Lab 2 - Seek, Arrive and Align Steering using Raylib
AI Programming for Games
COMP10068

Paul Keir

Date Issued: January 12, 2022
Instructor: Dr. Paul Keir

In this week's lab we will continue to use the Raylib videogame programming library to explore dynamic steering algorithms from Section 3.3 of the AI for Games book by Ian Millington (eBook is in the online library).

Edit the CMakeLists.txt file in the lab-steering2 directory as before using Notepad++. You can copy from the top of last week's CMakeLists.txt file if you haven't moved anything. Ensure your antivirus software is not removing the DLLs and binary files it finds within the Vcpkg directories.

After you have used CMake to configure and generate your Visual Studio solution, open it in Visual Studio and you will find a single skeleton project. We will work through each stage of development together.

Steering Three (steering3.cpp)

1.. We will use our own vector library to afford `const` correctness and a sensible `normalise` `Vector` method. The library is in the vec.hpp header, and entities are declared in the `ai` namespace. Start by adding a global `using` declaration:

```
using Vector = ai::Vector3;
```

2.. Create a `Kinematic` and a `SteeringOutput` C++ class using the pseudocode from slides 14 & 15 (book page 82). Add the `public` access specifier on the first line of each class for now.

3.. Now modify the `Kinematic` C++ class as in the pseudocode from slide 36 (page 95). The `update` member function should use `const` references for class type parameters. Note that our `normalise` method is spelled with an "s".

4.. Add a `Seek` class for the dynamic seek behaviour encoded in the pseudocode on slide 38 (page 96). Ensure `character` and `target` are declared as C++ *references* (i.e. with an ampersand `&` after the type). Also, do not declare `result` using a `new` expression. Instead, just declare it as a simple local variable.

5.. Declare a `Ship` class, with a `draw` method, much like last week's `tri` class. But this time we will use 3D vectors, and use the z and x components to correspond to the left-right and up-down axes respectively. Ensure it either inherits from `Kinematic`, or has a `Kinematic` data member.

6.. Declare two `Ship` objects (perhaps called ship and enemy), before the `while` loop in the `main` function. Call the `draw` method on each before `EndDrawing`. Run your program to confirm they are where you expect them to be.

7.. Declare a `Seek` object, also before the `while` loop. Initialise it using the `Kinematic` component of your ship and enemy ship. Also choose a maximum acceleration value. Then, after `EndDrawing`, call the `Seek` object's `getSteering` method, and pass the result to the `update` method of the first ship object.

8.. Use `IsMouseButtonPressed` and `GetMousePosition` to set the position of the enemy ship each time the mouse button is pressed.

9.. Now switch from the seeking behaviour of the `Seek` class to the arriving behaviour of the `Arrive` class. Refer to slide 42, or page 99 of the book online. When picking radii, consider that the ships have a length of 30.

10.. Now you are ready to experiment with the remaining algorithms in section 3.3. Ensure you have firstly read and understood the section itself, and then begin by implementing the align behaviour.