Lesbrief: IoT en Object Orientatie

Op het moment zijn er ongeveer net zoveel IoT devices als mensen op de wereld, en is de verwachting dat dit zelfs verdubbelt de komende 5 jaar (https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/).

Deze devices bieden een ontwikkelaar de mogelijkheid om zelf fysieke concepten te bouwen. /

Als projecten echter groter worden, wordt de code als snel erg complex. Een gemiddelde auto bevat tegenwoordig zo'n 100 miljoen regels code (https://www.wired.com/2012/12/automotive-os-war/ , en ter vergelijking https://www.visualcapitalist.com/millions-lines-of-code/).

Zo groot zullen onze projecten waarschijnlijk niet worden, maar ook bij een project van een paar duizend regels code, is het eigenlijk al niet meer werkbaar om dit in één bestand te zetten.

Arduino is een veel gebruikt, open source platform, waarvoor veel code die je online vindt er precies zo uit ziet: één groot bestand met alle code.

Het is echter mogelijk om dit professioneler aan te pakken dan alles in een sketch (.ino) bestand te zetten. Arduino kan namelijk volledig in C++ geprogrammeerd worden. Daardoor kan je je code beter organiseren en overzichtelijker houden, omdat je hierin volledig object georiënteerd kunt werken.

Het is de bedoeling dat je aan de hand van deze lesbrief de basisbeginselen van C++ uitzoekt en demonstreert in een klein project op de Arduino, Circuit Playground of ander development board.

Uitzoeken

- Basisprincipes C++
- Classes en interfaces
- Verschil tussen interrupts en polling
- Sensoren en actuatoren aansluiten

Opdracht

Demonstreer op de Arduino of Circuit Playground een opstelling die aan onderstaande eisen voldoet:

- Code geschreven in C++
- Alle classes gescheiden in .cpp en .h bestanden

- Maakt gebruikt van minimaal 2 interfaces/abstract classes, en deze zijn beiden minimaal 2 keer geïmplementeerd
- Er wordt gebruikt gemaakt van input van zowel analoge (0-1023) als digitale (0/1) sensoren
- En er wordt zowel met interrupts als met polling gewerkt
  (NB. omdat het vrij lastig is interrupts in een OO project te implementeren, mag je er ook voor kiezen om interrupts in een apart kort voorbeeld te demonstreren)
- Er wordt output gestuurd naar één of meerdere actuatoren (led, servo, etc.)

(een voorbeeld-opstelling vind je onderaan deze lesbrief)

Leerdoelen

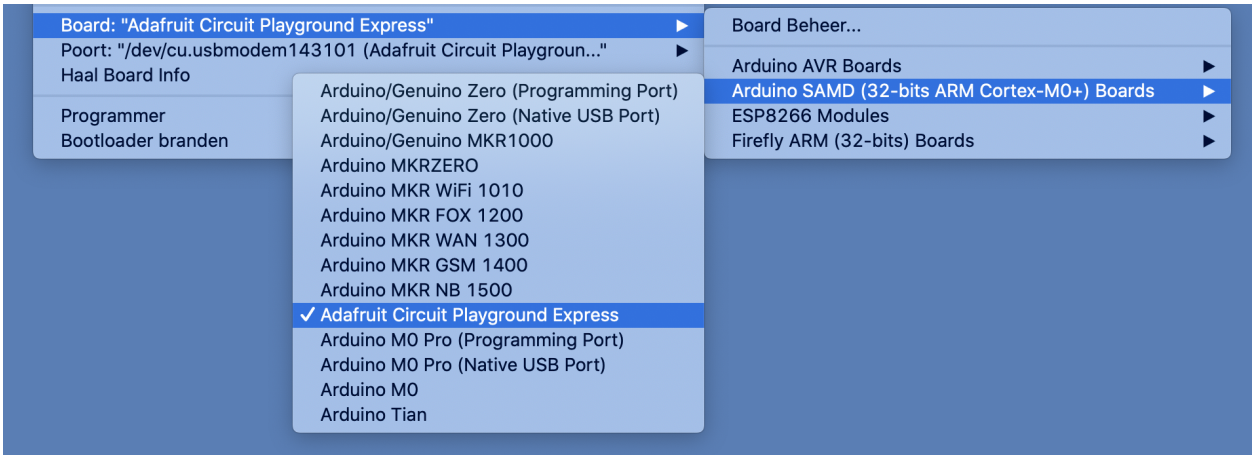
- Ik kan classes en interfaces maken in C++, en weet wat het verschil is tussen een .h en .cpp bestand
- Ik kan een interface/abstract class maken in C++ en deze implementeren
- Ik kan uitleggen wat sensoren en actuatoren zijn
- Ik kan uitleggen wat interrupts zijn en deze gebruiken in een prototype
- Ik kan uitleggen wat het verschil is tussen interrupts en polling
- Ik kan een functioneel testplan opstellen om de werking van mijn prototype te demonstreren

Toetsing

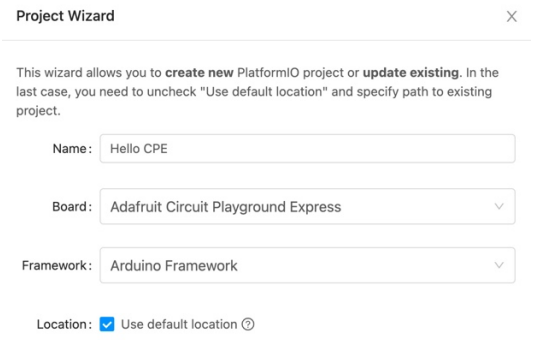
Om deze lesbrief te behalen moet je een werkend project in C++ op Arduino of Circuit Playground laten zien dat voldoet aan de opdrachteisen (zie boven), dit mag zowel een live demonstratie zijn als een filmpje.
Omdat het lastig is om interrupts en object oriëntatie samen te laten werken, mag je ervoor kiezen om 2 aparte demonstratieprojecten te maken voor polling (met OO) en interrupts (zonder OO).
Ook moet je de code mondeling kunnen toelichten.

Tools

- Arduino IDE
  De standaard IDE kan ook met C++ werken. Nieuwe bestanden binnen je project maak je aan door een ‘nieuw tabblad’ aan te maken. Je mist echter wel de extra functionaliteit van een geavanceerde IDE.
  Als je met de Circuit Playground Express werkt, let dan op dat je het juiste bord selecteert!



- Visual studio code
  - Er zijn verschillende plugins voor Visual studio code, zelf vind ik PlatformIO erg goed: https://docs.platformio.org/en/latest/integration/ide/vscode.html
  - Let op dat je ook hier weer het juiste platform selecteert als je met de Circuit Playground werkt.



  :

- Eigen IDE

Bronnen

Edx

https://www.edx.org/course/introduction-c-plus-plus-3
https://www.edx.org/course/intermediate-c-plus-plus-3

C++ docs

http://www.cplusplus.com/doc/tutorial/

Arduino docs

https://www.arduino.cc/
https://learn.adafruit.com/introducing-circuit-playground/overview
https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

- Let op: De Circuit Playground heeft een ATmega32u4 processor / board, dus kijk daar voor de juiste interrupt pins

Tips

Er zijn meerdere manieren om in C++ een instantie van een class te maken.

Automatisch memory management
Wordt opgeruimd aan het einde van de scope

```
Led l = Led(2);
l.turnOn();
```

Dynamisch memory management
Te herkennen aan keyword new, en een pointer variable (*). Als je new gebruikt moet je zelf met delete opruimen.

```
Led *l2 = new Led(3);
l2->turnOn();
delete l2;
```

Echte interfaces bestaan niet in C++. Het alternatief is een abstract class waarin je alle methods volledig virtual maakt.

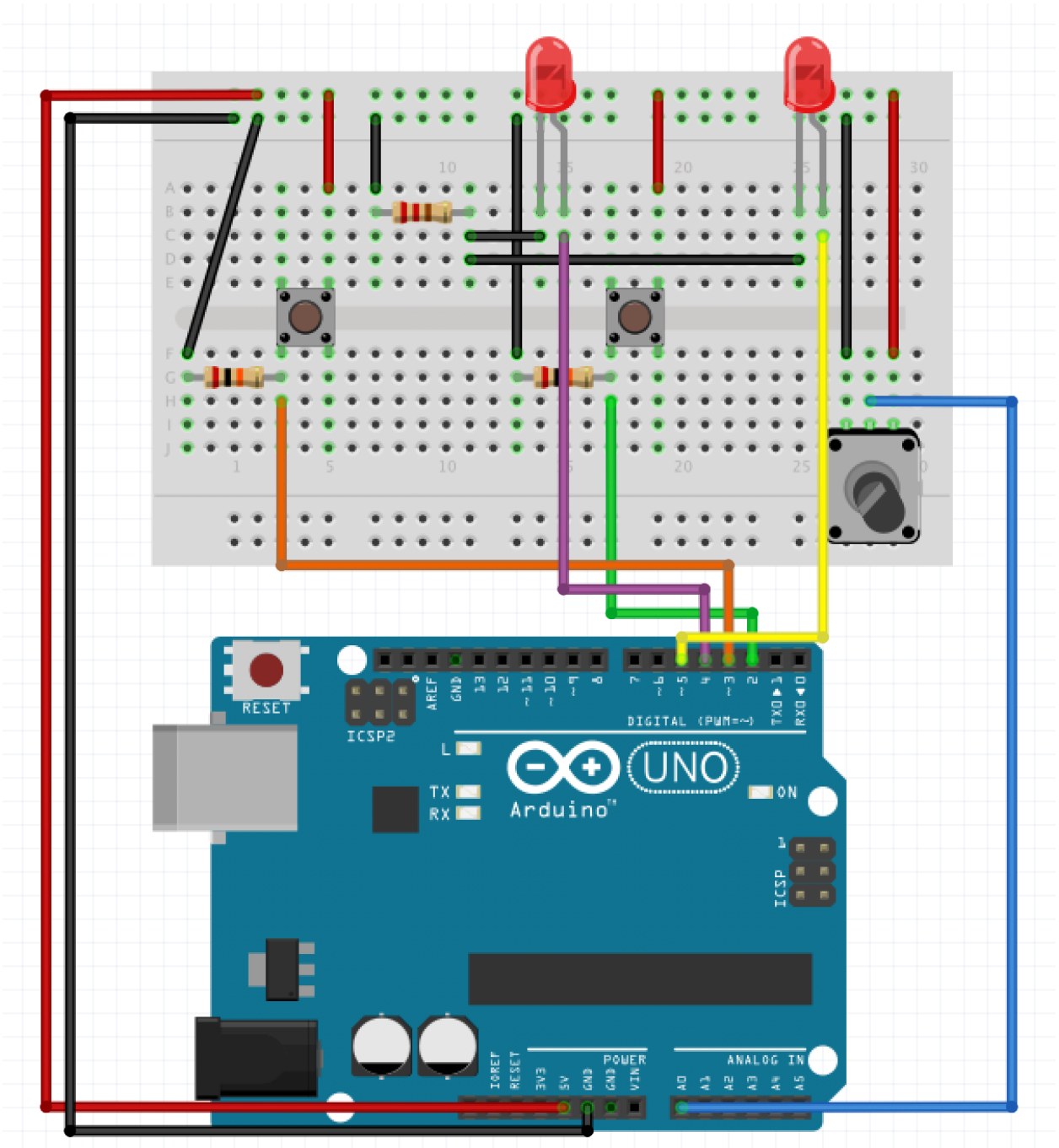
Als je een interface/abstract class als base class wilt gebruiken kan dit niet direct, maar moet dit via een pointer. Zie tip over dynamisch memory management.

Interrupts worden op een low level gekoppeld aan functies. Hierdoor is het helaas niet mogelijk om ze vanuit een object aan een method van dat object te koppelen. De workaround is dat je de interrupts attacht vanuit je main code.

Ook kan je maar 1 interrupt per pin gebruiken. Wil je zowel RISING als FALLING afvangen, dan kan je de CHANGE interrupt gebruiken.

Documentatie

De opdracht is vrij, zolang je je maar aan bovenstaande opdracht eisen houdt. Hieronder een voorbeeld van een opdracht die aan alle eisen voldoet, die je zou mogen gebruiken als je het lastig vindt om zelf iets te verzinnen.



Werking

- BUTTON1 zet LED1 aan, werkt als ‘deurbel’ of togglet (strategy-pattern)
- POT bestuurt LED2, werkt als dimmer of snel-langzaam knipperen (strategy-pattern)
- BUTTON2 wisselt strategieen.
- BUTTON1 en POT meter werken met polling.
- BUTTON2 werkt met interrupts (het is het makkelijkst om dit niet object georienteerd aan te pakken, zie tips)

Voorbeeld strategy in C++

https://sourcemaking.com/design_patterns/strategy/cpp/1