Out of Sight, Out of Mind: Improving Visualization of AI Info

Mika Vehkala
Principal Game Tech Programmer @ Guerrilla Games

AI ARTIFICIAL INTELLIGENCE SUMMIT

GAME DEVELOPERS CONFERENCE 2014
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21

Contents

- Introduction
- Part I – Visualizing Runtime Flow
- Part II – Record and Playback Data
- Part III – Visualizing Algorithms
- Behind the scenes

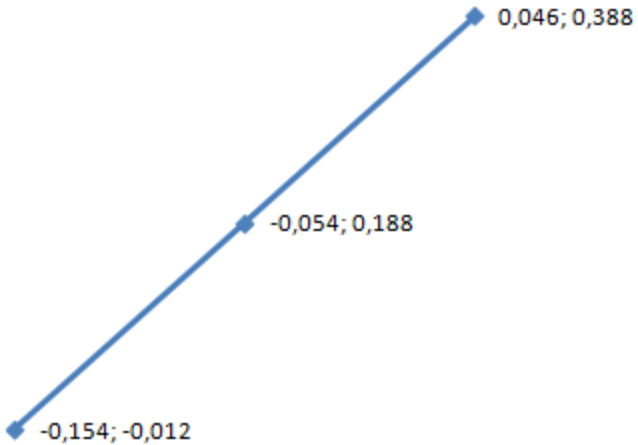
Introduction

- Improving visualization of your data helps in
  - Debugging
  - Verification
  - Understanding
- Challenge your workflows and tools

Simple example

Points	Count = 3
[0]	{X: -0,154 Y: -0,012}
[1]	{X: -0,054 Y: 0,188}
[2]	{X: 0,046 Y: 0,388}
Raw View	

This?



Or this?

Nice properties for a debug tool

- Minimize impact on client
  - Low memory requirement
  - Low processing power requirement
  - Lean API for minimal debug code

  => Separate process, communicates over network
- Clutter-free UI
  - Shouldn’t need user’s manual for the tool
  - Helps to keep visualization simple as well

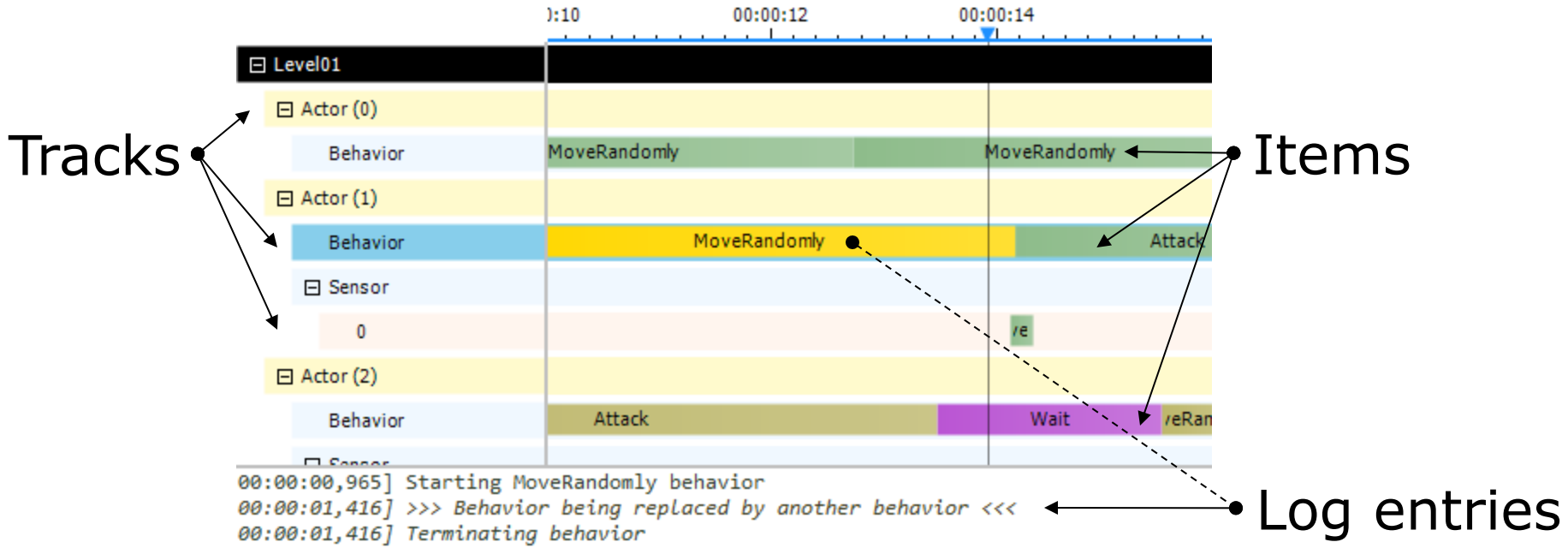
Part I – Visualizing Runtime Flow

- What are the main components
- Who manages the lifetime
- What is the lifetime
- What are the dependencies

Sequencer

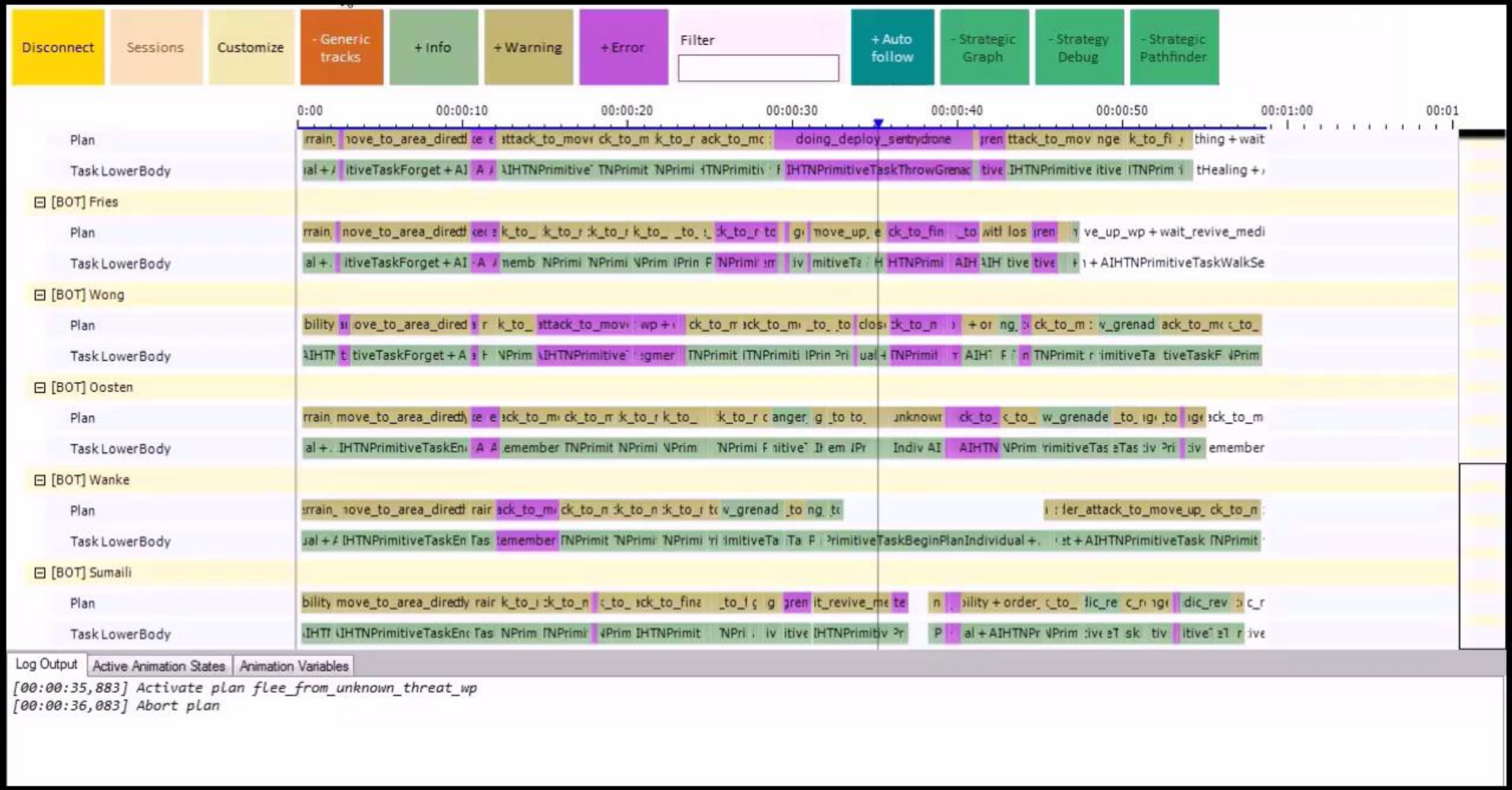
- Hierarchy to show structure
- Timeline and tracks to show history

Hierarchical Timeline View



Part II – Record and Playback Data

- Simple and data agnostic
- Register binary feed and callback
- Add arbitrary data { ID | Time | Byte[*] }
- Scrub timeline to send back to feed


Disconnect
Sessions
Customize
- Generic tracks
+ Info
+ Warning
+ Error
Filter
+ Auto follow
- Strategic Graph
- Strategy Debug
- Strategic Pathfinder
0:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01
Plan
Task LowerBody
[BOT] Fries
Plan
Task LowerBody
[BOT] Wong
Plan
Task LowerBody
[BOT] Oosten
Plan
Task LowerBody
[BOT] Wanke
Plan
Task LowerBody
[BOT] Sumaili
Plan
Task LowerBody
Log Output
Active Animation States
Animation Variables
[00:00:35,883] Activate plan flee_from_unknown_threat_wp
[00:00:36,083] Abort plan

Use in Killzone Shadowfall

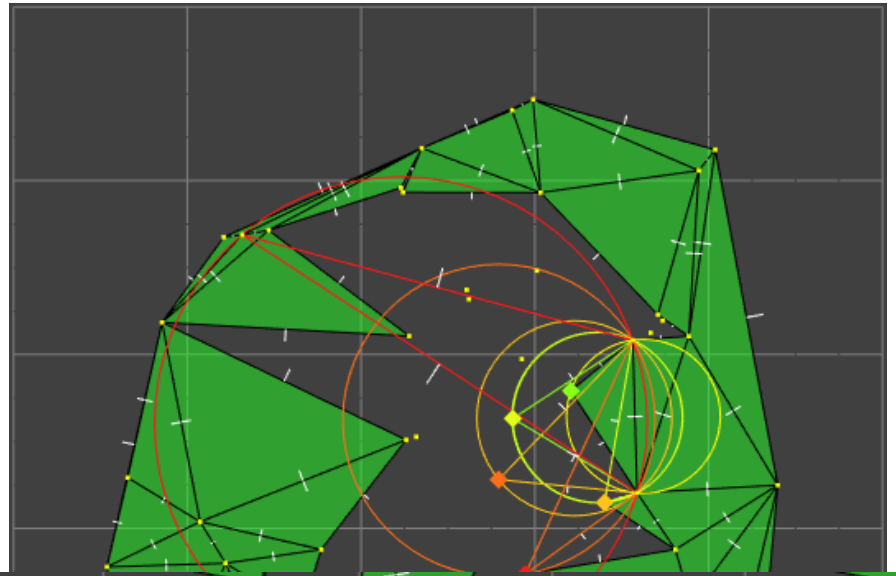
- MP Bot AI debugging and validation
- Gameplay animation debugging
  - Player
  - NPCs
- Took ~1 week to integrate and hook-up debug calls

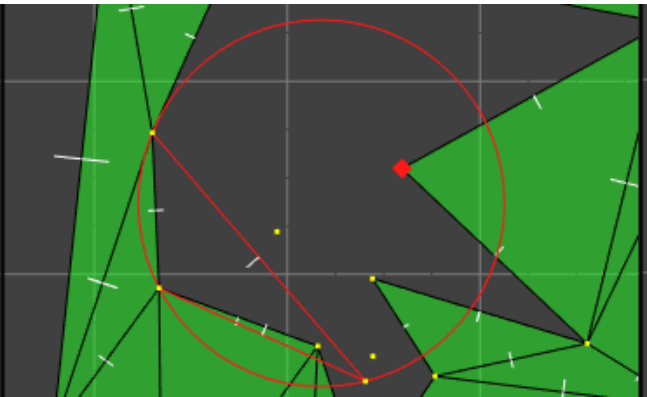
Part III - Visualizing Algorithms

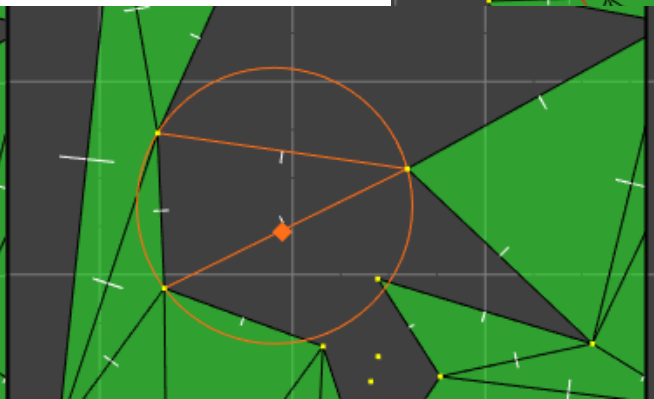
- How to visualize algorithms
  - Not just the end result but step-by-step
- No access to renderer
  - Long turn-around time to use in-game rendering
- Also, alternative viewport

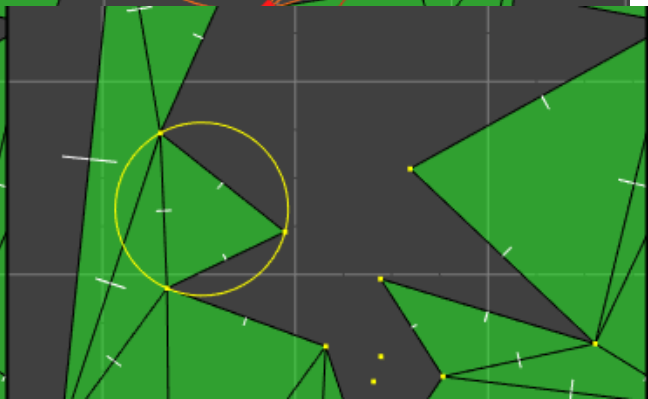
I've found out that...

- Visualizing data is not trivial
  - Iterate but keep it simple
- Time is of the essence
  - Collapse into single image
  - Series of snapshots









Behind the scenes

- ReView communicates using RPC over TCP/IP
  - Major contributor to extensibility!
- C# for building the tool

Quick look at the code

```
Feed.Connect("localhost", 5000);

track_id = Feed.AddTrack(parent_id, “Name”);
item_id = Feed.AddItem(track_id, time, ”Name”);
Feed.AddLog(item_id, time, flags, ”Log entry”);

box_id = Feed.AddBox(time, Inf, Matrix.Identity, center, size, Color.Green);
Feed.RemovePrimitive(box_id, later_time);

id = Feed.AddMesh(time, Inf, Matrix.Identity, center, flatShaded : true);
Feed.AddTriangle(id, time, pointA, pointB, pointC, Color.GreenAlpha);
```

Takeaway

Don’t guess what happened...

...know what happened!

That's All!

Follow @MikaVehkala

ReView can be found at www.reviewtool.net

Special thanks to Maurizio De Pascale

Suggested reading;
Edward Tufte, The Visual Display of Quantitative Information

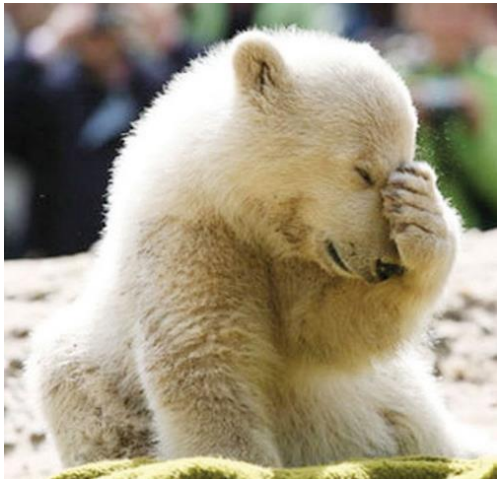
Out of Sight, Out of Mind: Improving Visualization of AI Info

Bill Merrill
Senior AI Engineer at Turtle Rock Studios

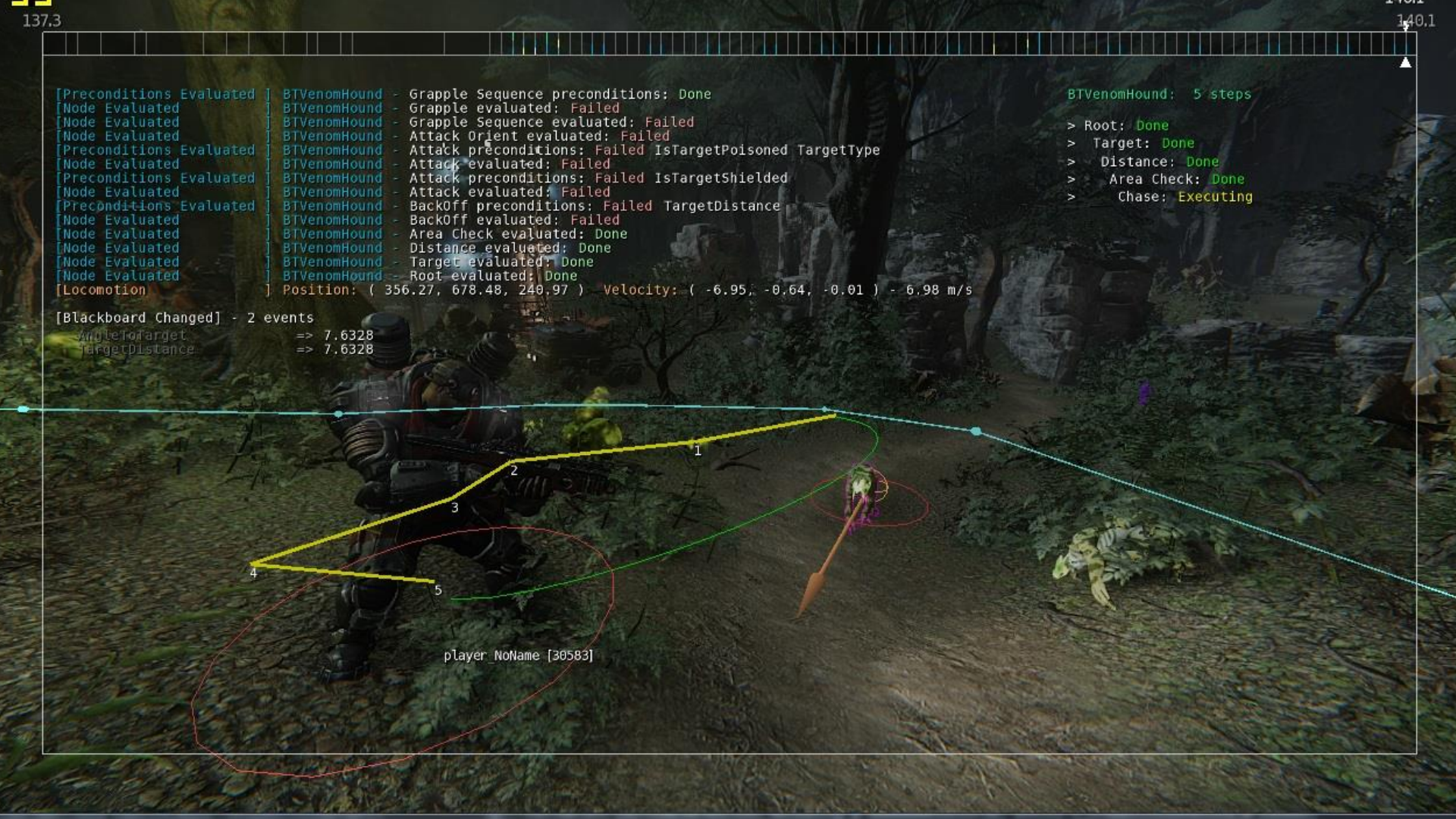
GAME DEVELOPERS CONFERENCE 2014
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21

Introduction

- Our version of a cheap, but powerful tool for historical debugging
- Tools are always worth the time, but it’s never too late
- Can be built at **very low cost**
- I should’ve done it sooner




100


[Preconditions Evaluated] BTVenomHound - Grapple Sequence preconditions: Done
[Node Evaluated] BTVenomHound - Grapple evaluated: Failed
[Node Evaluated] BTVenomHound - Grapple Sequence evaluated: Failed
[Node Evaluated] BTVenomHound - Attack Orient evaluated: Failed
[Preconditions Evaluated] BTVenomHound - Attack preconditions: Failed IsTargetPoisoned TargetType
[Node Evaluated] BTVenomHound - Attack evaluated: Failed
[Preconditions Evaluated] BTVenomHound - Attack preconditions: Failed IsTargetShielded
[Node Evaluated] BTVenomHound - Attack evaluated: Failed
[Preconditions Evaluated] BTVenomHound - BackOff preconditions: Failed TargetDistance
[Node Evaluated] BTVenomHound - BackOff evaluated: Failed
[Node Evaluated] BTVenomHound - Area Check evaluated: Done
[Node Evaluated] BTVenomHound - Distance evaluated: Done
[Node Evaluated] BTVenomHound - Target evaluated: Done
[Node Evaluated] BTVenomHound - Root evaluated: Done
[Locomotion] Position: (356.27, 678.48, 240.97) Velocity: (-6.95, -0.64, -0.01) - 6.98 m/s
[Blackboard Changed] - 2 events
AngleToTarget => 7.6328
TargetDistance => 7.6328
BTVenomHound: 5 steps
> Root: Done
> Target: Done
> Distance: Done
> Area Check: Done
> Chase: Executing
1
2
3
4
5
player NoName [30583]

Background on Evolve and TRS

- Online cooperative/competitive first/third-person shooter
- Always plenty of AI, even in full online games
- AI agents also must play **all roles** in liu of human players
- Rapid development; need to leverage lots of playtest data

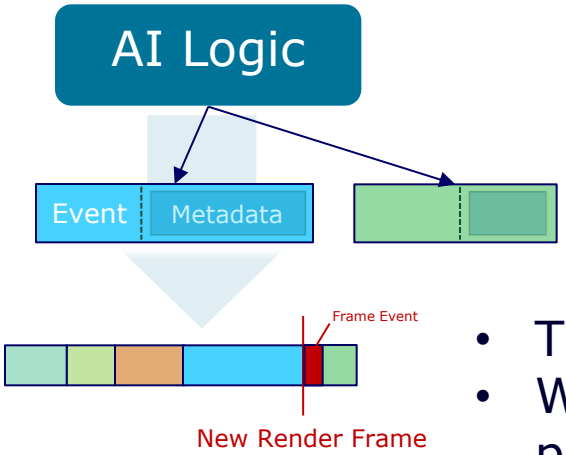
Bare Bones Requirements

- Get it up and running in a man-week
  - Took almost as long to prepare this presentation ☺
- Rapidly and safely add data; vis comes second
- A dedicated server recording should feel like a local session
- Runs on server, so minimum CPU overhead during recording

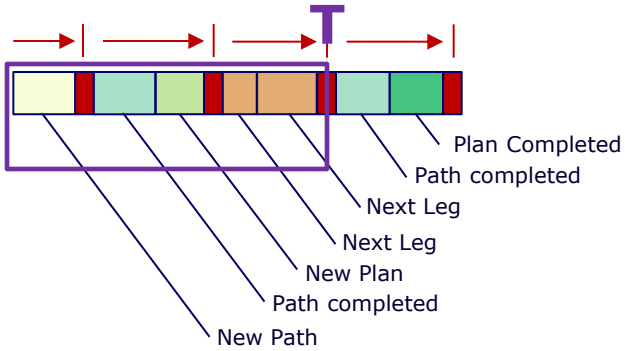
Stupid-Simple Data Stream

- Self-contained events and metadata in a contiguous memory stream
- Metadata typically very small, and easily quantized
- Store frame markers to establish timeline
- If the stream is nearly full, we purge old contents
  - "Version 2.0" would handle this more intelligently
- **The data's all there** - reconstruct and render later on a visual client

Writing the Stream



Interpreting the Stream



- Timeline scrubs between render frames
- We always know what happened in the past, relative to **T**
- Turn small atomic events into useful data
- Higher granularity than this example (details later)

Versioning

- We simply distinguish between a current version and a last-readable version – pretty standard
- Each event type's serialization handler can support multiple versions

```
SERIALIZE()
{
    VEC3_RANGE(m_pointStart,4000.f);
    VEC3_RANGE(m_pointEnd,4000.f);

    if(version >= 3)
    {
        FIELD(m_flightType);
    }
}
```

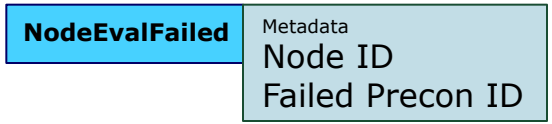
- Periodically strip old version support, just so the code is tiny

Game Data Compatibility

- We always know the originating build's stamp; sync to data as necessary to reference large data
- When possible, events store *inputs,* and re-execute during timeline scrubbing
  - Determinism is important, but only needed in a small subset of systems
- Some events just serialize results if they're tiny

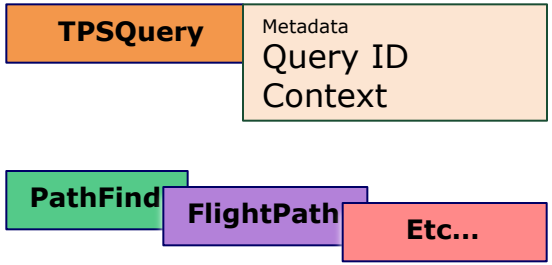
Minimizing Metadata

Referencing Static Data



- Minimal, just relates directly to static BT data
- Sync to older game data as necessary

Re-Query With Stored Input

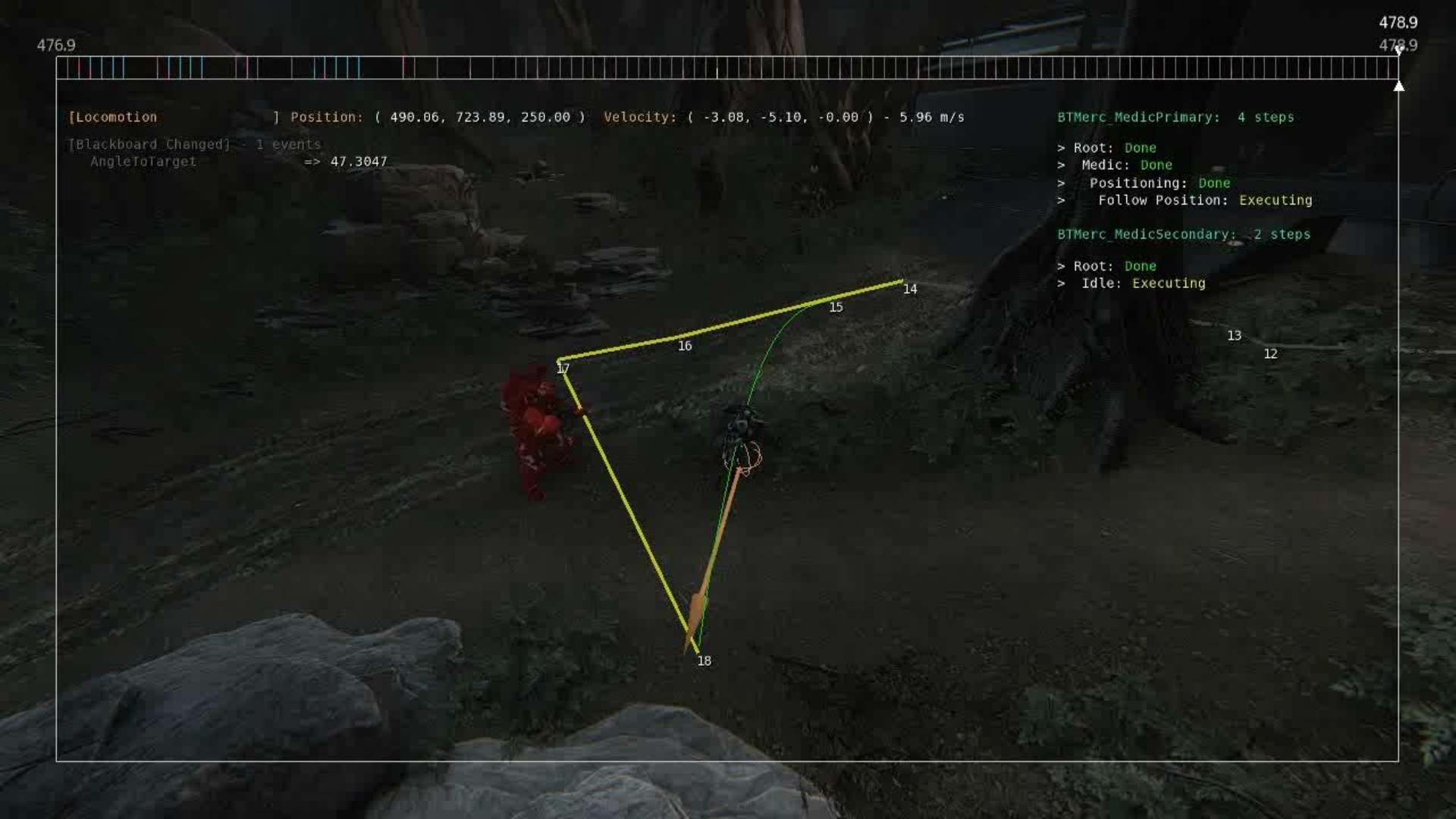


- For a tactical query, we need to see all candidates and their scores (tons!)
- Way too much to store, so we store the context used to conduct the query
- Just re-execute tactical query; metadata as input

Playback & Scrubbing

- Timeline shows a range of time with color-coded markers
- Linearly process entire stream up to the displayed frame
- Use gamepad to scrub back and forth, detach camera, select different agents
- Aggregating larger context under the hood for a complete picture
- Anything traditional debug displays can show… but with history


100


476.9
478.9
478.9
[Locomotion] Position: (490.06, 723.89, 250.00) Velocity: (-3.08, -5.10, -0.00) - 5.96 m/s
[Blackboard Changed] - 1 events
AngleToTarget => 47.3047
BTMerc_MedicPrimary: 4 steps
> Root: Done
> Medic: Done
> Positioning: Done
> Follow Position: Executing
BTMerc_MedicSecondary: 2 steps
> Root: Done
> Idle: Executing
14
15
16
17
13
12
18

Version 2.0

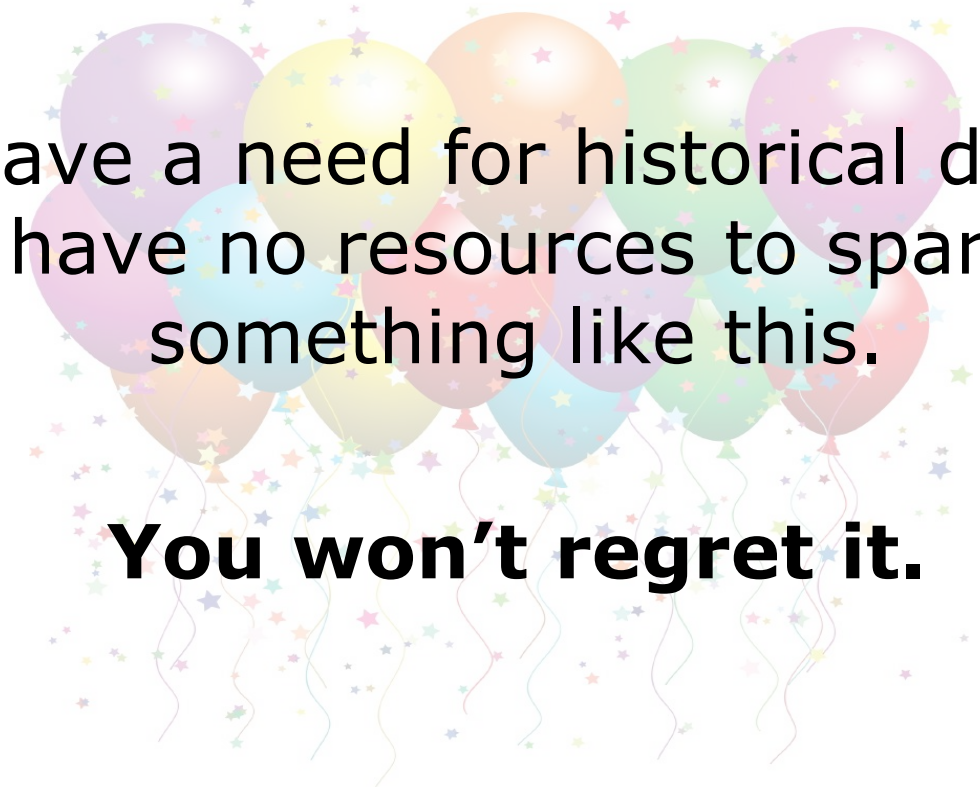
- Obvious next step is to visualize in external app
  - Though, something to be said about being in-game
- Stream over the network, “infinite” history
- Or write events to a DB, such as a free NoSQL key/value store
  - Visualize on the web or anywhere else
- Better visualization, animation/position rewind
  - In the works, bit-by-bit as necessary

Conclusions

- So much data: from any bug report, we have recent history for **all** active agents
- We see everything* that's happened on a remote dedicated server
- Engineers new to the team were able to jump in and track down tricky bugs in a fraction of the time… says Troy
- We observed and fixed bugs we weren't even looking for
- Replaced all the disjointed visualization junk we had before

* almost

Conclusions



If you have a need for historical debugging and have no resources to spare, try something like this.

You won’t regret it.