is a catch

Traits

- “There is one instance of the *infoclass* shared across all actors of the same type. Each actor gets its own instance of the *trait class* itself.” [1]
- “*Infoclasses* are responsible for instantiating their corresponding trait class -- see `ITraitInfo`, and `TraitInfo` for the trivial implementation of this. [1]”

TraitInfo

- The *info class* seems like a builder that `Create` a *trait class*

```
public abstract class TraitInfo : ITraitInfoInterface {
  // Value is set using reflection during TraitInfo creation
  [FieldLoader.Ignore]
  public readonly string InstanceName = null;

  public abstract object Create(ActorInitializer init);
}
```

Info Class Example

```
// OpenRA.Mods.Cnc/Traits/Attack/AttackLeap.cs
// namespace OpenRA.Mods.Cnc.Traits
//
// inherits TraitInfo class
[Desc("Move onto the target then execute the attack.")]
public class AttackLeapInfo : AttackFrontalInfo, Requires<MobileInfo> {
  // ...
  public override object Create(ActorInitializer init)
  {
    return new AttackLeap(init.Self, this);
  }
}
```

Trait Class Example

```
// OpeOpenRA.Mods.Cnc/Traits/Attack/AttackLeap.cs
// namespace OpenRA.Mods.Cnc.Traits
public class AttackLeap : AttackFrontal { // inherits Actor class
  // ...
  public override Activity GetAttackActivity(
    Actor self, AttackSource source, in Target newTarget,
    bool allowMove, bool forceAttack, Color? targetLineColor
  ) {
    return new LeapAttack(self, newTarget, allowMove,
      forceAttack, this, info, targetLineColor);
  }
}
```

Activity

- “Things an actor can be doing are represented as Activity subclasses. Actor has a queue of these.” [1]
- Seems like actions between actors. Can probably think of it as `systems` that operate on two or more actors.

```
OpenRA.Mods.Cnc/Activities
├── Infiltrate.cs
├── LayMines.cs
├── LeapAttack.cs
├── Leap.cs
├── Teleport.cs
└── VoxelHarvesterDockSequence.cs
```

Traits and Inheritance

`TraitInfo` is the base class for all info classes

Problem:

- Complexity
- How to decouple?
- Can we reduce the complexity?
- Do we have to recreate the complexity in bevy?

Trait inheritance example: `AttackLeapInfo`, and `AttackLeap`

Thoughts on the Next Step

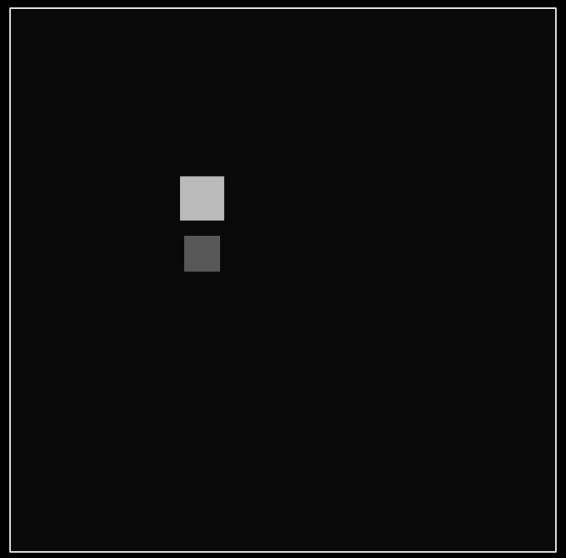
- A deeper dive into the implementation
- A min playable mods

Bevy ECS

bevy_snake as an example

Demo

- GitHub: https://github.com/marcusbuffett/bevy_snake
- Tutorial: https://mbuffett.com/posts/bevy-snake-tutorial/



ECS

Entities	Components
(Snake)	SnakeHead, SnakeSegment, (DefaultComponents), Position, Size
(Food)	Food, (DefaultComponents), Position, Size

Systems	Components	Non-component Args
setup		Commands, ColorMaterial
spawn_snake		Commands, Materials, SnakeSegments
Snake_eating	Position, Food, SnakeHead	Commands, GrowthEvent

Incomplete….