

Practical HFSM

Andrew Gresyk @ Splash Damage

Hi!

TOM CLANCY'S
SPLINTER CELL
CONVICTION

ASSASSIN'S
— CREED —
REVELATIONS

BATTLEFIELD
4

SNIPER ELITE III



ZOMBIE
ARMY
TRILOGY



SNIPER ELITE 4



Hi!



Anniversary



C++ London
London



PAST MEETUP

Meeting[1] // "A Polymorphic Value Type" & "Anti: Signed for Life"



Wednesday, January 25
19:00

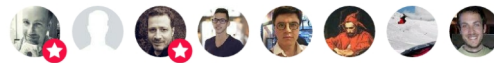


Skills Matter | Code Node
10 South Place EC2M 2RB, London, United Kingdom



Hosted by Phil Nash, Jonathan Coe, Tony Lewis

80 people went




After the phenomenal success of our first meet-up we now have another pair of talks lined up for January.

Once again we're being hosted by the great folks at SkillsMatter - so please register on the SkillsMatter page too as they require this for entry (<https://...>)

[Read more](#)


Evolution





C++ London
London

PAST MEETUP


Hierarchical State Machine Framework for Videogames

 Tuesday, April 11
19:00

 Skills Matter | Code Node
10 South Place EC2M 2RB, London, United Kingdom

 Hosted by Phil Nash, Andrew Gresyk

96 people went

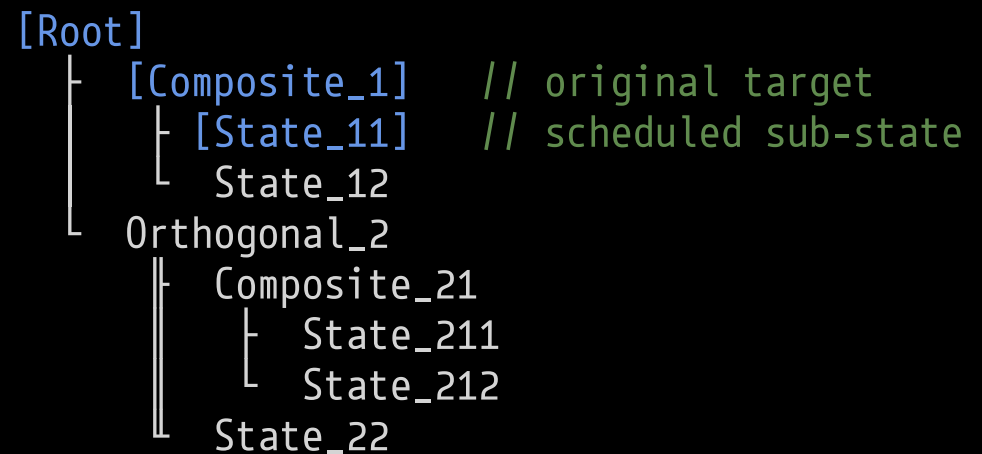
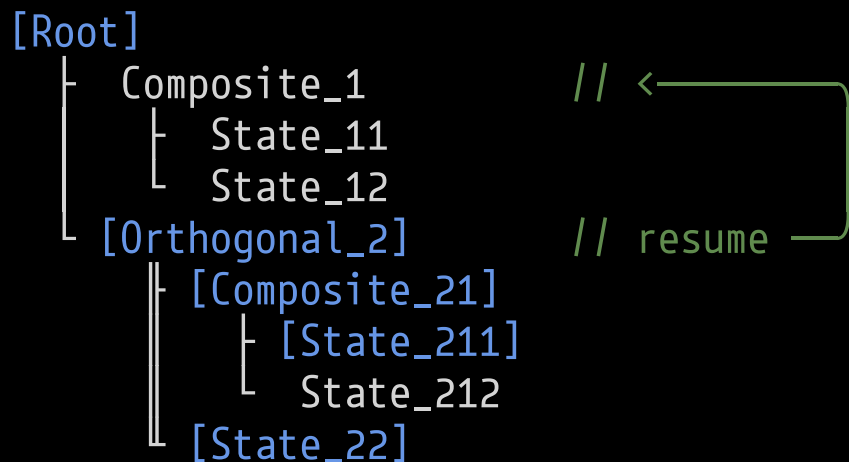
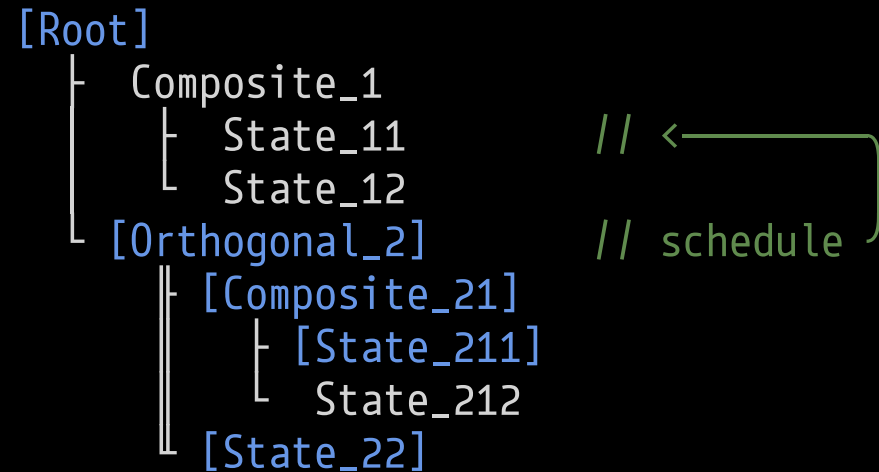
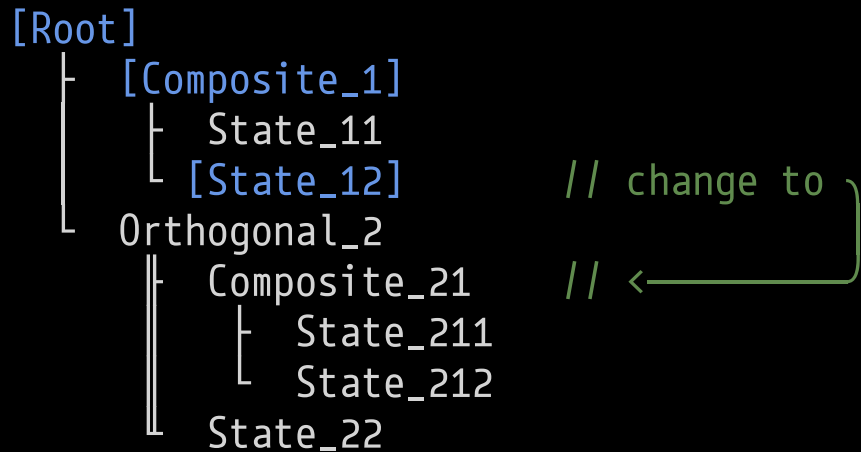


This month we have just one speaker, Andrew Gresyk, with a longer talk that relates to games programming, which has been an often requested topic!

(As before, please also register on the SkillsMatter page (<https://skillsmatter.com/meetups/9200-hierarch...>))

[Read more](#)

Hierarchy+Transitions



Complexity Intuition

There's a math concept that resembles a feature in terms of complexity and interaction:

- * Feature \sim Matrix
 - * State Variable \sim Matrix Component
 - * Conditional Expression on State Variables \sim Matrix Component Product
 - * Feature Composition / Interaction \sim Matrix Multiplication
-

Feature composition using plain state variables (feels like):

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \times \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31} \\ a_{21} \times b_{11} + a_{22} \times b_{21} + a_{23} \times b_{31} \end{bmatrix} \begin{array}{l} // a_{11}, b_{11} \sim \text{State Variables} \\ // a_{11} \times b_{11} \sim \text{Conditional Expression} \end{array}$$

Feature composition using FSM framework (feels like):

$$A \times B = C$$

Disclaimers

- Game code was sacrificed to showcase FSMs!

Acknowledgments



SFML

- <https://www.sfml-dev.org/>

Acknowledgments



<https://didigameboy.itch.io/jambo-jungle-free-sprites-asset-pack>

HFSM

- header-only
- template
- without external dependencies
 - except standard c++ headers
- fully static
 - no dynamic memory allocations
- gamedev friendly
- minimal boilerplate
- scalable

Minimal State Machine

```
#include "kaynine/machine/machine.hpp"

struct Context { /* shared variables */ }; // fsm data interface
using M = k9::Machine<Context>; // just a tagged namespace

struct HeadState : M::Base { /* state methods */ };
struct State1 : M::Base { /* state methods */ };
struct State2 : M::Base { /* state methods */ };
using FSM = M::Root<HeadState, SState1, State2>; // root region

void example() {
    Context _; // context instance
    FSM fsm(_); // state machine instance
    fsm.update(0.0f); // updates, transitions, everything
};
```

Structural Diagram

- Code:

```
struct TopState : M::Base {};  
struct State1   : M::Base {};  
struct State2   : M::Base {};
```

```
using FSM = M::Root<TopState,  
                  State1,  
                  State2  
>;
```

- Diagram:

```
Root(TopState)  
├ State1  
└ State2
```

State Overridables

```
struct State : M::Base {  
    void substitute(Control&, Context&);  
  
    void enter(Control&);  
    void leave(Context&);  
  
    void update(Control&);  
    void transition(Control&, Context&);  
  
    void linger(Control&, Context&);  
  
    template <typename TEvent>  
    void react(const TEvent&, Control&, Context&);  
};
```

State Transitions

```
struct AnotherState : M::Base {};
```

```
struct State : M::Base {  
    void transition(Control& control, Context&, const Time) const {  
        control.schedule<AnotherState>();  
        control.changeTo<AnotherState>();  
        control.resume<AnotherState>();  
        control.catchUp<AnotherState>();  
    }  
};
```

External Transitions

```
struct State1 : M::Base {};  
struct State2 : M::Base {};  
  
void example() {  
    Context _;  
    M::PeerRoot<State1, State2> fsm(_); // State1 is activated  
                                        // initially  
  
    fsm.schedule<State2>();  
    fsm.changeTo<State2>();  
    fsm.resume<State2>();  
    fsm.catchUp<State2>();  
};
```


Event Handling

```
struct HandledEvent {}; struct UnhandledEvent {};
```

```
struct State : M::Base {  
    void react(const HandledEvent&, Control&, Context&, const Time) {  
        // control.changeTo<...>();  
    }  
};
```

```
void example() {  
    Context _;  
    M::PeerRoot<State> fsm(_);  
  
    fsm.react(HandledEvent{}, 0.0f); // State::react() is called  
    fsm.react(UnhandledEvent{}, 0.0f); // no handlers exist, ignored  
};
```

Scene 0

Scene 1

Scene 2

Scene 3

Scene 4

Scene 5

The End

@andrew_gresyk
<https://github.com/andrew-gresyk>