**Lab 5**

**An Example of a Behaviour Tree implementation in C++**

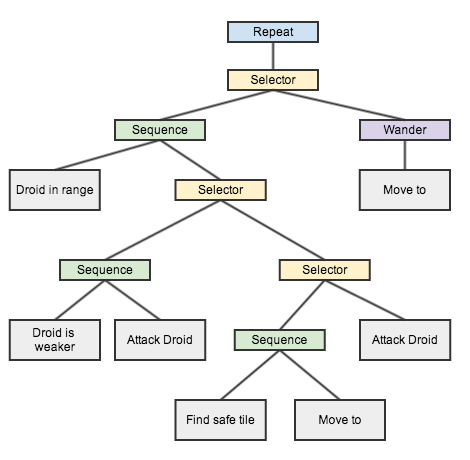
**Due Feb 11th**

**Worth 5%**

**Setup**

Please ensure you have the following installed:

* Your preferred version of Visual Studio (I suggest 2022)
* SFML (Simple and Fast Multimedia Library) at https://www.sfml-dev.org/download.php)
  + I suggest we use at least version 2.6.0)
  + Please setup an environment variable called SFML\_SDK. My sample application expects it. It should be set to the directory where you installed the SFML libraries e.g. SFML\_SDK=C:\SFML-2.6.0
  + The source code for this Lab is on blackboard (ABS.zip). ABS stands for Agent Based System which is a topic we explore at the end of the module. It is based on the java tutorial found [here](https://www.javacodegeeks.com/2014/08/game-ai-an-introduction-to-behaviour-trees.html) which I have reproduced in C++ and fixed his bugs (because I’m nice like that). It includes a partial implementation of this behaviour tree including demonstrations of various other behaviours:



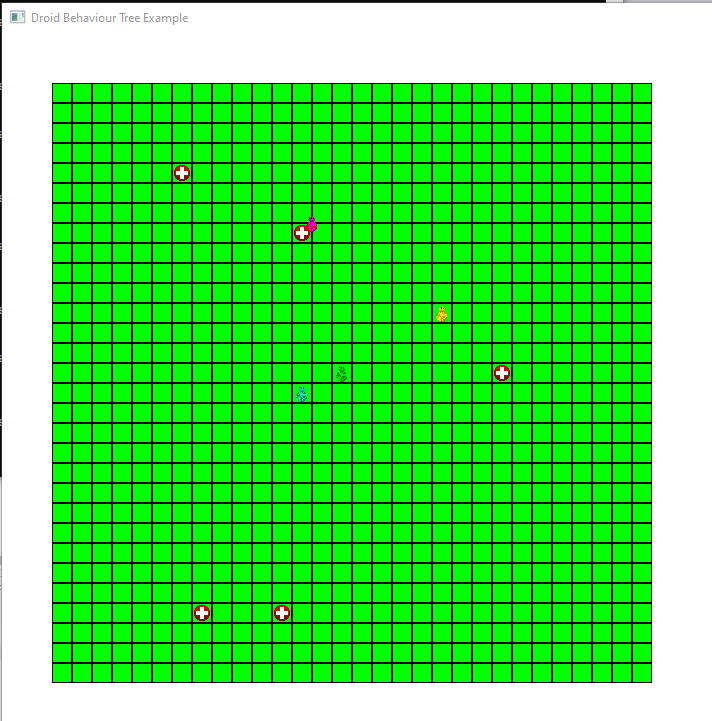
In **Game::setupDroids()** I have some Droids created with different behaviours assigned to them. Most are commented out to remove clutter on the screen.

The base Behaviour is called a **Routine**, and each other behaviour inherits from that. I have implemented a number of leaf Behaviours (e.g. **MoveTo**, **Wander** and **IsDroidInRange**). I have also implemented **Selector** node, **Sequence** node and **Repeat** node types. The key to a behaviour tree is of course the return value (SUCCESS, FAILURE, RUNNING) which I have implemented as a sort of finite state machine in the base behaviour.

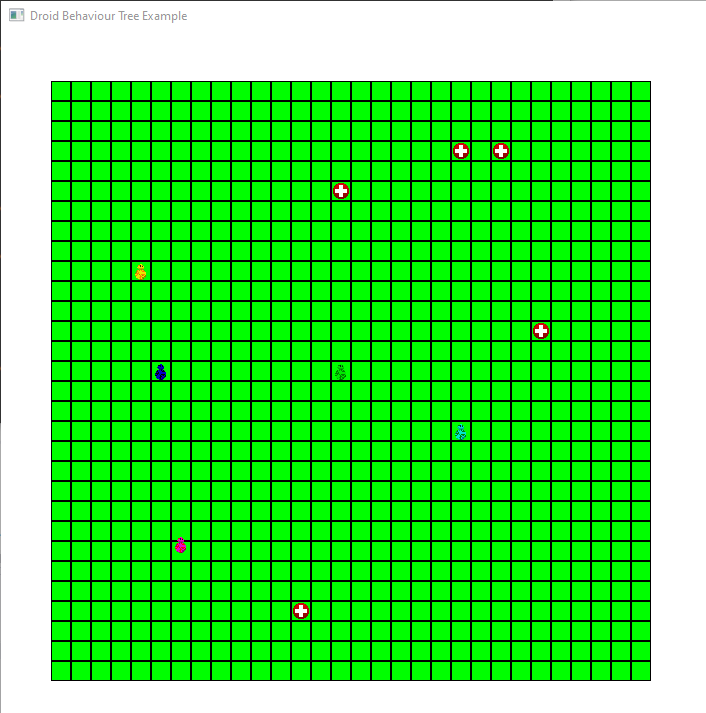
**To do:**

1. Have a read of the tutorial explanation on line.
2. Interrogate my code and understand how the behaviours are implemented via inheritance.
3. Run the program. You should see my little BB-8s moving around the screen.

They stop when their behaviour finishes (if it finishes). Debug info to the output screen shows what’s going on.



1. Play with the droids and behaviours to see how they work (and what restrictions exist with this implementation)
2. I have also created a behaviour called **Hide**. It takes two int parameters referring to two other droids A and B. The behaviour is to hide from droid A behind droid B. Droid D4 (light blue above) is running that behaviour. It is hiding from D1 (Yellow) behind D2 (Green). Move Yellow (D1) by clicking somewhere in the grid. D4 should adjust its position.
3. Don’t forget, a behaviour runs until SUCCESS or FAILURE and then stops. That is why you have to Reset() the behaviour if you want it to run again. See how I do this in updateDroids().
4. Just like in the Excel exercise, create a new behaviour called **Protect** similar to **Hide** except the behaviour is to position the droid between A and B thus protecting droid A from droid B. Droid D5 below (Blue) is protecting D1 (Yellow) from D3 (Magenta). Magenta is running the Wander behaviour, so Blue will continuously update its position due to its behaviour (as long as I reset it when it completes).



1. Instantiate a droid which performs this behaviour.
2. Invent and implement your own behaviour.
3. Show me when you have it running.
4. Submit your minimum zipped project to Blackboard.