

Cross-Platform Determinism in Warhammer Age of Sigmar: Realms of Ruin

Bradley Pollard
Engineering Lead
Frontier Developments

Outline

- Introduction
- Protecting the programmer
- Cross-platform determinism
- Networking
- Post-mortem



Who are Frontier Developments?

- Founded in 1994 by David Braben
- Cobra™ engine
- Best known for park management games









Realms of Ruin

- Warhammer Age of Sigmar universe
- Released: Nov 17th, 2023
- Platforms: PC, PS5, Xbox
 Series X|S





Making our first RTS

- Frontier has never made an RTS before
- Building on prior GDC talks

Determinism:

- Common approach
- High learning curve
- Unfamiliar dev team

Goal: Build a robust deterministic simulation framework (DSim)



Why determinism?

Commands (inputs) generate state

- Network synchronisation
- Replays
- Anti-cheat

Inspired by these talks:

- 'Overwatch' Gameplay Architecture and Netcode GDC 2017
- Back to the Future! Working with Deterministic Simulation in 'For Honor' – GDC 2019



Network synchronisation

- Ultra-low bandwidth
- Low development overhead
- Limited multiplayer experience



Replays

- Simple to generate
- Debugging aid
- Development tool



Example of a looping replay



Anti-cheat

- Competitive multiplayer
- Detect local modifications
- Command validation



Determinism considerations...

Desynchronisations (desyncs):

- Caused by non-deterministic code
- State will diverge after
- Hard to debug!

- Separate gameplay state and visuals
- Programmers need to adjust

How can we help them?



Outline

- Introduction
- Protecting the programmer
- Cross-platform determinism
- Networking
- Post-mortem



Protecting the programmer

Built for gameplay programmers

Fail fast

Three steps to prevent desyncs...

Protecting the programmer

Component Compiler

Block nondeterministic types

Template metaprogramming

Define data access

Deterministic ordering

 Assert on startup that parallel reads and writes are ordered



About the DSim Framework

- Gameplay state only
- Entity Component System (ECS) model
- Components only contain data
- Systems only contain methods
- This made it easier to enforce determinism



Protecting the programmer

Component Compiler

Block nondeterministic types

Template metaprogramming

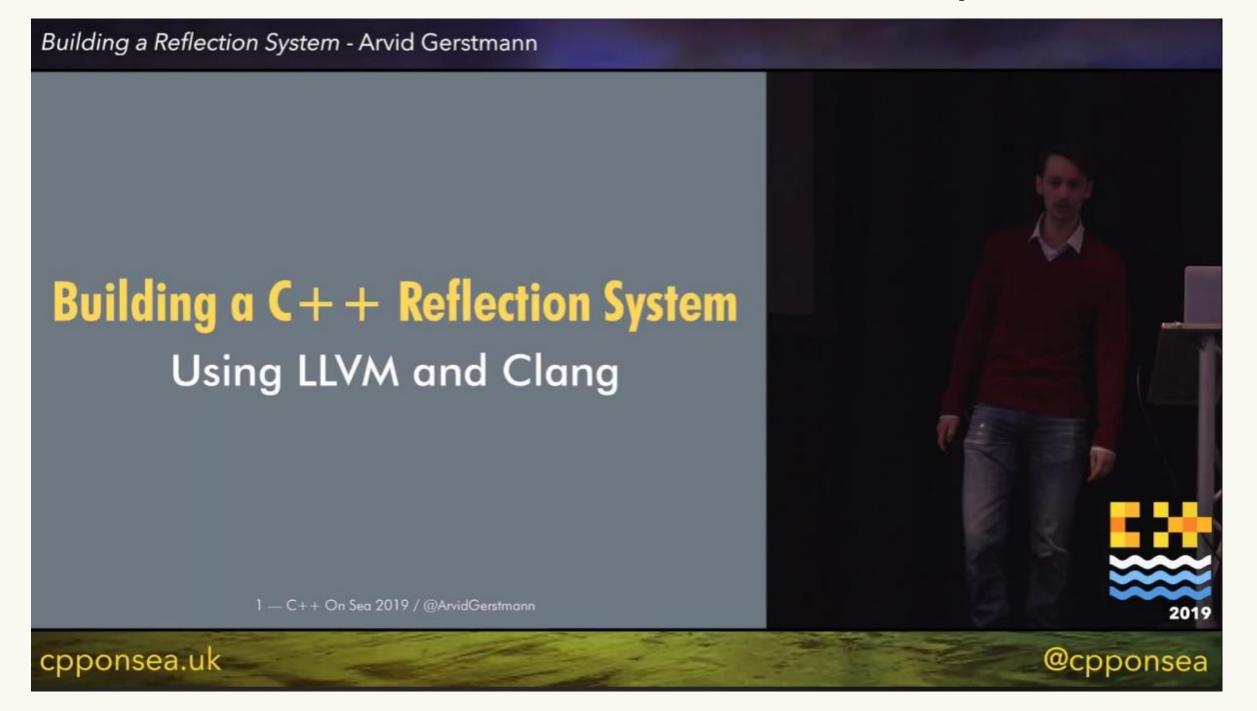
Define data access

Deterministic ordering

 Assert on startup that parallel reads and writes are ordered



Building a C++ Reflection System in One Weekend Using Clang and LLVM by Arvid Gerstmann



Component Compiler

Pre-compilation step:

- Generate an Abstract Syntax Tree (AST)
- Create reflection functions

Runtime:

Populate reflection database

Serialisation

Cyclic Redundancy Check (CRC) calculation



```
// Register ::FP::DSim::Test::SystemTestComponentA reflection data.
□namespace GeneratedReflection
     struct __FP__DSim__Test__SystemTestComponentARegistrar :
         public ReflectionRegistrar
         __FP_DSim__Test__SystemTestComponentARegistrar()
                                                                                                                                         storage.
             ReflectionClass& reflected = static_cast<ReflectionClass &>(*GetReflectionTypeImpl
                                                                                                                                        < ::FP::DSim::Test::SystemTestComponentA > )
               ( ReflectionTypeWrapper< ::FP::DSim::Test::SystemTestComponentA >{} ));
             RegisterConstructionFuncs( reflected, ConstructClass< ::FP::DSim::Test::SystemTestComponentA >,
                                                                                                                                       onentA, 1, 0, 0 > storage( [] ( auto self )
               DestructClass< ::FP::DSim::Test::SystemTestComponentA > );
             RegisterBaseClass( reflected, ReflectionRegistry::GetClass< ::FP::DSim::Component >() );
             RegisterField( reflected, 0, ReflectionName("m_value"), ReflectionRegistry::GetType< uint32 >(), offsetof( REFLECT_EXPAND
               ( ::FP::DSim::Test::SystemTestComponentA ), m_value ), 0 );
     namespace // Inaccessable global variable to run registrar code at startup
         __FP__DSim__Test__SystemTestComponentARegistrar g___FP__DSim__Test__SystemTestComponentARegistrar;
                                                                            ReflectionClass::TemplateArgIterator(nullptr),
                                                                             nullptr,
                                                                             ReflectionType::Flags::ProhibitAsMember
                                                                         return &reflected;
```

Reflection System

- A list of permitted types
- Includes primitives and container types



Compilation error!

```
121
            @brief Reflect primitive types
122
        REFLECT TYPE( bool )
123
        REFLECT_TYPE( char )
124
125
        REFLECT TYPE( int8 )
        REFLECT TYPE( uint8 )
126
        REFLECT_TYPE( int16 )
127
128
        REFLECT TYPE( uint16 )
129
        REFLECT_TYPE( int32 )
130
        REFLECT_TYPE( uint32 )
        REFLECT_TYPE( int64 )
131
        REFLECT TYPE( uint64 )
132
        REFLECT TYPE( fFPReal )
133
134
        REFLECT TYPE( fFPVector2 )
        REFLECT_TYPE( fFPVector3 )
135
        REFLECT TYPE( fFPVector4 )
136
137
        REFLECT TYPE( fFPRotationAboutY )
        REFLECT TYPE( fFPTransform2D )
        REFLECT TYPE( fFPTransformXZPlanePlusY )
        REFLECT_TYPE( fFPComplex )
        REFLECT_TYPE( ReflectionID )
        REFLECT_TYPE( FP::DSim::ComponentClassID )
```

Primitive types that are permitted in the DSim

Missing reflection type! `float` has not been reflected and is therefore not permitted within the DSim.



Protecting the programmer

Component Compiler

Block nondeterministic types

Template metaprogramming

Define data access

Deterministic ordering

 Assert on startup that parallel reads and writes are ordered



Template metaprogramming

```
⊟class SystemTest::VaryingSystemAll :
           public System
          VaryingSystemAll,
           ComponentList
              ComponentPtr< DSim::Test::SystemTestComponentA, ComponentAccess::Required >,
                                                                                                                                                System update
              ComponentConstPtr< DSim::Test::SystemTestComponentB, ComponentAccess::Required >,
           >, // Varying
          ComponentList
                                                                    // From SystemTemplated
                                                                    void Update(Query& i_query) override
                                                         44
          >, // Simulation
          ComponentList
                                                                        fAssert( i query.Size() == NumObjects );
                                                                        // Test to retrieve components on the returned objects
                                                         47
           > // Adhoc
                                                                        for ( ObjectList:: Value value : i query )
                                                                            ObjectList::Data const& componentTuple = value.Data();
System definition
                                                         51
                                                                            FP::DSim::ID const id = value.Key();
                                                         52
                                                                            // Write access - allowed:
                                                                            SystemTestComponentA* systemTestCompA = componentTuple.GetComponent<SystemTestComponentA>();
                                                         54
                                                                            // Read access - allowed:
                                                                           SystemTestComponentB const* systemTestCompBRead = componentTuple.GetComponent<SystemTestComponentB const>();
                                                                            // Write access to const component - Compilation error:
                                                                            SystemTestComponentB* systemTestCompBWrite = componentTuple.GetComponent<SystemTestComponentB>();
                                                                            // Read access of component system does not have access to - Compilation error:
                                                                            SystemTestComponentC const* systemTestCompC = componentTuple.GetComponent<SystemTestComponentC const>();
```



Making the compiler work for you

```
public:
21
22
           System()
23
               static assert(sizeof(T DerivedSystem) == sizeof(System),
25
                   "Systems should not contain any state, does the derived system have member variables?");
27
               static assert(TupleComponentPtrAllBaseOf<Component, typename SystemT<T DerivedSystem, T Varying, T Simulation,
                 T Adhoc, T SysInjectors>::VaryingComponentList>::Value,
                   "This System's VaryingComponentList contains a class that does not inherit from Component. Check you haven't
                      accidentally added a ProxyComponent to your tuple." );
29
               static assert(TupleComponentPtrAllBaseOf<SingletonComponent, typename SystemT<T DerivedSystem, T Varying,
                 T Simulation, T Adhoc, T SysInjectors>::SimulationComponentList>::Value,
                   "This System's SimulationComponentList contains a class that does not inherit from SingletonComponent. Check 🤻
                      you haven't accidentally added a SingletonProxyComponent to your tuple." );
               static_assert(TupleComponentPtrAllBaseOf<Component, typename SystemT<T_DerivedSystem, T_Varying, T_Simulation,
31
                 T Adhoc, T SysInjectors>::AdhocComponentList>::Value,
                   "This System's AdhocComponentList contains a class that does not inherit from Component. Check you haven't
32
                     accidentally added a ProxyComponent to your tuple." );
      ⊟#if DSIM DEBUG TOOLS
           // Make sure the derived system cannot hide any member variables by padding out the end of this class
           F ALIGN( 16u ) int32 m memberDetectionPadding[4];
       #endif
```

Protecting the programmer

Component Compiler

Block nondeterministic types

Template metaprogramming

Define data access

Deterministic ordering

 Assert on startup that parallel reads and writes are ordered



Why parallelism?

- Simulation runs at 16Hz
- Interpolate to higher framerates
- Simulation ticks could cause spikes
- Can we run Systems in parallel?



Parallel problems

- A vector for non-determinism
- Random numbers are tricky!
- Ordering matters
- Component access dependencies matter
- Avoid serial execution



Deterministic parallelism

```
Dclass SystemTest::VaryingSystemAll:

public System

VaryingSystemAll,

ComponentList

ComponentPtr< DSim::Test::SystemTestComponentA, ComponentAccess::Required >,

ComponentConstPtr< DSim::Test::SystemTestComponentB, ComponentAccess::Required >,

ComponentList

Note: Test::SystemTestComponentB, ComponentAccess::Required >,

Note: Test::SystemTest::SystemTestComponentB, ComponentB, Component
```

- Systems are templated on the Components they access
- We can use this information to our advantage



Traversing the Schedule Graph

- Cobra™ provides an advanced scheduler
- Each System is scheduled as a task
- Leverage the schedule graph

```
const auto assertDependency = [&] ( ComponentClassID i_id )

if ( !m_query->IsConstAccess(i_id) )

{

if ( !m_query->IsConstAccess(i_id) )

{

// Assert there are dependencies between this and all other systems that have access (read or write) to the same component for ( SystemBase* system : i_systems )

if ( system != this )

for ( SystemBase* system : i_systems )

{

if ( system != this )

}

// Assert dependency for each ComponentPtr type in the query

fForEach( m_query->GetRequiredComponentTypes().Begin(), m_query->GetRequiredComponentTypes().End(), assertDependency );

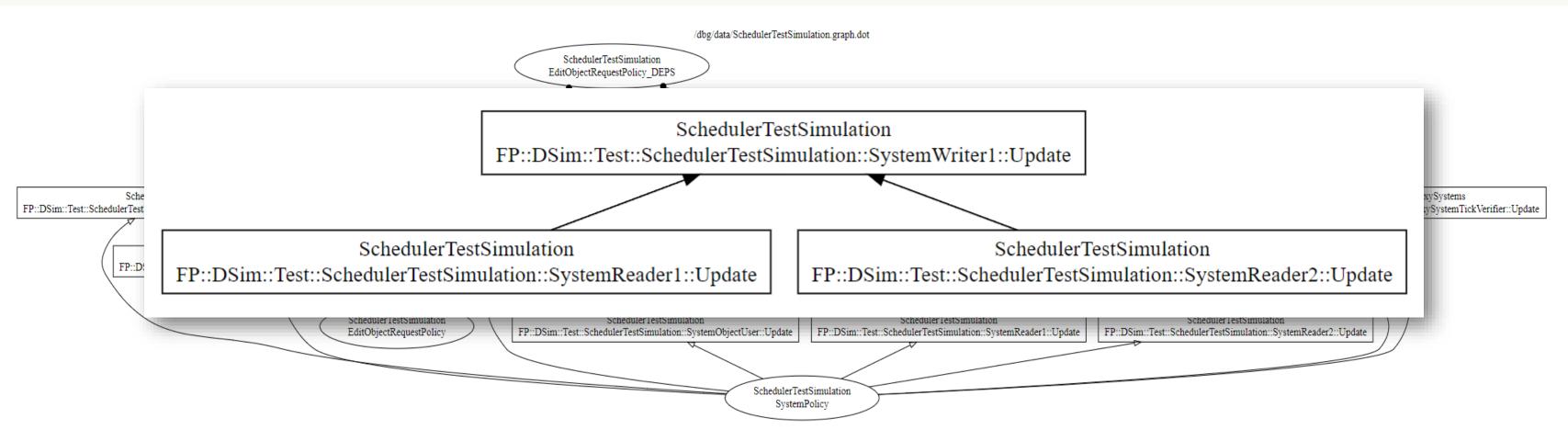
fForEach( m_query->GetOptionalComponentTypes().Begin(), m_query->GetSingletonComponentTypes().End(), assertDependency );

fForEach( m_query->GetAdhocComponentTypes().Begin(), m_query->GetSingletonComponentTypes().End(), assertDependency );

fForEach( m_query->GetAdhocComponentTypes().Begin(), m_query->GetAdhocComponentTypes().End(), assertDependency );
```

Visualising the graph

- Visualisations are key
- Scheduler can export graphs to show dependencies



Random numbers

Machine A

Cim.

RNG(42)

Syste n A

A = 10

System B

B = 20

System C

C - 4

System D

D = 7

А	В	C	
10	20	4	7

Madine B

Simulation

RNG(42)

Sy en A

A = 1

System B

B = 20

stem D

D = 4

System C

C = 7

A	В	С	D
10	20	7	4

Random numbers

Machine A

Simulation

RNG(42)

System A

RNG(10)

A = 5

System B

RNG(20)

B = 12

System C

RNG(4)

C =

System I

RNG(7)

D = 21

А	В	С	D
5	12	9	21

Machine B

Simulation

RNG(42)

System /

RNG(10

A = 5

Syster B

RNG(0)

B = 12

System D

RNG(7)

D = 21

System C

RNG(4)

C = 9

	В	С	D
5	12	9	21

Outline

- Introduction
- Protecting the programmer
- Cross-platform determinism
- Networking
- Post-mortem



Floating point, yay or nay?

- Floating point can be used...
- but requires *extreme* care
- We targeted 4 different platforms
- Take the performance hit use fixed-point!



Floating point in detail

Could we quantize/round? Probably not

- Multiple platforms = multiple compilers
- Robustness was the priority

https://randomascii.wordpress.com/2013/07/16/floating-point-determinism/



Data model differences

- We shipped on Linux and PS5
- Fundamental type size differences
- Component Compiler can assist us again!

```
# We only support 64 bit platforms. This is a bit tricky to check for, but we shouldn't ever be running on 32 bit or lower anyway.
                               LLP64,
dataModelProperties = [
                               "int8",
     "signed char",
                                           "int8" ),
     "unsigned char",
                               "uint8",
                                           "uint8" ),
                               "int16",
                                           "int16" ),
     "short",
                               "int16",
                                           "int16" ),
     "short int",
                                           "int16" ),
     "signed short",
                               "int16",
     "signed short int",
                               "int16",
                                           "int16" ),
     "unsigned short",
                               "uint16",
                                           "uint16" ),
     "unsigned short int",
                               "uint16",
                                           "uint16" ),
     "int",
                               "int32",
                                           "int32" ),
     "signed",
                               "int32",
                                           "int32" ),
     "signed int",
                               "int32",
                                           "int32" ),
                               "uint32",
                                           "uint32" ),
     "unsigned",
     "unsigned int",
                               "uint32",
                                           "uint32" ),
     "long",
                               "int32",
                                           "int64" ),
     "long int",
                               "int32",
                                           "int64" ),
     "signed long",
                               "int32",
                                           "int64" ),
    "signed long int",
                               "int32",
                                           "int64" ),
      "unsigned long",
                               "uint32",
                                           "uint64" ),
     "unsigned long int",
                               "uint32",
                                           "uint64" ),
                               "int64",
                                           "int64" ),
     "long long",
     "long long int",
                               "int64",
                                           "int64" ),
    "signed long long",
                               "int64",
                                           "int64" ),
     "signed long long int",
                               "int64",
                                           "int64" ),
      "unsigned long long",
                                "uint64",
                                           "uint64" ),
     "unsigned long long int", "uint64",
dataModelProperties = sorted(dataModelProperties, key=lambda i: len(i[0]), reverse=True ) # Sort by type length from longest to shortest
for dataModelProperty in dataModelProperties:
   # We need to make sure to only match types and not substrings part of a longer type name
   typeName = re.sub("(^|[^a-zA-Z0-9])(" + dataModelProperty[0] + ")([^a-zA-Z0-9]|\$)", "\1" + dataModelProperty[dataModelType] + "\\3", typeName)
```



Evaluation order

- A compiler issue, rather than a platform one
- Take for example MyFunc(RandInt(), RandInt())
- No guarantee which call to RandInt() happens first!
- No smart tricks here



Padding bytes

- The most common form of desync
- "False positive" desyncs
- Serialising non-trivial types is slow
- Component Compiler initialises padding bytes



Initialising padding bytes

- Component Compiler will initialise padding
- Allows for Components to be serialised trivially
- Padding bytes are then deterministic

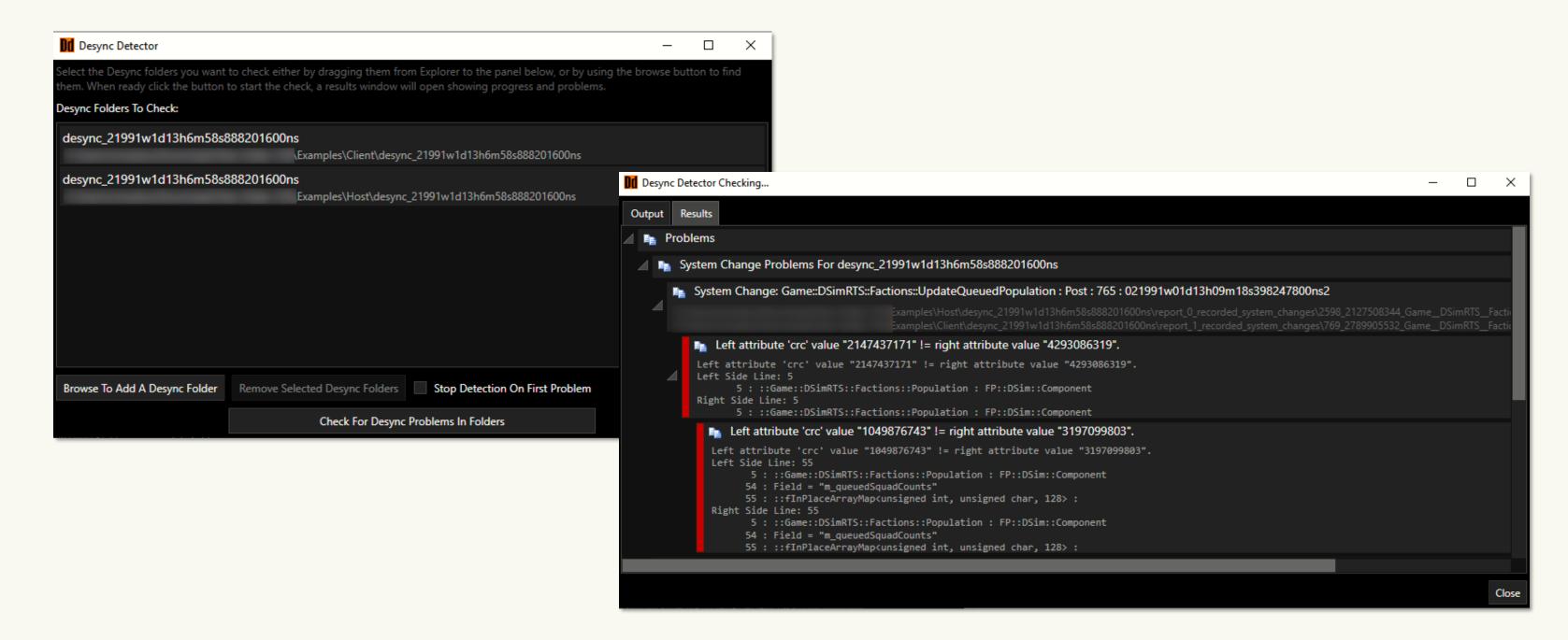


Tooling

- How to handle desyncs?
- Appropriate tools needed
- 'Desync Detector'
- A tool to analyse XML state dumps as a diff



Desync Detector





Outline

- Introduction
- Protecting the programmer
- Cross-platform determinism
- Networking
- Post-mortem



Dedicated server

- Another platform Linux!
- Preferable over peer to peer
- Blame clients for desyncs
- Gather debug info from the server



Optimising for latency

- Very little data is sent each frame
- Acks add too much latency
- Resend everything!
- Keep track of last confirmed tick

 Inspired by Gaffer On Games – Deterministic Lockstep https://gafferongames.com/post/deterministic lockstep/



Redundant retransmission

Time

Tick 10	Tick 11	Tick 12	Tick 13	Tick 14	Tick 15	Tick 16	Tick 17
Input A	Input A	Input A	Input A	Input B	Input B	Input C	
		Input B	Input B	Input C	Input C		
			Input C				

9	9	9	9	10	10	12	13
				Ack		Ack	Ack

Buffering

- Command buffer
- Buffer size increases during bad network conditions
- Increases input delay
- Speed up to shrink buffer



Future work

- Rollback
- Further optimisation needed
- Reduce the size of a snapshot

- "8 Frames in 16ms" by Netherrealm Games' Michael Stallone
 - https://gdcvault.com/play/1025021/8-Frames-in-16ms-Rollback



Outline

- Introduction
- Protecting the programmer
- Cross-platform determinism
- Networking
- Post-mortem

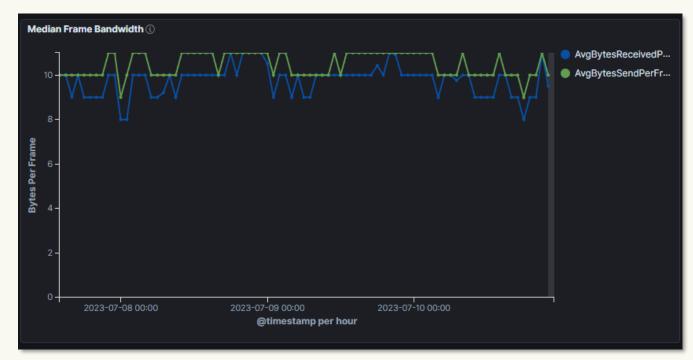


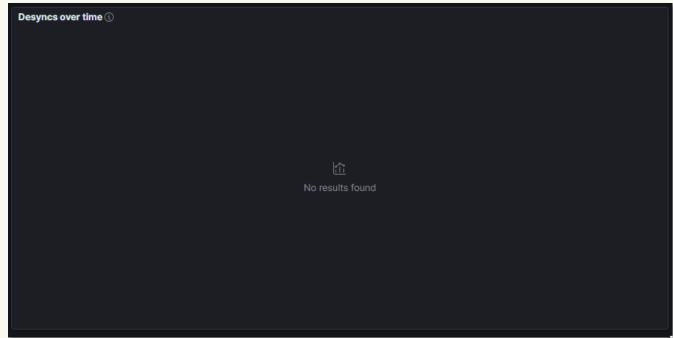
Open Beta post-mortem

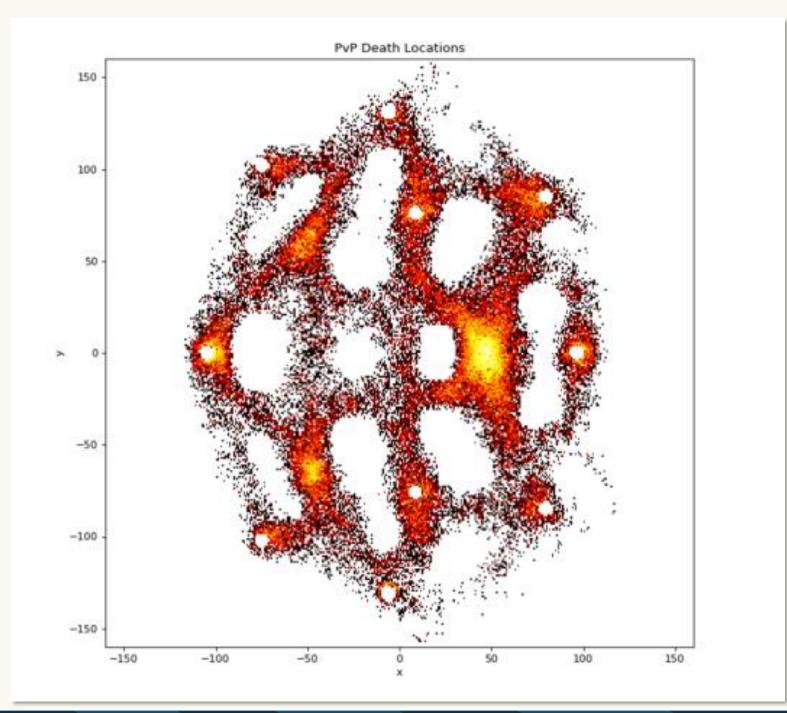
- Frontier's first ever cross-platform test
- Xbox Series X|S, PlayStation 5 and PC
- 50,000+ games played



Open beta post-mortem

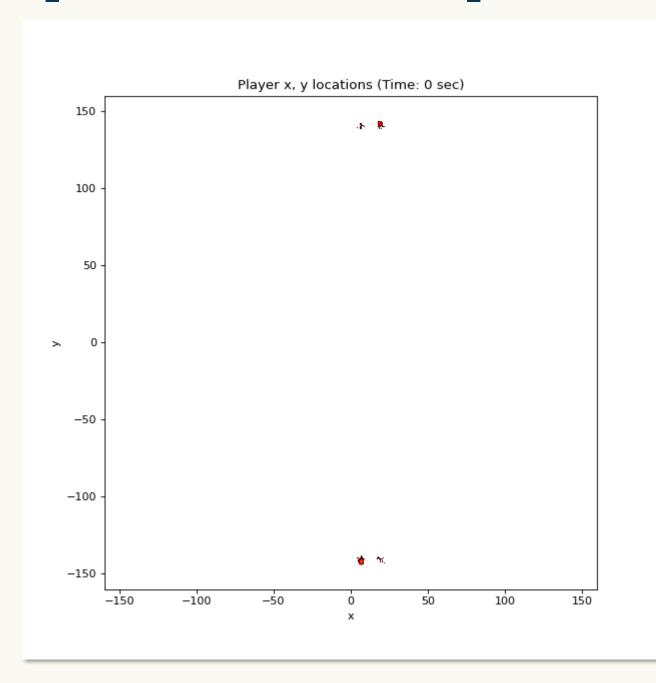


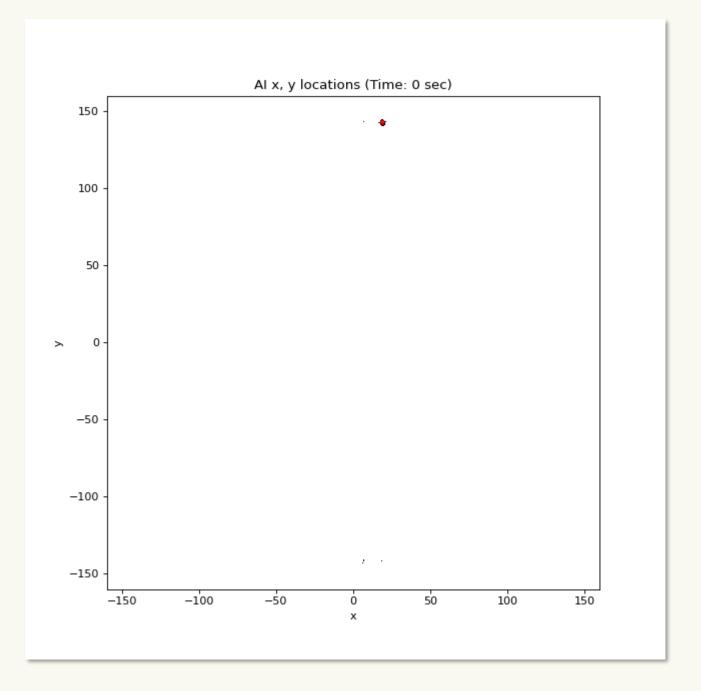






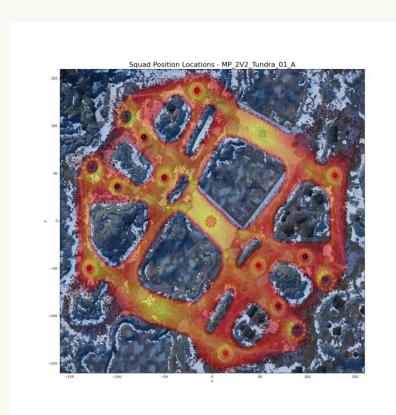
Open beta post-mortem





Release post-mortem

Story remains the same after release

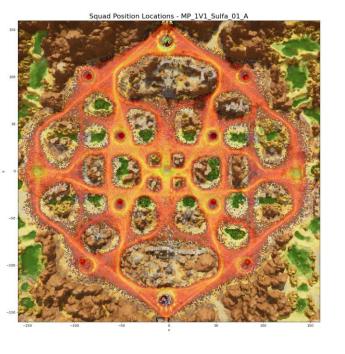


-130 -130 -50 0 10 110 120

2v2 telemetry

Squad Position Locations - MP_1V1_Sulfa_01_A

Squad Death Locations - MP_1V1_Sulfa_01_A





Key takeaways

- Protect the programmer to avoid desyncs
- Cross-platform determinism is possible given the right constraints
- Proper tooling is critical
- Determinism can provide benefits



Thanks to the team at Frontier

Core Code team:

- Menno Markus
- Daniel Whittaker

And the Shared Technology Group

Talk advisors:

- Matt Simper
- Owen McCarthy

Questions?

Email additional questions to:

bpollard@frontier.co.uk

