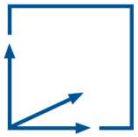
Development of a High-Performance Engine to Execute Statecharts for Games

Matthias Ellerbeck

2021/04/29



Final: Bachelor Informatics: Games Engineering

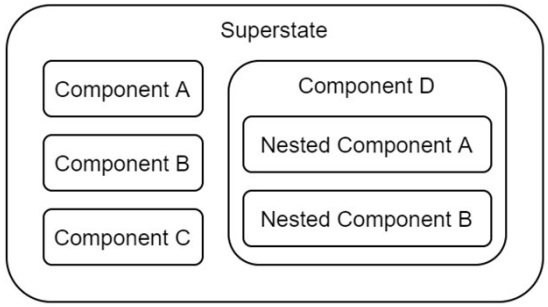
Supervisors: Prof. Gudrun Klinker, Phd; Daniel Dyrda

Introduction

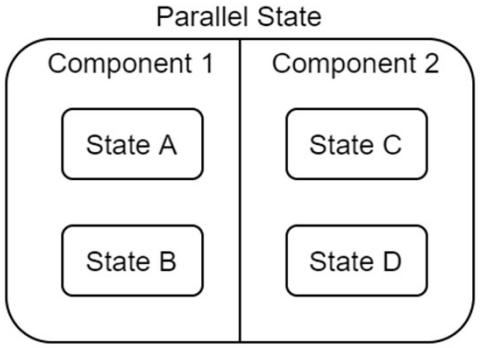
- Games are reactive systems
- Game components (e.g. character controllers) are reactive systems
- Statecharts are designed to model reactive systems

Statecharts

Hierarchy



Orthogonality



Statecharts

- Configuration
  - Set of active states
  - Valid if certain requirements are met
  - Knowledge of basic states in configuration sufficient
    → Basic Configuration

- Status
  - Current state of a statechart
  - Contains configuration and events (and more)
  - Change between two statuses: Step

Related Work

- Statecharts in standards
  - UML Behavior State Machines
    Add initial pseudo-states
  - SCXML
    Statecharts in XML format
- Statechart execution
  - 3 - 460 times slower than compiling
  - Still sufficiently fast

Related Work

- Statecharts in games
  - DEAL
    - SCXML
    - Natural language processing
    - Convincing interactive dialogue
  - AI
    - Statechart models AI
    - Components can be reused

Goals of this Thesis

Execute statecharts

- In Unity
- At least 1000 statecharts
- In realtime (< 16.6 ms)
- Usable in a game

Critical Issues

Problems

- Statecharts are complex
- Many statecharts may be active simultaneously
- Games usually require 60 frames per second

Of interest

- Average execution time
- Maximum execution time

Performance Testing

- Test Computer
  - AMD Ryzen 9 3950X (16 Cores, 3.5 – 4.7 GHz)
  - 4x G-Skill F4-2133C15-8GRB (32 GB total)
  - Microsoft Windows Home 64 Bit 20H2
- C# Stopwatch
  - Measurement overhead not measurable
  - 100 ns resolution
  - At least 10,000 samples per run, data from the last 5000 samples
  - Result: minimum, maximum, mean, median of the samples
- Complex test statechart
- Built project

Naive Implementation

MonoBehaviour Scripts

- StatechartInstance
  - Contains status
  - Contains action bindings
  - Contains stepping code
- StatechartEngine
  - Singleton
  - Contains lists of instances
  - Performs steps

Naive Implementation

Asset Script: Statechart

- Contains statechart in custom format
- Parses XML file to custom format

Naive Implementation

Other Classes

- Statechart elements
  - Basic state, compound state, parallel state, pseudo-state, and superclasses
  - Transitions
- Event
- Guard
- Status

Naive Implementation

Step algorithm

- Prepare step
- Search active transitions
- Validate transitions and fill sets
- Execute actions, compute new configuration

Naive Implementation

Algorithm measurements in µs for a single instance

	Minimum	Maximum	Mean	Median
Prepare	1.4	237.9	2.3	2.0
Search	0.9	133.6	2.1	1.7
Validate	1.7	232.2	6.9	4.9
Execute	2.2	251.7	6.6	4.3

Optimization: Caching

Cache computed data

- Scope of transitions
- Name of nodes

Optimization: Caching

Algorithm measurements after caching in µs

	Minimum	Maximum	Mean	Median
Prepare	1.5	152.2	2.3	2.1
	+0.1	-85.7	0.0	+0.1
Search	0.8	160.6	2.1	1.8
	-0.1	+27.0	0.0	+0.1
Validate	1.8	276.9	6.4	4.7
	+0.1	+44.0	-0.5	-0.2
Execute	1.5	150.4	4.6	3.3
	-0.7	-101.3	-2.0	-1.0

Optimization: Locality

Keep data in the CPU cache: Classes to structs

- Node
  - 160 Bits / 20 Bytes
  - One struct for all types; contains type identifier
- Statechart
  - Array of nodes
  - Array of node component indices
  - Array of transitions
  - Indices instead of references
- Statechart Instance
  - Indices instead of references

Optimization: Locality

Algorithm measurements after locality improvements in µs

	Minimum	Maximum	Mean	Median
Prepare	1.2	126.0	1.9	1.7
	-0.3	-26.2	-0.4	-0.4
Search	0.5	146.2	1.8	1.7
	-0.3	-14.2	-0.3	-0.1
Validate	1.2	226.6	4.9	3.9
	-0.5	-50.3	-1.5	-0.8
Execute	1.2	222.1	2.4	1.7
	-0.5	+71.7	-2.2	-1.6

Optimization: Other

- Remove dictionary lookups for guards:
  String keys to array indices
- Reduce dynamic array memory allocations:
  Initialize with minimum capacity
- Reduce garbage collector runs:
  Reduce reference type allocations

Optimization: Other

Algorithm measurements after other improvements in µs

	Minimum	Maximum	Mean	Median
Prepare	0.8	25.1	1.5	1.5
	-0.4	-100.9	-0.4	-0.2
Search	0.4	130.5	1.6	1.6
	-0.1	-15.7	-0.2	-0.1
Validate	1.1	227.7	4.0	3.5
	-0.1	+11.0	-0.9	-0.4
Execute	1.0	234.5	1.8	1.4
	-0.2	+12.4	-0.6	-0.3

Discussion

Algorithm measurements after all optimizations in µs

	Minimum	Maximum	Mean	Median
Prepare	0.8	25.1	1.5	1.5
	-0.6	-212.8	-0.8	-0.5
Search	0.4	130.5	1.6	1.6
	-0.5	-3.1	-0.5	-0.1
Validate	1.1	227.7	4.0	3.5
	-0.6	-5.2	-2.9	-1.4
Execute	1.0	234.5	1.8	1.4
	-1.2	-17.2	-4.8	-2.9

Discussion

Engine measurements after all optimizations in µs

	Minimum	Maximum	Mean	Median
Engine naive	14825.9	18871.7	16536.1	16529.2
Engine optimized	7667.6	10220.0	8872.0	8874.0
Difference	-7158.3	-8651.7	-7664.1	-7655.2
	-48.3%	-45.8%	-46.3%	-46.3%

Discussion

Problems:

- Test statechart tends heavily towards a single state
- Measurements valid for single statechart with many instances

Limitations:

- Pesudo-states in certain configurations may produce invalid configurations
- Step does not isolate statecharts

Future Work

- Reduce struct sizes
- Rework actions
- Finish threading
- Improve algorithm
- Investigate GPUs

Conclusion

- Engine to execute statecharts in Unity
- Can execute 1000 statecharts in 8.8 ms
- Optimizations used:
  - Cache results
  - Improve locality
  - Avoid dictionary lookups/complex hash functions
  - Avoid unnecessary allocations
- Code available at https://github.com/file-not-found42/StatechartEngine

List of References

1. D. Harel and A. Pnueli. On the Development of Reactive Systems. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.
2. David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
3. David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Trans. Softw. Eng. Methodol.*, 5(4):293–333, October 1996.
4. Jenny Brusk, Torbjörn Lager, Anna Hjalmarsson, and Preben Wik. DEAL: Dialogue Management in SCXML for Believable Game Characters. In *Proceedings of the 2007 Conference on Future Play*, Future Play ’07, page 137–144, New York, NY, USA, 2007. Association for Computing Machinery.
5. Christopher Dragert, Jörg Kienzle, and Clark Verbrugge. Toward High-Level Reuse of Statechart-Based AI in Computer Games. In *Proceedings of the 1st International Workshop on Games and Software Engineering*, GAS ’11, page 25–28, New York, NY, USA, 2011. Association for Computing Machinery.

List of References

6. Edzard Höfig. *Interpretation of Behaviour Models at Runtime: Performance Benchmark and Case Studies*. Doctoral thesis, Technische Universität Berlin, Fakultät IV - Elektrotechnik und Informatik, Berlin, 2011.
7. Microsoft Corporation. Hashset class. https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.hashset-1?view=netstandard-2.0, 2021. Accessed: 2021-04-09.
8. Microsoft Corporation. Stopwatch class. https://docs.microsoft.com/en-us/dotnet/api/ system.diagnostics.stopwatch?view=netstandard-2.0, 2021. Accessed: 2021-03-31.
9. Microsoft Corporation. Structure types (c# reference). https://docs.microsoft.com/en-us/dotnet/csharp/languagereference/builtin-types/struct, 2021. Accessed: 2021-04-09.
10. Object Management Group, 9C Medway Road, PMB 274 Milford, MA 01757 USA. *OMG® Unified Modeling Language®* (OMG UML®), 2.5.1 edition, 2017.
11. W3C. State Chart XML (SCXML): *State Machine Notation for Control Abstraction*, w3c recommendation 1 september 2015 edition, September 2015. Available at https://www.w3.org/TR/2015/REC-scxml-20150901/.