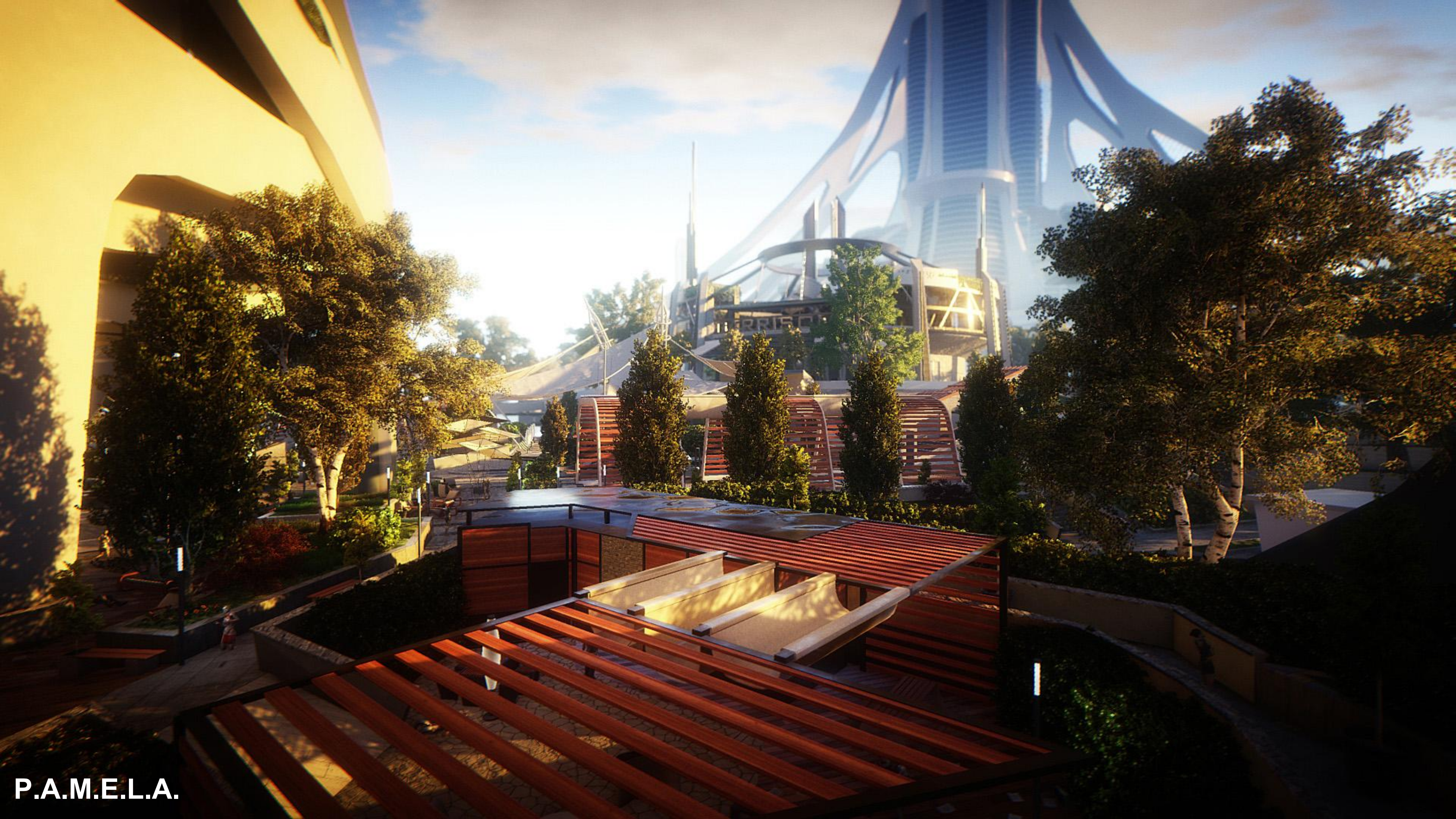
Scriptable Render Pipeline

Future of Rendering in Unity

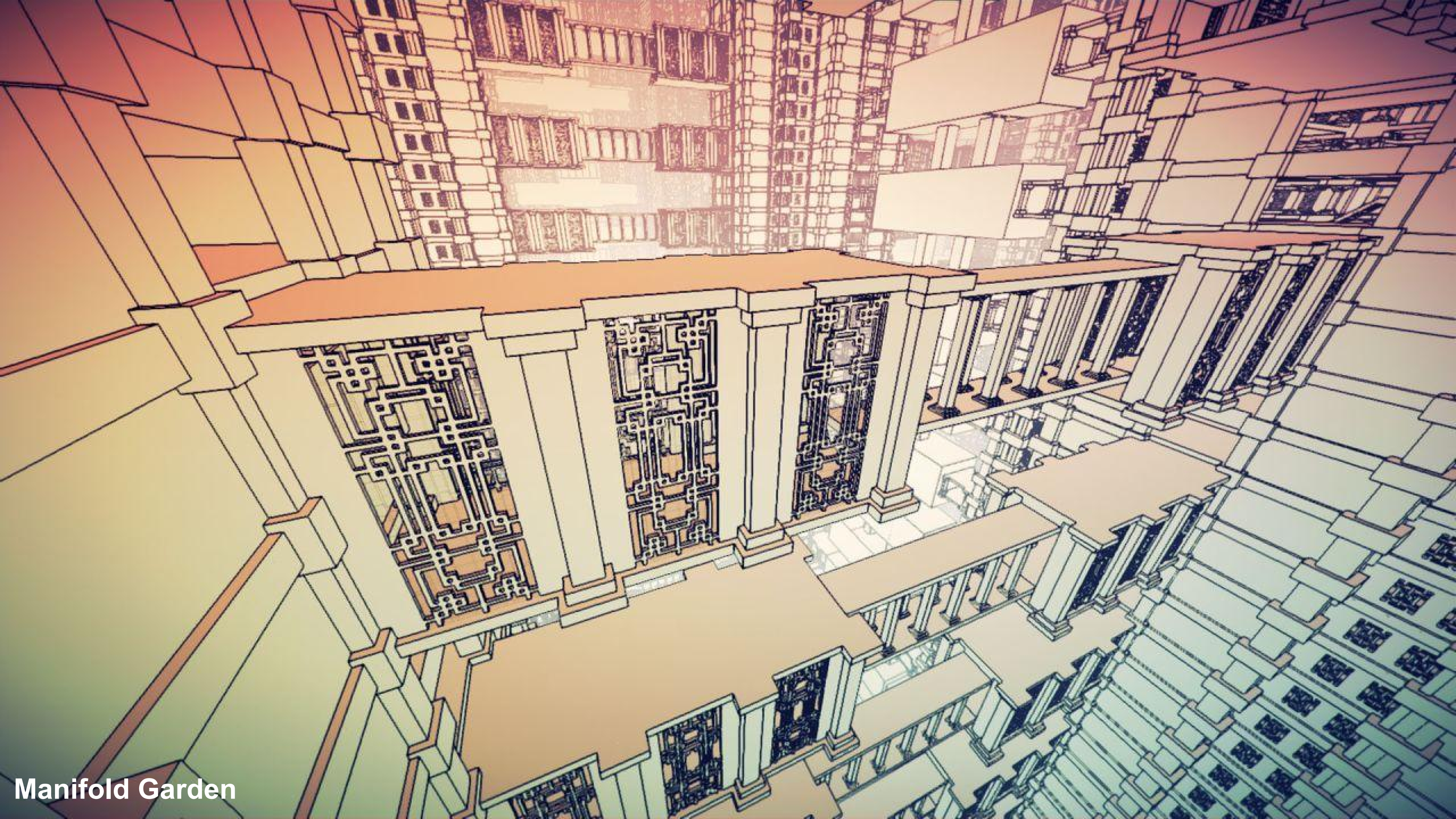
Aras Pranckevičius

Problem In Pictures

Ori And The Blind Forest

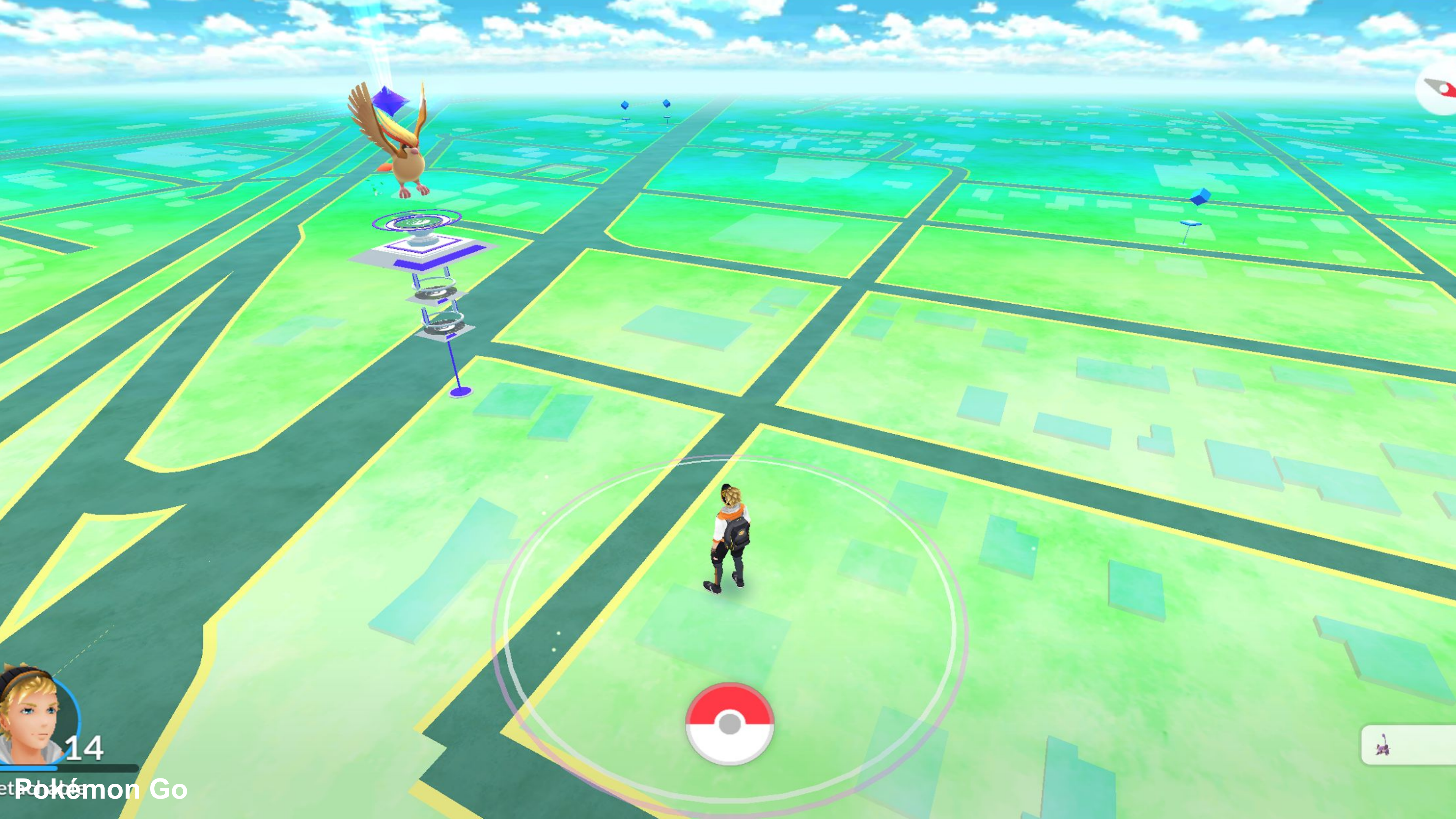

P.A.M.E.L.A.

Night In The Woods

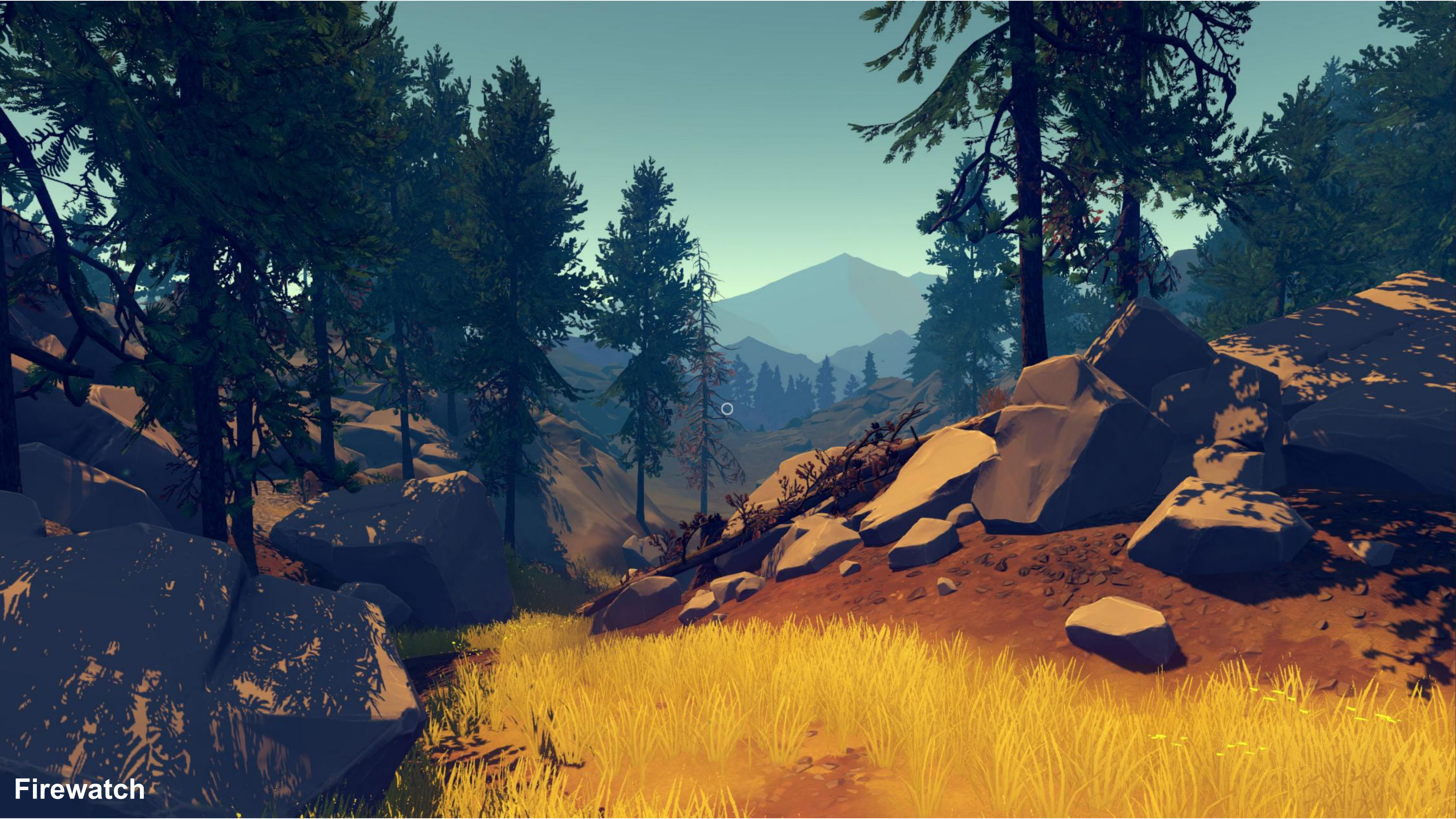

Manifold Garden

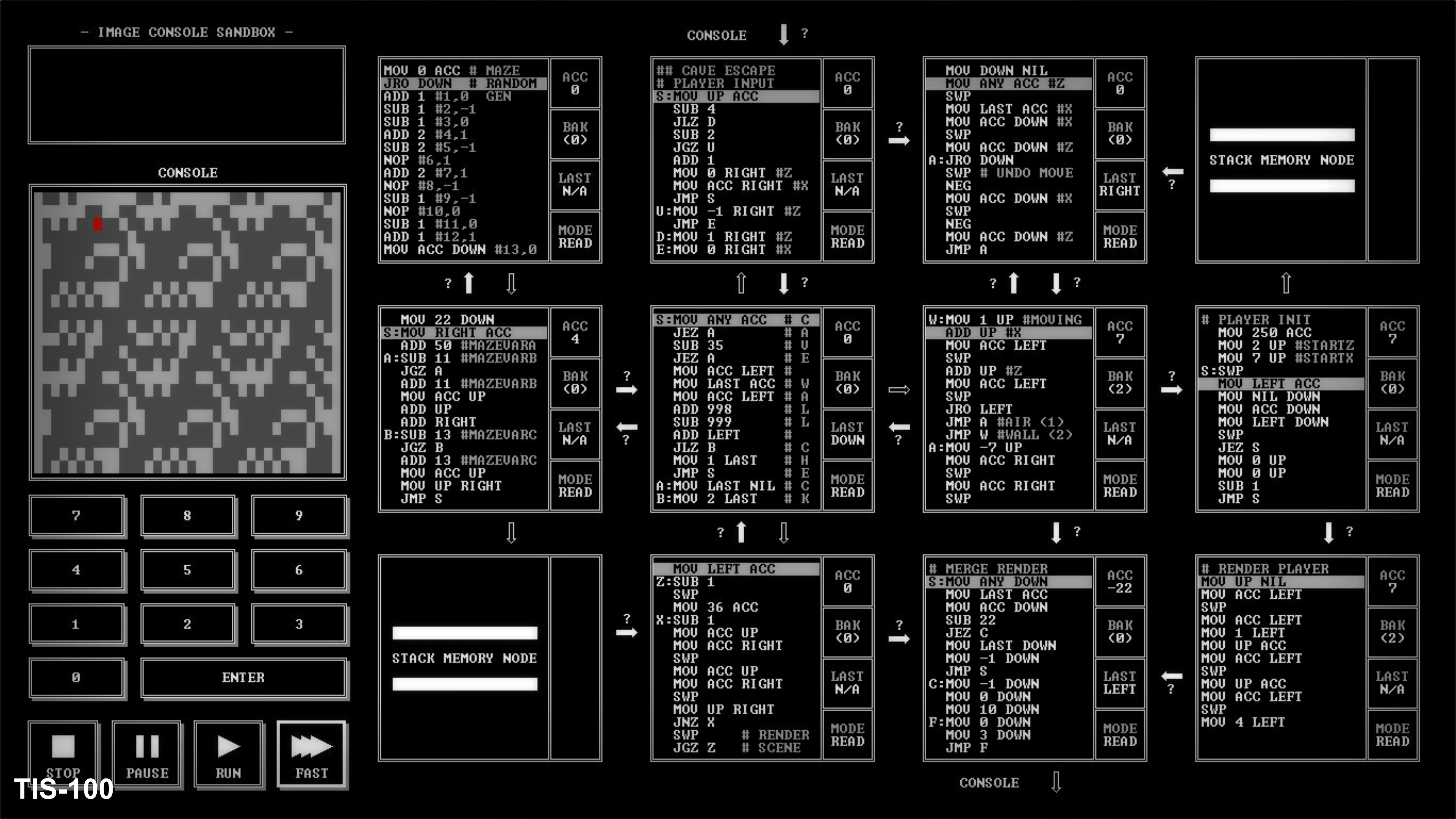
INSIDE

Osiris: New Dawn



Pokémon Go


Firewatch



- IMAGE CONSOLE SANDBOX -

CONSOLE

```
MOV 0 ACC # MAZE
JRO DOWN  # RANDOM
ADD 1 #1,0  GEN
SUB 1 #2,-1
SUB 1 #3,0
ADD 2 #4,1
SUB 2 #5,-1
NOP #6,1
ADD 2 #7,1
NOP #8,-1
SUB 1 #9,-1
NOP #10,0
SUB 1 #11,0
ADD 1 #12,1
MOV ACC DOWN #13,0
```
ACC 0 · BAK (0) · LAST N/A · MODE READ

CONSOLE

```
## CAVE ESCAPE
# PLAYER INPUT
S:MOV UP ACC
  SUB 4
  JLZ D
  SUB 2
  JGZ U
  ADD 1
  MOV 0 RIGHT #Z
  MOV ACC RIGHT #X
  JMP S
U:MOV -1 RIGHT #Z
  JMP E
D:MOV 1 RIGHT #Z
E:MOV 0 RIGHT #X
```
ACC 0 · BAK (0) · LAST N/A · MODE READ

```
  MOV DOWN NIL
  MOV ANY ACC #Z
  SWP
  MOV LAST ACC #X
  MOV ACC DOWN #X
  SWP
  MOV ACC DOWN #Z
A:JRO DOWN
  SWP # UNDO MOVE
  NEG
  MOV ACC DOWN #X
  SWP
  NEG
  MOV ACC DOWN #Z
  JMP A
```
ACC 0 · BAK (0) · LAST RIGHT · MODE READ

STACK MEMORY NODE

```
  MOV 22 DOWN
S:MOV RIGHT ACC
  ADD 50 #MAZEVARA
A:SUB 11 #MAZEVARB
  JGZ A
  ADD 11 #MAZEVARB
  MOV ACC UP
  ADD UP
  ADD RIGHT
B:SUB 13 #MAZEVARC
  JGZ B
  ADD 13 #MAZEVARC
  MOV ACC UP
  MOV UP RIGHT
  JMP S
```
ACC 4 · BAK (0) · LAST N/A · MODE READ

```
S:MOV ANY ACC   # C
  JEZ A         # A
  SUB 35        # V
  JEZ A         # E
  MOV ACC LEFT  #
  MOV LAST ACC  # W
  MOV ACC LEFT  # A
  ADD 998       # L
  SUB 999       # L
  ADD LEFT      #
  JLZ B         # C
  MOV 1 LAST    # H
  JMP S         # E
A:MOV LAST NIL  # C
B:MOV 2 LAST    # K
```
ACC 0 · BAK (0) · LAST DOWN · MODE READ

```
W:MOV 1 UP #MOVING
  ADD UP #X
  MOV ACC LEFT
  SWP
  ADD UP #Z
  MOV ACC LEFT
  SWP
  JRO LEFT
  JMP A #AIR (1)
  JMP W #WALL (2)
A:MOV -7 UP
  MOV ACC RIGHT
  SWP
  MOV ACC RIGHT
  SWP
```
ACC 7 · BAK (2) · LAST N/A · MODE READ

```
# PLAYER INIT
  MOV 250 ACC
  MOV 2 UP #STARTZ
  MOV 7 UP #STARTX
S:SWP
  MOV LEFT ACC
  MOV NIL DOWN
  MOV ACC DOWN
  MOV LEFT DOWN
  SWP
  JEZ S
  MOV 0 UP
  MOV 0 UP
  SUB 1
  JMP S
```
ACC 7 · BAK (0) · LAST N/A · MODE READ

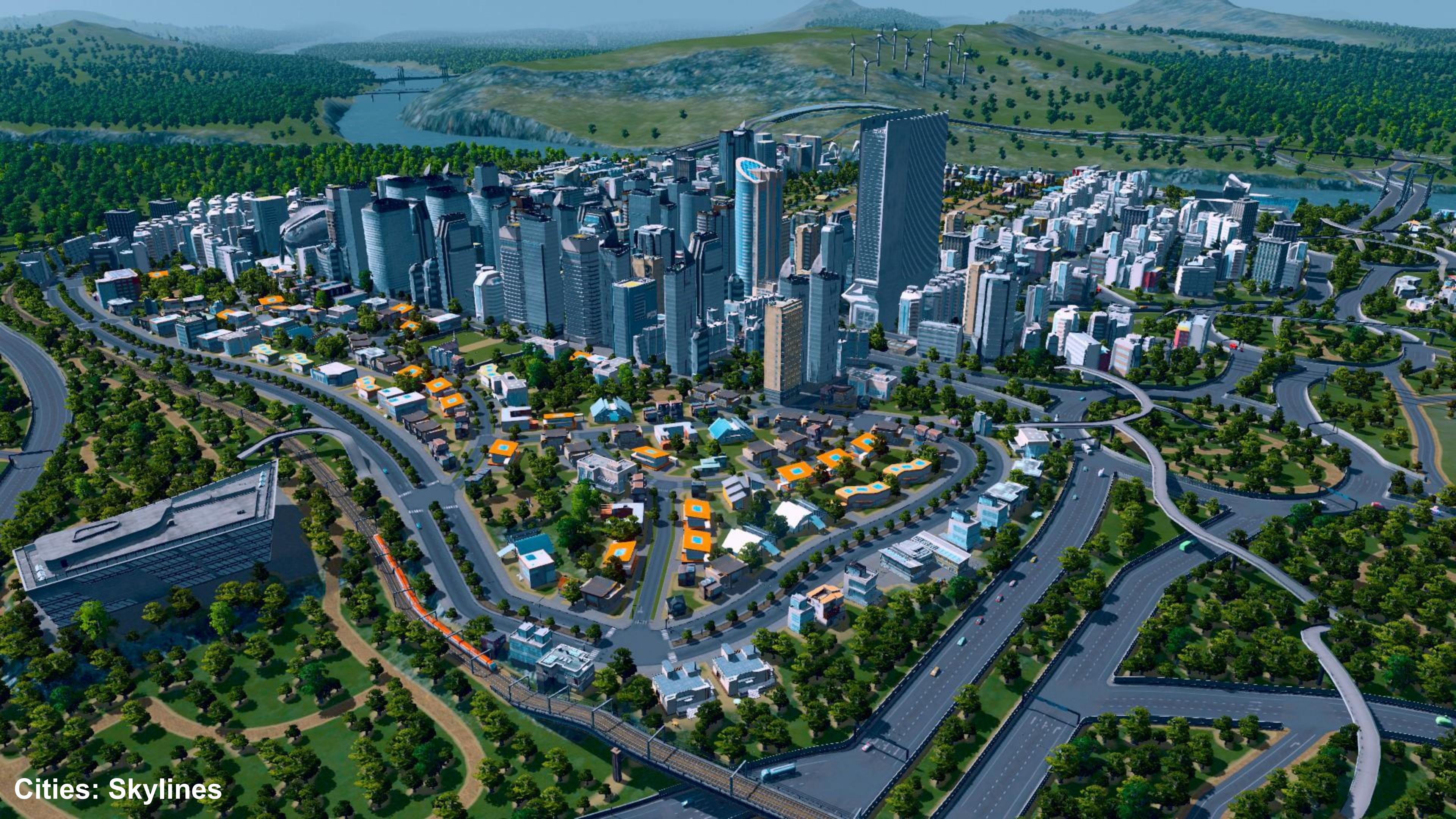
STACK MEMORY NODE

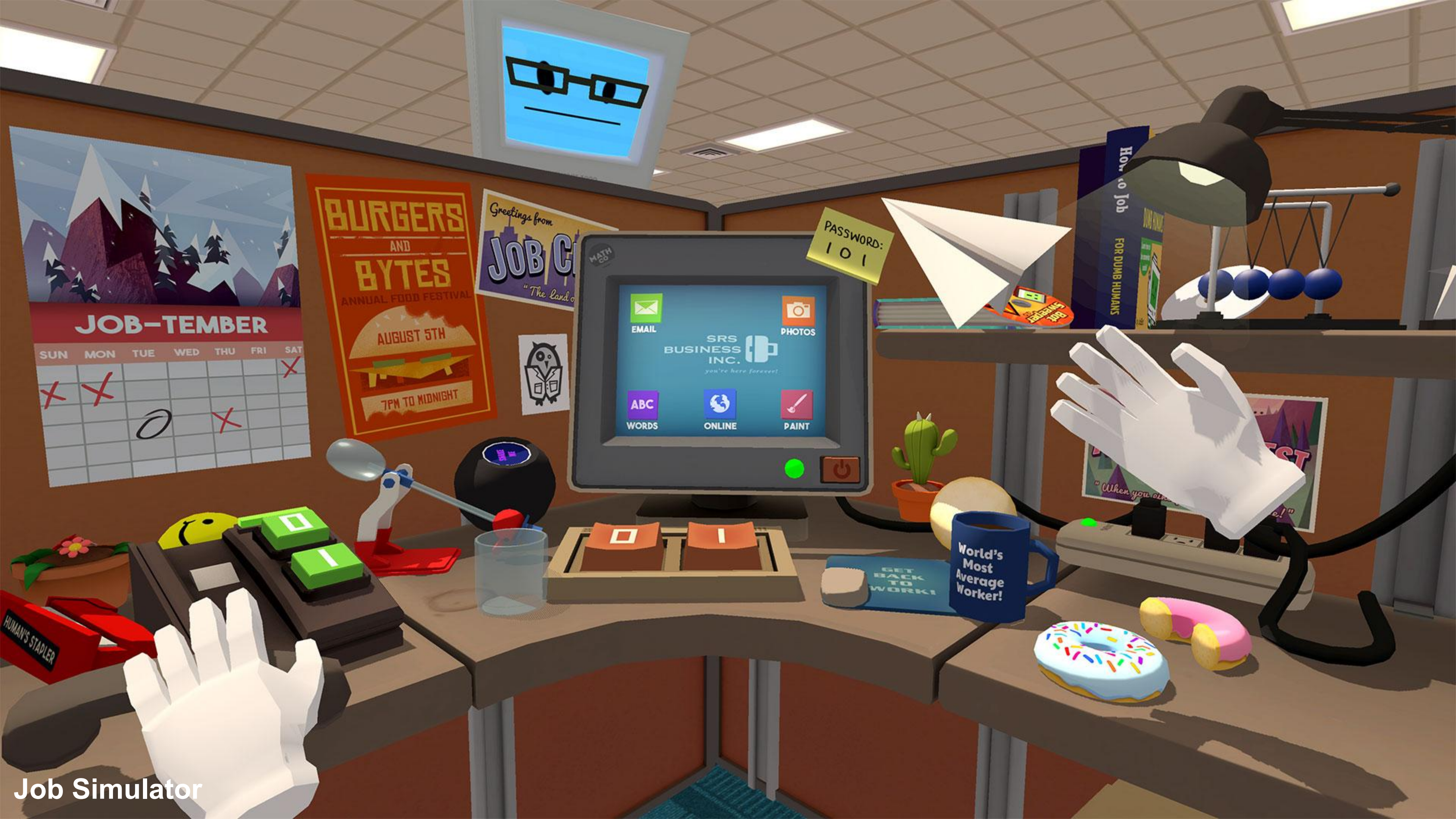
```
  MOV LEFT ACC
Z:SUB 1
  SWP
  MOV 36 ACC
X:SUB 1
  MOV ACC UP
  MOV ACC RIGHT
  SWP
  MOV ACC UP
  MOV ACC RIGHT
  SWP
  MOV UP RIGHT
  JNZ X
  SWP       # RENDER
  JGZ Z     # SCENE
```
ACC 0 · BAK (0) · LAST N/A · MODE READ

```
# MERGE RENDER
S:MOV ANY DOWN
  MOV LAST ACC
  MOV ACC DOWN
  SUB 22
  JEZ C
  MOV LAST DOWN
  MOV -1 DOWN
  JMP S
C:MOV -1 DOWN
  MOV 0 DOWN
  MOV 10 DOWN
F:MOV 0 DOWN
  MOV 3 DOWN
  JMP F
```
ACC -22 · BAK (0) · LAST LEFT · MODE READ

```
# RENDER PLAYER
MOV UP NIL
MOV ACC LEFT
SWP
MOV ACC LEFT
MOV 1 LEFT
MOV UP ACC
MOV ACC LEFT
SWP
MOV UP ACC
MOV ACC LEFT
SWP
MOV 4 LEFT
```
ACC 7 · BAK (2) · LAST N/A · MODE READ

CONSOLE


Cities: Skylines



Job Simulator

Crossy Road

Eagle Flight

Why is that a problem?!

Unity’s Render Pipeline Today, In Theory

- Forward or Deferred
- Configurable
    - Custom shaders, both for materials and lighting
    - Compute shaders
    - Custom post-processing effects
    - Command Buffers
- Works well on all platforms

Unity’s Render Pipeline Today, In Practice...

- Big black box system
- Hard to configure right
- Flexibility is not awesome
- Performance is not awesome
- “One Size Fits All” trap
- Often does not use platform specific strengths
- Changing the behavior is hard

:(

New Goals!

- Small C++ core
- Expose APIs
- High level “render loop” logic in C#

What do we want our renderer to be?

Lean

- Minimal surface area
- Testable
- Loosely coupled

What do we want our renderer to be?

User Centric

- Lives as extension or in user’s project directly
- Debuggable
- Extend and modify
- Fast iteration time for changes

What do we want our renderer to be?

Optimal

- Perform fast, duh
- Optimal for:
  - Particular platform
  - Particular application type
- Allow removing things your project does not need

What do we want our renderer to be?

Explicit

- Does exactly what you tell it. Nothing more. Nothing less.
- No magic
- Clean API

Scriptable Render Pipeline

Engine (C++) vs userland (C#) split

- If it's perf critical, it's done in engine/C++
  - Future: maybe in C# if we can make it fast (ongoing research)
- Engine C++ code:
  - Culling
  - Sorting / Batching / Rendering sets of objects
  - Internal graphics platform abstraction
- C# / shader code:
  - Camera setup
  - Lighting / shadows setup
  - Frame render passes setup / logic
  - Shader & compute code

This is not rocket surgery

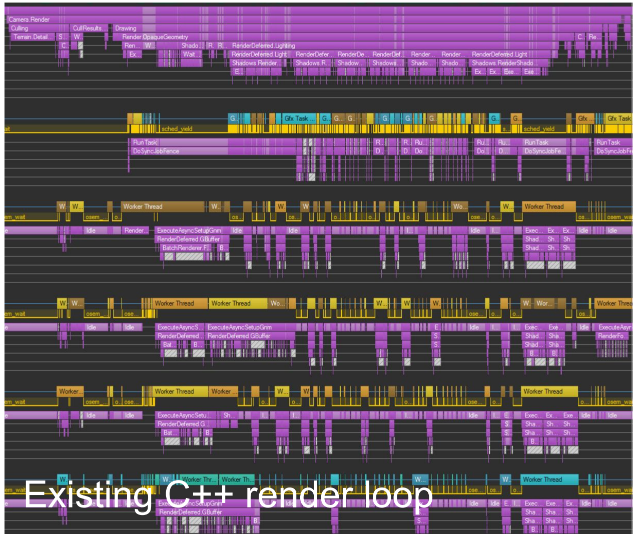
- High level code / config to describe rendering idea is not new:
  - "Benefits of a data-driven renderer", Tobias Persson, GDC 2011
  - "Destiny's Multi-Threaded Rendering Architecture", Natalya Tatarchuk, GDC 2015
  - "Framegraph: Extensible Rendering Architecture in Frostbite", Yuriy O'Donell, GDC 2017
- Should it be data (graph / config files) or code (C# / Lua / …)?
  - We went for code
  - Programmers like code more than noodle graphs :)
  - Some decisions are branchy and game state dependent

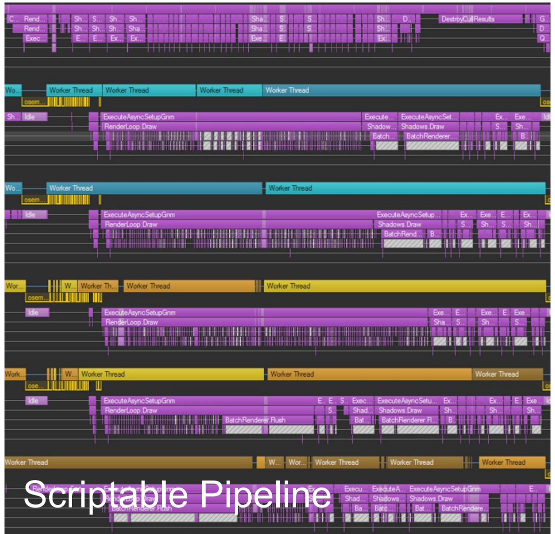
Main C# APIs

- Cull specific views
- Render subset of visible objects
  - With info on what material/shader passes to use
  - With sorting flags
  - With “what kind of per-object data to setup” (light probes, per-object light lists, etc.) to set up
- Already existing APIs for:
  - Setting up render passes / render targets
  - Setting up shader constants / global resources
  - Dispatching compute shaders
  - Rendering individual meshes (for special fx / post fx)
- APIs build a “command buffer” that is later analyzed/executed

C#?! U MAD?!?!

- This is high-level code operating on frame structure
- No per-visible-object C# bits
- Actually runs faster and schedules better than our old C++ render loops!
- We also have a bunch of threading / no-GC things cooking for C#, *soon...*



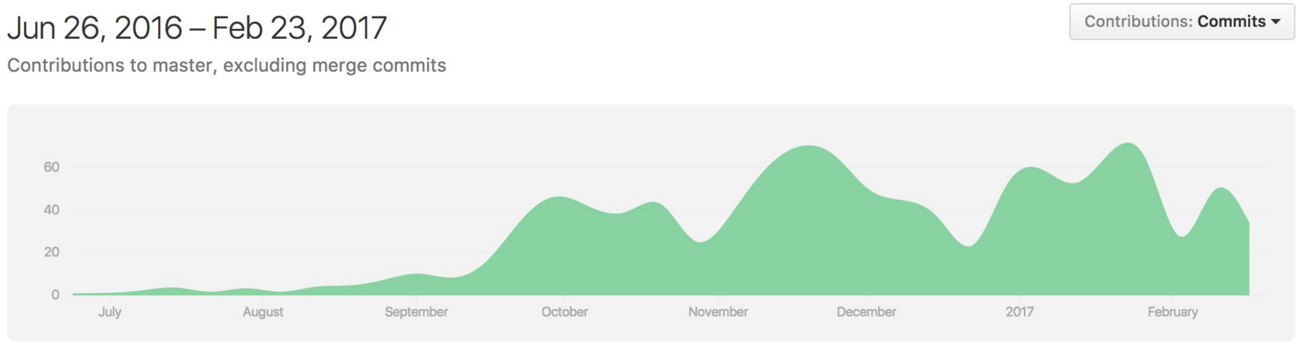


Want to ship out of the box

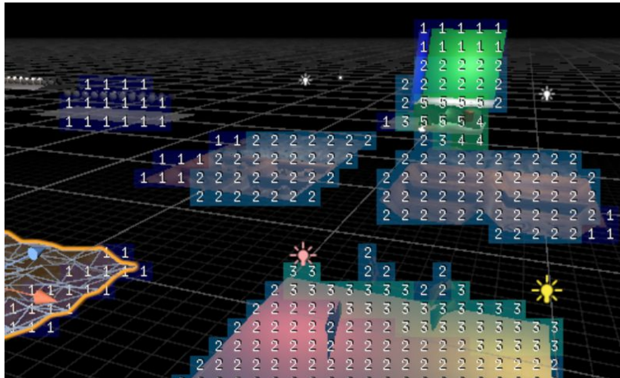
- PC/Console/High-Mobile pipeline *(codename "HD"... naming is hard!)*
- Low-end mobile pipeline
- VR

HD Pipeline

- PBR, GGX, area lights, FPTL/clustered, aniso GGX, layered, SSS, …
  - All the buzzwords :)
- Requires compute shader support
- Watch it live! github.com/Unity-Technologies/ScriptableRenderLoop


Jun 26, 2016 – Feb 23, 2017

Contributions to master, excluding merge commits



Great. When?

- “Experimental” in Unity 5.6 since *last year*!
  - unity3d.com/unity/beta
  - github.com/Unity-Technologies/ScriptableRenderLoop
  - API keeps on changing
- Want to ship “for reals” in release after 5.6