



Content fueled gameplay programming in Frostpunk

 **Aleksander Kauch**
Lead Gameplay Programmer



Who am I?

ANOMALY
WARZONE EARTH

ANOMALY2

Funky Smugglers

**THIS
WAR
OF
MINE**



FROST
PUNK

Frostpunk

- Society survival
- City builder
- Survival game
- Focused on society
- Narrative heavy



The setup

- Liquid Engine
 - 11 bit's own engine
 - Developed since 2010
- The team
 - 5 gameplay programmers
 - 5 engine/tools programmers





Challenges

- Few references
- Lots of research
- Rapid prototyping
- Fast iterations
- Biggest project for 11 bit studios



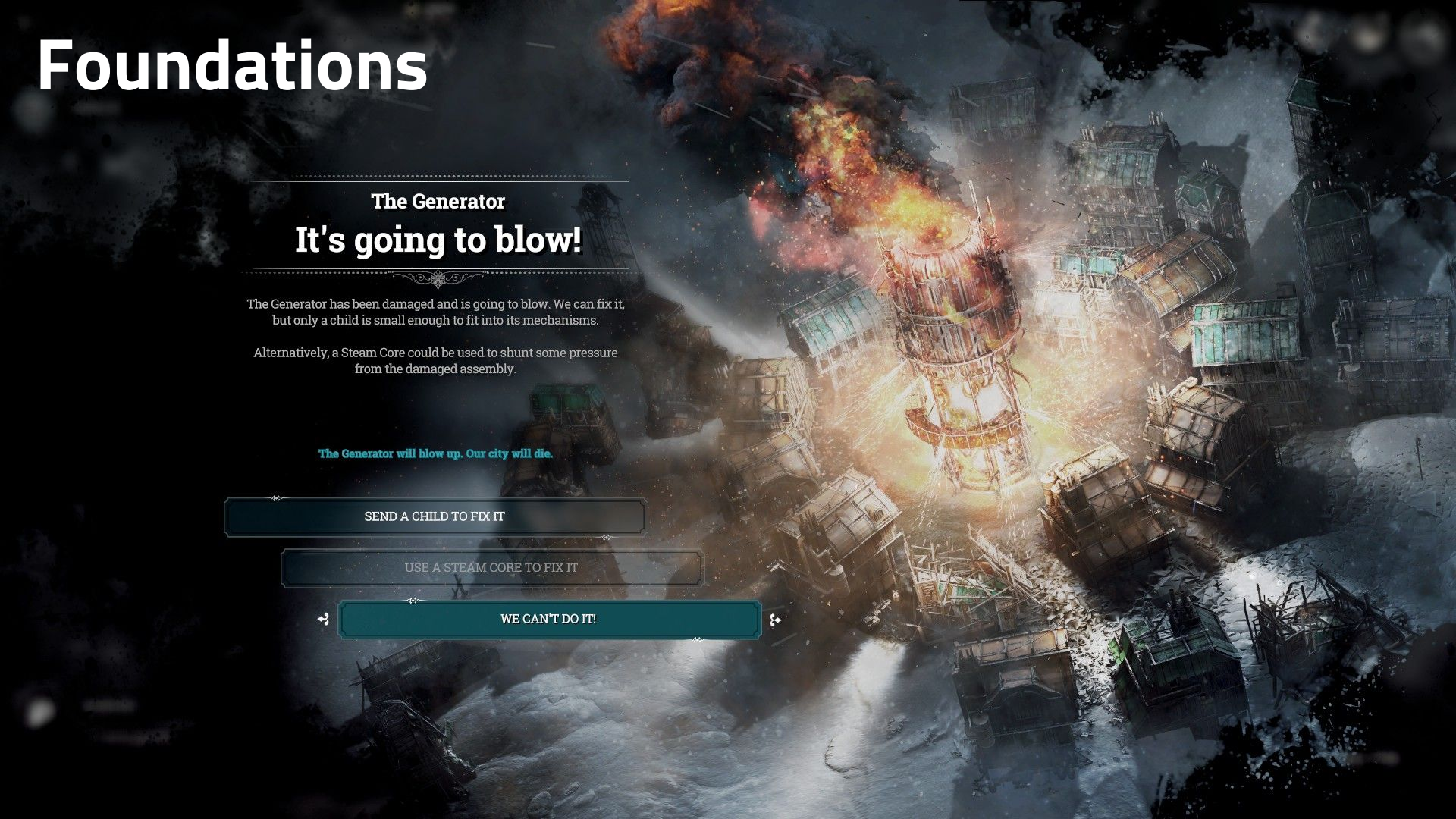
Data driven architecture

- Based on Entity Component System (ECS)
- Flexible, easily expandable configuration
- Content created by design and art teams
- Programmers only provided tools
- Architecture ready for many iterations and experiments



**Without
data driven architecture
creating the unique society builder
would not be possible**

Foundations



The Generator It's going to blow!

The Generator has been damaged and is going to blow. We can fix it, but only a child is small enough to fit into its mechanisms.

Alternatively, a Steam Core could be used to shunt some pressure from the damaged assembly.

The Generator will blow up. Our city will die.

SEND A CHILD TO FIX IT

USE A STEAM CORE TO FIX IT

WE CAN'T DO IT!

Requirements

- Designer-defined entities
- Quick reconfiguration
- Sharing features between entity types
- Scaling

Data structure



■ RTTI Classes

- Runtime type identification
- C++ class serializable into XML node or binary
- XML is used in editor, binary format is used in-game



Data structure



■ GUID *(globally unique identifier)*

- Automatically generated ID based on exact time of creation
- Has a fixed-size text form:

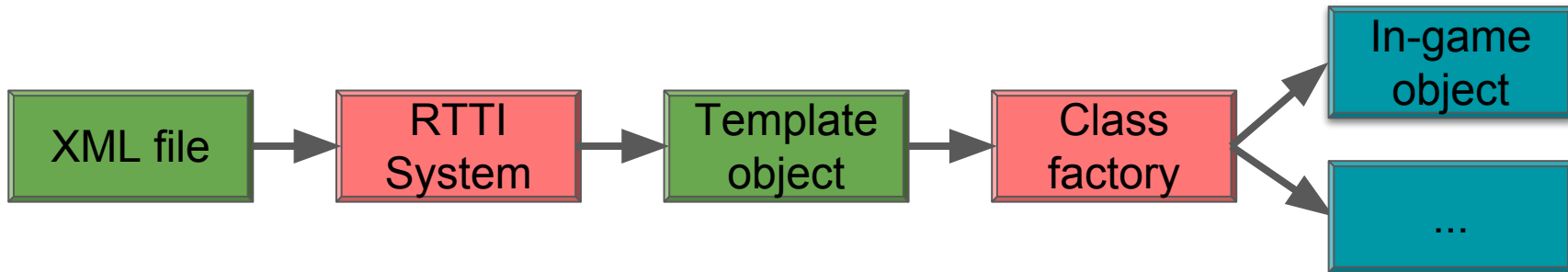
D2F63677-A6D3-42D3-A68C-C149166D2A0E



Data structure

■ Template

- File with RTTI Class defining an in-game object
- Identified by GUIDs
- Editor support



Data structures

■ Template examples

- Entity template - defines in-game object type
- Component template - defines a component
- Mesh template - defines mesh and import data
- UI Recipe - UI Element / UI Screen definition

ECS Implementation



	ENTITY	ENTITY	ENTITY	ENTITY
Move System	COMPONENT		COMPONENT	COMPONENT
Weapon System	COMPONENT			COMPONENT
Container System		COMPONENT	COMPONENT	COMPONENT
Repair System		COMPONENT		COMPONENT



ECS Implementation

- State/Data kept in components
- Logic implemented in systems
- Systems can keep global state

- Some features outside ECS (UI, Input...)



ECS Implementation



- Components and component templates are RTTI Classes
- Entity template has a list of component templates
- Component keeps reference to its template object
 - Const data kept in the template object
 - Mutable data kept in the component object



ECS Implementation



- One component of a type per entity
 - Clean configuration
 - Less configuration errors
- One system per component type
 - Single responsibility principle
 - Simplifies architecture



ECS Implementation

- No System / Component classes inheritance
 - Consequence of one system per component type rule
 - Bitmask IDs possible

MoveComponent ID = 1 << 0

WeaponComponent ID = 1 << 1

RepairComponent ID = 1 << 2

Entity ComponentMask = 0x1

Entity ComponentMask = 0x3

Entity ComponentMask = 0x6

- Expanding functionality via adding new component types

ECS Implementation



■ Component dependency

- Attribute `RequireComponent<Type>` marks component type as a prerequisite

`GeneratorComponent` - `RequireComponent<BuildingComponent>`

■ Component initialization order

- Attribute `AddAfterComponent<Type>` marks component type that

`WorkplaceComponent` - `AddAfterComponent<PlaceForAgentComponent>`



ECS Implementation



■ Component Sets

- Designer-defined batches of components
- Contain components defining most used types of entities
e.g. buildings, citizens, resource piles
- Can be nested
- If duplicated - component definition from the highest level is used



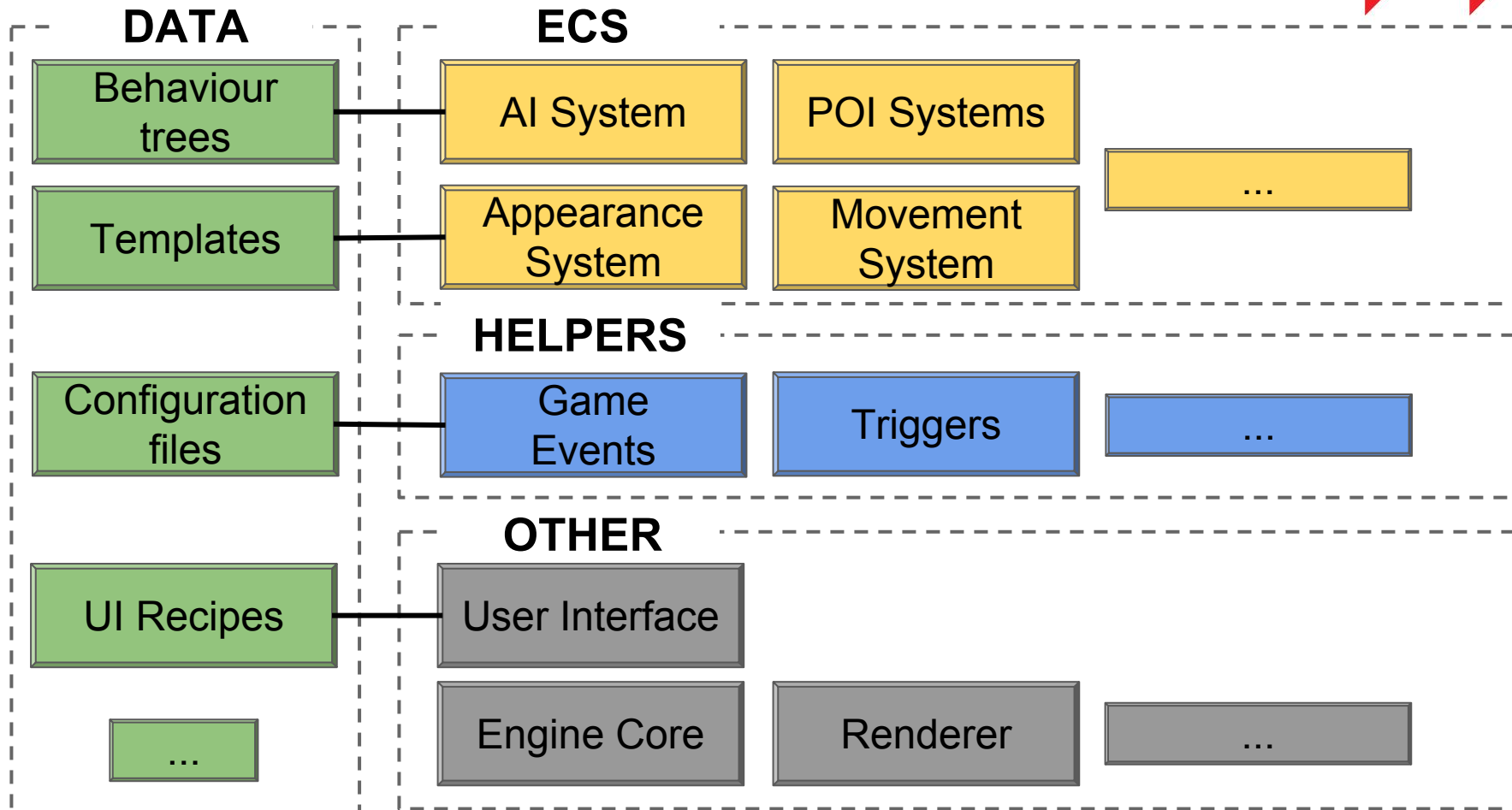
Example component class



```
class GeneratorComponent : public ComponentImpl<GeneratorComponent,  
                                         GeneratorComponentTemplate>  
{  
    DECLARE_PROPERTIES (GeneratorComponent, ComponentBase)  
    {  
        DECLARE_ATTRIBUTES (Attr::RequireComponent<BuildingComponent>());  
        DECLARE_ATTRIBUTES (Attr::AddAfterComponent<HeaterComponent>());  
        DECLARE_PROPERTY (SteamGenerationUpkeep, 0);  
        DECLARE_PROPERTY (SteamPower, 0);  
        ...  
    }  
    ...  
    Map<EntryLink<ResourceEntry>, int> SteamGenerationUpkeep;  
    float SteamPower = 0.0f;  
  
    friend class GeneratorSystem;  
};
```



Architecture overview

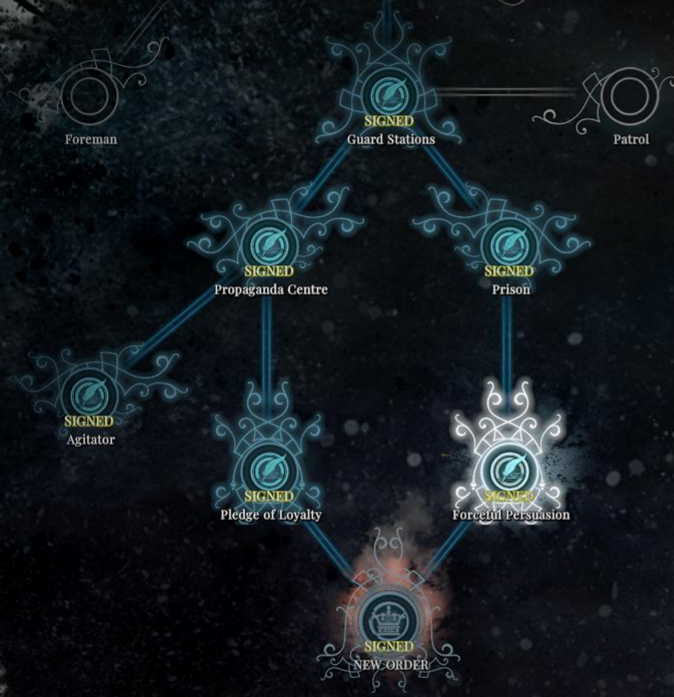


GO BACK

Purpose

Book of Laws

Morning Gathering



CLOSE



Forceful Persuasion

Explaining to inmates the enormity of their wrongdoings in a more direct manner will return them to the society faster.



After introducing this law, another one can be passed after 1d 21h

- NEW ABILITY: Forceful Persuasion (Prison)
- reduces the time prisoners spend in Prison
- prisoners may get hurt or killed
- discontent will rise

Controlling the flow



Requirements

- Narration via in-game events
- Some events scripted, some emerging from game state
- Easy way to check game state elements
- No redundancy in configuration

Configuration

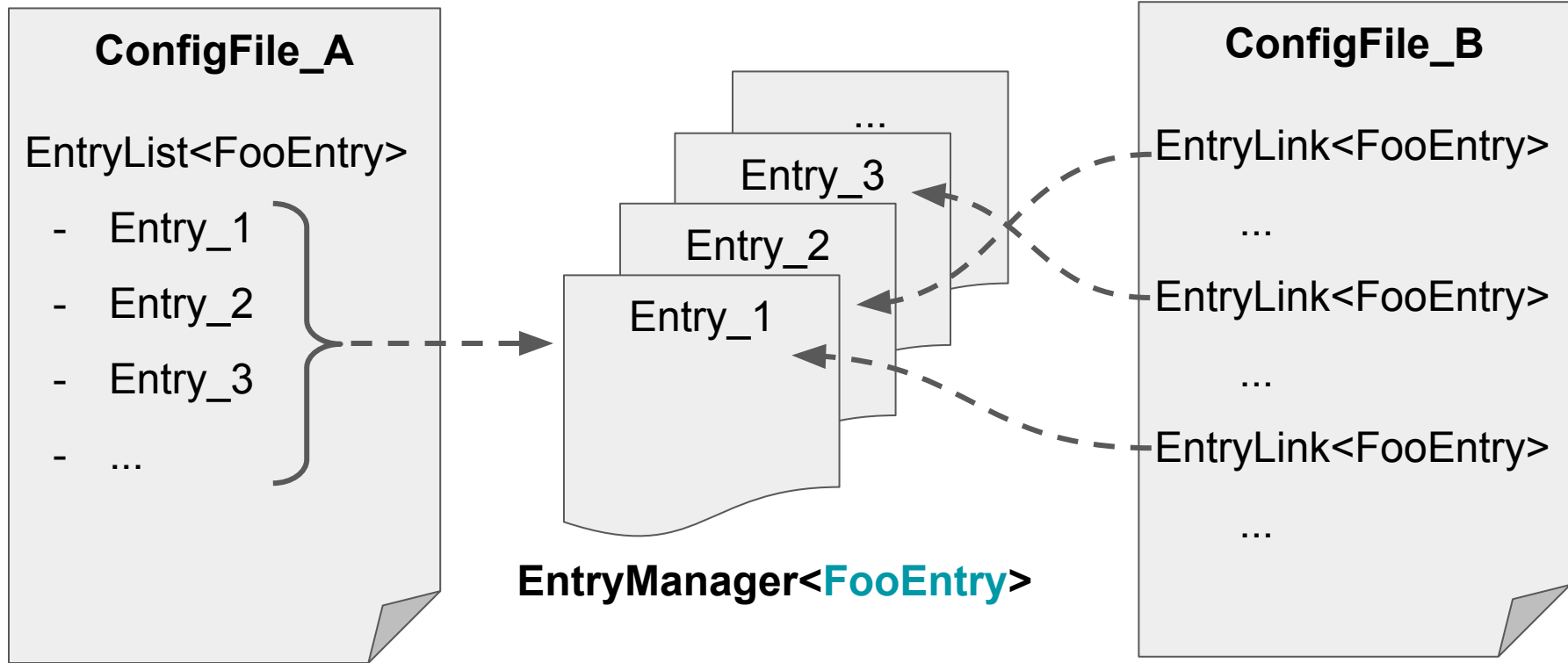


■ Entry

- RTTI Class with GUID and readable name
- Used in configuration files to link configurations
- One “source” of entries per entry type
- Global EntryManager provide lists of available entries
 - one manager per one entry type
- EntryLinks used to reference entries in another part of configuration



Configuration



Game events system

- “Scripts” configured in XML
- Translated to C++ during load
- No scripts interpreted in-game
- Designed for quick prototyping but valid in final content

Game events system



■ Game Event

- Entry built from three lists of element types:

Triggers

Conditions

Effects

- Fully configurable by the design team
- With wide variety of elements it's almost a programming language



Game events system

■ Triggers

- Registered for specific “incident” in-game
e.g.: specific time of day, death of a citizen, new technology ...
- When an in-game “incident” happens - the triggers fire

Triggers	Game[1]
■ [0]	Construction of building Beacon finished
(Class)	Building work
Building template	GameEntities/Buildings/Expedition/Beacon
Building work	Construction
Event	Finished

Game events system

■ Conditions (Boolean expressions)

- Evaluate to true or false
- Utilise custom helper classes that return game state values
- Take form of polish notation expression

1 < Amount of Bodies	1 < Amount of Bodies
(Class)	Integer comparison
Comparison	<
□ Left side	1
(Class)	Constant integer
Value	1
□ Right side	Amount of Bodies
(Class)	Resource amount
Resource	Bodies

Game events system



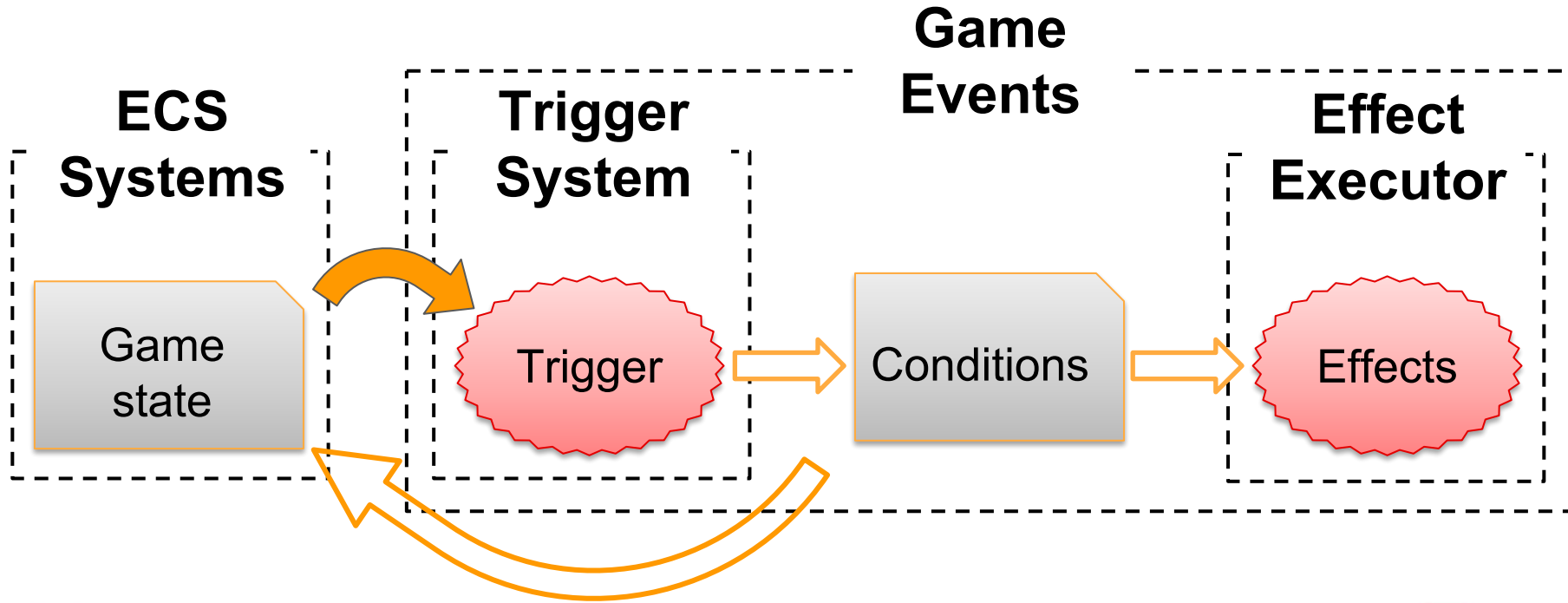
■ Effects

- Custom RTTI Classes that change game state
- They're simply configurable C++ function calls



Game events system

■ Flow:



Examples

- TRIGGER: Citizen died from starvation
CONDITION: > 30 deaths from starvation
EFFECT: Create manifestation

- TRIGGER: New sick citizen
CONDITION: > 10 sick citizens without medical care
EFFECT: Start quest to solve healthcare problem

- TRIGGER: Day 3, 12:00
CONDITION: No hunters' huts
EFFECT: Create hint to construct hunters' hut

Game events system



■ Contexts

- Events have an additional parameter - context Entity
- Triggers provide base context for the event
- Effects can take form of selectors that pick context
- Selectors have their own effect array and conditions (picking criteria)



Game events system

GameConfig: Events

death

Events

Events

_Consequences [common]

Sequences

Utility - General

S: Death - First death

Triggers

Citizen died (loss type: (Any))

Conditions

(This event) execution count < 1

Count of buildings of type Cemetery (constructed: Any, active: Any) == 0

Count of buildings of type SnowPit (constructed: Any, active: Any) == 0

Effects

Create dilemma Sociotech Hints/[Sociotech Hints]S: FirstDead

Invoke event priority group

U: Death - Death out of hunger

U: Death - Stadnard death event

Utility - Music & Sounds

(Config entry)

(Class)	Game event
Name	S: Death - First death

Różne

GUID	abe220a6-8dad-47c6-92aa-fecaa9183a37
Comments	
Enabled	☒ True
Probability	1
Triggers	Game[1]
Conditions	Boolean[3]
Effects	Effect entry[2]

Effects

Game events system

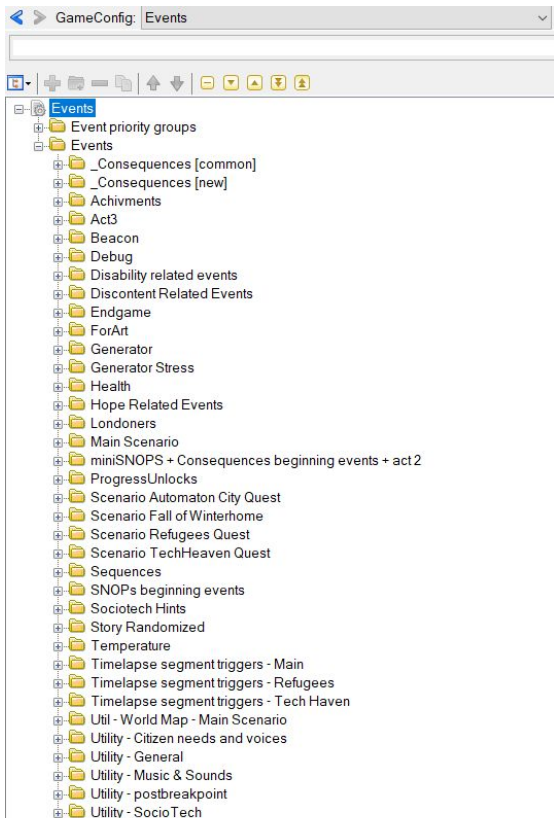
- Parts of this system were reused
 - Conditions in requirements for research
 - Effects in technologies and social laws
 - Effects in hard decision choices
- Entry system allowed creating libraries of predefined conditions/triggers/effects
- Useful debug tool - e.g. trigger firing on console command

Game events system

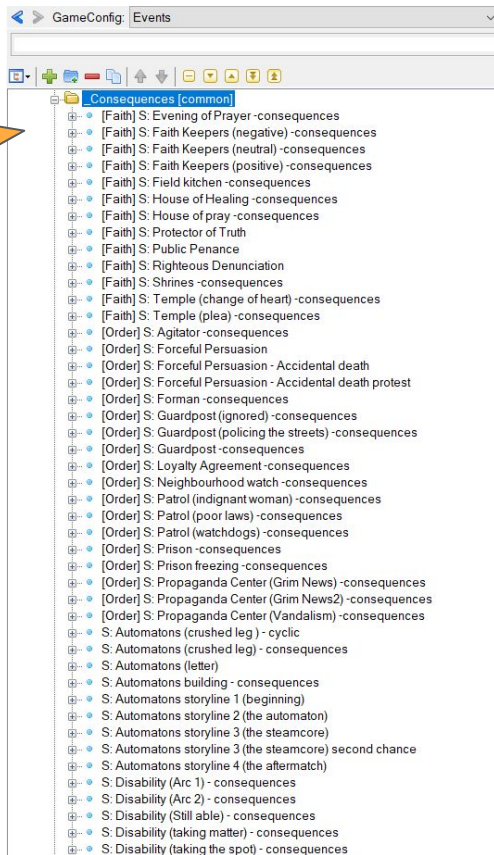
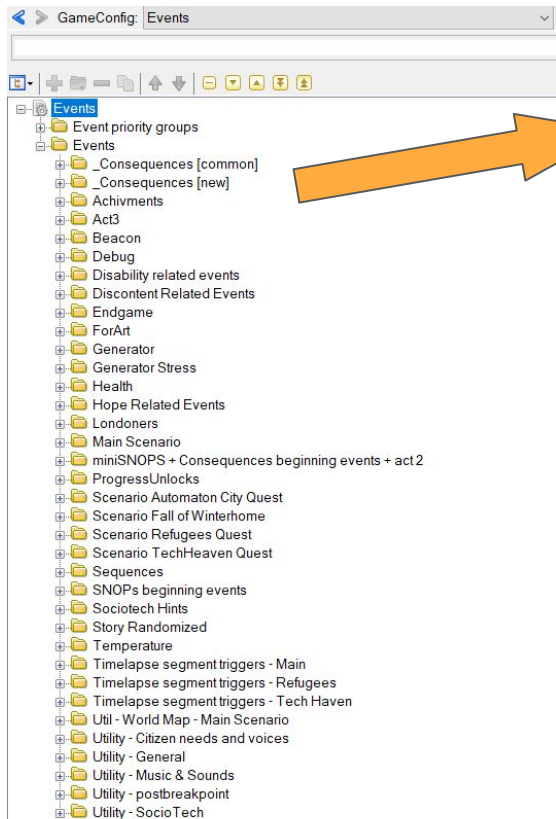
- All of the game “flow” controlled by the Game Events System
- Powerful but can get very complicated to configure
- Needs better tool to simplify the configuration



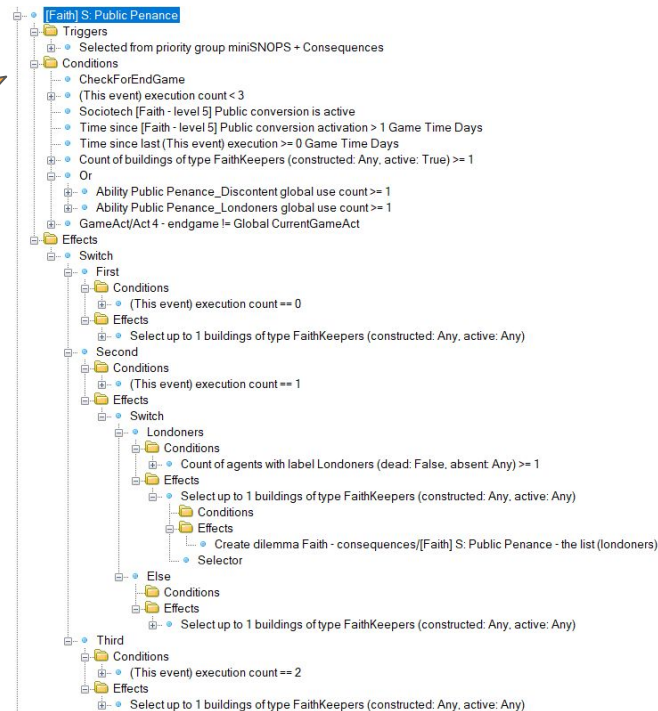
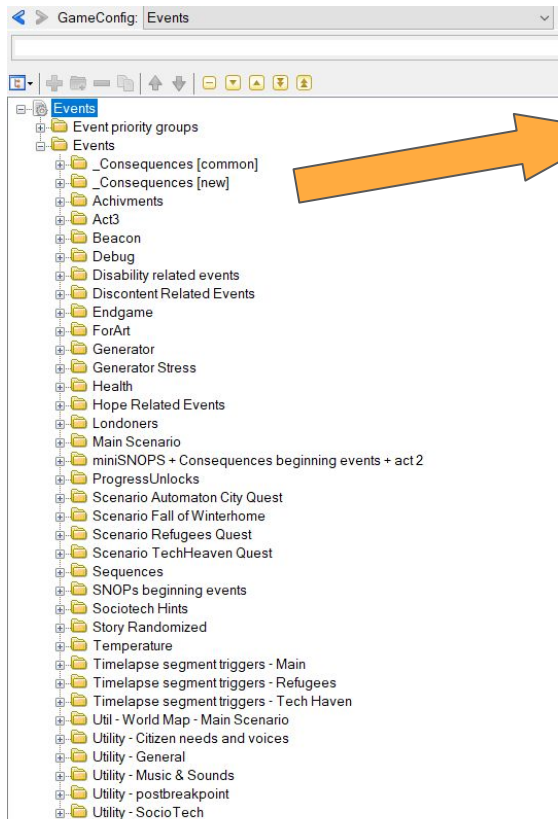
Game events system - scale



Game events system - scale



Game events system - scale





Defining Appearance

Requirements

- Changing entity visuals depending on game state
- Artists define visuals (obviously) and conditions
- Bulk changes of conditions and visual elements

Appearance system

■ Buildings change visuals depending on game state

- Temperature
- Activity
- Policies
- Time of day
- ...



Appearance system



Appearance system

- Appearance state
 - String label, config entry
 - Defined by designers or artists
 - Added/removed by the game event or condition set
- Building can have multiple active appearance states

Active; totalitarian building; during night

Appearance system

- Some states need to be excluded for visuals , e.g.:

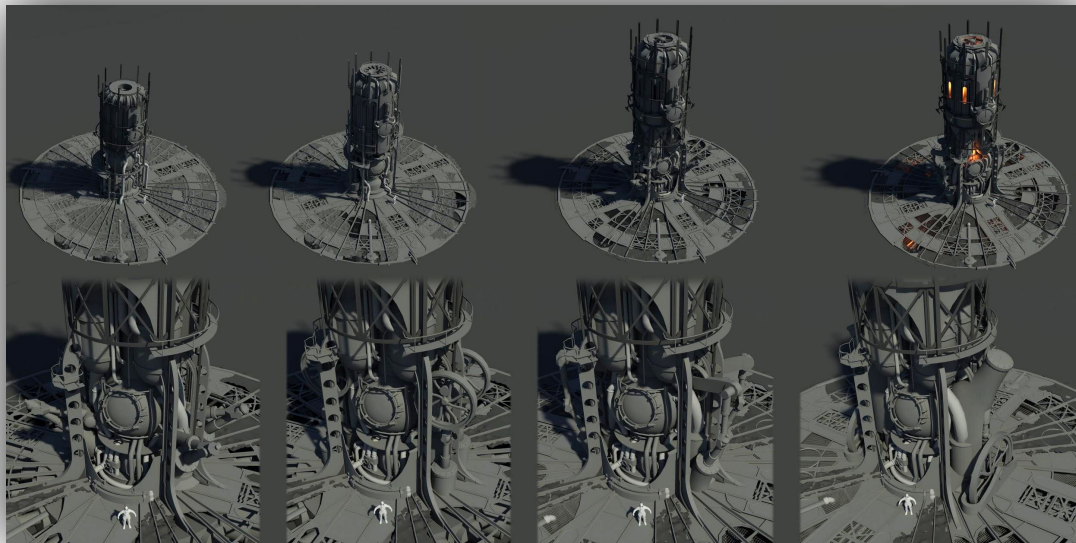
Building; during night; not under construction

- Final visuals of the building depends on the combination of appearance states

Appearance system

■ Visual elements (props) changed by appearance states:

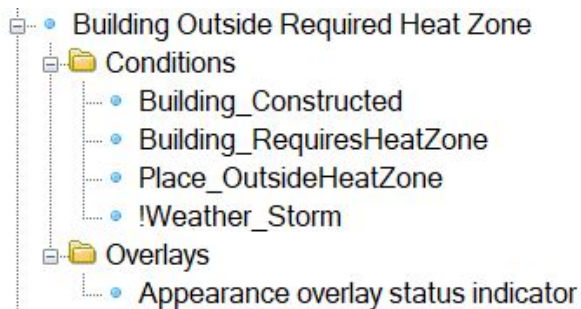
- Mesh elements
- Lights / Textures
- Outlines / Shaders
- Animations
- UI Icons
- Sounds
- ...



Appearance system

- Appearance overlay - simple change in props, e.g.:
 - Visibility
 - Animation start/stop
 - Shader change
- Appearance conditions - combination of appearance states and negations
 - Checked when appearance state list changes
- When conditions match overlay is applied, if not it is removed

Example of appearance config



⊞ (Config entry)	
(Class)	Appearance overlay state
⊞ Rózne	
CustomName	Building Outside Required Heat Zone
Overlay duration type	Persistent
⊞ Conditions	Appearance condition[4]
⊞ [0]	Building_Constructed
⊞ [1]	Building_RequiresHeatZone
⊞ [2]	Place_OutsideHeatZone
⊞ [3]	!Weather_Storm
⊞ Overlays	Appearance overlay base[1]
⊞ [0]	(Appearance overlay status indicator)
(Class)	Appearance overlay status indicator
Text	
Description	
Display overhead	☒ True
Display overhead in Debug	☒ True
Blinking icon	☐ False
Hide overhead icon in distant zoom	☒ True
⊞ Overhead icon	"UI\Tiles\Icons.dds" tile (12 11) of (16 16)
Display in panel	☐ False
⊞ Panel icon	"" tile (0 0) of (1 1)
Custom preset	



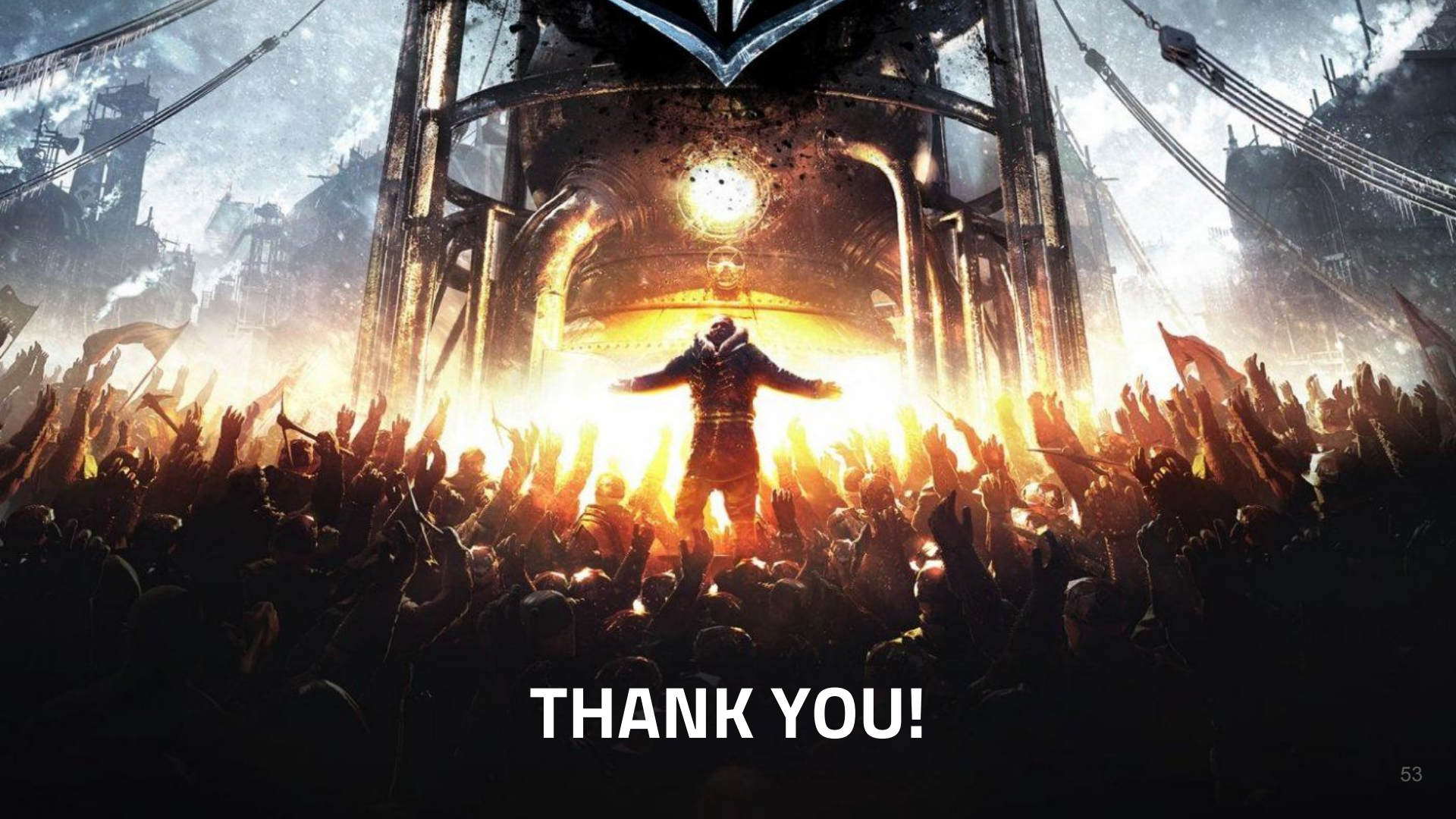
Closing

- Data-driven architecture provided
 - Designer-defined content
 - Rapid prototyping
 - Reusability of features
 - Control for design and art teams
 - Less work for programmers =)
- ECS efficiently glued everything together



Q & A

aleksander.kauch@11bitstudios.com
www.facebook.com/Kauch



THANK YOU!