Project Evaluation Report

Artificial Intelligence for Games

Academy of Interactive Entertainment

Lewis Comstive, July 2021

# Project Overview

*Slime Survival™* is a simulation involving innocent slime creatures that are just trying to live in harmony, until *skeletons* start attacking and destroying their population!

Initial development of the application went well, which was then stretched out over many tedious hours with minor bugs and issues. With the plan to implement behaviour trees as the “brain” of the NPCs, accompanied with the A\* (AStar) algorithm for pathfinding around a grid-based map, I went full steam ahead and developed the pathfinding first – to stress test was a 1000x1000 unit grid that a single point could use A\* to navigate to any other point within a few frames.

Next on the list was implementing behaviour trees, which had very little information on implementation but rather *a lot* of higher-level concepts (predominantly Unreal Engine’s system…).  
Gamasutra came in clutch as per usual, with [Chris Simpson’s article](https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php) it started making more sense and I was able to develop (*what I believe is*) a solid, flexible framework that used a hashmap per tree as a contextual pool of information for each node (*akin to a blackboard, but per AI instead)*.

While trying to test the behaviour tree I realised that a physics library would be beneficial, so [*Box2D*](https://box2d.org/) was added as a submodule to the GitHub repository; now the basic game framework had collisions and the ability to raycast, which was then implemented in a behaviour tree node.

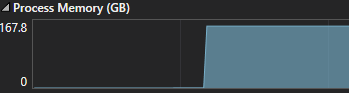
With the blackboard implemented and some chicken sprites running after each other using viewcones with raycasting, everything was coming to life. Next was to combine the tree with pathfinding, with absolutely no idea on how to do so...

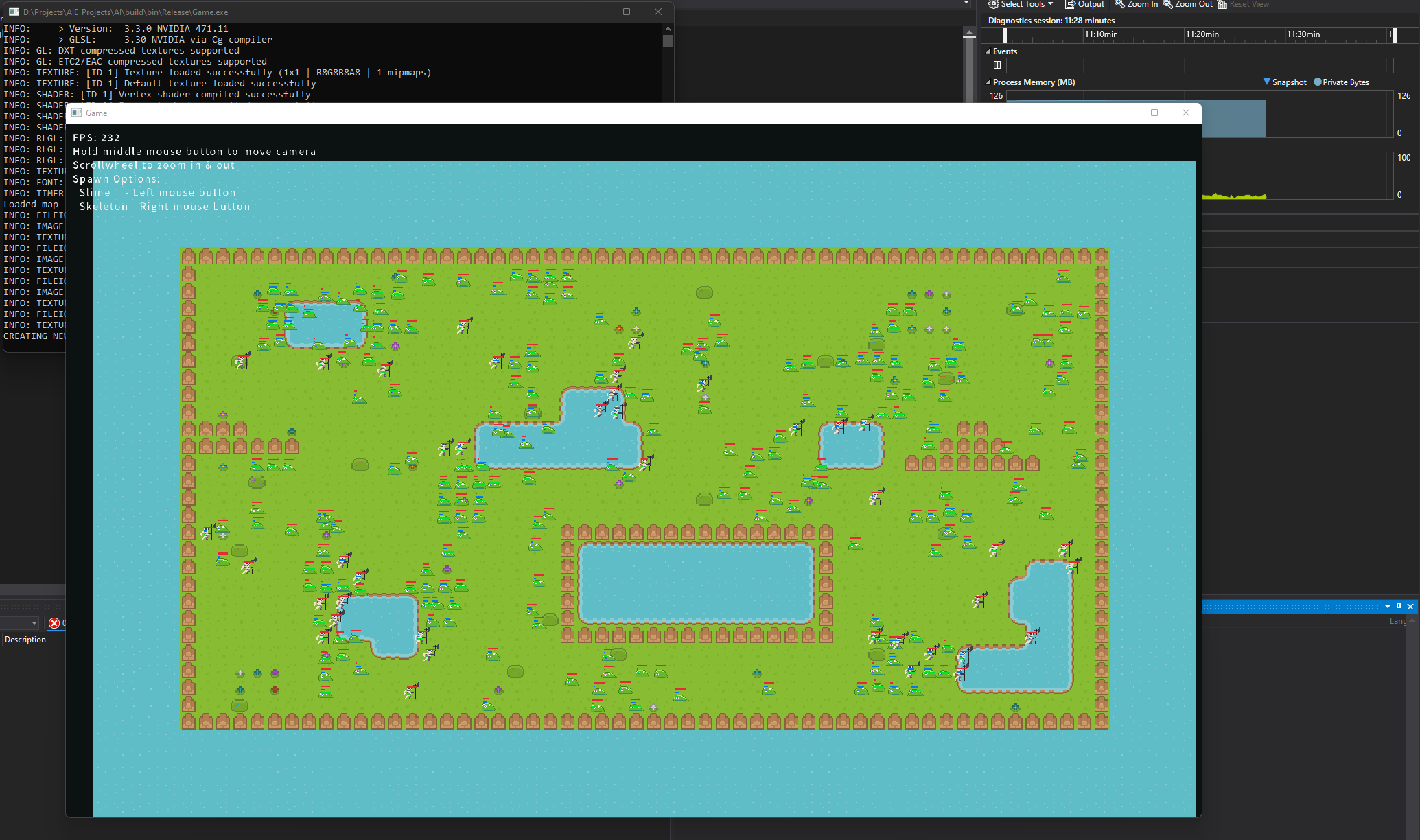
After many, many tedious hours of trial and error the behaviour nodes now had options for navigation via pathfinding, as well as finding a path to certain other entities within the scene.

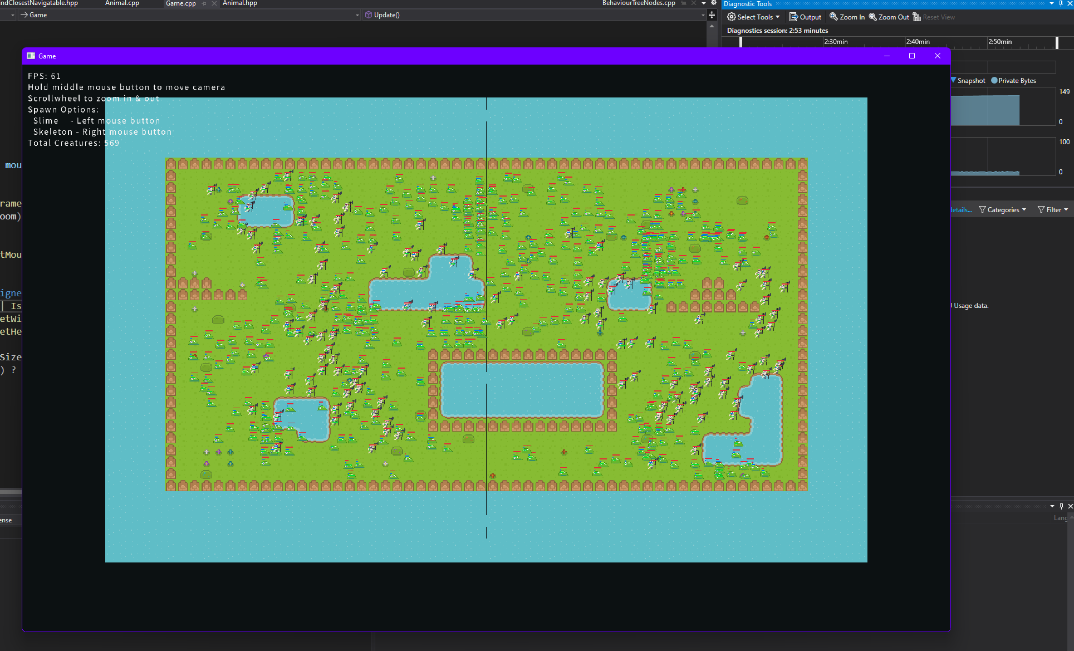
With enough time an animation system was planned to be implemented by changing a value in the contextual map, akin to states, that the sprite would read and change the spritesheet offset it was reading from; but alas, spending around 60 hours on this project over two weeks I had to let personal life commitments take priority closer to the submission date.

There is a bug that has lingered throughout the codebase for the past week or so that has caused the application to crash randomly, but *only* in release mode on the Windows platform. Tests have been done in Debug mode and Release mode w/ Debug info, but the crash only happens when there are *no debug symbols loaded* – the issue was not present while testing on an M1 Mac OS system.  
Interestingly, after satisfying some pedantic errors from the clang compiler on Mac OS then pushing the codebase back to Windows the issue has been resolved.

Another bug that occurred was very high memory usage, for instance after spawning about a hundred units the memory usage rose to sometimes a gigabyte or more – once Visual Studio crashed and reported 164GB used by the application (on a 32GB memory machine)!  
The issue was resolved thanks to the magic of pointers, and now the simulation has a consistent sub-150MB memory footprint during stress-testing.

  
A once-off issue..

  
Memory usage sitting around 100MB in Release mode with many units pathfinding



Stress-testing the simulation – 569 creatures at ~60 FPS

# Performance Analysis

Using sizeof in C++ revealed that the Animal class holds 296 bytes of data, which includes a behaviour tree (& context map), an instance of A\* and all parameters.  
If the simulation ran with 1,000 animals, this would only use ~296KB of memory.

The pathfinding should not be done every frame, so it is only calculated at the beginning of navigation or every 1.5 seconds thereafter, incase the target has moved (e.g. predator seeking prey).

To improve performance of the application, multithreading may be utilized for the A\* searching (*which was tested, but unsuccessful due to time constraints and lack of knowledge*).

The algorithms used within the simulation appeared to be efficient, the A\* especially when stress-tested during the beginning of development.

The final simulation was stress-tested and could surpass 500 agents navigating before hitting under 30FPS consistently (*ignoring the stuttering when they all went to calculate a new A\* path within a few frames of each other*)

# Future Improvements

Originally there were going to be more creatures, such as an omnivore, and with enough time also the ability for animals to mate and produce offspring with slight variations and mutations.

An animation state value in the behaviour tree context was in the works for the sprite animation system to read, but alas time was not on my side.

Overall, I believe behaviour trees are an extremely flexible system and will most definitely be using them again in the future. A visual node-based editor could make creation of game AI incredibly intuitive, so I may attempt to implement that in my free time for an existing engine such as Unity3D.