Sampo Vänninen, Sakari Ropponen, Emma Turkulainen, Joona Sauramäki

Boid Simulator 3D

https://courses-git.comnet.aalto.fi/CPlusPlus/steering-behaviours-2019-2

Overview

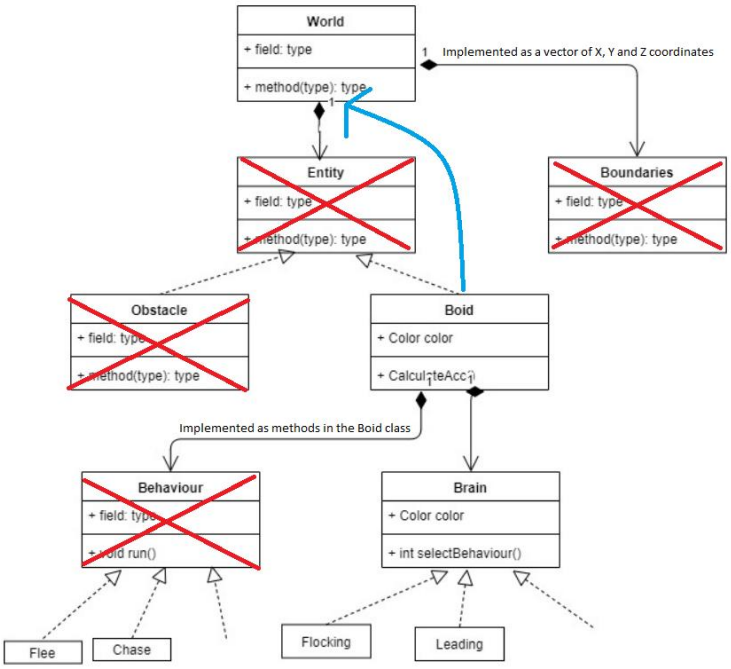
The program is a 3D Boid simulator. It can be used to observe Boid behavior in an cuboid area, for example flocking, wandering, obstacle avoidance and seeking. The program supports dynamic variable changing while running (through the console), and adding/removing boids with variable behavior. The main way to interact with the program is through the graphical renderer, in which you can control the camera and move around, and the terminal, which is used to change/add objects in the world.

In its current version, the program does not support changing the bounds into a non-cuboid nor adding static obstacles inside the world. A GUI for interacting with variables and boids, as described in the project plan, is also missing.

Structure

The program is split into three core parts: The main loop, which deals with rendering the world, the core loop, which deals with boid movement calculation, and the console loop, which handles user input. Each loop is given their own thread for concurrent operation, separating the rendering frames from physics calculation frames.

The basic class hierarchy is similar to the one described in the project plan, however a lot of relations were stripped as our implementation turned out to be simple enough to not warrant a class to represent.



As the program does not support obstacles (apart from the world bounds), we did not require a parent class to represent entities; Instead, Boids are directly contained inside the World class.

Because our bounds are merely a set of X, Y and Z coordinates, it was sufficient to represent them as a vector, instead of requiring a class for them.

The Behaviour-class was stripped as we found implementing them directly to the Brain class was simpler.

So, the implemented class structure is as follows: The program has a global class World, which all threads can access, and which holds all the information about the current state of the program. The World has a vector of bounds and a vector of Boids. The world also has some global physics constants, for example the max speed allowed.

The Boid class has a brain attached to it, which defines its behavior. Boids do not have a Color; all colors are defined in the different types of Brains. They do, however, have some physical constants for the maximum force and mass, allowing different boids to have different speeds.

The external libraries used are GLFW (v. 3.3) for rendering and GLM for vectors. GLFW integration is only in the main loop, while GLM is used wherever position, velocity and orientation vectors are required.

Sampo Vänninen, Sakari Ropponen, Emma Turkulainen, Joona Sauramäki

Usage instructions

To compile the program on Windows, a thread-supporting C++ compiler is required (ex. Cygwin, MSYS2).

Otherwise, all required libraries are included in the source, so running CMake (Requires version 3.11+) in the source folder and “make” inside the generated build folder should be sufficient. The program

On Linux one must install two required libraries:

sudo apt-get install xorg-dev

sudo apt-get install freeglut3-dev

After installing these the workflow is identical to the Windows part of building & running.

Usage

After running the .exe, a window should open, showing the world bounds as a transparent cube, with a few flocker boids in it. The window has the following keybindings:

Pause movement	K
Return movement	I
Move Forwards	W
Move Backwards	S
Move Left	A
Move Right	D
Move Up	Space
Move Down	Ctrl
Faster movement	Shift
Fullscreen	F11/Tab
Exit	Esc

Changing the properties of the world is done in the terminal the .exe was launched from. Commands are entered as a series of parameters split by spaces, for ex. “add 100 flocker”. The commands and their parameters and functionality are as follows:

add [int] [identifier]	Adds [int] boids in random locations, with [identifier] brain.	Identifiers are: “wanderer”, “flocker”, “hunter”
clear [identifier]	Removes all boids from the world	Same identifiers as add. No identifiers clears all boids.
bounds [int]	Changes bounds to a cube with side length of [int]. Very small and very large bounds produce erratic boid behavior.	Default = 50
setfps [int]	Changes the boid update frequency to [int] times a second.	Valid range is 1 to ∞, however very large numbers will not do anything, as the computations cannot be calculated fast enough. Default = 60
maxspeed [int]	Sets the maximum speed allowed inside the world to [int] / 100.	Default = 500
maxforce [int]	Sets the maximum steering force for boids to [int] / 100.	Default = 300

Large amounts of boids (>500) do work, but rapidly decrease the physics frames per second.

Wanderer boids do nothing but wander aimlessly, avoiding other Boids and the bounds.

Flocker boids “flock” into groups of ~40 boids, and traverse as a whole in a fairly straight line, turning sharply at the world bounds, creating a circular pattern for the flock.

Sampo Vänninen, Sakari Ropponen, Emma Turkulainen, Joona Sauramäki
Hunter boids wander around the world, targeting other non-hunter boids, and seeking them until they reach their position, at which point the boid is “killed” and removed from the world.

Testing

The project does not have any unit testing. Most of the testing as done the classic console printing way while implementing new functionality. This does mean implementations which were working before sometimes broke afterwards when functionality was changed somewhere else, which was not ideal and took some debugging time. Implementing some kind of testing would most likely have saved some time overall.

The remnants of console print testing can be seen in the source code in the from of commented-out std::cout lines.

In the last few days we had a nasty crashing error which took quite some time to fix, and unit testing would have helped to pinpoint the source.

Work log

The work was divided into three larger parts: The rendering, the Boid logic, and the GUI.

The rendering was solely created by Joona, and he also created the multithreading code (including mutexes).

The Boid logic was done by Emma & Sakari. Sakari also set up the CMake for the project.

The GUI (which ended up being the console interface) was implemented by Sampo. He also wrote this documentation.

Most of the work was done in the last 3 weeks of the project, contrary to what we had planned in the original plan. However, the project didn’t feel rushed and we had enough time to implement most of the things we initially wanted. This also reflects somewhat in the work breakdown, where most of the early weeks are quite empty.

Sampo Total: ~44h

Week 44	Roles, project plan	~2h
Week 45		
Week 46	Set up work environment (Eclipse)	~3h
Week 47		
Week 48	Switched from Eclipse to VSCode, meeting & refined project plan	~4h
Week 49	Created basic class structure	~15h
Week 50	Created console interface, unified render & core data, refactored parts of classes. Wrote the documentation.	~20h

Steering Behaviors Group 2

Sampo Vänninen, Sakari Ropponen, Emma Turkulainen, Joona Sauramäki

Sakari Total: ~87h

Week 44	Established roles, and created project plan	~3h
Week 45	Learning CMake & linking essential libs.	~15h
Week 46	Removed redundant libraries + meeting	~3h
Week 47	Waiting for 3d environment	0h
Week 48	Added review of other team and creating the core of project architecture together with others.	~6h
Week 49	Creating boid behavior with Emma	~35h
Week 50	Refining boid behavior & bugfixes with Emma	~25h

Emma Total: ~57.5h

Week 44	Attended project meeting, established roles	~2h
Week 45	Waiting for testing environment	
Week 46	Attended project meeting, installed needed compiler	~2h
Week 47	Waiting for 3d environment	
Week 48	Project meeting, created core of the project architecture with the others	~3.5h
Week 49	Created seeking, wandering, obstacle avoidance and flocking algorithms with help from Sakari	~35h
Week 50	Created behavior for hunter with Sakari, bug fixes, cleaning code, adding comments	~15h

Steering Behaviors Group 2

Sampo Vänninen, Sakari Ropponen, Emma Turkulainen, Joona Sauramäki

Joona Total: ~52h

Week 44	First project meeting, project plan	~2h
Week 45	Learning openGL and GLFW	~6h
Week 46	Learning openGL and GLFW	~6h
Week 47	Creating 3D environment	~4h
Week 48	Project meeting (architecture)	~4h
Week 49	Reworking the 3D scene, adding bounds, orientation	~10h
Week 50	Finishing the project, cleanup, last meetings, fixing bugs, reworked boids to one vector with mutexes	~20h