Plan van aanpak voor IPASS

Naam: Peter Schenkels

Student nummer: 1741497

Email: [Peter.schenkels@student.hu.nl](mailto:Peter.schenkels@student.hu.nl)

Datum: 19-6-2019

Wat ik ga gebruiken:

* Arduino Due
* Arduino Nano
* L3g4200 Gyroscoop

<https://www.elecrow.com/download/L3G4200_AN3393.pdf>

<https://github.com/jarzebski/Arduino-L3G4200D>

* 8x8 Led matrix met MAX7219 (4x ?)

<https://datasheets.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf>

<https://github.com/squix78/MAX7219LedMatrix>

Beschrijving van project:

Naam: soGyro lib

Beschrijving: Een Library die gyroscoop data ophaalt volgens een Master en Slave principe.

Je kan via een Master arduino gyroscoop data aanvragen aan een Slave Arduino Nano via SPI die aangesloten is met een L3G4200 Gyroscoop chip via I2C.

Het voordeel hiervan is dat de Slave Arduino Nano Zijn “Pitch, Roll en Yaw” axis bijhoudt, die doet dat veel accurater dan als je de Master Arduino direct zou aansluiten op de Gyroscoop. Dit komt omdat je uit de chip de grades per seconden uitleest, de berekening ziet er dan als het volgende uit

Verlopen tijd sinds vorige berekening (In seconden) / Graden per seconden = Aantal bewogen graden

Als de Slave Arduino Nano er constant (grof gezegd) 4ms overdoet om te berekenen wat zijn “Pitch, Roll en Yaw” axis en het ontvangen van die waardes en versturen naar de Master Arduino. Krijg je veel accuratere waardes dan als de Master Arduino direct aangesloten is op de chip. Die Arduino moet ook nog andere dingen berekenen. Hij kan er soms (heel grof gezegd) 5ms over doen of soms 16ms, omdat hij bezig is met andere dingen zoals: het versturen en ontvangen van een aantal bytes. En dan krijg je inaccuratere waardes.

Methoden:

Class naam: gyroscope

Variabelen:

Gyro\_x: float

Gyro\_y: float

Gyro\_z: float

bus: i2c\_bus

Address: uint8\_t

Constructor parameters

Bus: i2c\_bus,

Address; uint8\_t,

Wake:

Activeerd de chip.

Sleep:

Activeerd sleep modus op de chip.

Stop:

Deactiveerd de chip.

get\_temprature:

Volgens de datasheet heeft de gyroscoop een tempratuur register. Ik heb me nog niet goed ingelezen waarom er überhaupt een temperatuur sensor op de chip zit, maar dat zou vast een goede rede hebben.

get\_x\_axis:

Returnt de X as waarde die de Arduino heeft geregistreerd (sinds hij aan staat) als een float.

get\_y\_axis:

Returnt de Y as waarde die de Arduino heeft geregistreerd (sinds hij aan staat) als een float.

get\_z\_axis:

Returnt de Z as waarde die de Arduino heeft geregistreerd (sinds hij aan staat) als een float.

get\_dgs\_x:

Returnt het aantal graden per seconde van de X as als een float.

get\_dgs\_y:

Returnt het aantal graden per seconde van de Y as als een float.

get\_dgs\_z:

Returnt het aantal graden per seconde van de Z as als een float.

get\_dir:

returnt de richting waar de chip naar toe draait. Voor de Roll en Yaw axis zou je links en rechts kunnen gebruiken en voor de Pitch zou je omhoog en omlaag kunnen gebruiken. En hij zou dan de characters kunnen returnen in een byte waarde, zoals: U: up (omhoog), D: down (omlaag), L: left (links), R right(rechts).

En nog voor als de demo applicatie het niet doet.

Get\_all:

Deze functie zal gyro\_x, gyro\_y, gyro\_z waarden printen

Wat gaat deze library uitdagend maken?

Omdat ik twee verschillende arduino ’s ga gebruiken die wel het zelfde doeleinde hebben zal ik twee verschillende libraries moet gaan maken.

Ik zal een library moet maken voor de Arduino Nano die Alle berekeningen doet voor de gyroscoop. En ik zal ook nog Slave code moeten schrijven, zodat de Arduino Nano kan communiceren met de Master Arduino.

Voor de master Arduino zal ik een library moeten schrijven die de data aanvraagt van de Arduino Nano als ik een bepaalde functie oproep, zoals: gyro.get\_x\_axis. Dan zal deze functie een byte versturen naar de Arduino Nano en dan zal de Arduino Nano ook weer de goede waarden terugsturen.

Versies:

1e versie: alleen functies voor het halen van de data uit de registers.

2e versie: Het omzetten van de gyroscoop waardes naar “Pitch, Roll en Yaw” axis.

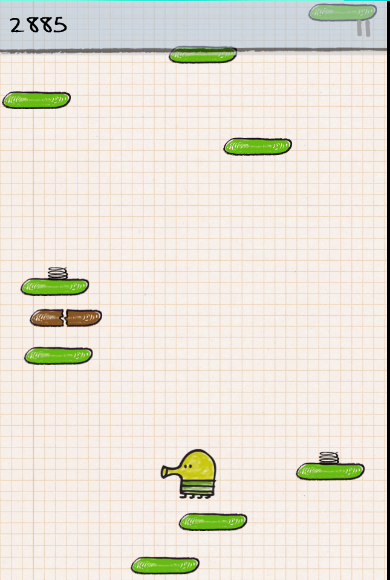
3e versie: Slave library en Master Library.

Applicatie:

Naam: Frog Jump

Beschrijving: Het doel van frog jump zo hoog mogelijk te komen door van platform naar platform te springen en niet naar beneden te vallen.

Frog jump is eigenlijk doodle jump. Het poppetje blijft altijd springen en je bestuurd het poppetje met de gyroscoop.



Wat maakt deze Applicatie uitdagend?

Voor deze applicatie moet ik een game engine schrijven die ook een physics engine heeft. Het moeilijke van spelletje is ook dat ik een manier moet vinden zodat je voor altijd omhoog kan blijven gaan en dat dan ook op geheugen efficiënte manier doet. Dus ik zou iets van een platform generator moeten maken die werkt met 3 verschillende zones. Daarnaast moet ik ook een library schrijven die de data omzet naar visuele elementen voor meerdere 8x8 led matrixes.

Risicobeheersing:

Gelukkig ben ik al een paar weken geleden (tijdens oopc) begonnen met de engine. Daar ben ik nu eigenlijk al bijna klaar mee, op een paar bugs na. En ik heb vorige week ook nog een kleine library kunnen schrijven voor de max7219 led matrix. Dus daar ben ik ook al klaar mee.

Nu kan ik me dus volledig focussen op het spelletje en de gyroscoop. Voor als het spelletje niet lukt kan ik de gyroscoop al zijn berekeningen kunnen laten printen. En voor als de gyroscoop niet lukt zou ik het spelletje wel kunnen laten zien maar dan met knopjes in plaats van gyroscoop. Ook zou ik nog een physics demo kunnen laten zien.

Versies:

1e versie support alleen dat een character naar links en rechts kan bewegen en op een paar voor geprogrammeerde platforms omhoog kan springen

2e versie support dat platforms gegeneerd worden en dat je eindeloos omhoog kan gaan.

3e Dat er bewegende platforms zijn als een bepaalde score bereikt en dat er “enemies” zijn.

Hieronder documenteer ik de 2 libraries die ik ga schrijven voor de applicatie. Ik weet niet of daar behoefte aan is, maar ik doe het toch voor de zekerheid.

Naam: Simple Matrix

Beschrijving: Een library die via SPI met een 8x8 led matrixje kan communiceren die een MAX7219 chip hebben. Je kan er meerdere pixels op aan zetten met de functie:

write(int x, int y, bool enable)

en je kan er nog een character op displayen door de functie: display(char character) aan te roepen.

Methoden:

**Class naam:** ledmatrix

Variabelen

Bus: spi\_bus

Cs: pin\_out

display\_buffer[8]: uint8\_t

Functies:

write(int x, int y, bool enable)

tekent een pixel op een xy locatie

display(char character)

displayt de meegeven character.

Wake:

Activeerd de chip.

Clear\_buffer:

Zet alle waardes in de display\_buffer naar nul

Clear\_screen:

Zet alle pixels op het scherm naar nul.

Clear:

Voert clear\_buffer en clear\_screen uit.

Flush();

Tekent de pixel buffer op het scherm.

(Ik kan begrijpen dat dit erg mager is voor een library maar daar heb ik de onderstaande library voor gemaakt)

En dan de library waar de matrix in samenwerking gaat. Deze heb ik al eerder tijdens OOPC uitgewerkt dus hoef ik hier niet veel tijd aan te besteden.

Naam: Array Engine

Beschrijving:

Met deze library kan je pixeltjes rond laten bewegen en animeren en het heeft ook nog eens een primitieve “physics engine” waardoor je de pixeltjes tegen de grond en tegen elkaar aan kan laten stuiteren en zwaartekracht kan simuleren op de pixeltjes. Waardoor je in principe makkelijk een leuk spelletje kan maken.

Daarnaast heeft deze library een camera object. Deze kan je aansluiten aan een ledmatrix.

Een camera object heeft een eigen locatie en kan daarom ook een pixeltje (object) volgen. Hierdoor zou je makkelijk twee schermpjes kunnen maken waarbij je twee spelers tegen elkaar kan laten spelen. Dan moet je voorstellen dat bij speler 1 het ledmatrixje gefocust is op Object1 en bij speler 2 het ledmatrixje gefocust is op Object2. Ook zou je een 16x16 schermpje kunnen maken door ledmatrix1 op positie xy(1,0) te zetten en ledmatrix2 op positie(8,0) en ledmtarix3 en 4 op positie xy(1,8) en xy(8,8).

Ik zal nog iets dieper op in gaan wat ik nou precies bedoel met een object/pixeltje.

Een object heeft een xy positie op een 2d array en kan vrij rond bewegen. Je kan in de Constructor van een object vermelden wat voor physics eigenschappen het heeft, bijvoorbeeld: Of het kan stuiteren of het zwaartekracht/massa heeft en wat zijn xy snelheid is. Deze object kunnen vrij rond bewegen in een class genaamd: scene. Dit is gewoon een 2d array van char’s.

Daarnaast heb je nog een obstacle object. Het verschil tussen een obstacle en een object is dat een obstacle twee xy posities heeft, waardoor je een lijntje kan maken, en dat het statisch is. Objecten kunnen niet door een obstacle heen bewegen.

Methoden:

**Class naam**: Object

Variabelen:

X, Y: int

Character: char (identificatie van object in een scene)

Bounce, Gravity

X\_speed, Y\_speed: int

Functies:

Get\_collision:

Returnt aan welke kant(en) hij iets aan gaat raken.

Move\_up(n: int):

Beweegt het object een n aantal pixels oomhoog als er niks in de weg staat.

Move\_down(n: int):

Beweegt het object een n aantal pixels oomhoog als er niks in de weg staat.

Move\_left(n: int):

Beweegt het object een n aantal pixels oomhoog als er niks in de weg staat.

Move\_right(n: int):

Beweegt het object een n aantal pixels oomhoog als er niks in de weg staat.

Set\_pos(x: int, y: int):

Zet de positie van het object naar gegeven x en y waardes.

**Class naam:** Obstacle

Variabelen:

X\_start, Y\_start, X\_end, Y\_end

**Class naam:** scene

Variabelen:

Playground[X\_SIZE][Y\_SIZE]: char

Functies:

clear\_scene:

reset alle waardes van playground zodat na het renderen van een frame op de ledmatrix (of terminal) de nieuwe xy posities van objecten en obstacles gezet kunnen worden in playground.

**Class naam:** Camera

Variablen:

X, Y: int

Display: ledmatrix

Functies:

Follow(obj: object):

Zet X en Y positie naar de positie X-4 en Y-4 positie van een object.

Render(field: scene):

Rendert alle objecten die in positie: X tot en met X\_size + x en Y tot en met Y\_size + Y van field voorkomen naar een ledmatrixje. X\_size en Y\_size zal altijd 8 blijven als er een ledmatrixje wordt gebruikt. Ook zouden de X\_size en Y\_size waardes veranderen als deze buiten de 2d array komen.

**Class naam:** Physbox

Functies:

Calculate\_gravity(object obj, scene field):

Berekent obj zijn nieuwe snelheid met zwaartekracht en voert dan calculate\_bounce uit.

Calculate\_bounce(object obj, scene field):

Kijkt of obj moet stuiteren en zo ja pas hij de speed aan van het object en voert dan calaculate\_movement uit.

Calculate\_movement(object obj, scene field):

Berekent de nieuwe positie van obj in field.

(Er zijn nog een aantal kleine classes maar dat wordt te veel moeite om dat hier allemaal te documenteren)