Content

Summary

Projektet er udarbejdet i samarbejde mellem Mads, Kaloyan og Christian i 3-ugers perioden 02-01-2018 til 19-01-2018 i kurset “62505 Introduktion til indlejrede systemer”. Projektoplægget gik ud på, at lave et Arduino projekt. Gruppen besluttede sig for, at lave arduinoen om til en spillekonsol, som kan kommunikere med en RPI. På RPI siden er der databehandling af det modtagne data. Data’en skulle så efterfølgende sorteres og blive tilskrevet i et tekstdokument, der indeholdte en highscore liste. En tidsplan over projektet blev udarbejdet og den aktuelt “brugt tid” blev løbende tilskrevet tidsplanen, så der løbende blev holdt øje med projektet. Projektet var opdelt i Arduino og RPI. Mads og Christian samarbejde om Arduino delen og Kaloyan stod for databehandling på RPI siden, kommunikation med FTP serveren og TWI kommunikationen mellem RPI og Arduinoen.
Produktet endte med at opfylde de kravspecifikationer som blev opstillet i starten af processen.

1. Introduktion

1.1 Problemformulering

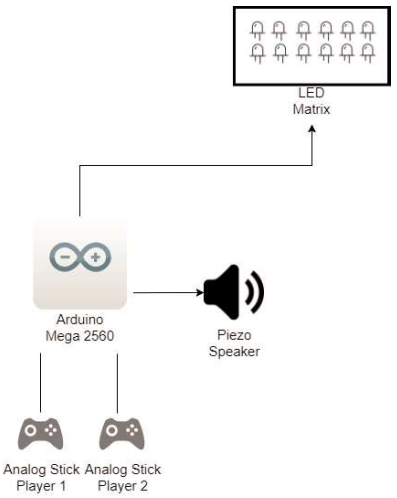
En arduino Mega 2560 ønskes programmeres til, at den kan spille spil som en spillekonsol. Arduinoen skulle kommunikere med en Raspberry Pi 3 (RPI) som skulle stå for, at holde og opdatere en highscore liste som bliver uploaded til en webside. Det første og simpleste spil der kan laves er Kryds og bolle, men problemet er at lave spillet på en måde så et nyt spil kan skrive udfra de standard funktioner som systemet indeholder. Derudover skal systemet opbygges modulært, så det er trivielt at tilføje nye spil og funktionaliteter til spillekonsollen og kunne interagere med det hardware som sidder i systemet.

2. Problem Løsning

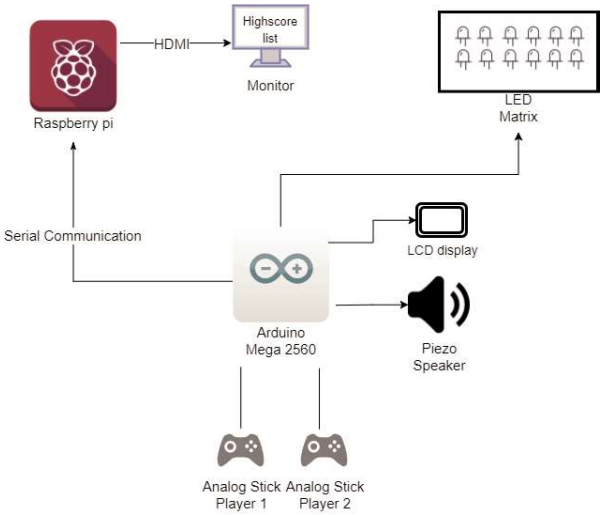
2.1 Analyse og design (Mark)

2.1.1 Konceptet

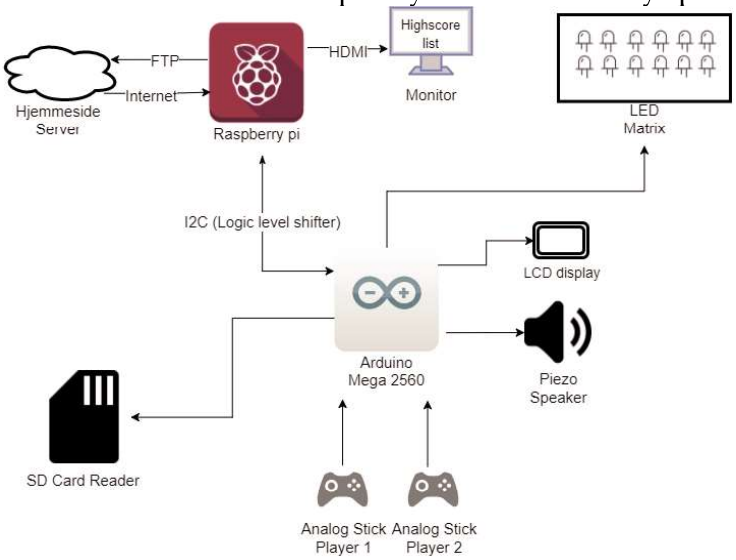
Gruppen startede ud med, at brainstorme omkring, hvad der skulle bruges til at kunne spille et spil fra arduinoen. Det blev pga. gruppens erfaring med brugerinput valgt, at der skulle bruges nogle "analog sticks" som man kender dem fra en Playstation controller, til at styre spillet. Derudover blev der diskuteret mulighederne for, at brugerne kunne have noget bruger feed f.eks. i form af lyd, lys og vibration udfra spillets tilstand. Spillet skulle foregå på en form for skærm og en mulighed for, at holde noget highscore liste blev ønsket implementeret.
Gruppen startede ud med, at udarbejde koncepttegninger over, hvilket hardware der skulle indgå i produktet. Denne koncept tegning gennemgik 3 iterationer som kan ses på de følgende sider:



(Koncept tegning version 1: Arduino modtager Inputs fra spillerne og outputter spillet til LED matrix og noget lyd på en Piezo højtaler.)



(Koncept tegning version 2: Arduino giver brugerfeedback til et LCD display og kommunikere med en Raspberry Pi som holder styr på en highscore liste)



(Koncept tegning version 3: Arduino holder lokal backup af highscore listen og kommunikere med RPI med I²C. Highscore listen uploades via FTP til en Server, så listen kan vises på en hjemmeside.)

I²C blev valgt til kommunikationen mellem arduino og RPI pga. et af gruppemedlemmernes interesse i, at sætte sig ind i protokollen og evt. skrive sit eget bibliotek til I²C kommunikationen.

2.1.2 At køre spil på en arduino (Mark)

Noget som gruppen fra starten af lagde vægt på var, at spillet skulle kunne køre så hurtigt som muligt, mens brugeren stadig skulle kunne styre rundt på banen som ønsket. Man snakker typisk om "responsiveness" i et spil og om "inputlag" som hentyder til, hvor lang tid der går fra, at brugeren trykker på en knap til, at man kan se der sker noget på skærmen. Dette betyder rigtigt meget for selve spiloplevelsen og kan i nogle spilgenre som f.eks. skydespil eller andre spil der kræver "hurtige reflekser" være med til - i grove tilfælde - at ødelægge spille oplevelsen for brugeren. I forhold til spil på arduino'en kunne det have indflydelse på spil som "snake" eller andre "auto-run platformer" spil, hvor man konstant er i bevægelse og skal tage valg hurtigt.[1]

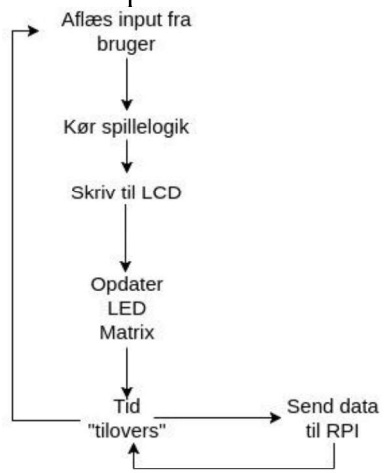
Udover brugerens spiloplevelse, kan man kigge på noget andet vigtigt og meget relevant for gruppens projekt: "Hvad er runtime på programmet og hvad kan overskydende CPU tid kan bruges på?"
Hvis man f.eks. skulle lave en meget overfladisk sammenligning mellem en PS4 Pro og en arduino mega 2560 og deres evne til, at afvikle et spil er dette netop et meget relevant og interessant spørgsmål. Arduino'en[2] er på den ene side: er en single-core processor der kører ved 16 MHz, har 8 KB Sram, 256 KB plads til et program, har ikke et styresystem eller et dedikeret grafikkort.
Hvorimod en PS4 Pro[3] har en 8 kernet processor der kører ved 2,13 GHz, 8 GB ram, 1 TB plads til styresystemet og divs. spil som bliver kørt på CPU'en og det dedikerede grafikkort skaber grafikken til en skærm. Men udover, at PS4 Pro'ens CPU kører ved en væsentlig højere clock er der også flere kerner og et styresystem der gør, at der kan blive kørt flere ting parallelt og ikke bare køre programmet sekventielt som det sker på en arduino. Dette gør, at PS4's CPU kan lave flere ting på en gang: aflæse/modtage brugerinput fra controllerne, køre spillet og sætte GPU'en igang med, at arbejde på grafikken som skal ud på en skærm. Arduino'en har intet styresystem og kører derfor sekventielt igennem det loop som der er i det program der er uploaded til den sidst.

Det er derfor i forhold til projektet vigtigt for gruppen, at programmet der ligger i main loopet gennemløbes hurtigt, da der udover spillet også skal bruges tid på at kører

[1] Definition af "Auto-run platformer" - http://www.mobygames.com/game-group/genre-auto-run-platformer (Besøgt 17-01-2018 kl 14)
[2] Specifikationer på en Arduino mega 2560 - https://store.arduino.cc/usa/arduino-mega-2560-rev3 (Besøgt 17-01-2017 kl 15)
[3] Specifikationer på en PS4 Pro - www.playstation.com/en-gb/explore/ps4/tech-specs/ (Besøgt 17-01-2018 kl 15)

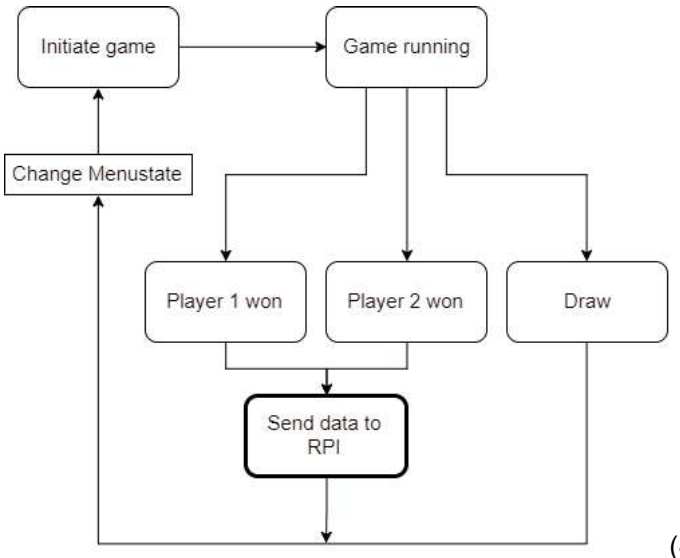
nogle “andre opgaver”: skrive til et LCD display, et LED matrix display, skrives til et SD-kort og sendes noget data til RPI under programkørsel. Det hele skulle kunne køre i et kompromi mellem spilleoplevelsen beskrevet tidligere og hvor meget tid “tid tilovers” der var til, at udfører de “andre opgaver”.

I starten af designprocessen blev det foreslået, at kommunikationen mellem RPI og Arduino skulle ske mens der blev spillet et spil for, at teste “worst-case scenario” da det her, hvor alt spil logikken skal/kan risikere at blive gennemløbet. På diagrammet ses konceptet:



(1. Koncepttegning: Beskrivelse af programmet skal afvikles under et spil)

Konceptet og selve behovet for, at sende data til RPI blev løbende justeret og tilpasset til et simplere koncept. På tilstandsdiagrammet herunder vises simplificeret at der er et spil igang. Tilstandsmaskinen og spillet vil blive beskrevet senere, men er her fremhævet for, at vise hvornår i programmet der bliver sendt data til RPI. Grunden til, at konceptet blev justeret og at der kun sendes data når der er nogle der har vundet, er fordi gruppen besluttede sig for, at det vil være det mest optimale. Til videreudvikling er det muligt, at udvikle så data kan sendes hele tiden.



(Simplificeret koncepttegning: Beskrivelse af, hvornår data skal sendes til RPI i programmet under afvikling af et spil)

2.2 Flowkontrol and delays (Mads)

Der er to måder at styrer kontrollen i et program, der skal reagerer på brugerindput, på. Enten kan programmet vente på at brugeren genererer et indput, eller det kan kører hele koden igennem og hver gang tjekke om brugeren har genereret et indput. Ved ikke at vente, men derimod kører hele koden, er det muligt at holde en fast opdateringshastighed i programmet og dermed på outputtet, som kunne være at opdaterer en skærm. Det gør det også nemmere at lave interval betingede handlinger. Derfor bruges dette normalt i spil, som for eksempel i snake hvor slangen bevæger sig selvom brugeren ikke foretager sig noget.
En anden af fordel ved dette er ,at hvis koden kører tilpas hurtigt, er det muligt at bruge noget af køretiden for hver gennemgang på at gøre andre ting.
Som tidligere nævnt har en Arduino Mega 2560 kun en enkelt cpu kerne, så kun et program af gangen kan afvikles. Derfor må der bruges tidsstyring, for at kunne lave flere ting på en gang.
Derfor bruges en flow-kontrol model, som kører hele programkoden igennem hele tiden for at opdaterer outputtet og tjekke brugerindput.

Da det, der skrives til de to visuelle outputs, vil blive der indtil de opdateres, er den langsommeste hastighed hvormed, det er muligt at opdaterer, bestemt ved det mest tidskritiske, i den tilstand programmet befinder sig i. I ”menu_running” tilstanden er dette den tid der går mellem der bliver reageret på brugerinputet. Det samme er det i ”game_running” tilstanden, selvom disse to hastigheder er forskellige. Grunden til dette vil forklares senere i rapporten.
Den endelige implementering af programmet har en variabel forsinkelse i bunden af kildekoden. Denne er implementeret ved brug af arduino komandoen ”*millis()*” som

retunerer antallet af mikrosekunder på siden arduinoens opstart. Dette tal holdes i en unsigned integer og vil derfor overflowe efter ca. 50 dage[4].
I dette tidsinterval kører processoren en løkke som kun samler nye værdier til "currentMillis". I dette tidstinterval er det muligt at udvide koden med ekstra funktionalitet.

```
354   //while this runtime for the program is less than the inteval
355   while(currentMillis - previousMillis <= interval) {
356     currentMillis = millis();
357     //Serial.println(currnetMillis);
358
359     //Here is the place to add sender code
360   }
361
362   currentMillis = millis();             //update the current time, if we did not enter while loop
363   previousMillis = currentMillis;       //update previous time for next compare
```

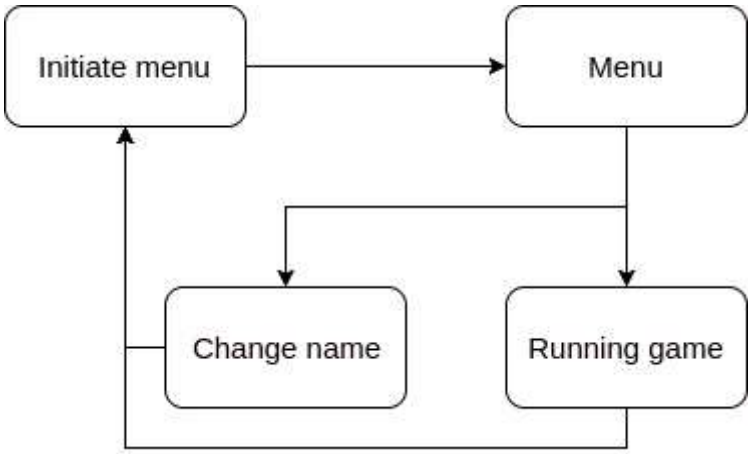
Det interval der ventes på afgøres af den tilstand programmet bevæger sig i, hvis programmet er i "menu_running" er det 250 millisekunder, mens det kun er 200 millisekunder hvis programmet er i tilstanden "game_running".

2.3 Tilstandsmaskiner

Programmet er overordnet opbygget af to tilstandsmaskiner som styrer programmets flow. De to tilstandsmaskiner er menu tilstandsmaskinen og spil tilstandsmaskinen. En af tilstandene i menu maskinen er at kører spil maskinen.
Ved at opbygge de to hoveddele af programmet af tilstandsmaskiner, er det nemt at holde styr på hvor programmet er på et hvilket som helst tidspunkt. Det gør det også muligt at kører hele programmet i et stort loop, som gennemløbes hele tiden.

Menu tilstandsmaskinen består af fire tilstande, "initiate menu" som initialiserer menuen og giver kontrollen til spiller et og hopper videre til "Menu" som kører menuen og ikke forlader denne tilstand før brugeren vælger en af menupunkterne hvilket vil ændre tilstanden. I "Menu" køres der med den flowkontrol som er forklaret tidligere. Tilstanden "Change name" gør det muligt for brugeren at skifte de navnene på spillerne, dette foregår ved to funktionskald og under indtastningen bliver programmet nede i funktionerne. Den sidste tilstand i menu tilstandsmaskinen er "Running game". I denne tilstand køres programmet og der skiftes først væk fra denne tilstand efter at spillet færdiggøres. I selve spillet køres der også med den flowkontrol som blev omtalt i sidste kapitlet.

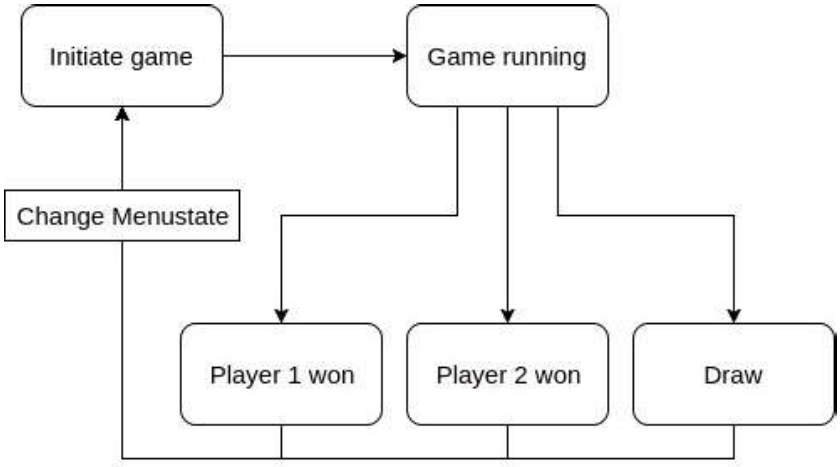
[4] Information om "*millis()*" komando: https://www.arduino.cc/reference/en/language/functions/time/millis/(Besøgt 17-01-2018 kl 14:00)



(Her ses et diagram over menu tilstandsmaskinen)

Spil tilstandsmaskinen sættes igang når menu tilstandsmaskinen er i "Running game" tilstanden.
Tilstandsmaskinen består af fem tilstande, "Initiate game" som initiere spillet og resetter alle variable og sætter maskinen i "Game running" tilstanden, som kører selve spillogikken og opdaterer skærmen, her bliver maskinen indtil der findes en vinder af spillet eller spillet bliver uafgjort. I denne tilstand køres der med den flowkontrol der omtales tidligere, således der kommer en fast opdateringsrate på spillet. De to tilstande "Player 1 won" og "Player 2 won" gemmer vinderens navn til highscore listen og sender dette til RPI'en, hvorefter spil tilstandsmaskinen retuneres til "Initiate game" og menu tilstandsmaskinen retuneres til "initiate menu", dette gør at programmet starter menuen. Det samme sker når spil maskinen kommer i "Draw" tilstand, dog uden der gemmes og sendes en vinder.

(Her ses et diagram over spil tilstandsmaskinen.)



2.4 Bruger input

Der findes to overordnede måder at fange indput på "polling" og "interrupts". Det samlede produkt indeholder begge dele. Det brugerinput som skal modtages af produktet er de to "analog sticks" controllers som hver består af to potentiometre og en knap. De modtager hver 5v og jord, mens de outputter to analog værdier, en for hver akse og en knap der er aktivt lav.

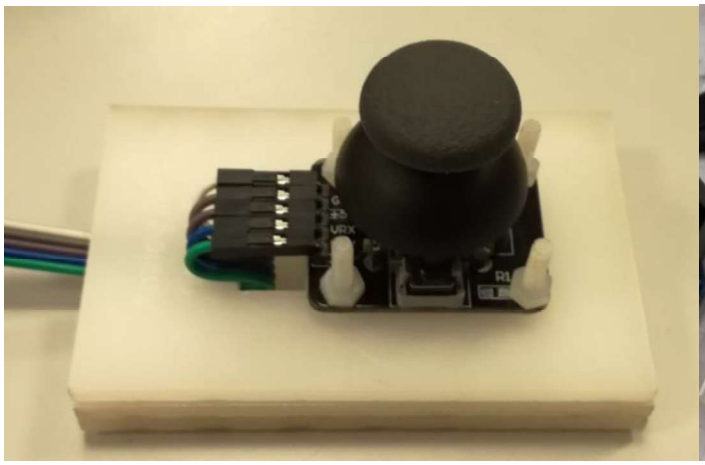
2.4.1 Interrupts

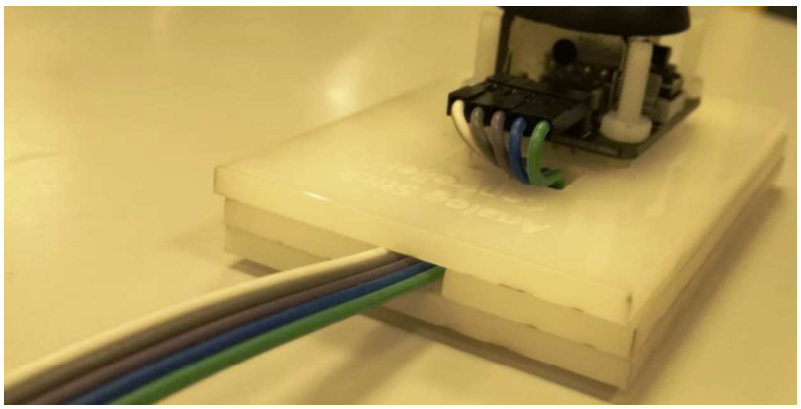
De to knapper er implementeret som interrupts på pin 2 for controller et og på pin 3 for controller to. Dette er gjort da, dette er to pins som kan sættes som interrupts, men også da der under opstart af produktet kommer et interrupt på pin 20 og 21, som tidligere var brugt. Disse interrupts formodes at skyldes, at de to pins er forbundet til Arduinoens TWI hardware.
De er på produkt siden opsat som "pull-up switches" hvilket vil sige, at de er aktivt lave, og derfor slår interruptet til når de falder. For at undgå "bounce" er der implementeret en beskyttelse der gør at de kun kører den kode der står i interrupt routinen, hvis der er gået 200 millisekunder siden sidste interrupt. Den kode, der kører i interrupt rutinen, sætter en boolsk værdi til sandt, sådan at der kan tjekkes efter den i koden. Dette gør at selv et tryk der bliver modtaget mens LED'erne opdateres vil registreres og reageres på når koden tjekker efter indput igen.

2.4.2 Analog controllere

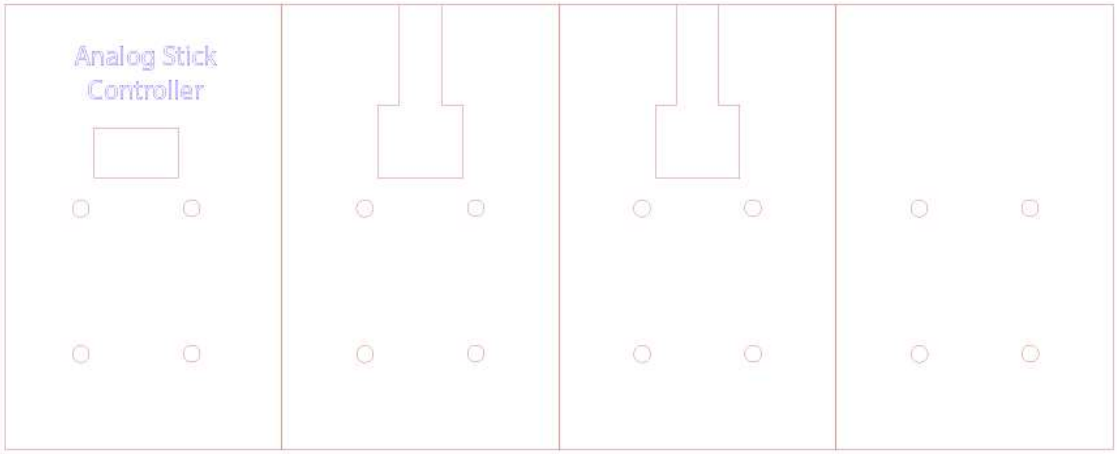
De to akser på "analog stick"'en måles af to potentiometre, som foretager en spændingsdeling af de 5v. Når der ikke røres ved controlleren står den i midten af de to potentiometre og returnere ca. 2.5v på dem. Dette læses som ca. 510 på arduinoens 10-bit ADC.
Potentiometrene "polles" ved brug af gruppens funktion "*detectInput()*" som tager en boolsk værdi, ved falsk måles inputtet fra controller et og ved sandt controller to.
Funktionen gør brug af funktionen "*analogRead()*" som er inkluderet i Arduino IDE'en. "*analogRead()*" gør brug af den indbyggede ADC for at returnere en 10 bits værdi. Denne værdi bruges til at se hvilken position "stick"-en er i, og ud fra det ændre i "*Move[2]*" arrayet, hvis der ikke er bevægelse skrives nul til begge pladser, ellers skrives -1 og 1 alt efter bevægelsesretningen. Index nul i arrayet bruges til bevægelse i x-aksen mens indeks et bruges til bevægelse i y-aksen.

For at kunne give den bedste brugeroplevelse er der lavet to simple holdere til controllerne. De består af udskåret akryl plader og holder controlleren fast på toppen af dem med fire bolte, mens ledningen går ned igennem pladerne og ud af siden.





Her ses den 2D model som én controller bliver skåret udfra på en laserskærer.



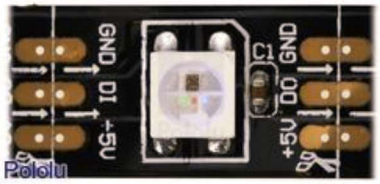
(billede fil af controlleren til de analog sticks til udskæring på laser skærene - Lavet i Adobe illustrator)

2.5 Bruger output (MARK)

Til bruger output eller brugerinterfacet har gruppen valgt at implementere 3 dele: en RGB LED matrix skærm, en LCD skærm og en piezo højtaler.

2.5.1 RGB LED matrix

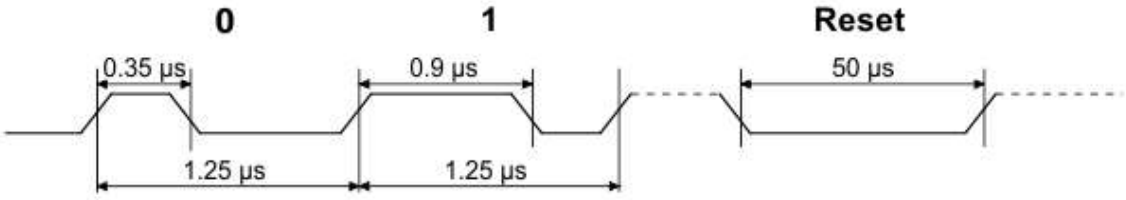
Til selve visning af spillet er der brugt en skærm bestående af 8x8 RGB LED'er som er af modellen WS2812B[5]. RGB modellen gruppen har valgt at bruge, er individuelt adresserbar hvilket betyder, at de kan tændes, opsættes til forskellige farver og slukkes individuelt. De bruger et "One-wire interface" kaldet "NZR communication mode" (Non-return-to-zero) som ikke har en dedikeret clock, men en data pin. Her ses et "close up" billede af komponenten:



(Billede af en WS2812B - http://forum.arduino.cc/index.php?topic=256946.0 besøgt 18-01-2018)

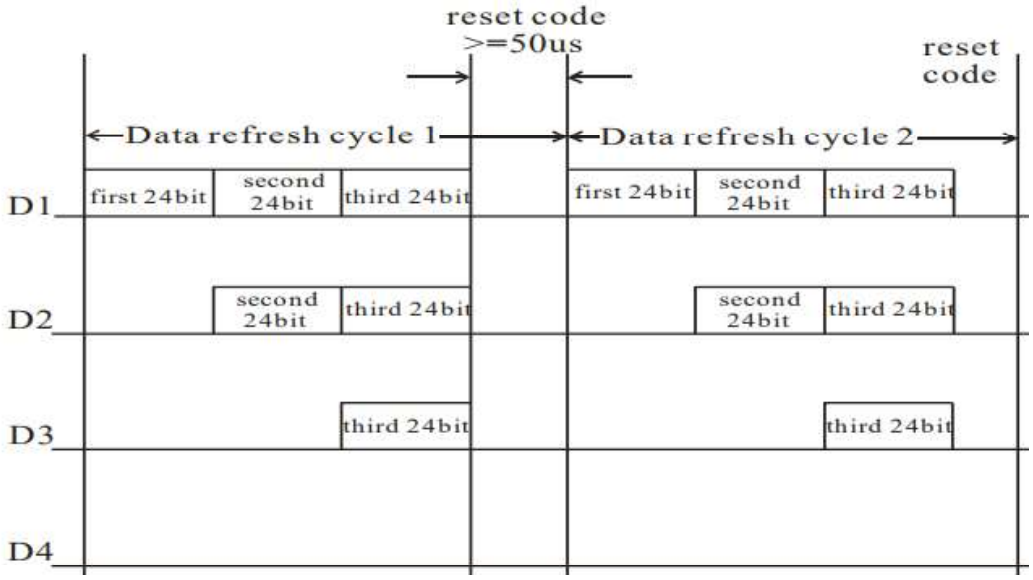
[5] Databladet kan findes på https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf

Da der ikke er en clock, men kun en data pin; encoder man signalet ved, at sende pulser ud med forskellige længder. På diagrammet vises, hvordan et '0' og '1' bliver sendt. Derudover vises et Reset signal som sendes efter, at data til alle LED'erne er blevet sendt ud. Dette Reset signal fortæller LED'erne, at de skal vise de nye farver de har fået tilsendt.[6]

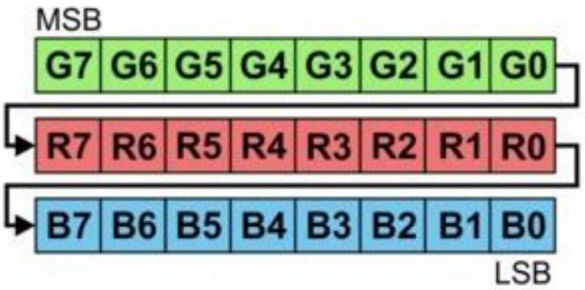


LED'erne er sat i serie eller "cascade method". Det vil sige, at data der skal til den sidste LED i rækken skal "rejse" længst da den skal "skubbes" igennem alle LED'erne, hvilket også kan ses på billedet herunder. Her ses alle LED'erne (D1 som den første, D2 som den anden osv.) og hvordan "third 24bit" pakken til D3 bliver sendt igennem D1 og D2 først.

Data transmission method:



Pakkerne der bliver sendt via protokollen er 24-bit farve kontrols signaler. 8-bit per farve styrer, hvor meget af hver farve der skal vises. De 2^{24} giver cirka 16.8 millioner forskellige nuancer LED'erne kan vise. På næste billede ses rækkefølgen på, hvordan farverne sendes som MSB først i Grøn-Rød-Blå rækkefølge.



Hastigheden på, hvor hurtigt man kan sende data til WS2812B LED'erne er fixed, da der ikke ligesom på andre "smarte" RGB LED'er (f.eks. apa102) er en clock pin der lader brugeren selv

[6] Forklaring af protokollen WS2812B bruger - https://www.pololu.com/product/2547

bestemme, hvor hurtigt LED'erne skal opdateres. WS2812B's opdateringsfrekvens sker ved 800 KHz og er bestemt udfra timings'ne der bruges til, at encode signalet med. Det kan simpelt udregnes og er vist her:

Biblioteket der bruges kan opsættes til at sende ved forskellige hastigheder. Her udregnes, at LED'erne kan opdateres ved de anførte 800 KHz, da det tager 1.25 µs at sende et bit

$$\frac{1}{1.25\mu s * 10^{-6}} = 800.000\ Hz\ \Rightarrow\ 800\ KHz$$

Derudover er det også blevet beregnet,at displayet (8x8 LED'er) kan blive opdateret ved 507.61 Hz da det tager 1970µs at sende de 1.536 bits ud + reset signalet det kræver, for at opdater alle LED'erne.[7]

2.5.1.1 Implementering af WS2812B

Til styring af LED matrixen anvendte gruppen tre biblioteker fra Adafruit:
#include <Adafruit_GFX.h>[8]
#include <Adafruit_NeoMatrix.h>[9]
#include <Adafruit_NeoPixel.h>[10]

De 3 biblioteker er alle sammen lavet til, at de tilsammen kan lave grafik og styrer en skærm. De understøtter mange forskellige RGB LED modeller og forskellige opsætninger af disse.
Her ses et udklip af linje 222 i tic_tac_toe main filen:

```
222 Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
223                                                NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
224                                                NEO_MATRIX_COLUMNS + NEO_MATRIX_ZIGZAG,
225                                                NEO_GRB            + NEO_KHZ800);
```

Her ses, forskellige macroer som bruges som parametre til opsætningen af gruppens 8*8 opsætning af LED matrixen som et objekt der hedder "matrix". Det ses også, at der bliver angivet at dataen skal angives som NEO_GRB (Grøn rød blå) og at bitstream'en skal være NEO_KHZ800.

Derudover er der blevet lavet nogle Macroer for nogle bestemte farver for spillerne, cursoren og spille "gitteret" som det kan ses på udklipet fra linje 80 i tic_tac_toe main filen:

```
80 ////////////COLOR CONFIG////////////
81
82 #define PLAYER_1_COLOR matrix.Color(0,0,255)
83 #define PLAYER_2_COLOR matrix.Color(0,255,0)
84 #define PLAYER_3_COLOR matrix.Color(150,150,150)
85 #define CURSOR_1_COLOR matrix.Color(255,0,255)
86 #define CURSOR_2_COLOR matrix.Color(255,255,0)
87 #define GRID_COLOR matrix.Color(150,150,150)
```

Player_3_COLOR macroen bruges til, at tegne animationen når spillet bliver uafgjort.

NeoMatrix biblioteket opretter ligesom LiquidCrystal et objekt, hvor man kan kalde metoder på det objekt. Ligeledes i setup() i koden initialisers matrixen også med en .begin() metode.

---

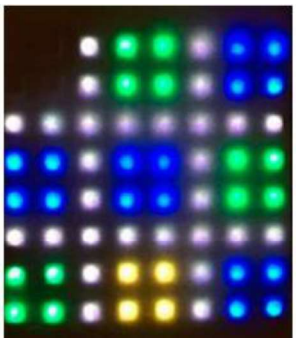
[7] Se bilag for udregning af strømforbrug på WS2812B
[8] https://github.com/adafruit/Adafruit-GFX-Library - Adafruit GFX Bibliotek
[9] https://github.com/adafruit/Adafruit_NeoPixel - Adafruit NeoPixel Bibliotek
[10] https://github.com/adafruit/Adafruit_NeoMatrix - Adafruit NeoMatrix Bibliotek

```
247   matrix.begin();
248   matrix.setBrightness(40);
249   matrix.fillScreen(0);
250   matrix.show();
```

Her ses også, at der bliver sat en lysstyrke til 40 udaf 255 (datatypen er "uint8_t") som gruppen fandt passende. Herefter bruges methode .fillScreen(0) som kan bruges til, at sætte hele skærmen/alle LED'erne til at vise en bestemt farve. I dette tilfælde skrives der '0' som betyder, at alle farverne skal slukkes. Herefter kaldes .show som sender data til LED'erne.

Når programmet kører i main loop bliver der kun skrevet til LED Matrixen når der spilles. Når spillet starter tegnes spille banen op. Under spillet bruges der en cursor på skærmen som kan flyttes rundt og når brugeren lægger en brik tegnes der enten en blå eller grøn spillebrik. Alle disse elementer bliver skrevet til LED matrixen, med .drawPixel(`int16_t x, int16_t y, uint16_t color)` metoden. Funktionen modtager et x og y koordiant i matrixen som biblioteket holder styr på samt en farve som koordinatet skal farves.

Matrixen der holder styr på spillets spilleplade er implementeret som et 3x3 array. Men da selve skærmen er 8x8 skaleres brikkerne op til 2x2. Dette sker ved, at drawPieceNM(..) kalder drawPixel(..) metoden 4 gange for, at sætte en 2x2 brik på banen.



```
255 /* NEW LED MATRIX FUNC
256  * Function draws a piece on the NEW LED matrix from
257  */
258 void drawPieceNM(char posX, char posY, int color){
259   matrix.drawPixel((posX * 3),(posY * 3),color);
260   matrix.drawPixel((posX * 3)+1,(posY * 3),color);
261   matrix.drawPixel((posX * 3),(posY * 3)+1,color);
262   matrix.drawPixel((posX * 3)+1,(posY * 3)+1,color);
263 }
```

Spillebanen består af 1x8 hvide "streger" som bliver tegnet med drawGridNM() funktionen hver gang et spil starter. Her benyttes samme .drawPixel() metode til, at skrive til hver enkelt LED.

```
264 extern void drawGridNM(){
265   for (char y = 2; y < 6; y += 3){
266     for (char x = 0; x < 8; x++){
267       matrix.drawPixel(x, y, GRID_COLOR);
268     }
269   }
270   for (char x = 2; x < 6; x += 3){
271     for (char y = 0; y < 8; y++){
272       matrix.drawPixel(x, y, GRID_COLOR);
273     }
274   }
275 }
```

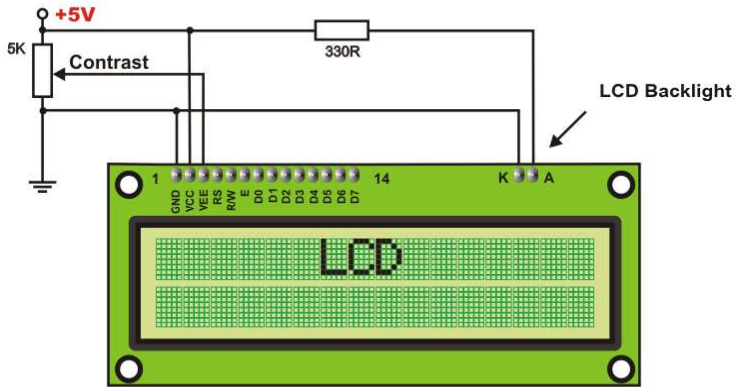
2.5.2 LCD skærm

Da RGB LED skærmen kun bliver brugt til at vise spillet på, manglede der en måde, at brugeren kunne interagere med menuen på. LCD displayet giver brugeren mulighed for, at se hvorhenne han er i programmet og dermed også, hvor henne i programmet en evt. fejl er opstået. LCD

displayet giver også brugeren mulighed for, at skifte navn og se, hvis tur det er. Dette vil blive forklaret senere i rapport.

På billedet ses, LCD displayet i menuen lige efter opstart af programmet.

Til styring af LCD displayet bruges "LiquidCrystal" biblioteket som kan styrer et text baseret LCD display controller af typen Hitachi HD44780 mm. LCD displayet er 16 linjer gange 2 række, har mulighed for Backlight og er meget nemt, at styrer da det er veldokumenteret på Arduino siden om biblioteket.



(Billede fra google søgning der viser opsætningen af Backlight på displayet - http://www.todopic.com.ar/foros/index.php?topic=39189.0)

2.5.3 Piezo buzzer

Udover LCD skærmen bruges der en Piezo buzzer til, at give lyd når en spiller placere en brik eller prøver, at placere en brik forkert. Dette er med til, at forbedre brugerens oplevelse af produktet, da det ikke er sikkert, at brugeren via LED matrix skærmen kan se, at han prøver at placere en brik forkert, men en "fejl" lyd indikere, når der bliver forsøgt et ugyldigt træk.

(http://maltarotors.com/wp-content/uploads/2016/04/piezo-buzzer.jpg)

2.6 Lokal highscore (Mads)

Den lokale highscore liste gemmes på et SD-kort som er tilsluttet en SD-kortlæser. Kortlæseren kommunikerer med arduinoen via SPI, og er forbundet på arduinoens SPI pins, 50 er MISO (Master In Slave Out) , 51 MOSI (Master Out Slave In), 52 SCK (Clock) og 53 er SS (Slave Select).

Der kommunikeres med kortlæseren ved hjælp af "SD.h" biblioteket, som er nogenlunde dokumenteret, men dette bibliotek er skrevet som en let implementation af "SDFat" biblioteket af Bill Greiman, som er detaljeret dokumenteret. Denne dokumentation giver en god mulighed for at få et overblik eller dykke ned i de enkelte funktionen, selvom "SD.h" ikke indeholder alle de klasser og muligheder som findes i "SDFat".
Interaktionen med SD kortet foregår når en spiller har vundet et spil, og spil tilstandsmaskinen kommer i "wonP1" eller "wonP2" tilstanden, her kaldes funktionen "*writeWinnerToSD()*" som tager en boolsk værdi som argument, hvis den modtager falsk bruger den spiller 1 som vinder og hvis sandt spiller to.
Funktionen gør brug af at filobjektet i det bibliotek der bruges (”SD.h”) er af klassen ”File” som arver fra ”stream” klassen. Dette gør det muligt at bruge ”stream” funktionen ”xx.*find()*” som leder efter en streng i en fil. Funktionen bruges til at se om en spillers navn allerede eksisterer i highscore listen, hvis dette er tilfældet så trækkes scoren ud og der lægges et point til. Hvis dette ikke er tilfældet skrives spillerens navn til listen med en score på 1.

Det filformat der gøres brug af i listen er, først tre karakterer til at holde point, dette giver en maksimal scorer på 999. Herefter kommer et mellemrum til at separerer point og navn. Herefter kommer fem karakterer til at holde navnet, efterfulgt af en ”newline” karakter, som viser at det der kommer efter er en anden spiller. Dette vil altså se således ud:

score.txt | README

```
037 MADS
109 MARK
002 NBETB
004 MARKA
003 MADSB
002 MADSX
003 KARK
004 NADS
001 MCRK
001 LAES
002 QEXOF
001 NB Q
002 PB RB
001 UI QH
014 KALO
002 NARK
002 NADSA
004 NARKA
```

Ved at bruge et fast format er det nemmere at flytte sig rundt i filen, da alt har en bestemt længde. Dette gør at fil objektets pointer kan flyttes med faste og skalerbare intervaller som afhænger af score og navne længde, for at udskifte scoren.

2.7 Menuen (Mads)

Som tidligere beskrevet er menuen implementeret som et "switch" udtryk med fire tilstande. I tilstanden "running_menu" vil brugeren på LCD skærmen se på øverste linje "Welcome…" og på nederste linje vil der stå hvor I menuen brugeren er. Det første menupunkt hedder "Play Tic Tac Toe", her vil menuen altid stå til at starte med, og dette menupunkt starter spillet. Det andet menupunkt hedder "Change name", og vil sætter brugeren i gang med at ændre begge navne. Programmet holder styr på hvor brugeren er i menuen med en "*char toggle_menu*" værdi, som bruges til at printe det menupunt brugeren er på, når den ændres og som vendes rundt når man når enden af listen. Dette design gør det nemt at tilføje nye menupunkter, som for eksempel et nyt spil.
Når menuen initieres sættes en boolsk værdi "*inMenu*" til sand. Denne værdi sørger for at interrupt rutinerne reagerer på spiller 1's knaptryk, når menuen forlades sættes den til falsk igen.

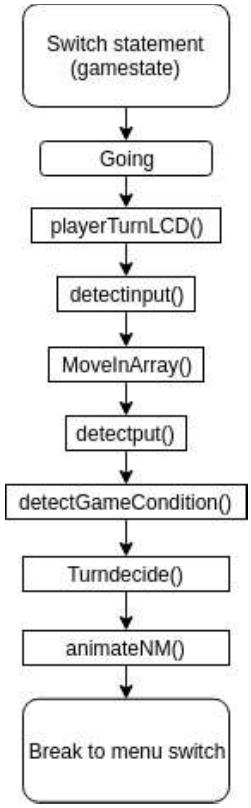
2.7.1 Navneskift

Når brugeren vælger at ændre navne, vil der på øverste linje i displayet stå hvilken bruger, der er ved at blive indtastet navn for. Navnet indtastes ved at rykker controlleren fra side til side og på den måde vælge et bogstav. Det er muligt at vælge mellem alle store bogstaver i det engelske alfabet og mellemrum. Når der trykkes på knappen vælges bogstavet, og når det sidste bogstav er indtastet gemmes navnet. Karakter cursoren vil altid starte på det bogstav, som står på samme plads i det navn der er gemt i forvejen.
I koden kaldes funktionen "*inputName()*" to gange, en for hver spiller. Denne funktion tager en boolsk værdi for hvilken spiller der skal ændre navn. Funktionen holder også styr på hvis tur det er når den kaldes, sådan så den ikke påvirker "*bool turn*" variablen (som bruges til, at holde styr på hvis tur det er i spillet), selvom der skiftes navn. Dette bliver den nød til, da den bruger variablen til at give kontrollen til hver af de to spillere således de ikke kan påvirke hinandens navne.
I koden rykkes cursoren ved at bruge "*detectInput()*" funktionen og derefter se på "*Move[0]*" værdien om der er rykket på cursoren langs x-aksen. Ud fra denne værdi tillægges eller fratrækkes karakter cursoren en. Derefter køres den gennem et "switch" udtryk som sørger for at holde den inde for parametrene. Det er derfor muligt at undslippe parametrene hvis der indtastes navne som indeholder andet end store engelske bogstaver eller mellemrum. Dette bør ikke gøres, da det kan give problemer i highscore formatet hvis der indsættes kontrol karakterer i stedet for bogstaver.

2.7.2 Spil

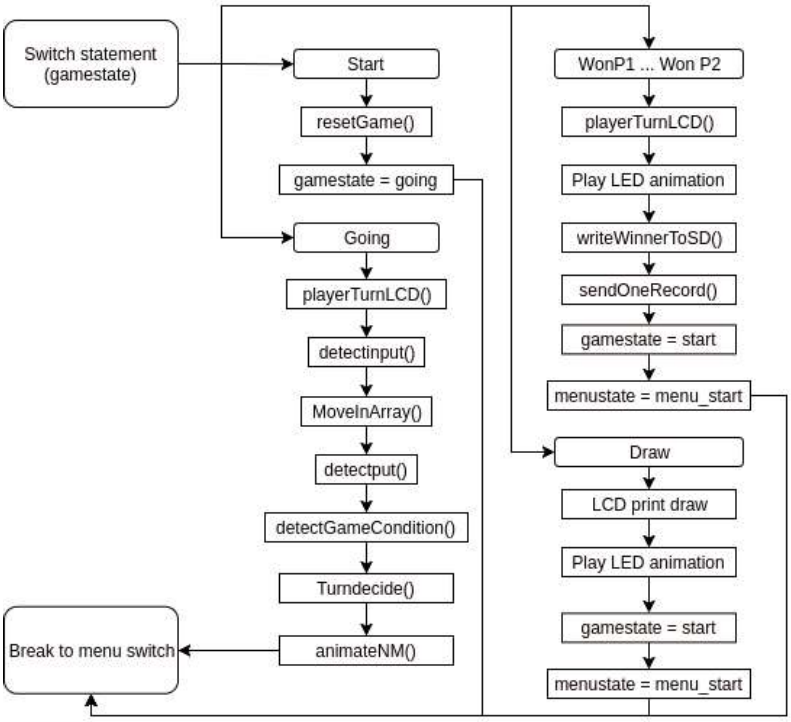
Når menupunktet "Play Tic Tac Toe" vælges vil menu tilstandsmaskinen gå til tilstanden "running_game" og spil tilstandsmaskinen vil aktiveres.
For brugeren vil et spil kryds og bolle begynde, med to farver i stedet for symboler. Spiller et vil være blå og have en lilla cursor mens spiller to vil være grøn og have en gul cursor. Det er

muligt at flytte cursoren over hele banen, men ikke muligt at lægge en brik hvor der allerede ligger en. Hver gang der lægges en brik vil turen overgå til den anden spiller, og på LCD skærmen vil det kunne ses hvis tur det er. Spillet vil fortsætte til en spiller vinder eller alle felter er fyldt ud, i hvilket tilfælde det er blevet uafgjort. Når spillet er afgjort vil vinderen skrives på LCD skærmen og cursoren forsvinde et sekund. Herefter vil der spilles en vinder animation over LED matrixen i farven på vinderens brikker eller hvid for uafgjort. Så sendes der data til RPI'en og bagefter vil brugeren ende tilbage i menuen.



Spil tilstandsmaskinen vil når spillet startes være i tilstanden "start" her vil koden nulstille alle spillets variable og sætte spil tilstandsmaskinen til "going".
I denne tilstand vil programmet gennemløbe de to tilstandsmaskiner og køre den primære spilkode. Spilkoden består af syv funktioner som styrer spillet og opdaterer outputtet i form af LED matrixen, ***Piezo buzzeren*** og LCD skærmen. I diagrammet til højre kan der ses et udklip af flowchartet for spil tilstandsmaskinens "going" del, og den del der udgører spillets logik.
Den første funktion der køres er "*playerTurnLCD()*" denne funktion skriver hvis tur det er til LCD displayet. Funktionen er også lavet sådan den skriver vinderen når nogle har vundet.
Funktionen "*detectinput()*", som omtalt i kapitlet om indput måler den på inputtet for den spiller hvis tur det er og bruger målingerne til at ændre i arrayet "*Move[2]*".
Næste funktion er "*moveInArray()*". Denne funktion bruger det der står i "*Move[2]*" arrayet, tjekker om det flytter cursoren til en plads inden for spillepladen og hvis det er, flytter den cursorens position til den nye plads.
Derefter kaldes "*detectInput()*" som også tager turen som argumentet. Denne funktion kigger på om knap interruptsne er blevet aktiveret, alt efter hvis tur det er. Hvis der er trykket tjekker den om det er en ledig plads på spillepladen og hvis det er, ligger funktionen en brik der for den spiller.
"*detectGameCondition()*" tjekker om der er blevet lagt en brik, hvis der er, tjekker funktionen rækken og kolonnen som brikken ligger i for at se om spilleren har vundet. Hvis brikken er lagt på en diagonal tjekkes der om der er vundet på denne måde. Hvis der ikke er fundet en vinder, tjekker funktionen inden den returnerer den om spillet er blevet uafgjort. Funktionen ændre tilstanden i spil maskinen til enten "wonP1", "wonP2" eller "draw".
"*turnDecide()*" da den variabel der holder stur på hvis tur det er, bliver brugt meget af de andre fnuktionen er det først her i den sidste del af spillogikken den ændres i. Funktionen ser om der er blevet lagt en brik og hvis så giver den turen til den anden spiller.
Den sidste funktion som kaldes i spillogikken er: "*animateNM()*" Denne funktion funktion tager spillepladen som holdes styr på som et 3x3 array og skriver det ud til LED matrixen, samt tegner de linjer som opdeler felterne. Derudover hvis spillet er ikke er slut printer den cursoren.

Når spilmaskinen vindes går den ind i "wonP1" tilstanden eller "wonP2" tilstanden, disse to er implementeret i den samme "case" i "switch" udtrykket. I denne "case" printer først vinderen med "*playerTurnLCD()*" funktionen, herefter spilles en win animation i den vindende spillers farve. Herefter skrives vinderen til SD kortet, som beskrevet i kapitlet om den lokale highscore liste. Herefter kaldes funktionen "*sendOneRecord()*" som sender navnet på vinderen til RPI'en. Herefter sættes spil tilstandsmaskinen til "Start" således den altid starter i start og menu tilstandsmaskinen til "menu_start" således brugeren retuneres til menuen korrekt.
Hvis spillet bliver uafgjort spiller en animation med "PLAYER_3" farven, hvorefter den på samme måde retunerer brugeren til menuen.



forklar funktionerne overordnet for at give en ide om hvordan spillet fungerer og medtag spil switch case diagram.
Hele spilkoden er mellem 13 og 14 millisekunder om at køre en gang, også selvom der modtages indputs. Denne opdatereingshastighed er for høj. Derfor er det indlagt i tidsstyringen, at under spillet vil spillogikken køres med 200 millisekunders mellemrum. Dette gør det muligt for spilleren at flytte sig et enkelt felt af gangen, således at koden ikke når at tjekke to gange og vælger at flytte cursoren to gange, inden spilleren kan nå at løfte fingeren fra controlleren. Den samme form for tidsstyring er indsat i menuen og i navneændrings funktionen. Selvom den i navneændrings funktionen blot er et simpelt delay, og ikke den overordnede tidsstyring.

2.8 Hardware opsætning

Hardwaren er opdelt i 2 hoveddele: Arduino delen, RPI delen, LED matrix og strømforsyning.

2.8.1 Arduino (MARK)

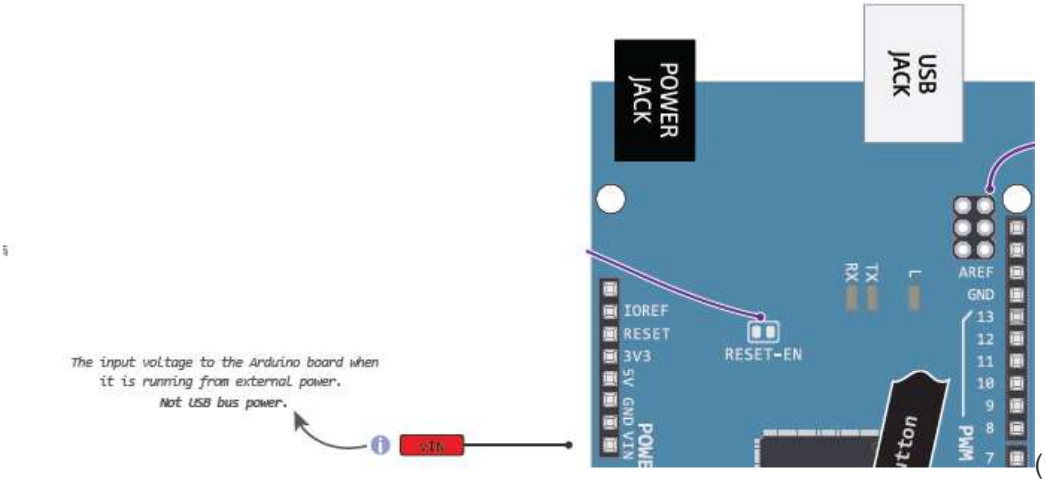
Arduino delen består af følgende komponenter:

Antal	Navn	Anvendelse
2 stk.	Analog sticks	Styrer spillet og i menuen.
1 stk.	Piezo buzzer	Give bruger feedback
1 stk.	SD kort læser	Lave lokal backup af highscore listen.
1 stk.	Potentiometer	Justering af kontrast på display.
1 stk.	LCD Hitachi HD44780 display	Bruger feedback og navigering af menuen

Se Appendix B for hardware opsætning af Arduinoen

2.8.2 Strømforsyning og RGB LED Matrix (Mark)

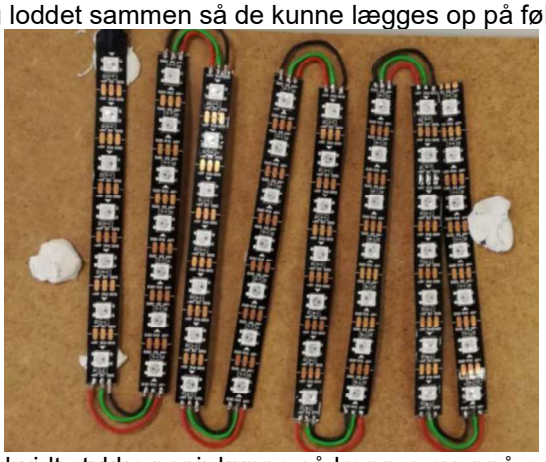
Arduino kan blive forsynet med strøm på forskellige måder:
Igennem USB stikket kan den blive forsynet med 5V og gør at den kan blive programmeret.
På Vin og igennem "Power jack" kan arduino få strøm fra en ekstern strømforsyning der er på mellem 7-12V.[11]



(Billedet taget og klippet til fra: http://3.bp.blogspot.com/-q1ACdRHilmY/V5irRui-L1I/AAAAAAAAEeA/8OzDljqKdSk_Iak5EtDEiZ8GuF6q4S6JwCK4B/s1600/pinOutMEGA2560.PNG)

Da arduinoen kun kan levere 20 mA per port skal der bruges en ekstern strømforsyning til, at levere strøm til RGB LED Matrixen. Den skal have mellem 500 mA og op til 2-3A alt efter hvilken lysstyrke den er sat til.[12] Gruppen startede ud med, at arbejde på LED Matrixen og den strømforsyning som skulle levere strøm til dioderne.

Den første prototype af matrixen blev lavet af en 1 meter RGB strip som blev klippet i 6x7 dele og loddet sammen så de kunne lægges op på følgende måde:

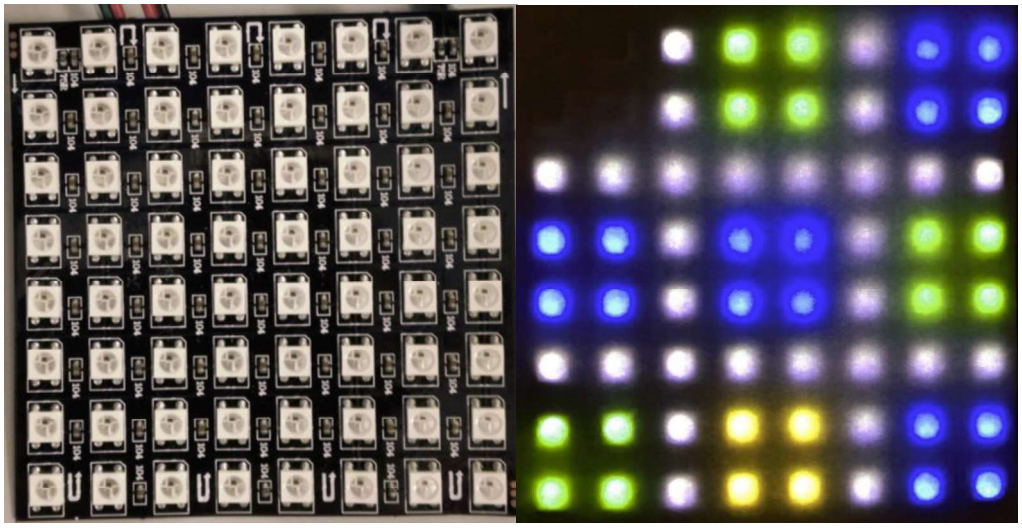


Et hvidt stykke papir kunne så lægges ovenpå og man kunne få et 6x7 spilleplade, hvor hver LED havde forskellige farver. Dette setup var ikke helt optimalt da gruppen gerne vil have et større spilleområde, men det var stort nok til at begynde at arbejde på spil logikken.

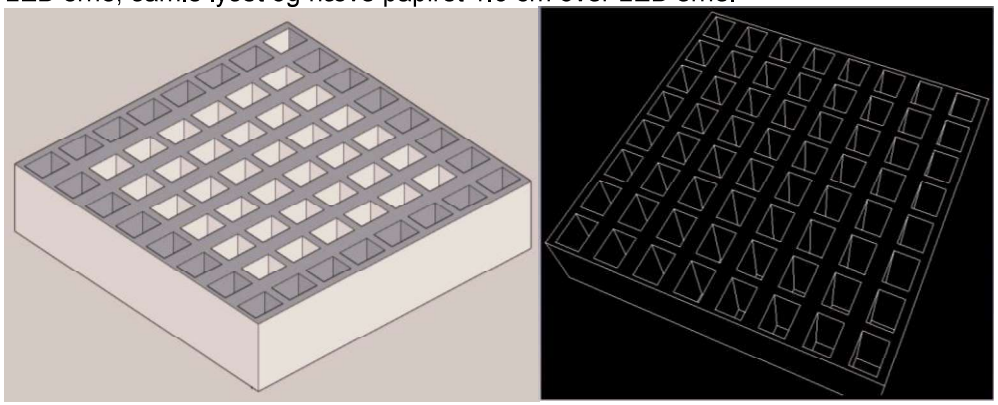
---

[11] Ardunio mega 2560 dokumentation - https://store.arduino.cc/usa/arduino-mega-2560-rev3 (Besøgt 17-01-2018)
[12] Se bilag for udregning af strømforbrug på WS2812B

Halvvejs i projektet modtag gruppen et 8x8 RGB LED matrix[13] som passede bedre i størrelsen og som i forvejen var loddet sammen som et Matrix. Her ses et billede af matrixen og spillepladen oplyst. Som det kan ses fordeler lyset sig ikke så pænt på papiret.

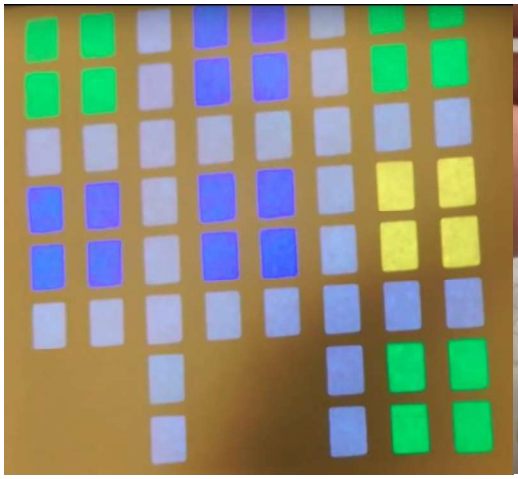
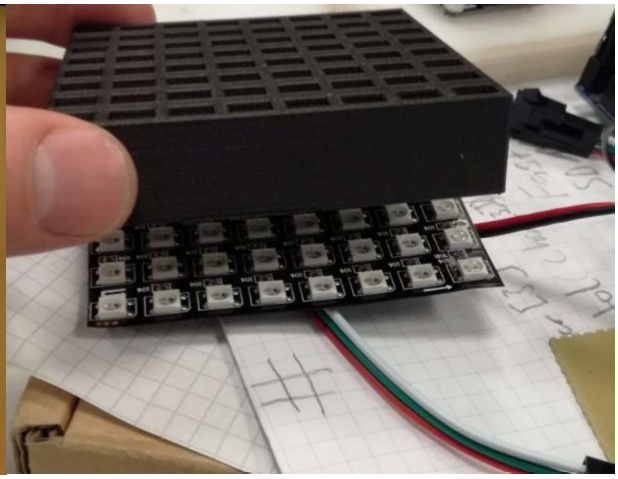


Så gruppen designede i et 3D-program "SketchUp" en matrix som kunne lægges ovenpå LED'erne, samle lyset og hæve papiret 1.5 cm over LED'erne.



Matrixen gjorde at LED'erne gav en rigtigt pæn farve på det hævede papir og den før runde lyskegle nu blev centreret og gjort firkantet.

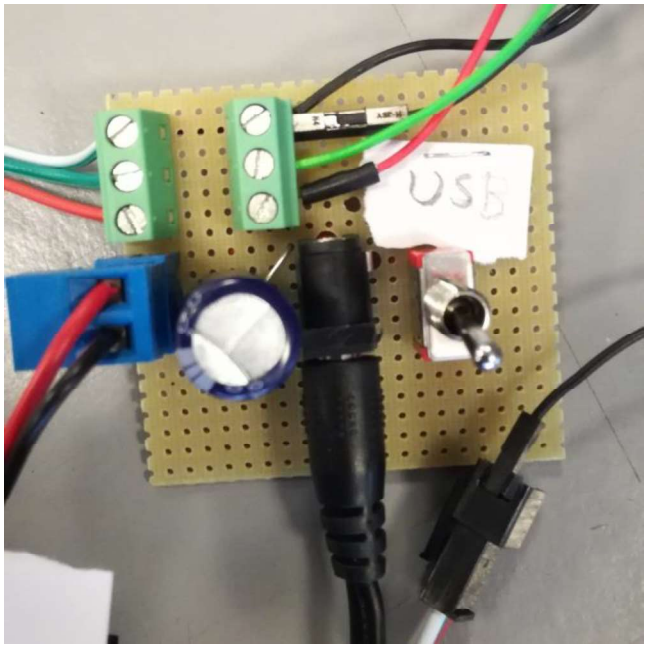
[13] https://www.aliexpress.com/item/1pcs-lots-WS2812B-5050-RGB-SMD-8-8-pixels-digital-flexible-dot-matrix-individually-addressable-led/32818045969.html?spm=a2g0s.9042311.0.0.WPWhDt

Som beskrevet tidligere, skulle matrixen have en ekstern strømforsyning. Gruppen fandt en 5V og 3A adapter der kunne sættes i et stik nede fra komponent shoppen der kunne blive loddet på et vævboard.

Se Appendix B “Strømforsyning og LED matrix setup:” for billede af opsætningen i fritzing.

På fritzing tegningen af gruppens vævboard modtages 5V fra en ekstern strømforsyning og levere strøm til arduino og LED Matrixen. Det kan ses, at der bl.a. fra arduino kommer GND og en grøn ledning som er DATA til at styrer LED matrixen.



Her ses strømforsyningen der modtager 5V fra en adapter. Strømforsyningen forsyner altid LED matrixen med strøm. Hvis knappen ikke står i “USB” positionen driver strømforsyningen også arduino’en. “USB” positionen skal bruges når USB kablet er sat til arduino’en og skal programmers. Denne switch er sat på så arduino ikke både får strøm fra USB kablet og fra den 5V ekstern strømforsyning på samme tid, hvilket helst skal undgås da det kan skade arduino’en.

2.9 Two Wire Interface between Raspberry Pi 3 and Arduino mega 2560(Kaloyan).

Kaloyan Penovs original idea was to develop the I2C library in C for the TWI module in Arduino mega. The teacher, Kurt accepted the idea. He extended the idea with solving a computationally heavy mathematical problem by sharing the workload between a couple of Arduinos so that the work could be done faster and communicate the pieces of the result to the requesting (master) Arduino via I2C. Kurt also added the requirement to work in group with someone else.
Kaloyan had to find a project where his topic of I2C was relevant. Mads and Christians project to build a Snake game with led matrix and joystick on Arduino and communicate the results to a Raspberry Pi via USB seemed to fit Kaloyans idea if the USB communication was exchanged with I2C communication. That was agreed to be possible and Kaloyans task was changed from the original idea to the new one in communicating results from Arduino to Raspberry Pi, data processing and sending the data to a web server for visualization in the web. One can see that this new task because of the group work requirement is minimum one abstraction level higher than the original idea, which was much closer to the hardware. That as will be seen in the end provided applying the learnings in the course to different embedded systems and programming languages different than Arduino and C with its differences and similarities. This new situation extended the number of clients to two more in the face of Mads and Christian, who can be looked as internal clients or co-workers with different tasks but also common aim and therefore meeting points of coupling different implementations. This gets the things closer to what is really happening in the industry, where different departments experience the challenges of incorporating each other's work to reach the common goal of delivering a product on time and keep the end client, Kurt, satisfied.

2.9.1 Two wire interface module on Arduino mega 2560.

2.9.1.1 Overview(Kaloyan)

From the data sheet for the Atmel ATmega 8 bit microcontroller based on the AVR architecture, basically combining 32 general purpose registers that are directly connected to the arithmetic logic allowing for two registers to be accessed per clock cycle. Along with a lot of fantastic features there is the needed byte oriented 2-wire Serial Interface or called TWI. One can shut down the clock to the module by writing 1 to bit 7 in the power reduction register (PPR0 (0x64)). The TWI module connects to the world outside via pins 20(SDA) and 21(SCL) mapped to port pins PD1 and PD0 bits or bits 1 and 0 of Port D. It is important to mention here that these pins are normally used for external interrupt source to the mc, but by setting the TWEN (two wire enable) bit of the TWCR (two wire controller) register to 1 we can change their functionality to input output pins for the two wire serial interface. The documentation describes that in that mode an input signal is filtered for spikes less than 50 ns. Interrupt vector for the TWI is defined with number 40 and program address $004E (Table 14-1, page 102 of the data sheet), effectively if the interrupt vector for the TWI is enabled its address is calculated by adding the program address to the start address of the Boot Flash section so that the typical TWI handler address is 0x004E (page 103).

The features provided by the TWI are: -2 bus line communication interface; - Master and Slave modes; - the module can operate as transmitter and as receiver; - up to 128 different slave devices plus a general call from the master if all 7 bits are 0; - up to 400kHz data transfer speed; - address recognition wakes-up the AVR if in sleep mode.

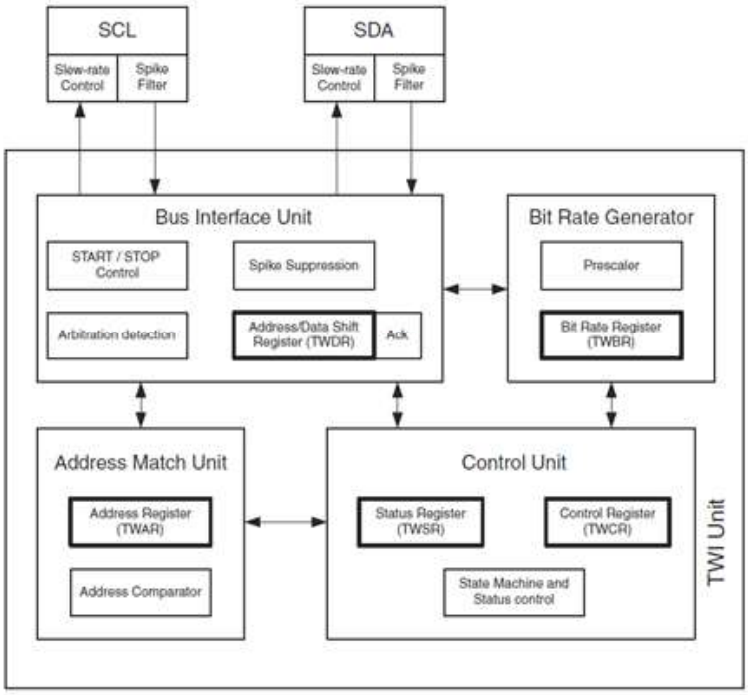
The serial data and clock lines need to be pulled up to be able to work as a bus for the TWI, and at this moment the bus is considered free. Arduino mega have internal pull up resistors that will be activated by writing logical high to the SDA and SCL pins as seen later, this is exactly what is done in the initialization phase of the typical library for the module twi.h twi.c used by the wrapper Wire.cpp library to initialize the module.

The general picture is that each read or write request can be initiated and terminated only by the master, which is also the device that generates the clock. Each slave device plays a passive role, but can send acknowledge signals and can write data on or read data from the

bus. A data bit is considered to be valid if the data line (SDA) is stable for the whole period when the clock line (SCL) is high. Because the master is the one that starts and stops the data transmission, the slaves will listen for a start condition which is considered when the master pulls SDA low while the SCL is high or a stop condition when the master pulls the SDA line up while SCL is high. In between the start and stop condition is the data transfer process and the bus is therefore considered busy. It's the TWI module job to listen for start and stop conditions and reflect that to its registers and takes the appropriate action. A start condition is followed by 9 bits address packet, where the first 7 bits are the address of the slave, bit 8 is the read (1) or write (0) operation bit and bit 9 is where the slave TWI module will pull the SDA line low on the 9th clock cycle to indicate that it acknowledges (ACK), confirms the acceptance of the address packet.

There is no discussion for general calls or arbitration between masters here as it is not relevant for the current scenario. Prolonging the SCL duty cycle by holding the SCL low by the slave to reduce transfer speed is also not discussed here.

When the master sends or receives data to/from the slave a data pack is send/received consisting of 9 bits, where the first 8 is the data byte starting from the most significant bit and during the 9 clock cycle the receiver need to acknowledge the received byte by pulling the SDA line low or let it be high if the data byte was not received, basically considered as not acknowledged (NACK). Interesting here is that the receiver needs to issue NCK when the last or only byte of data has been received. This is a bit counter intuitive, but this is the way the slave says I am done and as the master is the controller of the communication it is also its responsibility to know the current state and take the appropriate action (if 5 bytes need to be send but the master got NACK after the first 2 bytes then the master needs to decide what to do).



The picture above was adopted form Atmel 8bit microcontroller data sheet(2549Q–AVR–02/2014), page 242.

In order to initialize the TWI module one need to first be sure that the module is not shut down by writing 0 to bit 7 in the power reduction register (PPR0 (0x64)). After that the internal pullups can be activated by using the Arduino digitalWrite function and write HIGH for SDA and SCL pins. The next task is to calculate and assign the Bit Rate Register in the Bit Rate Generator, the calculation requires a prescaler value which is defined by the first 2 bits in the status register of the Control Unit. As seen from the table on the right below the prescaler value is defined so that for the 2 bits can hold the values 1, 4, 16 and 64. Setting the prescaler to 1 is

achieved by setting the status register to 0 during initialization (TWSR = 0x00, where #define TWSR (*(volatile uint8_t *)(0xB9))), together with the rest of the writable status bits.

Bit	7	6	5	4	3	2	1	0	
(0xB9)	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

TWPS0	TWPS1	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

The bit rate calculation can now be expressed from the following equation (2549Q–AVR–02/2014, page 242):

$$SCL_{freq} = \frac{F_{CPU}}{16 + 2 * TWBR * 4^{prescaler}}$$

$$SCL_{freq} = \frac{F_{CPU}}{16 + 2 * TWBR * 4^{prescaler}} \rightarrow TWBR = \frac{\left(\frac{F_{CPU}}{SCL_{freq}} - 16\right)}{2} = \frac{\left(\frac{16\,000\,000}{100\,000} - 16\right)}{2} = 72, for\ 100kHz$$

$$TWBR = \frac{\left(\frac{F_{CPU}}{SCL_{freq}} - 16\right)}{2} = \frac{\left(\frac{16\,000\,000}{400\,000} - 16\right)}{2} = 12\,, for\ the\ maximum\ clock\ of\ 400kHz$$

```
21    // Set bit rate
22    TWBR = ((F_CPU / TWI_FREQ) - 16) / 2;
```

(TWIlib.c by Nicholas Zambetti, Modified 2012 by Todd Krein)

myI2C.h | myI2C.c | twi.c | Wire.cpp | TWIlib.c

```
1  #ifndef MY_I2C__H__
2  #define MY_I2C__H__
3  // TWI bit rate
4  #define FREQ 100000L
5  #define F_CPU 16000000L   // in Hz
```

By default, the minimum clock frequency of 100kHz is defined.

The above myI2C.h and myI2C.c were partly developed by looking and understanding the implementations found in twi.c, twi.h and TWIlib.c, TWIlib.h. They will be applied as appendix to the current document, but the code is mainly rewritten aiming to understand it and slightly modified from the existing implementations.

2.9.1.2 TWI module internal units description and operations(Kaloyan)

The bus interface unit holds the TWDR (address and data shift register), as said earlier in this register data can be read or written to. This unit also contains the acknowledge register that is indirectly accessible: - in receiving mode the software can set it in the control register (TWCR), in a way to indicate ACK or NACK; - in transmitter mode the TWSR (status register) can read to determine it. There is also a start/stop controller in this unit, responsible for detection and generation of start and stop signals.

The Address match unit has the task to detect general calls or direct addressing to the local address (only if configured in the control register) and inform the control unit if addressed in one of the two ways described above.

The control unit holds the control register, this unit has the responsibility to “monitors the TWI bus and generates responses corresponding to setting in the TWI Control Register” (2549Q–AVR–02/2014, page 243). TWI interrupt flag (TWINT) is set in the control register when an event requiring action from the software occurs on the bus. In the following clock

cycle the TWSR (status register) can be read and compared to the possible event codes based on the mode and role of the device. One can keep in mind that the information in the status register is only relevant if the TWINT flag was set, and this also means that at that moment the module will keep the SCL line low and wait for the software to finish its work, to continue transmission the software need to clear the TWINT flag. The TWI module will issue interrupts after all type of bus events as start condition was transmitted or byte was received.

The following code has the purpose to illustrate how this is achieved in software. If we imagine that the master is in transmitting mode and has transmitted 1 of 5 bytes of data already and the 1 byte was successfully received by the slave and the slave pulled the SDA line low on the 9th clock cycle to indicate ACK, this means that if interrupts were enabled, the TWIE (interrupt enabled) bit in the TWCR, after the masters TWI module detect the ACK will call the interrupt vector (the interrupt service routine on line 77 here, myI2C.c). STATUS here is the status of the most significant 5 bits of the status register defined as:

```
8  #define STATUS  (TWSR & 0xF8)
```

The switch statement will read the current status and reach the case on line 85, because this was our scenario, that is: the master is transmitting data and ACK was send by the slave, this scenario was extracted as code from the data sheet (2549Q–AVR–02/2014) on page 249 says:

0x28	Data byte has been transmitted; ACK has been received

This case as many others were predefined as macros in the header file, like this: `56 #define MT_DATA_ACK 0x28 //Data sent and ACK` In line 86 if the max buffer is not reached line 88 will be executed and a byte from the transmitting buffer will be loaded in the data register ready to be transmitted over the bus. Line 89 there is another macro from the header file, responsible for setting the control register up so that the TWI module to proceed with the data transfer over the bus

```
46  #define SEND_TRANSMIT() (TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWIE) )
```

What is really happening here maps to the following TWCR = (1<<7) | (1<<2) | (1<<0) => (1000 0000) | (0000 0100) | (0000 0001) => (1000 0101) The (1<<0) is just to indicate that this bit is set, shifting 0 places to the left does nothing of course. The definition of the control register itself looks like this: #define TWCR (*(volatile uint8_t *)(0xBC)) , the macro is basically a cast to a volatile pointer to unsigned char (8 bits) that is dereferenced to be assigned a value, volatile because is assigned in a interrupt service routine and have to be written safely.

```
76  //this will be called after each i2c hardware operation
77  ISR(i2c_vector){
78      switch(STATUS){//this is the status register vale(TWSR & 0xF8)
79          //Master is transmiter or writing address
80          case MT_SLAW_ACK: //master transmitted -> slave address and write flag, and got ACK
81              //set mode to master transmitter
82              i2c_info.mode = MASTER_TRANSMITTER;
83              //there is no break here so jump to the next case
84          case START_SENT: //start signal was transmitted
85          case MT_DATA_ACK: //one byte of data has been transmitted, and got ACK
86              if(bufferIndexTX < bufferLenghtTX){
87              //load data to the transmit buffer
88                  TWDR = transmitBuffer[bufferIndexTX++];
89                  SEND_TRANSMIT(); //macro -> sent current byte signal to the i2c hardware
90              }else if(i2c_info.repStart){//this transmission is complete but do not relese the bus yet
91                  i2c_info.errorCode = 0xFF;
92                  SEND_START();//macro -> send start signal to the i2c hardware
93              }else{ //all transmiffions are done , its time to exit
94                  i2c_info.mode = READY;
95                  i2c_info.errorCode = 0xFF;
96                  SEND_STOP();//macro -> send stop signal to the i2c hardware
97              }
98          break;
```

Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TWINT - interrupt flag TWI module sets it to 1 when work is finished and waits response from the software.

TWEA – Enable ACK If TWEA is 1 the MCU will respond with ACK if: a byte was received; - master calls the local address; - master calls general: 000 0000.

TWSTA - start flag, TWSTA = 1 means transmit start. Start is send first after the bus is released. TWSTA must be set to 0 by software after start was send.

TWSTO - stop flag, TWSTO = 1 means transmit stop, TWSTO is set to 0 automatically after stop was send.

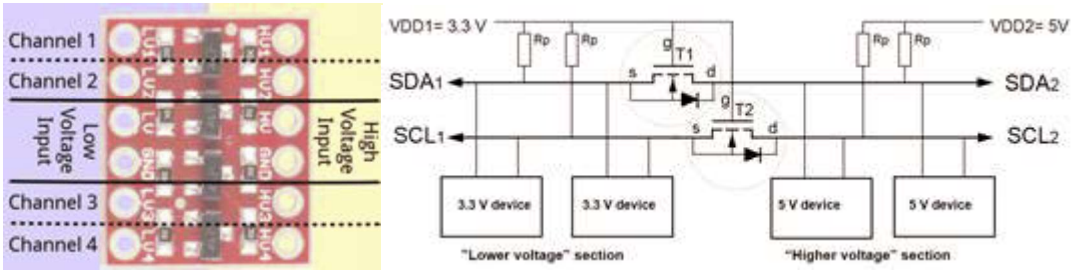
TWEN – is the TWI enable group, TWEN set to 1 means the control of SDA and SCL is given to TWI hardware, TWEN set to 0 stops the TWI hardware.

TWIE – interrupt enabled. If TWIE is 1, when TWINT is set the MCU will execute the TWI interrupt vector.

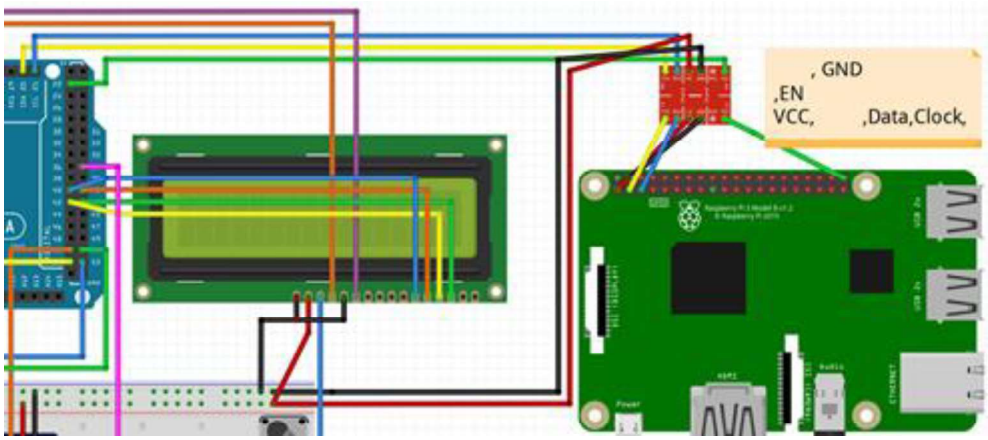
2.9.1.3 TWI module on Raspberry Pi 3 and logic level converter (Kaloyan)

The Raspberry Pi 3 or CM3(compute module 3) as defined in the data sheet contains BCM2837 processor, at the moment of writing this processor is not specifically documented but instead the data sheet refers to processor BCM2835 and its peripherals as they are essentially the same (K14 page 5). TWI is implemented following the Philips I2C version 2.1 January 2000 (K14, page 28), the documentation also says master only operation (there are GPIO supporting slave operation as well and this was also tested in slave operation in the end of the project and worked). In Raspberry Pi the authors refer to this module as Broadcom Serial Controller or BSC. The RPi implementation provides the option for 3 BSC masters, with the following starting addresses: BSC0: 0x7E20_5000, BSC1: 0x7E80_4000 and BSC2: 0x7E80_5000, so for example the status registers address can be calculated as offset from each of the master addresses, the same for the control register and the other more specific register for the RPi implementation (K14, page 28), we are not going into more depth here but briefly mention some.

As Raspberry Pi 3 uses 3.3 V on its SDA (GPIO02 master, GPIO18 slave) and SCL (GPIO03 master, GPIO19 slave) pins (K22 and K23) and Arduino mega uses 5 V, thus there is the need for logic level converter from 5V to 3.3V, to keep the Raspberry Pi 3 GPIO save, when in slave mode. The 4 channel Bi-directional logic level converter shown below was used for all scenarios.



4 channel Bi-directional logic level converter,
Adopted from: - https://cdn.sparkfun.com/tutorialimages/BD-LogicLevelConverter/an97055.pdf



Master mode connection with the converter.
Adopted from: Christian Mark, Modified by: Kaloyan Penov

3. Softwareudvikling og test(Kaloyan side)

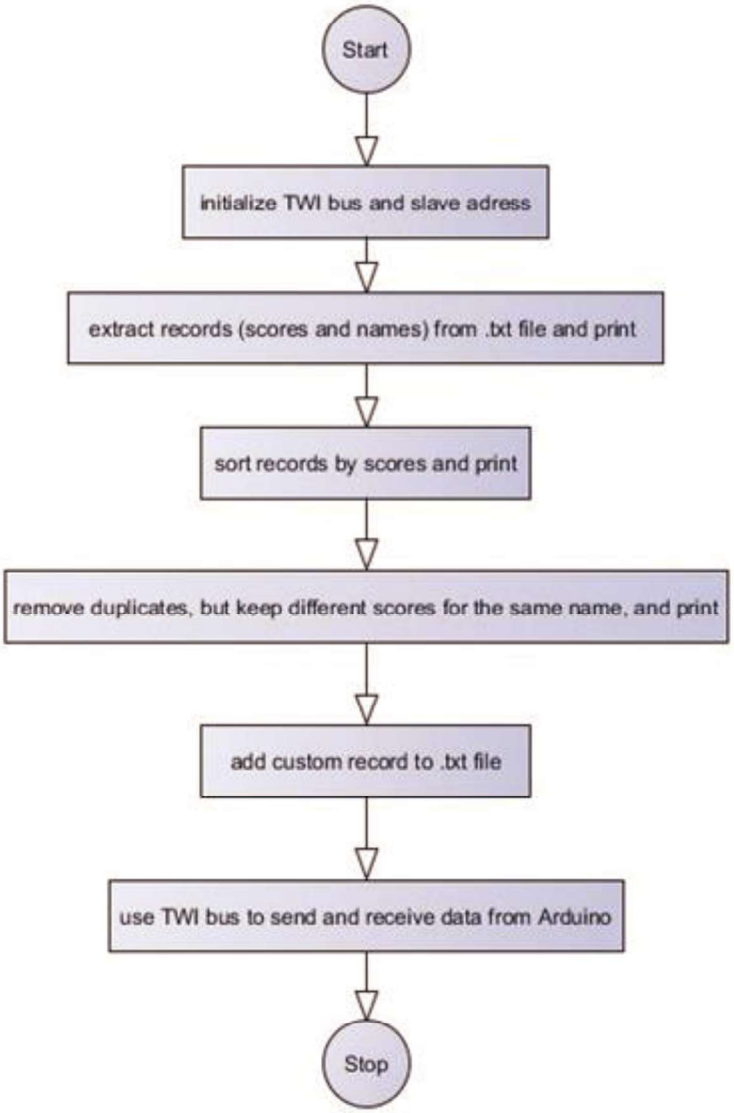
3.1 Scenarios tested with the game simulated on the Arduino side.(Kaloyan)

The code in the Raspberry Pi was written in Python. The reason is that Python became very popular and is today used widely in the industry and therefore there are a variety of libraries available for the hardware. Applying Python on the Raspberry Pi side and applying C on the Arduino side is a very good way to compare the two languages and experience different approaches for solving similar problems, and acquiring new skills.

3.1.1 Scenario 1: Arduino plays Snake - Master RaspberryPi (RPi) and Slave Arduino(Kaloyan)

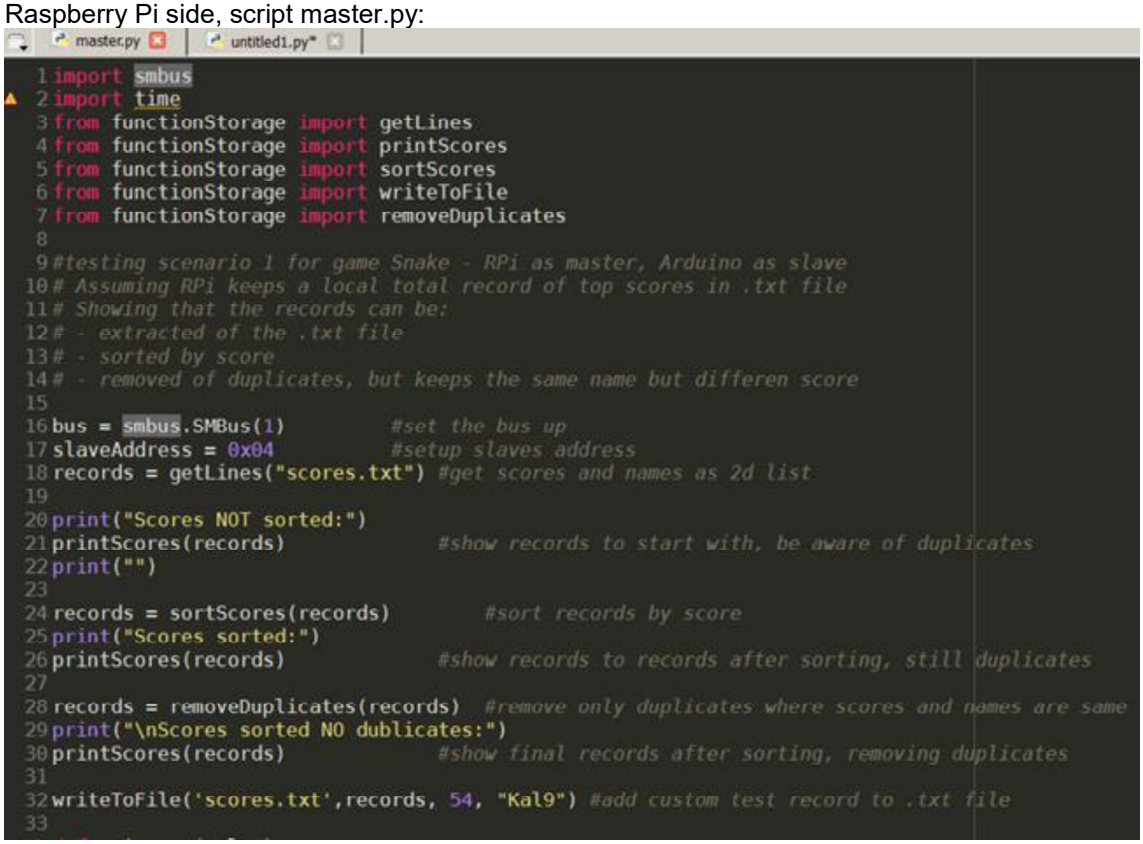
Code Arduino side: - written in C where the game is simulated by writing to and reading from SD card for 5 records and sending 5 (n)records requested from Raspberry Pi via TWI.
Code Raspberry Pi side: - written in Python for requesting 5 (n)records from Arduino via TWI, receiving records adding to local .txt file, removing duplicates and sorting by score.
Playing snake requires removal of duplicates as only different scores for the same player are kept.
Synchronization between Raspberry Pi and Arduino agreed by format of data send, number of records fixed (but can be changed).

3.1.1.1 Flow diagram for data processing testing on the Raspberry Pi side.(Kaloyan)



3.1.1.2 Code (Kaloyan)

Raspberry Pi side, script master.py:



```
master.py | untitled1.py*

1  import smbus
2  import time
3  from functionStorage import getLines
4  from functionStorage import printScores
5  from functionStorage import sortScores
6  from functionStorage import writeToFile
7  from functionStorage import removeDuplicates
8
9  #testing scenario 1 for game Snake - RPi as master, Arduino as slave
10 # Assuming RPi keeps a local total record of top scores in .txt file
11 # Showing that the records can be:
12 # - extracted of the .txt file
13 # - sorted by score
14 # - removed of duplicates, but keeps the same name but differen score
15
16 bus = smbus.SMBus(1)          #set the bus up
17 slaveAddress = 0x04           #setup slaves address
18 records = getLines("scores.txt") #get scores and names as 2d list
19
20 print("Scores NOT sorted:")
21 printScores(records)              #show records to start with, be aware of duplicates
22 print("")
23
24 records = sortScores(records)         #sort records by score
25 print("Scores sorted:")
26 printScores(records)              #show records to records after sorting, still duplicates
27
28 records = removeDuplicates(records)  #remove only duplicates where scores and names are same
29 print("\nScores sorted NO dublicates:")
30 printScores(records)              #show final records after sorting, removing duplicates
31
32 writeToFile('scores.txt',records, 54, "Kal9") #add custom test record to .txt file
33
```

In this first part of the script is shown mainly the calls to the functions doing the reading of the .txt file and operations on the records as sorting and removing duplicates. In the end a new hard-coded record is added to the .txt file. This is the first implementation where Python was tested for the required operations expected for the game Snake. Something not so obvious from the code above are Lines 3 to 7 that import the names of the used custom functions implemented in another Python script called functionStorage.py so that they can be called from here.

```
34 def writeInt(value):
35     bus.write_byte(slaveAddress, value) #use smbus library to write int to slave
36     return -1
37
38 def readInt():
39     byte = bus.read_byte(slaveAddress) #use smbus library to read int from slave
40     return byte
41
42 #initial test if Arduino responds via TWI
43 while True:
44     var = input("Enter 1 - 9: ") #get command to be send Arduino
45     if not var:
46         continue
47     writeInt(int(var))   #send a byte to Arduino
48     print("RPI sent Arduino number: ", var)
49     time.sleep(1)
50
51     var = readInt()
52     print("Arduino sent number : ", var)
53     print()
```

Here lines 35 and 39 use the TWI bus to write a byte to slave by passing the address and the value and to read a byte from the slave. Here we can for the first time see how Python is different from C, functions are defined with the keyword “def”, there is no need for return type

definition, there is no need for parameter type, there is no need for ";" in the end of each statement. The block scope is not defined by "{ }" but instead a 4 spaces indentation is used to define the scope, this can create some hard to find errors in more complex algorithms, therefor function creation followed the single responsibility principle(*K3) to make one function take care of only one task, which also improved reusability. Lines 43 and 45 shows that Pythons conditionals does not require "( )" around the expression but ":" in the end of it. This was the first touch of the Python syntax.
The code was written to be self-explanatory with carefully chosen variable and function names, but a lot of comments were still added. Singe line comments in Python use "#" in front of the comment and multiple line comment used as shown below. ''' before and ''' after the commented lines.

```
'''
#initial test if Arduino responds via TWI
while True:
    var = input("Enter 1 - 9: ") #get command to be send Arduino
    if not var:
        continue
    writeInt(int(var))   #send a byte to Arduino
    print("RPI sent Arduino number: ", var)
    time.sleep(1)

    var = readInt()
    print("Arduino sent number : ", var)
    print()
'''
```

The code below shows the functionStorge.py script, where functions were hidden from the user in master.py

```
#bubble sort function, gets 2d list,
#where comulm 0 is a string representing the score
#and   column 1 is a string representing the name
def sortScores(listToSort):
    count = len(listToSort) #get number of row in the 2d list
    if count > 1:    #if more than one row, it might need sorting
        i = 0
        while (i < count-1):
            if int(listToSort[i][0]) < int(listToSort[i+1][0]):
                tempScore = listToSort[i][0]
                listToSort[i][0] = listToSort[i+1][0]
                listToSort[i+1][0] = tempScore

                tempName = listToSort[i][1]
                listToSort[i][1] = listToSort[i+1][1]
                listToSort[i+1][1] = tempName

            if i > 0:
                j = i
                while j > 0:
                    if int(listToSort[j-1][0]) < int(listToSort[j][0]) :
                        tempScore = listToSort[j-1][0]
                        listToSort[j-1][0] = listToSort[j][0]
                        listToSort[j][0] = tempScore

                        tempName = listToSort[j-1][1]
                        listToSort[j-1][1] = listToSort[j][1]
                        listToSort[j][1] = tempName
                    j -= 1
            i += 1
    return listToSort

```

The logic for the bubble sort function above was adopted from my assignment in week 4, 2017 in the course Hardware-Oriented Programming in C.

```
33 # prerequest: 2d list should be sorted
34 def removeDuplicates(sortedList):
35     count = int(len(sortedList))
36     countToRemove = 0
37     if count > 1:
38         i = 0
39         while (i < count-1):
40             currentScore = sortedList[i][0]
41             nextScore = sortedList[i+1][0]
42             currentName = sortedList[i][1]
43             nextName = sortedList[i+1][1]
44 
45             if currentScore == nextScore and currentName == nextName:
46                 sortedList[i][0] = "" #mark to be removed by setting empthy string
47                 sortedList[i][1] = ""
48                 countToRemove += 1
49             i += 1
50 
51     #create the new list with less rows
52     countNew = count - countToRemove
53     newList = [["0","none"] for _ in range(countNew)]
54     if countNew > 0:
55         iNew = 0
56         i = 0
57         while (i < count):
58             score = sortedList[i][0]
59             if score: #if score string is not empty add record to new list
60                 newList[iNew][0] = sortedList[i][0]
61                 newList[iNew][1] = sortedList[i][1]
62                 iNew += 1
63             i += 1
64 
65     return newList
```

The above function for removing duplicates uses a while loop to find only adjacent duplicates and write empty string instead to indicate that they need to be skipped in the second part of the function where a new list is created only for the required amount of records. At the moment of writing this description, the author can see that this function can be divided into two functions one to mark the duplicates and one to extract only the needed records. The logic for marking the duplicates is only valid if the list is already sorted. An interesting fact here is that Python does not provide the pre- or post -increment/-decrement operators which are available in C out of the box. One can use for example the addition assignment operator "+=" in Python. Line 53 above shows one of the ways to define and initialize 2D list in Python, it is basically a shorthand way to run a for loop, the function range(number) creates a list of numbers from 0 to number-1 which is then iterated over to create the elements in the list.

The function below getLines(source) reads a .txt file and extracts the content as 2D list where each record in the list is a list of 2 strings the score and the name of the player. The file is open for reading with the key 'r' and after the file is processed it is closed, very similar to C. In line 84 as "strings" is a list of strings the for loop here uses iterator of the list of strings to iterate over each string and therefor one can call ".isdigit()" on the string object to find out if the string consist of only digits if yes then it is set on the score place if no it is set on the name place, new line is read from the file in the end of each iteration over the strings in the line.

On line 98 the function "writeToFile" is defined where the file is open for appending by using the key 'a' and a line of record is written to the file in formatted form in line 100, and the file is closed in the end.

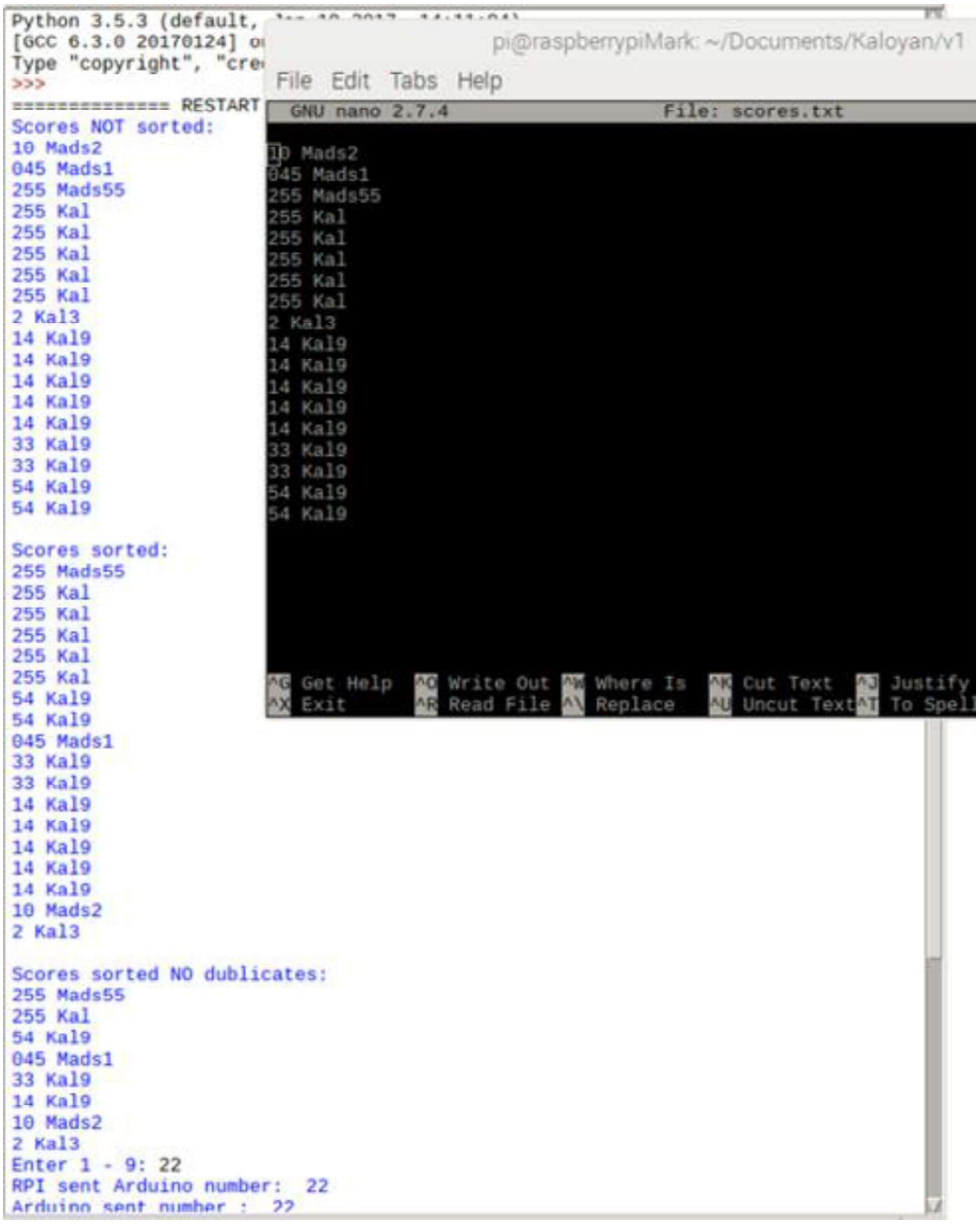
```python
def getLines(source):
    #get number of records, assumming not empty
    linesCount = sum(1 for line in open(str(source)))

    readFile = open(str(source),"r") #open file for reading

    r = 0
    lines = [["0","none"] for _ in range(int(linesCount))]

    #first line
    line = readFile.readline()
    count = 0
    while count < linesCount:
        #get rid of whitespace
        line.rstrip()
        #gets list of strings originally separated by " "
        strings = line.split(" ")
        for string in strings:
            if string.isdigit(): #be sure sting is made of digits
                lines[r][0] = string.strip()
            elif string:
                lines[r][1] = string.strip()

        r += 1; #get the index for the new row
        line = readFile.readline() #get new line
        count += 1 #keep rolling
    readFile.close()
    return lines

#appends one record to file,
# ignore the "lst" argument it is from TODO
def writeToFile(fileName, lst, score, name):
    appendFile = open(str(fileName),'a')
    appendFile.write("{} {}\n".format(score, name))
    appendFile.close()

def printScores(records):
    for row in records:
        print("{0} {1}".format(row[0],row[1]))
```

Arduino side slaveArduino.ino file: This is just setting up to test if bytes can be send from RaspberryPi to Arduino and back. receiveData(int) and sendData() will be called on events Wire.onReceive and .onRequest.

slaveArduino

```
#include <Wire.h>
#define SLVAE_ADDRESS 0x04
void receiveData(int byteCount);
void sendData();
int number = 0;
void setup(){
    Serial.begin(9600);
    Wire.begin(SLVAE_ADDRESS);
    //attach function to be called when data need to be received
    Wire.onReceive(receiveData);
    //attach function to be called when data need to be send
    Wire.onRequest(sendData);
}
void loop(){
    delay(100);
}
void receiveData(int byteCount){
    number = Wire.read();
    Serial.print("data received: ");
    Serial.print(number);
}
void sendData(){
    Wire.write(number);
    Serial.print("sendData() was called \n");
}
```

3.1.1.3 Testing scenario 1 (Kaloyan)



In the first part of the output one can see that the data was successfully extracted from the .txt file as it maps to the content of the file, therefor reading from file was successful. Next block of output shows that the sorting is also working. The last block of output shows the result after the duplicates were removed, this is also working correctly, one can see that Kal9 appears 3 times but with different scores, which is what is required for the game of Snake. In the end of the output one can also see that transmitting a byte to the Arduino and back was also successful.

3.1.1.4 Extending Scenario 1 (Kaloyan)

The incremental development strategy used, allows extending the scenario to that, the master RPi can now request more than just a number but all 5 or more records (agreed in advance). On the Arduino side the records were extracted from a .txt file on a SD card to a 2D array of strings in the setup section and this array is afterwards used to send the data over the bus on request of the RPi, more about that later.

The only change in the RPi side of the code was in master.py:

```
45 numberOfRecords = 5
46 records = [["0","none"] for _ in range(int(numberOfRecords))] #empty list
47 while True:
48     for i in range(0,5): # 5 records
49         score = ""
50         name = ""
51         for n in range(0,9): # 9 chars
52             if n < 3:
53                 writeByte(n)
54                 time.sleep(0.1)
55                 score = score + str( chr(readByte()) )
56             elif n == 3:
57                 continue
58             elif n > 3:
59                 writeByte(n)
60                 time.sleep(0.1)
61                 name = name + str( chr(readByte()) )
62         records[i][0] = score
63         records[i][1] = name.rstrip()
64         print("Recieved records from Arduino: {}".format(i+1))
65     print("Arduino send 5 records:")
66     printScores(records)
67     break
```

The while loop was modified to request 5 records via a for loop starting on line 48, followed by a for loop to request the individual characters. This for loop sends the index of the required char to the Arduino(line 53), give time (0.1 seconds worked okay) for the Arduino to get the index and write it in a global variable on its side, so that the next request on line 55 is for the char added to the score or name string on basis of the previously sent index. Nothing fancy here, as strings are objects in Python and C++ and they can use "+" operator to concatenate strings lines 55 and 61, and in C we use "char *strcat(char *dest, const char *src)", as string in C is a pointer to a string of chars array followed by '\0' end of string character.

The "{1} some text {2}".**format**(parm1,parm2) function line 64, is part of the string class and can format the string called on by exchanging every occurrence of "{}" or "{1} some text {2}" with the arguments passed to it, it looks similar to the "printf" function in C, with that difference that there is no need to specify the type of the parameters. In the end line 67 the while loop is "break"(as in C) to simulate reading only 5 records .

Modifications in slaveArduino.ino: To map what is happening on the RPi side the C code in Arduino need to provide logic corresponding to the requesting one. To start with the records, it needs to be extracted from a .txt file on a SD card in Arduino and saved as a 2D char array during setup as shown below between lines 31 and 56. In the end of the setup function the Wire library is used to initialize the TWI module slave address, so that the TWI module can send an ACK if a master calls it on the bus, and be able to respond to requests. Line 60 gets a char pointer to the first record to be send over the bus.

slaveArduino

```
1  #include <Wire.h>
2  #include <SD.h>
3  //SD card
4  #define NUMBER_RECORDS 5
5  #define SD_CARD_PIN 53
6  #define SLAVE_ADDRESS 0x04
7  #define NUMBER_OF_BYTES 17
8
9  File scoreRecordsFile;
10 int numberOfRecords = 0;
11 volatile char * record;
12 char str[NUMBER_OF_BYTES+1] = {'\0'};
13 char arr2[NUMBER_RECORDS][10]={'\0'};
14
15 volatile int index = 0;
16 volatile int recordIndex = 0;
17
18 void receiveData(int byteCount);
19 void sendData();
20
21 void setup(){
22     Serial.begin(9600);
23     //start of records extraction
24     if(!SD.begin(SD_CARD_PIN)){
25       Serial.println("SD card init failed!");
26       while(true) //stay here forever
27         ;
28     }else{
29       Serial.println("SD card init OK!");
30     }
31     //open file for reading
32     scoreRecordsFile = SD.open("SCOR.txt");
33     if(scoreRecordsFile){ //chek if it was opened
34       Serial.println("SCOR.txt opened OK!");
35       //read the file
36       int i = 0;
37       while(scoreRecordsFile.available()){
38           Serial.print("record: ");
39           String str = scoreRecordsFile.readStringUntil('\n'); //returns a String
40           int n = str.length();
41           char chars[10] = {'\0'};    //fill with end of string character
42           strcpy(chars, str.c_str()); //copy the string read from file to char array
43           if((i < NUMBER_RECORDS) && (i >= 0)){    //read only the first 5 records, its top
44             for(int k = 0; k < n; k++){
45               arr2[i][k] = chars[k];  //copy local char to global char array
46                                       //so it can be used later to send data
47             }
48             Serial.println(arr2[i]);  //print the record
49           }
50           i++; //increase the count for records
51       }
52       scoreRecordsFile.close();
53     }else{
54       Serial.println("Error opening SCOR.txt !");
55     }
56     //end of records extraction
57     Wire.begin(SLAVE_ADDRESS);
58     Wire.onReceive(receiveData);
59     Wire.onRequest(sendData);
60     record = arr2[recordIndex++]; //get pointer to the first record
61 }
62 void loop(){
63     delay(0);
64 }
```

Here in line 67 the index of the next char to be sent over the bus is read and assigned to the index variable. This variable is later used to send the requested char to RPi via the bus, this is done on line 74. Arr2 is the 2D array where the records are kept and therefore after all characters of a records are sent the recordIndex variable need to be used to get the pointer to the next record, line 78. Post-increment operator is here used to first get the current value of recordIndex to get a pointer to the next record and first after that 1 is added to recordIndex. Line 80 sets the recordIndex to 0 when all 5 records were sent.

```
65⊟void receiveData(int byteCount){
66⊟    while(Wire.available()){
67       index = (int)Wire.read(); //read the next index
68       Serial.print("index received: ");
69       Serial.println(index);
70     }
71 }//end of receiveData
72⊟void sendData(void){
73   Serial.print("sendData() was called \n");
74   Wire.write((byte)record[index]); //send current char
75   Serial.print("char was sent: ");
76   Serial.println(record[index]);
77⊟   if(index+1 > 8){ //RPi will request 8 chars only
78     record = arr2[recordIndex++]; //get pointer to new record
79⊟    if(recordIndex > 4){ //there are only 5 records to send
80       recordIndex = 0;   //start again
81     }//start again
82   }
83 }//end of sendData
```

Here is the result of the manual test:

```
14 Kal9
14 Kal9
10 Mads2
2 Kal

Scores sorted NO dublicates:
255 Mads55
255 Kal
134 Kal9
54 Kal9
045 Mads1
14 Kal9
10 Mads2
2 Kal
Recieved records from Arduino: 1
Recieved records from Arduino: 2
Recieved records from Arduino: 3
Recieved records from Arduino: 4
Recieved records from Arduino: 5
5 records received from Arduino:
034 Mads2
245 Mark
632 Peter
003 Sisco
200 Rosoo
>>>
```

```
SD card init OK!
SCOR.txt opened OK!
record: 034 Mads2
record: 245 Mark
record: 632 Peter
record: 003 Sisco
record: 200 Ros
index received: 0
sendData() was called
index received: 1
sendData() was called
index received: 2
sendData() was called
index received: 4
sendData() was called
index received: 5
sendData() was called
index received: 6
sendData() was called
index received: 7
sendData() was called
index received: 8
sendData() was called
index received: 0
sendData() was called
index received: 1
sendData() was called
index received: 2
```

On the left in blue is the RPi side output and on the right side in black is info about what is going on in the Arduino. One can see that 5 records were read from the SD card in the Arduino, and then requested by the RPi, the same records appear in the RPi console, thus the test was successful.

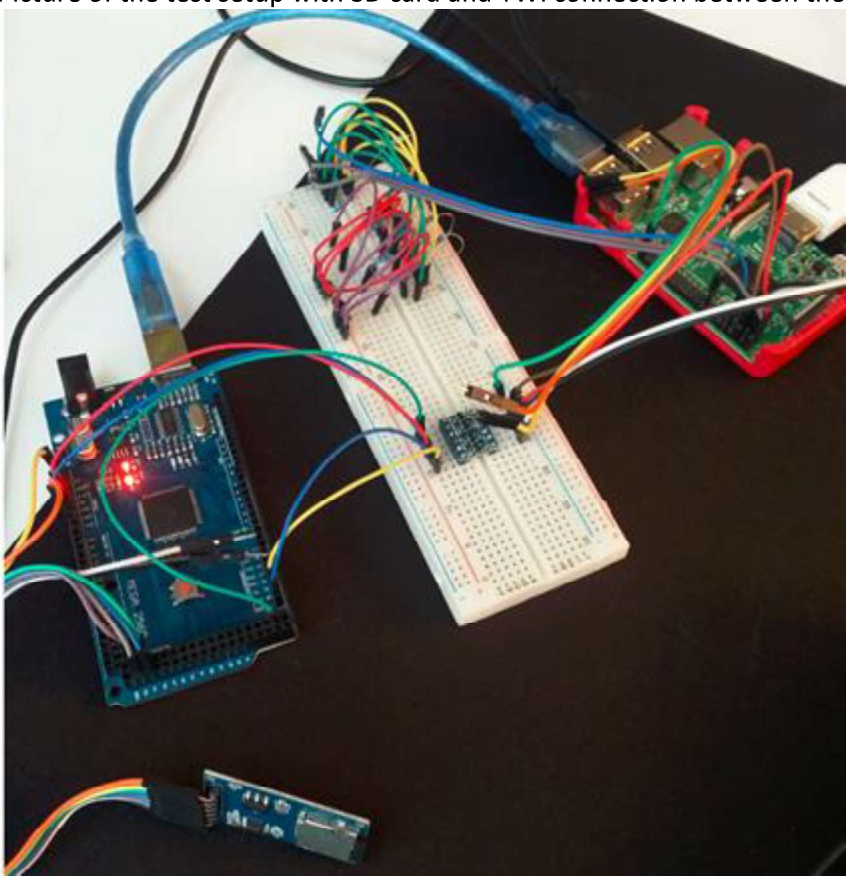
It seems as if the last records name in the RPi have some erroneous chars in the end of the name Ros, but this was just part of the test to see how Python works with end of string chars '\0' in C. A minor change in the python code corrects the problem as shown below.

It is good to mention that the python code changes background colour because at the moment of writing this document, I need to change physically to the Linux based RPi machine run the code, make the corrections and send the results to the Windows machine where the report is written.

```
numberOfRecords = 5
records = [["0","none"] for _ in range(int(numberOfRecords))]
while True:
    for i in range(0,5): # 5 records
        score = ""
        name = ""
        for n in range(0,9): # 9 chars
            if n < 3:
                writeByte(n)
                time.sleep(0.1)
                score = score + str( chr(readByte()) )
            elif n == 3:
                continue
            elif n > 3:
                writeByte(n)
                time.sleep(0.1)
                ch = chr(readByte())
                if ch == '\0':
                    ch = ' '
                name = name + str( ch )
        records[i][0] = score
        records[i][1] = name.rstrip()
        print("Recieved records from Arduino: {}".format(i+1))
    print("5 records received from Arduino: ")
    printScores(records)
    break
```

```
14 Kal9
14 Kal9
10 Mads2
2 Kal

Scores sorted NO dublicates:
255 Mads55
255 Kal
134 Kal9
54 Kal9
045 Mads1
14 Kal9
10 Mads2
2 Kal
Recieved records from Arduino: 1
Recieved records from Arduino: 2
Recieved records from Arduino: 3
Recieved records from Arduino: 4
Recieved records from Arduino: 5
5 records received from Arduino:
034 Mads2
245 Mark
632 Peter
003 Sisco
200 Ros
>>>
```

Picture of the test setup with SD card and TWI connection between the RPi and Arduino mega:



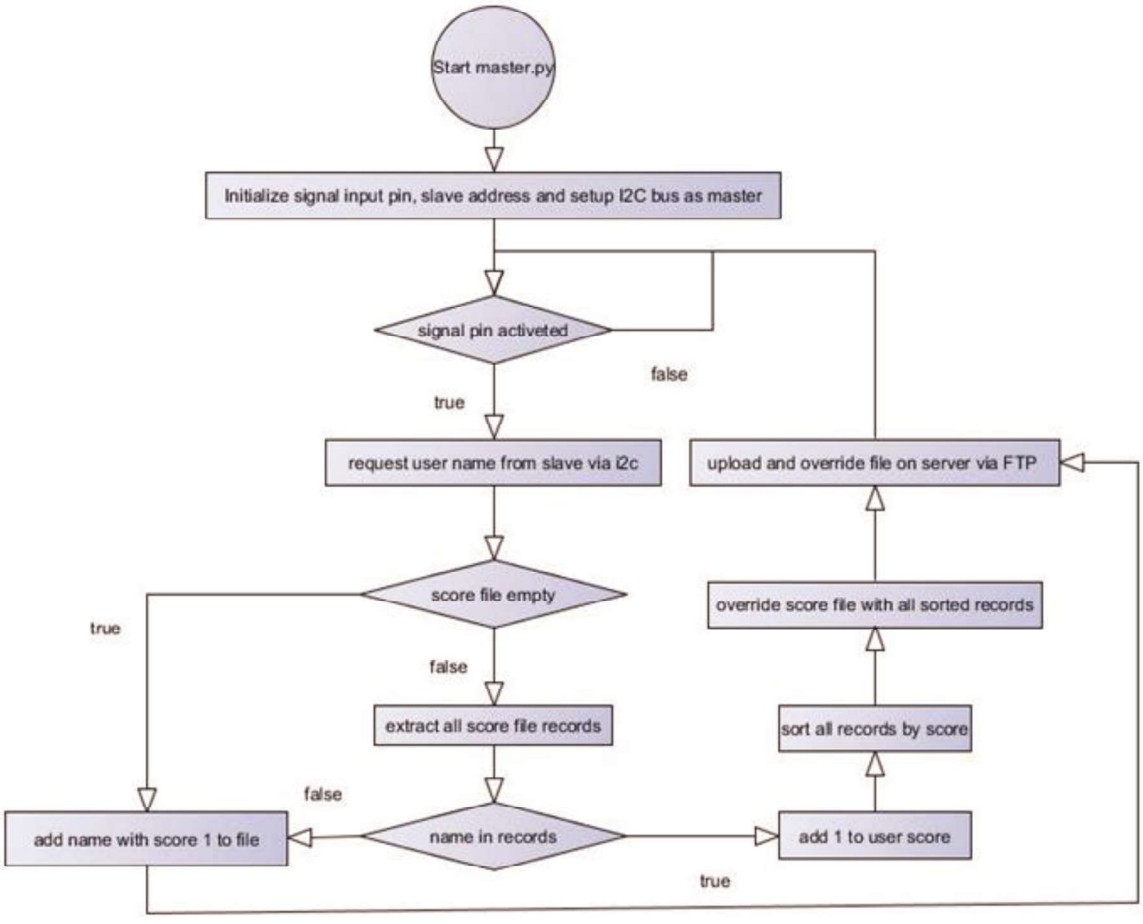
3.1.2 Scenario Final 2: Arduino Tic Tac Toe - Master RPi and Slave Arduino(signals RPi on end of game GPIOP) (Kaloyan)

In one of the first meetings of the working group, Kaloyan suggested that it will be wiser to follow the incremental development strategy principles and to start with an easier to implement game for example “Tic Tac Toe” and to keep the game of Snake for further development. The two other group members accepted this suggestion. Therefore, later in the process the focus was to implement “Tic Tac Toe”. This resulted in some changed requirements thus Scenario 1 was no longer valid. The new requirements are as follow:

Code Arduino side: - written in C where end of game simulated, signalling Rpi and the winners name sent via TWI.

Code Raspberry Pi side: - written in Python, waiting for Arduino to end a game, receiving winner name via TWI increasing score by name, sorting by score, writing to local .txt file, sending file to web server for visualization (playing tic tac toe requires increasing scores for player max score 999 fixed).

3.1.2.1 Flow diagram for data processing testing on the Raspberry Pi side(Kaloyan)



3.1.2.2 Pseudo-code(Kaloyan)

```
#if file is empty
    #append the name with score 1 to the empty file
#else
    #extract records as 2d list of score and name
    #if name in 2d list
        #add one to score of name if max score not reached else do nothing
        #if max score not reached
            #sort records by score
            #override file with updated records
    #else
        #append the name with score 1 to the file
        #(as the name is not in the file then it is last in the list)
```

3.1.2.3 Code (Kaloyan)

Smbus(system management bus) library(interface) is a derivative of I2C bus(by Philips) but developed by Intel. This library is imported and used for the I2C communication. GPIO was also imported for setting up the pin for taking signal from the Arduino in the end of each game. Empty string variable “name” was defined on line 21 for the name of the winner. The program is waiting in a while loop on line 24 for the Arduino to send an end of game signal. After signal is given, 5 consecutive characters are read from the bus and added to the string “name” variable on line 30 to form the winners name. Function addOneScore() on line 34 is called to do the logic for adding the score to the .txt file with records, more on this logic later. After a point was added to the score of the winner, on line 35 the function uploadFileToWeb() is called to upload the .txt via FTP(file transfer protocol) to the web server for visualization in www.marksauer.dk . The string variable “name” is reinitialized to empty string to be ready for the name of the winner next time.

functionStorage.py | master.py | untitled1.py*

```
1 import smbus
2 import time
3 import RPi.GPIO as GPIO
4
5 from functionStorage import addOneScore
6 from functionStorage import uploadFileToWeb
7
8 GPIO.setmode(GPIO.BCM)     #use BCM
9 GPIO.setup(21,GPIO.IN)     #use pin 21 as signal from slave to request data
10 bus = smbus.SMBus(1)       #activate the two way interface bus
11 slaveAddress = 0x04        #the address of the slave
12
13 def writeByte(value):
14     bus.write_byte(slaveAddress, value)
15     return -1
16
17 def readByte():
18     value = bus.read_byte(slaveAddress)
19     return value
20
21 name = ""
22
23 while True:
24     while not GPIO.input(21):              #check if slave signals
25         time.sleep(0.1)                    #delay
26         print("wait for slave to call")
27     print("slave to called")
28
29     for n in range(0,5):                   #request name of 5 chars
30         name = name + str( chr(readByte()) )
31         time.sleep(0.1)                    #enable delay during debug on arduino
32
33     print("Recieved name from Arduino: {}".format(name))
34     addOneScore(name, "scores_v2.txt")
35     uploadFileToWeb("scores_v2.txt",'files.000webhost.com','marksauer','V1IpWadgMOD',"public_html")
36     time.sleep(2)
37     name = ""
```

Function uploadFileToWeb is using the ftplib library to open a FTP session on line 4 below, open a .txt file as stream of bytes on line 6, and direct the bytes via the opened session to the server passing a query command STOR to save the file with the given name and location on lines 5, 7 and 8. As this function has so many tasks, and so many things can go wrong in production code, this whole function might be implemented as a separate class with the appropriate exception handling, where the main parameters injected via the constructor and the optional parameters as subdirectory path injected via public setter functions. This is the approach that will also facilitate extendibility and unit test writing. In order to keep the things compact in the visual range here all the operations are done in a single function.

functionStorage.py | master.py | untitled1.py*

```
1 import ftplib
2
3 def uploadFileToWeb(filename, server, user, password, subdir):
4     session = ftplib.FTP(str(server),str(user),str(password)) #open ftp session
5     session.cwd(str(subdir))              #define subdirectory on ftp
6     file = open(str(filename),'rb')       #open file in read binary mode
7     query = "STOR " + filename            #create override query
8     session.storbinary(query, file)       #execute query
9     file.close()
10    session.quit()
11    print("File was updated via FTP(www.marksauer.dk)\n")
12
13 def addOneScore(name, filename):
14    maxScoreReached = False
15    name = name.rstrip()
16    if isFileEmpty(filename):
17        appendRecord(name, filename)
18    else:
19        records = getRecords(filename)
20        if isNameInRecords(name, records):
21            #show that python can return more than one variable
22            records, maxScoreReached = addOneToScore(name, records)
23            if not maxScoreReached: #no need to sort and override
24                records = sortByScore(records)
25                overrideFile(records, filename)
26        else:
27            appendRecord(name, filename)
28
29 def overrideFile(records, filename):
30    file = open(str(filename),'w')
31    size = int(len(records))
32    for r in range(0,size):
33        score = records[r][0]
34        name = records[r][1]
35        file.write("{} {}\n".format(score, name))
36    file.close()
```

The function addOneScore is easy to read directly in code, as the variables and functions names are chosen carefully to reflect the purpose. In line 14, by default the max score variable is set to false indicating that the max score is not reached. The passed string variable “name” is cleaned from leading and ending space chars on line 15, by calling function .rstrip(), part of the string class library in Python. Function isFileEmpty on line 16 is checking if file is empty by given file name, assuming the file is in the location of the Python script and existing. And if the file is empty the function appendRecord is called to add the name of the winner with score 1 to the .txt file, and this is the end for this scenario. If the file is not empty the records in the file are extracted in a 2D list of strings, by calling the function getRecords on line 19. The Python documentation tell us that the List class is implemented with arrays. The important for us here is that it is an object that behaves as an array but can be extended in size. What is more important for our scenario is that if new instance of the object is created in a function (in another scope), thus we need to return the reference to it. This is because an object is passed as copy of its reference in Python (it’s not the same reference but is pointing to the same object instance in the heap), meaning that if we pass a reference to a function and do not return that reference, we can successfully change the list elements. But if we give a new instance to that copy of the original reference we essentially don’t change where the original reference is pointing to in the outer scope. So, the solution is to return the list, when a new instance is created in a function. A more expressive approach is to return even though no new instance is created for better readability. Because of that the function sortByScore on line 24 and shown below, returns the list even though this is not needed, it is not needed because operations involved in the bubble sort do not create a new instance of the list.

```
38 def sortByScore(records):
39     count = len(records)
40     if count > 1:
41         i = 0
42         while (i < count-1):
43             if int(records[i][0]) < int(records[i+1][0]):
44                 tempScore = records[i][0]
45                 records[i][0] = records[i+1][0]
46                 records[i+1][0] = tempScore
47 
48                 tempName = records[i][1]
49                 records[i][1] = records[i+1][1]
50                 records[i+1][1] = tempName
51 
52             if i > 0:
53                 j = i
54                 while j > 0:
55                     if int(records[j-1][0]) < int(records[j][0]) :
56                         tempScore = records[j-1][0]
57                         records[j-1][0] = records[j][0]
58                         records[j][0] = tempScore
59 
60                         tempName = records[j-1][1]
61                         records[j-1][1] = records[j][1]
62                         records[j][1] = tempName
63                     j -= 1
64             i += 1
65     return records
```

Line 20 calls a function isNameInRecords to check if the name of the winner is already in the list. It is not that easy to resist the temptation to just add one more point to the winner and to achieve the desired result but in the long term this would cause problems, so we keep applying the single responsibility rule here. If name is in records, we call addOneToScore function on line 22 to check if the max score is reached and add 1 to the record of the winner if not. Here we don't actually follow the single responsibility rule this is because we want to show that Python can return more than one value from a function, which is not directly possible in C or C++ (indirectly – encapsulated in struct or object). The function returns comma separated values line 88 and the variables are comma separated when assigned on line 22. Line 23 checks if the max score isn't reached and if that is true the records are sorted and the .txt file overwritten with the updated records. If the max score is reached there is no need of sorting or overwriting the .txt file and this is the end of the function. The last "else" line 26 is referred to isNameInRecords line 20 and reuses the function appendRecord on line 27 to add the new name with score 1 to the end of the .txt file as because of the game rules this is a first-time winner.

Below, line 67 is the code for the function addOneToScore, what is interesting here is how casting syntax is different than in C, on line 71 the maxScore integer value is first cased to string and then its length is got. On line 82, formatScore function is called to format the score in the defined format: "005" or "032". The function is built in a way that changing the maxScore on line 70 with a number of higher order of magnitude do not require to change the underlying implementation. And because of that the formatScore function can be said to follow the "open/closed principle"(*K2) that functions should be open for extension but closed for modifications. The leading character "0" could be also injected as an argument instead of hard coded as in line 94.

```
67 def addOneToScore(name, records):
68     maxScoreReached = False
69     count = int(len(records))
70     maxScore = 999
71     maxDigits = len(str(maxScore))
72     if count > 0:
73         i = 0
74         while (i < count):
75             currentName = records[i][1]
76             if currentName == name:
77                 strScore = records[i][0]
78                 #cast score to int
79                 intScore = int(strScore)
80                 if intScore != maxScore:
81                     intScore = intScore + 1
82                     records[i][0] = formatScore(str(intScore), maxDigits)
83                     break
84                 else:
85                     maxScoreReached = True #max score is reached
86                     break
87             i += 1
88     return records, maxScoreReached
89
90 def formatScore(strScore, maxDigits):
91     zerosToAdd = maxDigits - len(strScore)
92     if zerosToAdd > 0:
93         while zerosToAdd > 0:
94             strScore = "0" + strScore
95             zerosToAdd = zerosToAdd - 1
96     return strScore
```

The getRecords function is similar to the one used in Scenario 1 so no need to be discussed here.

```
98 def isNameInRecords(name, records):
99     nameInRecords = False
100    count = int(len(records))
101
102    if count > 0:
103        i = 0
104        while (i < count):
105            currentName = records[i][1]
106            if currentName.rstrip() == name.rstrip():
107                nameInRecords = True
108            i += 1
109    return nameInRecords
110
111 def getRecords(filename):
112    rowsCount = getCountRecords(filename)
113    readFile = open(str(filename),"r")
114    r = 0
115    records = [["0","none"] for _ in range(int(rowsCount))]
116    #first line
117    line = readFile.readline()
118    count = 0
119    while count < rowsCount:
120        line.rstrip() #get rid of whitespace
121        strings = line.split(" ")
122
123        for string in strings:
124            if string.isdigit():
125                records[r][0] = string.strip()
126            elif string:
127                records[r][1] = string.strip()
128
129        r += 1; #get the index for the new row
130        line = readFile.readline() #get new line
131        count += 1
132
133    readFile.close()
134    return records
```

Lines 141 and 144 define functions isFileEmpty and getCountRecords that extract one-line code and giving it a meaningful name, in this way the code is more expressive and reusable, this dramatically decreases the need of comments that need to change every time the code changes and very often are a source of confusion. It is more likely that when a function change its purpose, to change its name as well.

```
136 def appendRecord(name, filename):
137     file = open(str(filename),'a')
138     file.write("{} {}\n".format("001", name))
139     file.close()
140
141 def isFileEmpty(filename):
142     return    0 == getCountRecords(filename)
143
144 def getCountRecords(filename):
145     return sum(1 for line in open(str(filename)))
146
147 def printScores(records):
148     for row in records:
149         print("{0} {1}".format(row[0],row[1]))
```

The python code is written with the idea to follow some best programming practises, mainly the 3 main steps were followed: 1. Make it work. 2. Make it right. 3. Make it fast (*K1). At the moment of writing this document the “Make it work” was finished (this will be shown with some tests), the “Make it right” is also there for the scope of this project and the “Make it fast” hasn’t been seen as critical at the moment because it is running fast enough for its purpose.
Simulating end of game on the Arduino side with C in slaveArduino.ino:

```
76 void loop(){
77     digitalWrite(SIGNAL_PIN,HIGH);  //START signal to RPi
78     delay(2);
79     digitalWrite(SIGNAL_PIN,LOW);  //STOP signal to RPi
80     delay(2);
81 }
82 void sendData(void){
83   Serial.print("sendData() was called \n");
84   char myname[6] = "MARK ";                  //hard-coded for testing
85   Wire.write(myname[index2++]);
86   if(index2 == 5) index2 = 0;
87 }
```

The code above shows the simple test configuration with hard-coded winner name on line 84 that is send to the RPi via the TWI bus and the simulated end of game signal send to the RPi by writing the digital SIGNAL_PIN to high on line 77.

Code used in the actual game to achieve the required functionality is placed in a separate file called RPI_com (this logic is written by Mads) and imported in the file for the game.

RPI_com

```
1  void sendOneRecord(){
2    lcd.setCursor(0 , 1);
3    lcd.print("Sending data... ");
4    char i = 0;
5    digitalWrite(RPI_cp, HIGH);
6    while(digitalRead(RPI_cp)){
7      //Serial.print("This is value of done sending:");
8      //Serial.println(doneSending);
9      delay(200);
10     if (++i > 15){
11       digitalWrite(RPI_cp, LOW);
12       i=0;
13     }
14   }
15   digitalWrite(RPI_cp, LOW);
16   doneSending = false;
17 }
```

The sendOneRecord is called when a player wins the game, its called once and its purpouse is to give signal to the RPi that a player won a game. This is done by writing HIGH to the RPI_cp signal pin in line 5. The while loop on line 6 is basically waiting for the RPi to request the name of the winner by calling the event, the function sendData below. If the master RPi does not do that in 15 counts (with delay on line 9) for i in line 10 (sentOneRecord) then the new game can start as the control is returned to the game state machine.

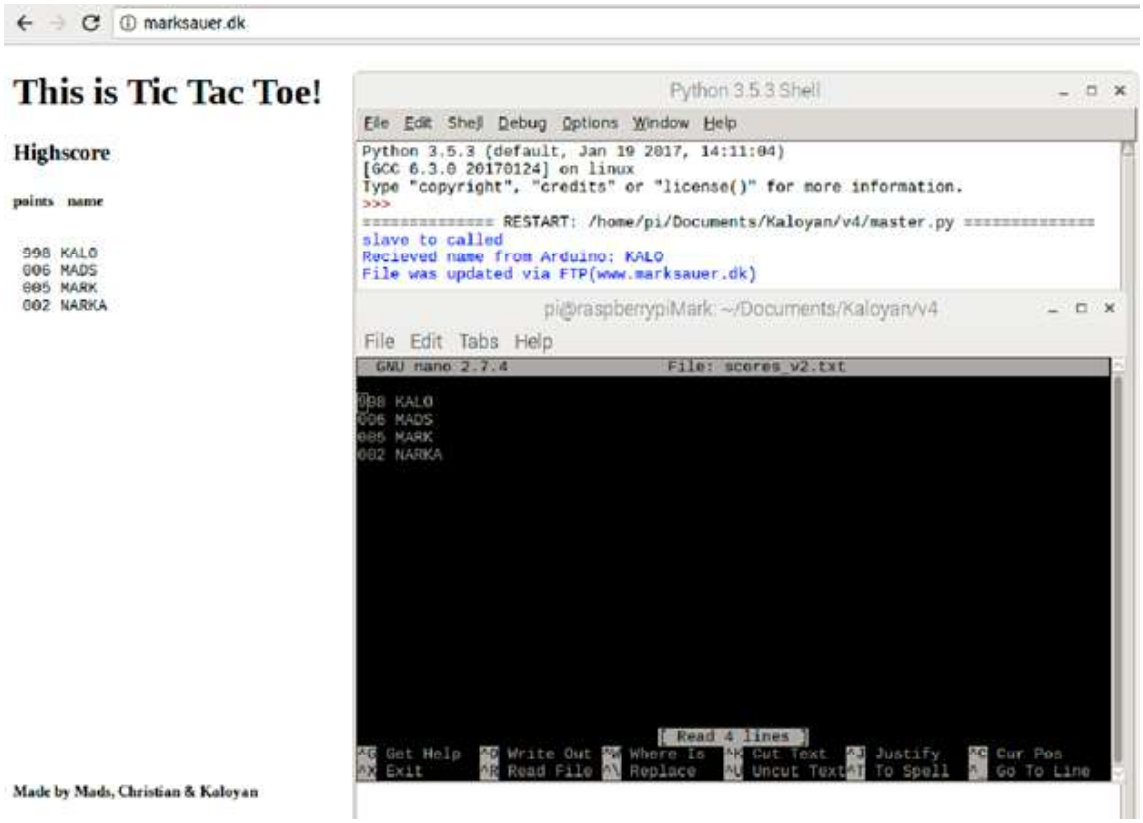
```
19 void sendData(void) {
20   static int index = 0;
21 #ifdef DEBUG_sendData
22   Serial.print("sendData() was called \n");
23 #endif
24   if (turn) {
25     Wire.write((byte)player1Name[index]);
26   } else {
27     Wire.write((byte)player2Name[index]);
28   }
29   index++;
30   //reset index and done sending
31   if (index >= NAME_LENGTH) {
32     index = 0;
33     doneSending = true;
34     digitalWrite(RPI_cp, LOW);
35   }
36 #ifdef DEBUG_sendData
37   Serial.print("char was sent: ");
38   if (turn) {
39     Serial.println(player1Name[index]);
40   } else {
41     Serial.println(player2Name[index]);
42   }//end if
43 #endif
44 }
```

sendData function above is called when the master RPI was given end of game signal from the Arduino and after that requests the name of the winner. The function is basically checking who of the two players won the game and then sending the chars of the name as a bytes, in the end of the transaction the signal to the RPI is stopped by writing it low on line 34. The index variable is static as the function is called for each char of the winners name and need to keep its value between the calls.

3.1.2.4 Hjemmeside og httpkode

Hjemmesiden er lavet i simpel http, den bruget et “embeded” tag til at importerer en formateret tekstfil(“score_v2.txt”), som udgør highscorelisten. Denne opdateres når RPI’en uploader en ny version af tekstfilen over FTP. Siden gør også brug af et “meta” tag som gør at siden opdateres med et interval på 20 sekunder.
Hjemmesiden er tilgændelig på “www.marksauer.dk”, og er en af gruppemedlemmernes egen test side, som står hos firmaet “000webhoste.com”.

3.1.2.5 Manual test results: (Kaloyan)



The start of the testing session started by testing for max score reached, where the player “KALO” was manually typed to “997” in the local .txt file, to save time, as shown above. After “KALO” won few games the score reached “999” and the followed wins of “KALO” didn’t change the top scores as expected.

The name of the winner “MARK” was sent a couple of times to simulate starting with a lower score and moving up on the list. It started with score “005” and after few more winning games passed the score of “MADS” and reached “007” before the script was stopped. This test was successful showing that sorting works as shown in the picture above from the updated web site of the top scores.

This was the testing in isolation of the Arduino with the real game.

3.1.3 Scenario 3: Arduino plays Tic Tac Toe - Master Arduino and Slave RPi (Arduino directly sends data to Rpi TWI)(Kaloyan)

Code Arduino side: - written in C where the end of the game was simulated, and the winner's name was directly sent to RPi via TWI.
Code Raspberry Pi side: - written in Python, waiting for Arduino to end the game and receive the winner's name via TWI and the name is printed for confirmation

3.1.3.1 **Code (Kaloyan)**

Arduino master, code in C:

arduinoMaster

```
#include <Wire.h>
void setup()
{
   Wire.begin(); // join the bus as master
}
void loop()
{
   char str[10];
   int x = 0;
   sprintf(str, "Mads \n", x);
   Wire.beginTransmission(0x09); //set slave address 0x09 in shift register
   Wire.write(str);              //put 10 bytes in the buffer
   Wire.endTransmission();       //send start and start transmitting
   delay(50);
}
```

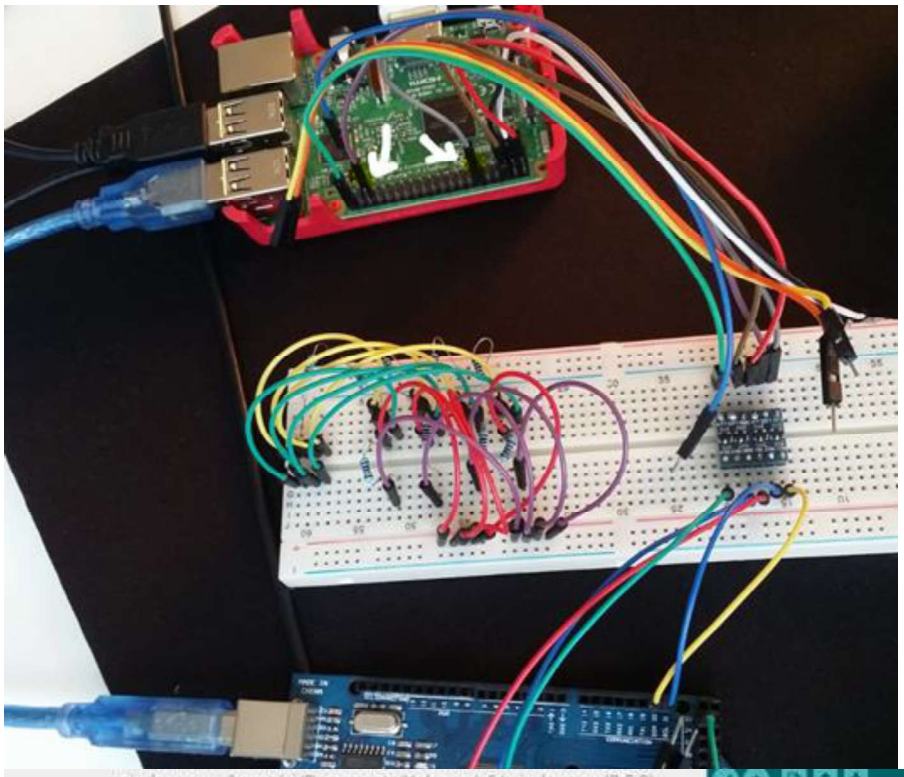
Raspberry Pi slave, code in python:

rpi_slave.py | untitled1.py*

```
import time
import pigpio

SDA=18
SCL=19

SLAVE_ADDRESS = 9

def onReceive(id, tick):
    global RPi
    s, received, data = RPi.bsc_i2c(SLAVE_ADDRESS)
    print("onReceive\n")
    if received:
        print(data[:-1])

RPi = pigpio.pi()

if not RPi.connected:
    exit()

# Add pull-ups in case external pull-ups haven't been added
RPi.set_pull_up_down(SDA, pigpio.PUD_UP)
RPi.set_pull_up_down(SCL, pigpio.PUD_UP)

# Respond to BSC slave activity
event = RPi.event_callback(pigpio.EVENT_BSC, onReceive)

RPi.bsc_i2c(SLAVE_ADDRESS) # Configure BSC as I2C slave

time.sleep(6)
event.cancel()
RPi.bsc_i2c(0) # Disable BSC peripheral
RPi.stop()
```

The code above was adopted from the guideline for working with the library and examples to fit for the current purpose. (K25: http://abyz.me.uk/rpi/pigpio/python.html#bsc_i2c)

3.1.3.2 Testing the implementation, with the dedicated for slave SDA and SCL as shown in the picture below(Kaloyan):



rpi_slave.py - /home/pi/Documents/Kaloyan/v6/rpi_slave.py (3.5.3)

File Edit Format Run Options Window Help

```
import time
import pigpio

SDA=18
SCL=19

SLAVE_ADDRESS = 9

def onReceive(id, tick):
    global RPi
    s, received, data = RPi.bsc_i2c(SLAVE_ADDRESS)
    print("onReceive\n")
    if received:
        print(data[:-1])

RPi = pigpio.pi()

if not RPi.connected:
    exit()

# Add pull-ups in case external pull-ups haven't been added
RPi.set_pull_up_down(SDA, pigpio.PUD_UP)
RPi.set_pull_up_down(SCL, pigpio.PUD_UP)

# Respond to BSC slave activity
event = RPi.event_callback(pigpio.EVENT_BSC, onReceive)

RPi.bsc_i2c(SLAVE_ADDRESS) # Configure BSC as I2C slave

time.sleep(6)
event.cancel()
RPi.bsc_i2c(0) # Disable BSC peripheral
RPi.stop()
```

arduinoMaster

```
void setup()
{
  Wire.begin(); // join the bus as master
}
void loop()
{
  char str[10];
  int x = 0;
  sprintf(str, "Mads \n", x);
  Wire.beginTransmission(0x09); //set slave address 0x09 in shift r
  Wire.write(str);              //put 10 bytes in the buffer
  Wire.endTransmission();       //send start and start transmitting
  delay(500);
}

/*
```

Python 3.5.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: /home/pi/Documents/Kaloyan/v6/rpi_slave.py ==========
onReceive

bytearray(b'Mads ')
onReceive

bytearray(b'Mads ')
onReceive

bytearray(b'Mads ')
onReceive
```

The name of the winner “Mads” was successfully sent from the Arduino master and received by the Raspberry Pi as a slave.

4. Test af produkt

4.1 Arduino delen

Test af spillet foregik på den måde, at gruppen fik folk til at afprøve spillet og fortælle noget om hvad de syntes om spillet og brugeroplevelsen.

Alle testerne sagde, at de syntes spillet virkede godt og brokkede sig ikke over hastigheden cursoren blev rykket rundt med. Gruppen har også filmet og lagt videoerne af produktet op på youtube. Disse videoer kan ses på følgende links:
Full demonstration video:
https://www.youtube.com/watch?v=zfV5iQ8ZYD8&feature=youtu.be

Python script:
https://www.youtube.com/watch?v=egbLoRT4I6c&feature=youtu.be

4.1.1 Strømforbrug

Gruppen ønskede at teste, hvor meget strøm LED matrixen (Uden at arduinoen var sat til) kunne trække til sammenligning af teoretiske udregnede værdier.[14]
Så gruppen målte med et multimeter, hvor meget strøm der blev trukket fra LED matrixen i forskellige tilstande. De tre første tilstande er målt ved normal lysstyrke og de to sidste tilstande er målt ved fuld lysstyrke, for at teste hvor meget de max kunne trække

Tilstand:	Strømforbrug:
Alle LED normal kraft	0,265A
draw animationen normal kraft	0,204A
Idel in menu no win, normal kraft	0,149A
Alle LED fuld kraft	1,585A
draw animationen fuld kraft	1,361

Hvis man sammenligner de målte tal og de teoretisk udregnede tal passer de ikke rigtigt. Det ser ud som om, at det reelle strømforbrug er det halve af den teoretiske udregnede strømforbrug

Tilstand	Teoretisk udregnet strømforbrug	Målte strømforbrug
Alle LED normal kraft	0,50A eller 0,602A*	0,265A
Alle LED fuld kraft	3,2A eller 3,84A*	1,585A

[14] Se bilag for udregning af strømforbrug på WS2812B

*Udregnet udfra to forskellige antalgelser om forskelligt angivet mA og W forbrug af LED'erne.

5. Konklusion

5.1 Løsningen på problemet

Problemet der blev stillet i problemstillingen er ud fra test af det samle produkt løst tilfredsstillende. Problemet med at forvandle en arduino mega 2560 til en spillekonsol blev løst ved at tilføje periferenheder til brugeindput og output, i form af 2 "analog sticks", en piezo buzzer, en LCD skærm og en RGB LED matrix som skærm. Konsollen blev modulopbygget med en overordnet struktur i koden der gør det nemt at lave nye spil til platformen. Kommunikations problemet er blevet løst ved at bruge I2C, med RPI'en som master og arduinoen som slave, og så tilføje en GPIO pin som arduinoen bruger til at fortælle RPI'en der skal sendes data. Databehandlingen på RPI'en er løst i python, som også uploader til en ekstern server.

5.2 Process vurdering

Se appendix A.2 Perspektivering af gruppens arbejde

Bibliography/References

Links fra Christian og Mads:

1) Specifikationer på en PS4 Pro - www.playstation.com/en-gb/explore/ps4/tech-specs/ Besøgt 17-01-2018 kl 15
2) Specifikationer på en Arduino mega 2560 - https://store.arduino.cc/usa/arduino-mega-2560-rev3 Besøgt 17-01-2017 kl 15
3) Definition af "Auto-run platformer" - http://www.mobygames.com/game-group/genre-auto-run-platformer Besøgt 17-01-2018 kl 14
4) Information om "*millis()*" komando: https://www.arduino.cc/reference/en/language/functions/time/millis/ Besøgt 17-01-2018 kl 14:00
5) Databladet kan findes på https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf
6) Billede af en WS2812B - http://forum.arduino.cc/index.php?topic=256946.0 besøgt 18-01-2018
7) Forklaring af protokollen WS2812B bruger - https://www.pololu.com/product/2547
8) https://github.com/adafruit/Adafruit-GFX-Library - Adafruit GFX Bibliotek
9) https://github.com/adafruit/Adafruit_NeoPixel - Adafruit NeoPixel Bibliotek
10) https://github.com/adafruit/Adafruit_NeoMatrix - Adafruit NeoMatrix Bibliotek
11) Billede fra google søgning der viser opsætningen af Backlight på displayet - http://www.todopic.com.ar/foros/index.php?topic=39189.0
12) http://maltarotors.com/wp-content/uploads/2016/04/piezo-buzzer.jpg Buzzer
13) Ardunio mega 2560 dokumentation - https://store.arduino.cc/usa/arduino-mega-2560-rev3 Besøgt 17-01-2018
14) Billedet taget og klippet til fra: http://3.bp.blogspot.com/-q1ACdRHilmY/V5irRui-L1I/AAAAAAAAEeA/8OzDIjqKdSk_Iak5EtDEiZ8GuF6q4S6JwCK4B/s1600/pinOutMEGA2560.PNG
15) https://www.aliexpress.com/item/1pcs-lots-WS2812B-5050-RGB-SMD-8-8-pixels-digital-flexible-dot-matrix-individually-addressable-

led/32818045969.html?spm=a2g0s.9042311.0.0.WPWhDt Produkt fra aliexpress 8x8 RGB LED Matrix

References from Kaloyan start with K followed by the number of the reference:

K1: wiki.c2.com/?MakeItWorkMakeItRightMakeItFast
K2: https://en.wikipedia.org/wiki/Open/closed_principle
K3: https://en.wikipedia.org/wiki/Single_responsibility_principle
K4: https://www.microchip.com/wwwproducts/en/ATmega2560
K5: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf
K6: https://oscarliang.com/raspberry-pi-arduino-connected-i2c/
K7: https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf
K8: https://github.com/raspberrypi/linux/blob/rpi-4.9.y/Documentation/i2c/slave-interface
K9: https://forum.43oh.com/topic/10265-implementing-an-i2c-slave-device/
K10: https://github.com/raspberrypi/linux/tree/rpi-4.9.y/Documentation/i2c
K11: https://alanbarr.github.io/RaspberryPi-GPIO/index.html
K12: http://abyz.me.uk/rpi/pigpio/python.html#bsc_i2c
K13: https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf
K14: https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf
K15: http://abyz.me.uk/rpi/pigpio/examples.html#Hardware
K16: https://raspberrypi.stackexchange.com/questions/76109/raspberry-as-an-i2c-slave
K17: https://garretlab.web.fc2.com/en/arduino/inside/avr/sfr_defs.h/_SFR_BYTE.html
K18: http://www.chrisherring.net/all/tutorial-interrupt-driven-twi-interface-for-avr-part1/
K19: https://www.blog.pythonlibrary.org/2012/07/19/python-101-downloading-a-file-with-ftplib/
K20: https://stackoverflow.com/questions/12613797/python-script-uploading-files-via-ftp
K21: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf
K22: https://pinout.xyz/pinout/pin35_gpio19#
K23: https://pinout.xyz/pinout/pin35_gpio19#
K24: http://abyz.me.uk/rpi/pigpio/
K25: http://abyz.me.uk/rpi/pigpio/python.html#bsc_i2c

Appendix A

A.1. Tidsplan

Tidsplan.
C - Christian Mark
M - Mads Astrup
K - Kaloyan Penov
A - Alle

	3-Ugers Projekt																	
	2017																	
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Projektplanlægning	C																	
- Brugt tid	C							C&M										
Analyse og design	A	A	A															
- Brugt tid	A	A	A															
Serial komm. mellem Arduino og Raspberry Pi		K	K	K														
- Brugt tid		K	K	K	K				K	K	K	K	K					
Skrive I2C protokol i C				K			K	K	K									
- Brugt tid						K	K	K										
LED Matrix	C	C																
- Brugt tid	C&M	C&M													C&M			
Spil				C&M			C&M	C&M	C&M	C&M	C&M							
- Brugt tid			C&M	C&M		M	C&M											
Controllere	C		C&M															
- Brugt tid			C															
Feedback (PZ og LCD)							C&M											
- Brugt tid							C&M	C&M										
Samling og test									A	A	A							
- Brugt tid											A	C&K	A					
Rapportskrivning						A					A		A					
- Brugt tid													A			A	A	A
Mødepligt	Al						Al							Al	Al			
Deadlines																		

Som det kan ses af tidsplanen blev planen omkring udviklingen af "Spil" ændret undervejs i projektperioden. Gruppen besluttede sig for, at lave kryds og bolle i stedet for snake og valgte at prioritere, at arbejde videre med tidsstyring og flowcontrol da det gav mere mening på daværende tidspunkt. Resultatet af dette var at der blev brugt færre dage på spiludvikling end først antaget. Derudover kan de to weekender nævnes, hvor der blev brugt lidt "interesse timer" også selvom det ikke var planlagt.

Til perspektivering og forbedring af forløbet kunne man tilføje, at det vil have gjort rapportskrivningen lettere, hvis der løbende var blevet testet, dokumenteret og skrevet teori omkring de forskellige dele mens de blev lavet.

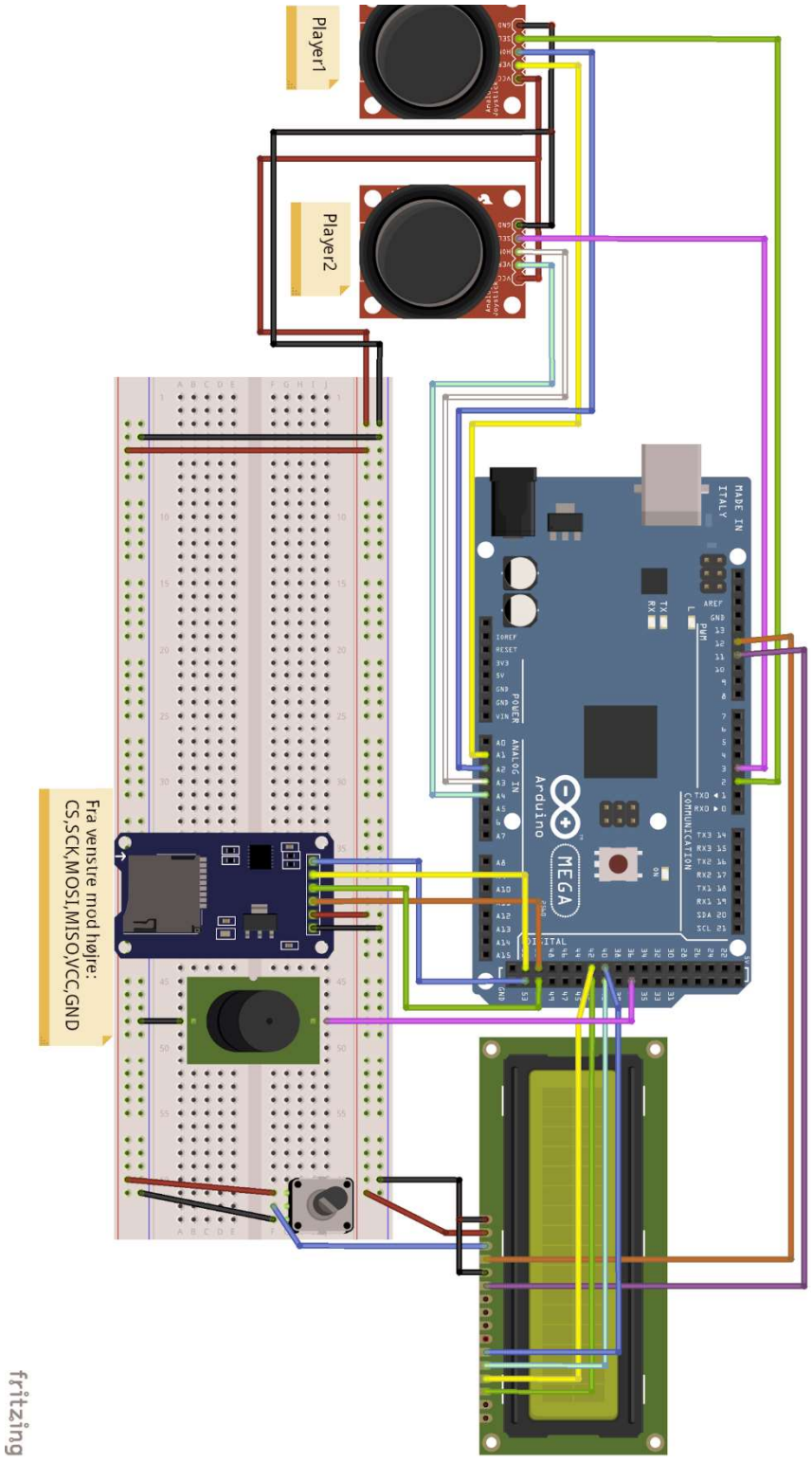
A.2. Perspektivering af gruppens arbejde

Gruppearbejdet er foregået opdelt efter hver del - Arduino og RPI delen for sig. Mads og Christian har fælles skrevet arduino kode og arbejdet med hardwaren på arduino siden. Mens Kaloyan har arbejdet på RPI delen og kommunikationen mellem RPI og arduino. Kommunikationen har været en udfordring, da grupperne undervejs i projektet ikke har været så gode til, at berette evt. ændringer eller udfordringer til deres del. Dette har delvist skyldes, at de to grupper har siddet fysisk opdelt og ikke lavet så mange "statusmøder" internt i gruppen, som der nok har været brug for.

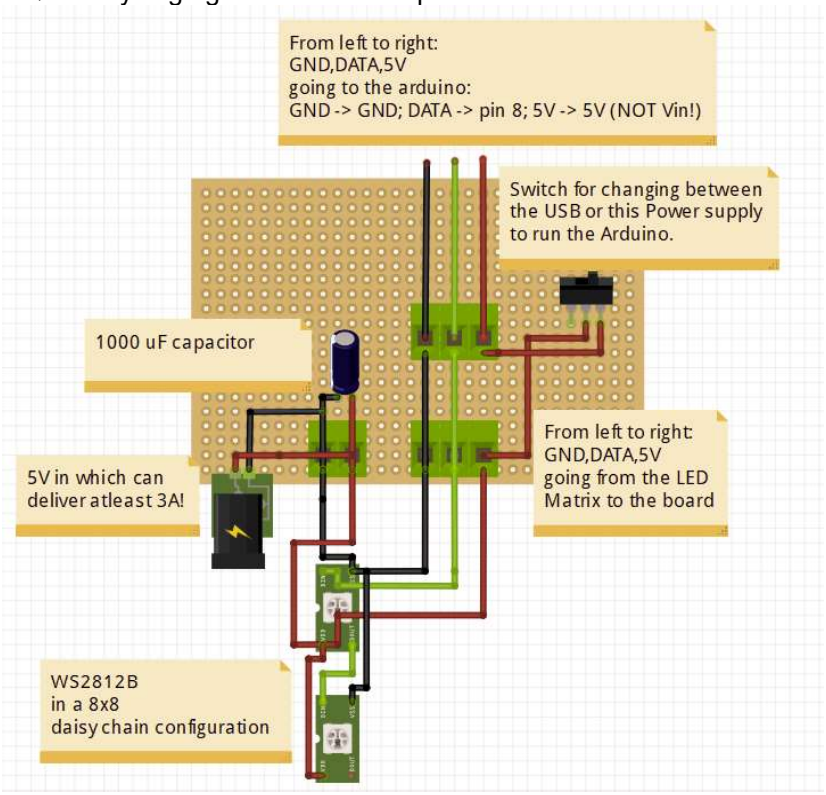
"Mads og Christian" del-gruppen har haft en lettere opdeling af Mads der har haft hovedfokus på Software-delen og Christian der har haft hovedfokus på hardware-delen. Dog har begge parter en dybdegående forståelse af Arduino delen og begge har taget del i udarbejdelse, testning og implementering af både hardware og software. Gruppen har lige fra starten af været gode til, at opdele arbejdet i "små bider" som så kunne løses uafhænigt af hinanden. Dette ses i opdelingen af programmet i mange små funktioner og de to tilstandsmaskiner der udgør programmets flow.

Appendix B - Hardware opsætning

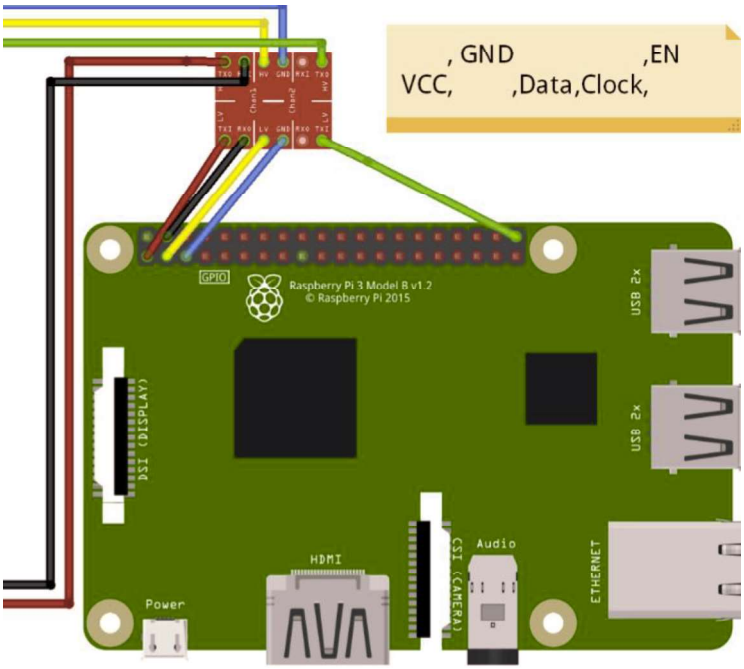
Arduino opsætningen:



Strømforsyning og LED matrix setup:



RPI og Level shifter



RPI opsætning mellem arduino

Player1
Player2
Fra venstre mod højre:
CS,SCK,MOSI,MISO,VCC,GND
Arduino
MEGA

Appendix C

Arduino koden

tic_tac_toe_v1.6.7_Final.ino filen

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>

#include <LiquidCrystal.h>

const int rs = 12, en = 11, d4 = 40, d5 = 41, d6 = 42, d7 = 43;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

//////////////PIN CONFIG////////////
#define PIN 8
#define PZ_PIN 37

#define VRX1 A1
#define VRY1 A2
#define Switch1 2

#define VRX2 A3
#define VRY2 A4
#define Switch2 3
////////////////////////////////////

////////////TONE MACROS/////////////
#define ERR_SOUND tone(PZ_PIN, 100, 75);
#define PRESS_SOUND
#define WIN_SOUND
#define DRAW_SOUND
#define PUT_SOUND tone(PZ_PIN, 2200, 100);


//bitSet(TIMSK5, TOIE5);  //sets "TOIEn" in register "TIMSKn" to
one. This bit enables timer overflow interrupts.

//    Check timer 5 registres
```

```
/*
 *   Serial.print("TCCR5A:");
 *   Serial.println(TCCR5A);
 *   Serial.print("TCCR5B:");
 *   Serial.println(TCCR5B);
 *   Serial.print("TCCR5C:");
 *   Serial.println(TCCR5C);
 *   Serial.print("OCR5AH:");
 *   Serial.println(OCR5AH);
 *   Serial.print("OCR5AL:");
 *   Serial.println(OCR5AL);
 *   Serial.print("OCR5BH:");
 *   Serial.println(OCR5BH);
 *   Serial.print("OCR5BL:");
 *   Serial.println(OCR5BL);
 */

//ISR for timer overflow:
/*
  ISR(TIMER5_OVF_vect){
    cli();
    OCR3A = sigTable[mode][i]; //PWM on pin 5 uses OCR3A as trigger
value, set it to defind value

    i+=1;
    if (i >= SAMPLESIZE - 1){
       i = 0;
     }//end if
    sei();
  }//end ISR
*/
//////////////SD CARD////////////////
#include <SD.h>
#define SD_PIN 53

//#define DEBUG_writeWinnerToSD //0.53s runtime extra

struct dataStructure {
  char points[3];
  char space = ' ';
  char nameSpace[6];
  char newlineC = '\n';
};
```

```
////////////COLOR CONFIG///////////

#define PLAYER_1_COLOR matrix.Color(0,0,255)
#define PLAYER_2_COLOR matrix.Color(0,255,0)
#define PLAYER_3_COLOR matrix.Color(150,150,150)
#define CURSOR_1_COLOR matrix.Color(255,0,255)
#define CURSOR_2_COLOR matrix.Color(255,255,0)
#define GRID_COLOR matrix.Color(150,150,150)
//////////////////////////////////
#define MENU_DELAY 250
#define GAME_DELAY 200

#define NAME_LENGTH 5
#define menuSize 2
#define SCORE_LENGTH 3
#define LINE_LENGTH (SCORE_LENGTH+NAME_LENGTH+1)
#define MAX_SCORE 999 // This can only be the number of digits
defined by SCORE_LENGTH

#define PLAYER1 false
#define PLAYER2 true

#define NM

#define SLAVE_ADDRESS 0x04

#define RPI_cp 22
//#define DEBUG_sendData
#define DEBUG_timing



///////Macros and variables/////////

#define CURSOR_POS_IN_AREA gameArea[cursorPos[1]][cursorPos[0]]

char gameArea[3][3] = {{ 0,0,0 },
                       { 0,0,0 },
                       { 0,0,0 }};
                       // 1'taller er player 1 og 0 er tom

boolean turn = PLAYER1; //FALSE eller 0 er player 1
```

```
boolean inMenu = false; // False if the game is started and 1 if the
statemachine is in the menu

char cursorPos[2] = {0,0};
char Move[2] = {0,0};
char latestPut[2] = {0,0};
char turns = 0;
unsigned long previousMillis = 0;        // will store last time LED
was updated
unsigned int interval = 0;           // interval at which to blink
(milliseconds)

volatile boolean button1_pressed_ISR = false;
volatile boolean button2_pressed_ISR = false;

boolean button1_detected = false;
boolean button2_detected = false;

enum gamestates {start, going, wonP1, wonP2, draw};
gamestates gamestate = start;


enum menustates {start_menu, running_menu, chnm_menu, game_menu};
menustates menustate = start_menu;
menustates previousMenustate = start_menu;
enum players {p1 = 0, p2 = 1};
players player;


char player1Name[NAME_LENGTH] = {'M','A','D','S',' '};
char player2Name[NAME_LENGTH] = {'M','A','R','K',' '};

volatile boolean sendingDataToRPI = false;
boolean doneSending = false;

///////Functions/////////
extern void MoveInArray();
extern bool gameAreaFiled(char xPos, char yPos);
extern bool outOfBounds(char xPos, char yPos);
extern void animate();
extern void drawPiece(char posX, char posY, int color);

//new matrix functions
extern void animateNM();
```

```
extern void drawPieceNM(char posX, char posY, int color);
extern void drawGridNM();

extern void detectinput(bool turn);
extern void detectput(bool turn);
extern boolean find_new_pos();
extern void Turndecide();
extern void detectGameCondition();
extern void p1win();
extern void p2win();
extern void drawWin();
extern void rungame();
extern void playerTurnLCD(boolean playerTurn);
extern void resetgame();
extern void inputName(bool chnmPlayer);

//SD card func
extern void serialPrintArr(char* arr, int len);
extern void writeWinnerToSD(boolean winner);

// Debug func
extern void updateD();
extern void printNamesSerial();

// Debug macros
//#define DEBUG_detectGameCondition
//#define DEBUG_Turndecide
//#define DEBUG_MoveInArray
//#define DEBUG_animate
//#define DEBUG_detectinput
//#define DEBUG_find_new_pos
//#define DEBUG_detectput
//#define DEBUG_winner

////////////////////////////////////
// MATRIX DECLARATION:
// Parameter 1 = width of NeoPixel matrix
// Parameter 2 = height of matrix
// Parameter 3 = pin number (most are valid)
// Parameter 4 = matrix layout flags, add together as needed:
//   NEO_MATRIX_TOP, NEO_MATRIX_BOTTOM, NEO_MATRIX_LEFT,
NEO_MATRIX_RIGHT:
//     Position of the FIRST LED in the matrix; pick two, e.g.
//     NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left corner.
```

```
//   NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs are arranged in
horizontal
//     rows or in vertical columns, respectively; pick one or the
other.
//   NEO_MATRIX_PROGRESSIVE, NEO_MATRIX_ZIGZAG: all rows/columns
proceed
//     in the same order, or alternate lines reverse direction; pick
one.
//   See example below for these values in action.
// Parameter 5 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812
LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811
drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel
products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA
pixels, not v2)


// Example for NeoPixel Shield.  In this application we'd like to
use it
// as a 5x8 tall matrix, with the USB port positioned at the top of
the
// Arduino.  When held that way, the first pixel is at the top
right, and
// lines are arranged in columns, progressive order.  The shield
uses
// 800 KHz (v2) pixels that expect GRB color data.
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
                                               NEO_MATRIX_TOP     +
NEO_MATRIX_LEFT +
                                               NEO_MATRIX_COLUMNS +
NEO_MATRIX_ZIGZAG,
                                               NEO_GRB            +
NEO_KHZ800);

void setup() {
  // put your setup code here, to run once:
  ////////////////////////////////////
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
```

```
  ////////////////////////////
  Serial.begin(9600);
  ////////////////////////////////////////
  while (!Serial) {};
  if (!SD.begin(SD_PIN)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  Serial.println("card initialized.");
  ////////////////////////////////////////


  matrix.begin();
  matrix.setBrightness(40);
  matrix.fillScreen(0);
  matrix.show();

  pinMode(VRX1, INPUT);
  pinMode(VRY1, INPUT);
  pinMode(Switch1, INPUT);
  digitalWrite(Switch1, INPUT_PULLUP);

attachInterrupt(digitalPinToInterrupt(Switch1),ISR_button1,FALLING);

  pinMode(VRX2, INPUT);
  pinMode(VRY2, INPUT);
  pinMode(Switch2, INPUT);
  digitalWrite(Switch2, INPUT_PULLUP);

attachInterrupt(digitalPinToInterrupt(Switch2),ISR_button2,FALLING);

  Wire.begin(SLAVE_ADDRESS);
  Wire.onRequest(sendData);
  pinMode(RPI_cp, OUTPUT);

}

void loop() {
  unsigned long currentMillis = millis();


  switch(menustate){
      // {start_menu, running_menu, chnm_menu, game_menu};
```

```
    case start_menu:
                    static char toggle_menu = 0;
                    inMenu = true; // The ISR of button1 now know
that the player1 is controlling the menu
                    //turn = 0;
                    button1_pressed_ISR = 0;
                    lcd.setCursor(0, 0);
                    lcd.print("Welcome...            ");
                    lcd.setCursor(0,1);
                    lcd.print("Play Tic Tac Toe");
                    toggle_menu = 0; // The menu is now on "Play Tic
Tac Toe"
                    menustate = running_menu; // Go to running_menu
                    break;

    case running_menu:
                      lcd.setCursor(0, 1);
                      detectinput(PLAYER1);

                      //move in menu
                      if(Move[0] > 0){

                        toggle_menu++;      //move right in menu
                        if(toggle_menu == menuSize){toggle_menu=0;};
//wrap around up

                      } else if (Move[0] < 0){

                        toggle_menu--;      //move left in menu
                        if(toggle_menu == -1){toggle_menu =
menuSize-1;}; //wrap around down

                      }//end if move left or right

                      //update menu display if changed
                      if (Move[0] != 0){
                          if(toggle_menu == 0){
                            lcd.print("Play Tic Tac Toe ");
                          }else if(toggle_menu == 1){
                            lcd.print("Change name      ");
                          }
                       } // if Moved

                       //update if the button is pressed
```

```
                        if(button1_pressed_ISR){
                           if(toggle_menu == 0){ // Go to Game
                             menustate = game_menu; // Go to the
game_menu next

                           }else if (toggle_menu == 1){ // Go to
"Change name"
                             menustate = chnm_menu;
                           }
                           button1_pressed_ISR = false;
                        }
                      if(menustate != running_menu){
                          inMenu = false; // The Menu StateMachine is
now changing to another menu than "start_menu"
                      }
                      break;

    case chnm_menu:
                      inputName(PLAYER1); // player1
                      inputName(PLAYER2); // player2
                      menustate = start_menu;
                      break;

    case game_menu:
      //{start, going, wonP1, wonP2, draw};
                      rungame();

                      break; // Break for the game state in the menu
SM
  } // Switch/Menu State Machine


  //if menustate chagnges, change delay
  if (previousMenustate != menustate){
    if( menustate ==  running_menu){
      interval = MENU_DELAY ;           //if in menu, apply menu
delay
    } else if (menustate == game_menu){
      interval = GAME_DELAY;            //if in game, apply game
delay
    }
    previousMenustate = menustate;
  }//end if - previousMenustate
```

```
  //while this runtime for the program is less than the inteval
  while(currentMillis - previousMillis <= interval) {
    currentMillis = millis();
    //Serial.println(currnetMillis);

    //Here is the place to add sender code
  }

  currentMillis = millis();             //update the current time,
if we did not enter while loop
  previousMillis = currentMillis;       //update previous time for
next compare

  #ifdef DEBUG_timing
  Serial.print("Ude, det tog:");
  Serial.println(currentMillis - previousMillis);
  #endif



} // void loop

////////////////////////////////////////////////
// ISR til knapperne
void ISR_button1(){
   static unsigned long last_interrupt_time = 0;
   unsigned long interrupt_time = millis();
   // If interrupts come faster than 200ms, assume it's a bounce and
ignore
   if (interrupt_time - last_interrupt_time > 200)
   {
      if(turn != true || inMenu){ // If it's the player 1's turn or
the Menu StateMachine is in Menu.
        button1_pressed_ISR = true;
      }
   }
   last_interrupt_time = interrupt_time;
}


void ISR_button2(){
   static unsigned long last_interrupt_time = 0;
   unsigned long interrupt_time = millis();
   // If interrupts come faster than 200ms, assume it's a bounce and
```

```
ignore
   if (interrupt_time - last_interrupt_time > 200)
   {
       if(turn){
          button2_pressed_ISR = true;
       }
   }
   last_interrupt_time = interrupt_time;
}
```

:D_funk.ino filen

```
/* Change name variables with the analog sticks. Only uses
 *  big letters and space.
 *  Call with a turn variabel, false for p1, and true for p2
 *
 */
void inputName(bool chnmPlayer){
  char temp = 0;
  char useOldName_index = 0;

  if(chnmPlayer){//Player 2
    temp = player2Name[useOldName_index];
  }else{//Player 1
    temp = player1Name[useOldName_index];
  }

  bool button_detect;

  bool old_turn;            //hold old global turn value
  old_turn = turn;        //save global turn
  turn = chnmPlayer;            //change global turn value

  chnmPlayer ? button_detect = button2_pressed_ISR : button_detect =
button1_pressed_ISR;

  lcd.clear();
  if (chnmPlayer){
    lcd.print("Player 2 name:");
  } else {
    lcd.print("Player 1 name:");
  }
```

```
  lcd.setCursor(0,1);
  lcd.cursor();
  for (char i = 0; i < (NAME_LENGTH); ++i){
    while (!button_detect){
      lcd.setCursor(i,1);
      lcd.write(temp);
      detectinput(chnmPlayer); //detect input for player 1
      delay(250);         //delay for responsive ness feeling

      //increment temp
      if (Move[0] > 0){
        temp++;
      } else if(Move[0] < 0){
        temp--;
      }//increment temp


      //made for looping char choise around
      switch (temp){
        case 91 :
            temp = 32;
            break;
        case 64 :
            temp = 32;
            break;
        case 31 :
            temp = 90;
            break;
        case 33 :
            temp = 65;
            break;
        default :
            break;
      }//end switch

      //poll for button press
      chnmPlayer ? button_detect = button2_pressed_ISR :
button_detect = button1_pressed_ISR;

    }//end while - not pressed

    //save char
    chnmPlayer ? player2Name[i] = temp : player1Name[i] = temp;
```

```
    useOldName_index++;
    if(chnmPlayer){//Player 2
      temp = player2Name[useOldName_index];
    }else{//Player 1
      temp = player1Name[useOldName_index];
    }

    //reset press ISR
    button1_pressed_ISR = false;
    button2_pressed_ISR = false;
    button1_detected = false;
    button2_detected = false;
    button_detect = false;
  }//end for - name
 lcd.noCursor();
  turn = old_turn; //restore global turn to previous value

}//end inputName



//////////////////////////////////////////// DEBUG FUNCTION
//////////////////////////////////
void printNamesSerial(){
  for (char i = 0; i < NAME_LENGTH; ++i){
    Serial.print(player1Name[i]);
  }
  Serial.println("");
  for (char i = 0; i < NAME_LENGTH; ++i){
    Serial.print(player2Name[i]);
  }
}
```

PI_funk.ino filen

```
void sendOneRecord(){

  lcd.setCursor(0 , 1);
  lcd.print("Sending data... ");
  char i = 0;
  digitalWrite(RPI_cp, HIGH);
```

```
  while(digitalRead(RPI_cp)){
    //Serial.print("This is value of done sending:");
    //Serial.println(doneSending);
    delay(200);
    if (++i > 15){
      digitalWrite(RPI_cp, LOW);
      i=0;
    }
  }
  digitalWrite(RPI_cp, LOW);
  doneSending = false;
}


void sendData(void) {

  static int index = 0;

#ifdef DEBUG_sendData
  Serial.print("sendData() was called \n");
#endif

  if (turn) {
    Wire.write((byte)player1Name[index]);
  } else {
    Wire.write((byte)player2Name[index]);
  }
  index++;

  //reset index and done sending
  if (index >= NAME_LENGTH) {
    index = 0;
    doneSending = true;
    digitalWrite(RPI_cp, LOW);
  }


#ifdef DEBUG_sendData
  Serial.print("char was sent: ");
  if (turn) {
    Serial.println(player1Name[index]);
  } else {
    Serial.println(player2Name[index]);
  }//end if
```

```
#endif
}

/*
  void receiveData(int byteCount){
    while(Wire.available()){
      index = (int)Wire.read();
    Serial.print("index received: ");
    Serial.println(index);
    }
  }
*/
```

)_func.ino filen

```
void writeWinnerToSD(boolean winner) {
  File writeFile;                 //for write file
  char score[SCORE_LENGTH];       //to contain score as string
  int finalPoints;                //to contain score as int
  bool foundName = false;         //used for searching name

  if (sendingDataToRPI) {
          //if sending data, write date to buffer file
    if (writeFile = SD.open("buffer.txt", O_RDWR)) {
#ifdef DEBUG_writeWinnerToSD
      Serial.println("Write file 'buffer.txt' opened");
#endif
    } else {
#ifdef DEBUG_writeWinnerToSD
      Serial.println("Error, could not open");
#endif
      return;
    }//end if - open SD
  } else {  //else open score file

    if (writeFile = SD.open("score.txt", O_RDWR)) {
#ifdef DEBUG_writeWinnerToSD
      Serial.println("Write file 'score.txt' opened");
#endif
    } else {
```

```
#ifdef DEBUG_writeWinnerToSD
      Serial.println("Error, could not open");
#endif
      return;
    }//end if - open SD
  }//end if - sendingDataToRPI



      //search for player 1 or 2, depending on who wins.
  if (winner) {
    foundName = writeFile.find(player2Name, NAME_LENGTH);
  } else {
    foundName = writeFile.find(player1Name, NAME_LENGTH);
  }//end if - winner find

#ifdef DEBUG_writeWinnerToSD
  Serial.print("The winner is: ");
  serialPrintArr(winner ? player2Name : player1Name, NAME_LENGTH);
#endif



      //if the name is found, 1 is added to players score
      //if not,  a new name has to be added to the list
  if (foundName) {

#ifdef DEBUG_writeWinnerToSD
    Serial.println("player found in highscore");
    Serial.print("This is pos after find: ");
    Serial.println(writeFile.position());
#endif

    //jump from end of line to begining of line to read score
    writeFile.seek(writeFile.position() - LINE_LENGTH);

#ifdef DEBUG_writeWinnerToSD
    Serial.print("This is pos after seek: ");
    Serial.println(writeFile.position());
#endif

        //read the SCORE_LENGTH char score into char arr
    for (char i = 0; i < SCORE_LENGTH; i++) {
```

```
      score[i] = writeFile.read();
    }//end for

#ifdef DEBUG_writeWinnerToSD
    Serial.print("This is pos after read: ");
    Serial.println(writeFile.position());
    serialPrintArr(score, SCORE_LENGTH);
#endif

        //jump cursor back to before read
    writeFile.seek(writeFile.position() - SCORE_LENGTH);

        //make the score into an int and add one
    finalPoints = atoi(score);
    finalPoints++;

    if(finalPoints > MAX_SCORE){ // The score can only be < MAX_SCORE
      finalPoints =  MAX_SCORE;
      Serial.println("The score has been set to MAX_SCORE");
    }

        //write finalPoints formatted to score again
    if(SCORE_LENGTH == 3){
      sprintf(score, "%03d", finalPoints);
    }else{
      Serial.println("The 'SCORE_LENGTH' MACRO has been changed from 3,
go change code in SD_FUNC");
    }


#ifdef DEBUG_writeWinnerToSD
    Serial.println(finalPoints);
    Serial.print("New score: ");
    serialPrintArr(score, SCORE_LENGTH);
#endif

    //write the score to the place in file
    writeFile.write(score);


  } else {
        //the playername was not found in highscore list
```

```
#ifdef DEBUG_writeWinnerToSD
    Serial.println("player not found in highscore");
    Serial.print("This is pos after find:");
    Serial.println(writeFile.position());
#endif
        //position is now at the end of the file, since we could not
find the player


        //now write the first point for player
    for(char i = 0; i < SCORE_LENGTH - 1; i++){
      writeFile.write("0");
    }
    writeFile.write("1 ");

    //writeFile.write("001 ");

        //now write playerName to file
    if (winner) {
      for (char i = 0; i < NAME_LENGTH; i++) {
        writeFile.write(player2Name[i]);
      }
    } else {
      for (char i = 0; i < NAME_LENGTH; i++) {
        writeFile.write(player1Name[i]);
      }
    }//end if - winner find

    writeFile.write("\n");
#ifdef DEBUG_writeWinnerToSD
    Serial.print("This is pos after write:");
    Serial.println(writeFile.position());
#endif
  }//if else - foundname

      //close the file as the file is only to be used in function
  writeFile.close();
}//end function - writeWinnerToSD



////////////////////////////////// DEBUG FUNCTION
////////////////////////////////////
```

```
void serialPrintArr(char* arr, int len) {
  for (char i = 0; i < len; i++) {
    Serial.print(arr[i]);
    Serial.print(", ");
  }
  Serial.println("");
}
```

;_tac_toe_funk.ino filen

```
void rungame(){
  switch(gamestate){
        case start:
                  resetgame(); // Restart the game variabels and
print
                  gamestate = going;
                  break;
        case going:
                //Print to the LCD
                playerTurnLCD(turn);

                //Get input from player
                detectinput(turn);

                //Move cursor in array
                MoveInArray();

                //Check for button pressed
                detectput(turn);

                //Detect if the game has been won or if its a draw
                detectGameCondition();

                // Decide if the turn should change
                Turndecide();

                // Send out the array to the LED Matrix
                #ifndef NM
                animate();
```

```
                #else
                animateNM();
                #endif

                //delay(100);
                break;
        case wonP1...wonP2:
                  delay(1000);
                  matrix.fillScreen(0);
                  matrix.show();

                  // When someone has won, the SM has already
shifted the turn to the other player with the Turndecide() func, so
we need to invert the turn.
                  playerTurnLCD(!turn);

                  if(gamestate == wonP1){
                    boolean toggle = true;

                    for(int j=0;j<3;j++){
                      if(toggle){
                        for(int i=0;i<3;i++){
                          #ifndef NM
                          drawPiece(i,j,PLAYER_1_COLOR);
                          #else
                          drawPieceNM(i,j,PLAYER_1_COLOR);
                          #endif
                          matrix.show();
                          delay(300);
                        }
                        toggle = !toggle;
                      }else{
                        for(int i=2;i>-1;i--){
                          #ifndef NM
                           drawPiece(i,j,PLAYER_1_COLOR);
                          #else
                           drawPieceNM(i,j,PLAYER_1_COLOR);
                          #endif
                           matrix.show();
                           delay(300);
                        }
                        toggle = !toggle;
                      }
                    }
```

```
                }else if(gamestate == wonP2) {
                  boolean toggle = true;

                  for(int j=0;j<3;j++){
                    if(toggle){
                      for(int i=0;i<3;i++){
                        #ifndef NM
                        drawPiece(i,j,PLAYER_2_COLOR);
                        #else
                        drawPieceNM(i,j,PLAYER_2_COLOR);
                        #endif
                        matrix.show();
                        delay(300);
                      }
                      toggle = !toggle;
                    }else{
                      for(int i=2;i>-1;i--){
                        #ifndef NM
                        drawPiece(i,j,PLAYER_2_COLOR);
                        #else
                        drawPieceNM(i,j,PLAYER_2_COLOR);
                        #endif
                        matrix.show();
                        delay(300);
                      }
                      toggle = !toggle;
                    }
                  }
                }
                // When someone has won, the SM has already
shifted the turn to the other player with the Turndecide() func, so
we need to invert the turn to get the winner.
                writeWinnerToSD(!turn);
                //send til RPI
                sendOneRecord();
                gamestate = start;
                menustate = start_menu;
                break;

            //The game has ended in a draw
            case draw:
                    delay(1000);
                    matrix.fillScreen(0);
                    matrix.show();
```

```
                    lcd.setCursor(0,1);
                    lcd.print("It's a draw!");

                    boolean toggle = true;

                    for(int j=0;j<3;j++){
                      if(toggle){
                          for(int i=0;i<3;i++){
                            #ifndef NM
                            drawPiece(i,j,PLAYER_3_COLOR);
                            #else
                            drawPieceNM(i,j,PLAYER_3_COLOR);
                            #endif
                            matrix.show();
                            delay(300);
                          }
                          toggle = !toggle;
                      }else{
                          for(int i=2;i>-1;i--){
                            #ifndef NM
                             drawPiece(i,j,PLAYER_3_COLOR);
                             #else
                             drawPieceNM(i,j,PLAYER_3_COLOR);
                             #endif
                             matrix.show();
                             delay(300);
                          }
                          toggle = !toggle;
                      }
                    }
                    gamestate = start;
                    menustate = start_menu;
                    break;
                } // Switch/ Game State Machine
}//end run-game

/*
 * Decides whos turn it is
 */
void Turndecide(){
    if(turn){// Player 2
        if(button2_detected ){
          #ifdef DEBUG_Turndecide
              Serial.println("Its now player 1's turn");
```

```
          #endif
          turn = PLAYER1;
          button2_detected = false;
        }
    }else{// Player 1
        if(button1_detected ){
          #ifdef DEBUG_Turndecide
              Serial.println("Its now player 2's turn");
          #endif
          turn = PLAYER2;
          button1_detected = false;
        }
    }
}

/*
 *  Function takes the move, sees if it is possible and if so, moves
the cursor
 */
void MoveInArray(){
  char newCursorPos[2] = {0,0}; // for checking
  newCursorPos[0] = cursorPos[0] + Move[0];
  newCursorPos[1] = cursorPos[1] + Move[1];

  if (outOfBounds(newCursorPos[0],newCursorPos[1])){
    #ifdef DEBUG_MoveInArray
        Serial.println("First bound check failed");
    #endif

    return;
  }

  cursorPos[0] = newCursorPos[0];
  cursorPos[1] = newCursorPos[1];
  return;
}

/*
 * returns true if the input position is out of the array
 */
bool outOfBounds(char xPos, char yPos){
  if (xPos > 2 || xPos < 0 || yPos > 2 || yPos < 0){
    return true;
  } else {
```

```
    return false;
  }
}

/*
 * returns true if the input position is filled already
 */
bool gameAreaFiled(char xPos, char yPos){
  if ((gameArea[yPos] [xPos]) != 0){
    return true;
  } else {
    return false;
  }
}

/* NEW LED MATRIX FUNC
 * Runs the animation funcktion, animates the board and cursor on
the new led matrix
 */
void animateNM(){
  matrix.fillScreen(0);

  drawGridNM(); //draw grid

  //run through game area and print it in players colors
  for(size_t i = 0; i < 3; i++){
    for (size_t j = 0; j < 3; j++){
      switch (gameArea[j][i]){
        case 1:
          #ifdef DEBUG_animate
              Serial.println("Player 1's tile is being printed");
          #endif
          drawPieceNM(i, j,PLAYER_1_COLOR);
          break;
        case 2:
          #ifdef DEBUG_animate
              Serial.println("Player 2's tile is being printed");
          #endif
          drawPieceNM(i, j,PLAYER_2_COLOR);
          break;
        default:
          break;
      }//end switch
    }//end for
```

```
  }//end for

  //animate cursor position
  if(gamestate == going){
      #ifdef DEBUG_animate
            Serial.println("The Cursor is being printed");
      #endif
      drawPieceNM(cursorPos[0], cursorPos[1], (turn ? CURSOR_2_COLOR
: CURSOR_1_COLOR));
  }
  matrix.show();
}

/* NEW LED MATRIX FUNC
 * Function draws a piece on the NEW LED matrix from a gameArea
coordinate
 */
void drawPieceNM(char posX, char posY, int color){
  matrix.drawPixel((posX * 3),(posY * 3),color);
  matrix.drawPixel((posX * 3)+1,(posY * 3),color);
  matrix.drawPixel((posX * 3),(posY * 3)+1,color);
  matrix.drawPixel((posX * 3)+1,(posY * 3)+1,color);
}
extern void drawGridNM(){
  for (char y = 2; y < 6; y += 3){
    for (char x = 0; x < 8; x++){
      matrix.drawPixel(x, y, GRID_COLOR);
    }
  }
  for (char x = 2; x < 6; x += 3){
    for (char y = 0; y < 8; y++){
      matrix.drawPixel(x, y, GRID_COLOR);
    }
  }

}
 /*Runs the animation funcktion, animates the board and cursor
  */
void animate(){
  matrix.fillScreen(0);
  //run through game area and print it in players colors
  for(size_t i = 0; i < 3; i++){
    for (size_t j = 0; j < 3; j++){
      switch (gameArea[j][i]){
```

```
        case 1:
          #ifdef DEBUG_animate
              Serial.println("Player 1's tile is being printed");
          #endif
          drawPiece(i, j,PLAYER_1_COLOR);
          break;
        case 2:
          #ifdef DEBUG_animate
              Serial.println("Player 2's tile is being printed");
          #endif
          drawPiece(i, j,PLAYER_2_COLOR);
          break;
        default:
          break;
      }//end switch
    }//end for
  }//end for

  //animate cursor position
  if(gamestate == going){
      #ifdef DEBUG_animate
            Serial.println("The Cursor is being printed");
      #endif
      drawPiece(cursorPos[0], cursorPos[1], (turn ? CURSOR_2_COLOR :
CURSOR_1_COLOR));
  }
  matrix.show();
}

/*Function draws a piece on the LED matrix from a gameArea
coordinate
 */
void drawPiece(char posX, char posY, int color){
  matrix.drawPixel((posX * 2),(posY*2),color);
  matrix.drawPixel((posX * 2)+1,(posY*2),color);
  matrix.drawPixel((posX * 2),(posY*2)+1,color);
  matrix.drawPixel((posX * 2)+1,(posY*2)+1,color);
}

/* DetectInput() recieves a boolean "turn" which indicades who's
turn it is. The func will change the Move[] array and return */
void detectinput(bool turn){
  int x,y = 0;
  if(turn){ // Chooses which analog stick that needs to be read from
```

```
        // Player 2
        x = analogRead(VRX2); // Read from the x-axis
        y = analogRead(VRY2); // Read from the y-axis

        #ifdef DEBUG_detectinput
            Serial.print("VRX2:");
            Serial.print(x);
            Serial.print("\tVRY2:");
            Serial.println(y);
        #endif

    }else{
        // Player 1
        x = analogRead(VRX1); // Read from the x-axis
        y = analogRead(VRY1); // Read from the y-axis

        #ifdef DEBUG_detectinput
            Serial.print("VRX1:");
            Serial.print(x);
            Serial.print("\tVRY1:");
            Serial.println(y);
        #endif
    }

    //Decides which direction the analog stick is facing on the x-axis
    if( x > 800){
        #ifdef DEBUG_detectinput
            Serial.println("The analog stick is facing left");
        #endif

        Move[0] = 1;
    }else if(x < 300){
        #ifdef DEBUG_detectinput
            Serial.println("The analog stick is facing right");
        #endif
        Move[0] = -1;
    }else{
        Move[0] = 0;
    }

      //Decides which direction the analog stick is facing on the y-
axis
    if( y > 800){
        #ifdef DEBUG_detectinput
```

```
            Serial.println("The analog stick is facing up");
        #endif
        Move[1] = 1;
    }else if(y < 300){
        #ifdef DEBUG_detectinput
            Serial.println("The analog stick is facing down");
        #endif
        Move[1] = -1;

    }else{
        Move[1] = 0;
    }
    return;
}

/*The function finds the first "legal" position in the gameArea to
place the cursor*/
boolean find_new_pos(){
        for(char i = 0;i<3;i++){
              for(char j = 0;j<3;j++){
                  if(gameAreaFiled(j, i) == 0){
                      cursorPos[0] = j;
                      cursorPos[1] = i;
                            #ifdef DEBUG_find_new_pos
                                Serial.println("A new position has
been found");
                            #endif

                      return 1;
                  }
              }
        }
        return 0; // No new position available
}

/*The function detects a button press, inserts a '1' or '2'
 *on the position of the cursor in the gameArea Array and
 *finds a new legal position for the cursor */
void detectput(bool turn){

    if(turn){
          if(button2_pressed_ISR){
                #ifdef DEBUG_detectput
                     Serial.println("Button 2 has been pressed");
```

```
            #endif

            button2_pressed_ISR = false; // These var must be
reset, because they can be toggled anytime by the ISR
            button1_pressed_ISR = false;

            if (!gameAreaFiled(cursorPos[0],cursorPos[1] )){
                PUT_SOUND
                //Serial.println("second filld check failed");
                button2_detected = true; // This var must be set,
because we need to know that the game has registered a button press
                turns++; // Increment the amount of turns
                CURSOR_POS_IN_AREA = 2; // Indsætter player 2's
træk på banen
                //Saves the cursorposition where a 1 or 2 has been
placed. This is used in the func "detectGameCondition()"
                latestPut[0] = cursorPos[0];
                latestPut[1] = cursorPos[1];
                #ifdef DEBUG_detectput    // Prints out the array
on the Serial monitor
                  updateD();
                #endif
                find_new_pos(); //Find a new "legal" position for
the cursor
                return;
            }  else {
              ERR_SOUND
            }

        } // end if
  }else{
         if(button1_pressed_ISR){
            #ifdef DEBUG_detectput
                 Serial.println("Button 1 has been pressed");
            #endif
            button1_pressed_ISR = false;// These var must be
reset, because they can be toggled anytime by the ISR
            button2_pressed_ISR = false;

            if (!gameAreaFiled(cursorPos[0],cursorPos[1] )){
              PUT_SOUND
              button1_detected = true;  //This var must be set,
because we need to know that the game has registered a button press
              turns++; // Increment the amount of turns
```

```
                CURSOR_POS_IN_AREA = 1; // Indsætter player 1's træk
på banen
                //Saves the cursorposition where a 1 or 2 has been
placed. This is used in the func "detectGameCondition()"
                latestPut[0] = cursorPos[0];
                latestPut[1] = cursorPos[1];
                #ifdef DEBUG_detectput
                  updateD();
                #endif
                find_new_pos(); //Find a new "legal" position for
the cursor
              } else {
                ERR_SOUND
              }// end if gameAreaField()
          } // end if
  } // end else
} // end func

/*
 * Make life great again
 * Detect the winner/draw and change game state.
 */
void detectGameCondition(){
  char xCheck = 0;
  char yCheck = 0;
  if (button1_detected || button2_detected){

    #ifdef DEBUG_detectGameCondition
        Serial.println("Check win begun");
    #endif
////////////////// Check column win /////////////////
    //first make x coordinate to check
    xCheck = latestPut[0] + 1;
    if (xCheck > 2){
      xCheck = 0;
    }//end if - xCheck overflow

    #ifdef DEBUG_detectGameCondition
        Serial.print("First X coordinate generatred:\t");
        Serial.println((int)xCheck);
    #endif

    if (gameArea[latestPut[1]][xCheck] ==
gameArea[latestPut[1]][latestPut[0]]){
```

```
      #ifdef DEBUG_detectGameCondition
         Serial.println("First test coordnate matches latest put");
      #endif

      //second make x coordinate to check
      xCheck++;
      if (xCheck > 2){
        xCheck = 0;
      }//end if - xCheck overflow

      #ifdef DEBUG_detectGameCondition
          Serial.print("Second X coordinate generatred:\t");
          Serial.println((int)xCheck);
      #endif

      if (gameArea[latestPut[1]][xCheck] ==
gameArea[latestPut[1]][latestPut[0]]){

        #ifdef DEBUG_detectGameCondition
           Serial.println("Someone won");
        #endif
        if (gameArea[latestPut[1]][latestPut[0]] == 1){
          p1win();
          return;

        } else if (gameArea[latestPut[1]][latestPut[0]] == 2){
          p2win();
          return;
        }//end if - change gamestate
      }//end if - second check "win check"
    }//end if - first check

////////////////// Check row win //////////////////
    //first make y coordinate to check
    yCheck = latestPut[1] + 1;
    if (yCheck > 2){
      yCheck = 0;
    }//end if - yCheck overflow

    #ifdef DEBUG_detectGameCondition
        Serial.print("First Y coordinate generatred:\t");
        Serial.println((int)yCheck);
    #endif
```

```
    if (gameArea[yCheck][latestPut[0]] ==
gameArea[latestPut[1]][latestPut[0]]){

      #ifdef DEBUG_detectGameCondition
          Serial.println("First test coordnate matches latest put");
      #endif

      //second make x coordinate to check
      yCheck++;
      if (yCheck > 2){
        yCheck = 0;
      }//end if - yCheck overflow

      #ifdef DEBUG_detectGameCondition
          Serial.print("Second Y coordinate generatred:\t");
          Serial.println((int)yCheck);
      #endif

      if (gameArea[yCheck][latestPut[0]] ==
gameArea[latestPut[1]][latestPut[0]]){

        #ifdef DEBUG_detectGameCondition
           Serial.println("Someone won");
        #endif

        if (gameArea[latestPut[1]][latestPut[0]] == 1){
          p1win();
          return;

        } else if (gameArea[latestPut[1]][latestPut[0]] == 2){
          p2win();
          return;
        }//end if - change gamestate
      }//end if - second check "win check"
    }//end if - first check

////////////////// Check diagonal win //////////////////
    if ((latestPut[0] == latestPut[1]) || ((latestPut[0] -
latestPut[1]) % 2 == 0)){
      //check if latest put is inside diagonals
      //might be more work to check if i should check, please
reasearch that! -Mads
```

```
      if ((gameArea[0][0] == gameArea[1][1] && gameArea[1][1] ==
gameArea[2][2]) || (gameArea[0][2] == gameArea[2][0] &&
gameArea[1][1] == gameArea[2][0])){

        #ifdef DEBUG_detectGameCondition
            Serial.println("Diagonal win detected");
        #endif
        //winner clearly has the middel
        if (gameArea[1][1] == 1){
          p1win();
          return;

        } else if (gameArea[1][1] == 2){
          p2win();
          return;
        }//end if - change gamestate
      }//end if - diagonal win
    }//end if - last put on diagonal
      if(turns == 9){
        drawWin();
        return;
      }
  }//end if buttons detected
}//end function detect win state



/* Printning out the Array on the Serial monitor (DEBUG)*/
void updateD(){
  for(size_t i = 0; i < 3; i++){
    for (size_t j = 0; j < 3; j++){
      Serial.print((int)gameArea[i][j]);
    }
    Serial.println();
  }Serial.println();

}//end updateD

void p1win(){
  gamestate = wonP1;
  #ifdef DEBUG_winner
       Serial.println("Player 1 won");
  #endif
```

```
}//end p1win

void p2win(){
  gamestate = wonP2;
  #ifdef DEBUG_winner
       Serial.println("Player 2 won");
  #endif
}//end p2win

void drawWin(){
  gamestate = draw;
  #ifdef DEBUG_winner
         Serial.println("draw detected, 9 moves used");
  #endif

}//end drawWin


void playerTurnLCD(boolean playerTurn){

   if(playerTurn){ // Player 2
      if(gamestate == going){
          lcd.setCursor(0,1);
          for(char i = 0;i<NAME_LENGTH;i++){
              lcd.print(player2Name[i]);
          }
      }else if(gamestate == wonP2){
          lcd.setCursor(0,1);
          for(char i = 0;i<NAME_LENGTH;i++){
              lcd.print(player2Name[i]);
          }
          lcd.print(" wins");
      }
   }else{ // Player 1
      if(gamestate == going){ // Prints the player name for whome
has the turn
          lcd.setCursor(0,1);
          for(char i = 0; i < NAME_LENGTH;i++){
              lcd.print(player1Name[i]);
          }
      }else if(gamestate == wonP1){ // Prints the player name for
who won.
          lcd.setCursor(0,1);
          for(char i = 0;i<NAME_LENGTH;i++){
```

```
                lcd.print(player1Name[i]);
            }
            lcd.print(" wins");
        }
    }
    lcd.noCursor(); // Remove the cursor
}

void resetgame(){

                    button1_pressed_ISR = false;
                    button2_pressed_ISR = false;
                    button1_detected = false;
                    button2_detected = false;
                    cursorPos[0] = 0;
                    cursorPos[1] = 0;
                    latestPut[0]= 0;
                    latestPut[1]= 0;
                    turns = 0;

                    //Write to LCD
                    lcd.clear();
                    lcd.setCursor(0, 0);
                    lcd.print("Tic Tac Toe");
                    //Reset the gameArea and the LED Maxtrix
                    for(int i = 0;i<3;i++){
                        for(int j = 0;j<3;j++){
                          gameArea[i][j] = 0;
                        }
                    }
                    matrix.fillScreen(0);
                    matrix.show();
}
```

Appendix D (Kaloyan, code)

Scenario 1: Arduino plays Snake - Master RaspberryPi (RPi) and Slave Arduino

master.py

master.py | untitled1.py*

```
import smbus
import time
from functionStorage import getLines
from functionStorage import printScores
from functionStorage import sortScores
from functionStorage import writeToFile
from functionStorage import removeDuplicates

#testing scenario 1 for game Snake - RPi as master, Arduino as slave
# Assuming RPi keeps a local total record of top scores in .txt file
# Showing that the records can be:
#  - extracted of the .txt file
#  - sorted by score
#  - removed of duplicates, but keeps the same name but differen score

bus = smbus.SMBus(1)          #set the bus up
slaveAddress = 0x04           #setup slaves address
records = getLines("scores.txt") #get scores and names as 2d list

print("Scores NOT sorted:")
printScores(records)              #show records to start with, be aware of duplicates
print("")

records = sortScores(records)         #sort records by score
print("Scores sorted:")
printScores(records)              #show records to records after sorting, still duplicates

records = removeDuplicates(records)  #remove only duplicates where scores and names are same
print("\nScores sorted NO dublicates:")
printScores(records)              #show final records after sorting, removing duplicates

writeToFile('scores.txt',records, 54, "Kal9") #add custom test record to .txt file

def writeInt(value):
    bus.write_byte(slaveAddress, value) #use smbus library to write int to slave
    return -1

def readInt():
    byte = bus.read_byte(slaveAddress) #use smbus library to read int from slave
    return byte

#initial test if Arduino responds via TWI
while True:
    var = input("Enter 1 - 9: ") #get command to be send Arduino
    if not var:
        continue
    writeInt(int(var))   #send a byte to Arduino
    print("RPI sent Arduino number: ", var)
    time.sleep(1)

    var = readInt()
    print("Arduino sent number : ", var)
    print()
```

```
'''
#initial test if Arduino responds via TWI
while True:
    var = input("Enter 1 - 9: ") #get command to be send Arduino
    if not var:
        continue
    writeInt(int(var))   #send a byte to Arduino
    print("RPI sent Arduino number: ", var)
    time.sleep(1)

    var = readInt()
    print("Arduino sent number : ", var)
    print()
'''
```

functionStorage.py

```
#bubble sort function, gets 2d list,
#where comulm 0 is a string representing the score
#and   column 1 is a string representing the name
def sortScores(listToSort):
    count = len(listToSort) #get number of row in the 2d list
    if count > 1:    #if more than one row, it might need sorting
        i = 0
        while (i < count-1):
            if int(listToSort[i][0]) < int(listToSort[i+1][0]):
                tempScore = listToSort[i][0]
                listToSort[i][0] = listToSort[i+1][0]
                listToSort[i+1][0] = tempScore

                tempName = listToSort[i][1]
                listToSort[i][1] = listToSort[i+1][1]
                listToSort[i+1][1] = tempName

            if i > 0:
                j = i
                while j > 0:
                    if int(listToSort[j-1][0]) < int(listToSort[j][0]) :
                        tempScore = listToSort[j-1][0]
                        listToSort[j-1][0] = listToSort[j][0]
                        listToSort[j][0] = tempScore

                        tempName = listToSort[j-1][1]
                        listToSort[j-1][1] = listToSort[j][1]
                        listToSort[j][1] = tempName
                    j -= 1
            i += 1
    return listToSort

```

```
# prerequest: 2d list should be sorted
def removeDuplicates(sortedList):
    count = int(len(sortedList))
    countToRemove = 0
    if count > 1:
        i = 0
        while (i < count-1):
            currentScore = sortedList[i][0]
            nextScore = sortedList[i+1][0]
            currentName = sortedList[i][1]
            nextName = sortedList[i+1][1]

            if currentScore == nextScore and currentName == nextName:
                sortedList[i][0] = "" #mark to be removed by setting empthy string
                sortedList[i][1] = ""
                countToRemove += 1
            i += 1

    #create the new list with less rows
    countNew = count - countToRemove
    newList = [["0","none"] for _ in range(countNew)]
    if countNew > 0:
        iNew = 0
        i = 0
        while (i < count):
            score = sortedList[i][0]
            if score: #if score string is not empty add record to new list
                newList[iNew][0] = sortedList[i][0]
                newList[iNew][1] = sortedList[i][1]
                iNew += 1
            i += 1

    return newList
```

```
def getLines(source):
    #get number of records, assumming not empty
    linesCount = sum(1 for line in open(str(source)))

    readFile = open(str(source),"r") #open file for reading

    r = 0
    lines = [["0","none"] for _ in range(int(linesCount))]

    #first line
    line = readFile.readline()
    count = 0
    while count < linesCount:
        #get rid of whitespace
        line.rstrip()
        #gets list of strings originally separated by " "
        strings = line.split(" ")
        for string in strings:
            if string.isdigit(): #be sure sting is made of digits
                lines[r][0] = string.strip()
            elif string:
                lines[r][1] = string.strip()

        r += 1; #get the index for the new row
        line = readFile.readline() #get new line
        count += 1 #keep rolling
    readFile.close()
    return lines

#appends one record to file,
# ignore the "lst" argument it is from TODO
def writeToFile(fileName, lst, score, name):
    appendFile = open(str(fileName),'a')
    appendFile.write("{} {}\n".format(score, name))
    appendFile.close()

def printScores(records):
    for row in records:
        print("{0} {1}".format(row[0],row[1]))
```

Arduino side the slave code:

slaveArduino

```
#include <Wire.h>
#define SLVAE_ADDRESS 0x04
void receiveData(int byteCount);
void sendData();
int number = 0;
void setup(){
    Serial.begin(9600);
    Wire.begin(SLVAE_ADDRESS);
    //attach function to be called when data need to be received
    Wire.onReceive(receiveData);
    //attach function to be called when data need to be send
    Wire.onRequest(sendData);
}
void loop(){
    delay(100);
}
void receiveData(int byteCount){
    number = Wire.read();
    Serial.print("data received: ");
    Serial.print(number);
}
void sendData(){
    Wire.write(number);
    Serial.print("sendData() was called \n");
}
```

Extending the scenario:

```
numberOfRecords = 5
records = [["0","none"] for _ in range(int(numberOfRecords))] #empty list
while True:
    for i in range(0,5): # 5 records
        score = ""
        name = ""
        for n in range(0,9): # 9 chars
            if n < 3:
                writeByte(n)
                time.sleep(0.1)
                score = score + str( chr(readByte()) )
            elif n == 3:
                continue
            elif n > 3:
                writeByte(n)
                time.sleep(0.1)
                name = name + str( chr(readByte()) )
        records[i][0] = score
        records[i][1] = name.rstrip()
        print("Recieved records from Arduino: {}".format(i+1))
    print("Arduino send 5 records:")
    printScores(records)
    break
```

Arduino side:

slaveArduino

```
#include <Wire.h>
#include <SD.h>
//SD card
#define NUMBER_RECORDS 5
#define SD_CARD_PIN 53
#define SLAVE_ADDRESS 0x04
#define NUMBER_OF_BYTES 17

File scoreRecordsFile;
int numberOfRecords = 0;
volatile char * record;
char str[NUMBER_OF_BYTES+1] = {'\0'};
char arr2[NUMBER_RECORDS][10]={'\0'};

volatile int index = 0;
volatile int recordIndex = 0;

void receiveData(int byteCount);
void sendData();
```

```
void receiveData(int byteCount);
void sendData();

void setup(){
    Serial.begin(9600);
    //start of records extraction
    if(!SD.begin(SD_CARD_PIN)){
      Serial.println("SD card init failed!");
      while(true) //stay here forever
        ;
    }else{
      Serial.println("SD card init OK!");
    }
```

```
    //open file for reading
    scoreRecordsFile = SD.open("SCOR.txt");
    if(scoreRecordsFile){ //chek if it was opened
      Serial.println("SCOR.txt opened OK!");
      //read the file
      int i = 0;
      while(scoreRecordsFile.available()){
          Serial.print("record: ");
          String str = scoreRecordsFile.readStringUntil('\n'); //returns a String
          int n = str.length();
          char chars[10] = {'\0'};    //fill with end of string character
          strcpy(chars, str.c_str()); //copy the string read from file to char array
          if((i < NUMBER_RECORDS) && (i >= 0)){    //read only the first 5 records, its top
            for(int k = 0; k < n; k++){
              arr2[i][k] = chars[k];  //copy local char to global char array
                                      //so it can be used later to send data
            }
            Serial.println(arr2[i]);  //print the record
          }
          i++; //increase the count for records
      }
      scoreRecordsFile.close();
    }else{
      Serial.println("Error opening SCOR.txt !");
    }
    //end of records extraction
```

```
57      Wire.begin(SLAVE_ADDRESS);
58      Wire.onReceive(receiveData);
59      Wire.onRequest(sendData);
60      record = arr2[recordIndex++]; //get pointer to the first record
61  }
62  void loop(){
63      delay(0);
64  }
```

```
65  void receiveData(int byteCount){
66      while(Wire.available()){
67        index = (int)Wire.read(); //read the next index
68        Serial.print("index received: ");
69        Serial.println(index);
70      }
71  }//end of receiveData
72  void sendData(void){
73    Serial.print("sendData() was called \n");
74    Wire.write((byte)record[index]); //send current char
75    Serial.print("char was sent: ");
76    Serial.println(record[index]);
77    if(index+1 > 8){ //RPi will request 8 chars only
78      record = arr2[recordIndex++]; //get pointer to new record
79      if(recordIndex > 4){ //there are only 5 records to send
80        recordIndex = 0;   //start again
81      }//start again
82    }
83  }//end of sendData
```

Appendix E (Kaloyan, code)

Scenario Final 2: Arduino Tic Tac Toe - Master RPi and Slave Arduino (signals RPi on end of game GPIOP)

master.py

```
import smbus
import time
import RPi.GPIO as GPIO

from functionStorage import addOneScore
from functionStorage import uploadFileToWeb

GPIO.setmode(GPIO.BCM)     #use BCM
GPIO.setup(21,GPIO.IN)     #use pin 21 as signal from slave to request data
bus = smbus.SMBus(1)       #activate the two way interface bus
slaveAddress = 0x04        #the address of the slave

def writeByte(value):
    bus.write_byte(slaveAddress, value)
    return -1

def readByte():
    value = bus.read_byte(slaveAddress)
    return value

name = ""

while True:
    while not GPIO.input(21):              #check if slave signals
        time.sleep(0.1)                    #delay
        print("wait for slave to call")
    print("slave to called")

    for n in range(0,5):                   #request name of 5 chars
        name = name + str( chr(readByte()) )
        time.sleep(0.1)                    #enable delay during debug on arduino

    print("Recieved name from Arduino: {}".format(name))
    addOneScore(name, "scores_v2.txt")
    uploadFileToWeb("scores_v2.txt",'files.000webhost.com','marksauer','V1IpWadgM0D',"public_html")
    time.sleep(2)
    name = ""
```

functionStorage.py

```python
import ftplib

def uploadFileToWeb(filename, server, user, password, subdir):
    session = ftplib.FTP(str(server),str(user),str(password)) #open ftp session
    session.cwd(str(subdir))            #define subdirectory on ftp
    file = open(str(filename),'rb')     #open file in read binary mode
    query = "STOR " + filename          #create override query
    session.storbinary(query, file)     #execute query
    file.close()
    session.quit()
    print("File was updated via FTP(www.marksauer.dk)\n")

```

```python
def addOneScore(name, filename):
    maxScoreReached = False
    name = name.rstrip()
    if isFileEmpty(filename):
        appendRecord(name, filename)
    else:
        records = getRecords(filename)
        if isNameInRecords(name, records):
            #show that python can return more than one variable
            records, maxScoreReached = addOneToScore(name, records)
            if not maxScoreReached: #no need to sort and override
                records = sortByScore(records)
                overrideFile(records, filename)
        else:
            appendRecord(name, filename)

def overrideFile(records, filename):
    file = open(str(filename),'w')
    size = int(len(records))
    for r in range(0,size):
        score = records[r][0]
        name = records[r][1]
        file.write("{} {}\n".format(score, name))
    file.close()
```

The logic for sortByScore function was adopted from the C code in assignment (page 15) in week 4 of the course 62507 Hardware-oriented programming (in C) in 2017 written by Kaloyan Penov.

```
def sortByScore(records):
    count = len(records)
    if count > 1:
        i = 0
        while (i < count-1):
            if int(records[i][0]) < int(records[i+1][0]):
                tempScore = records[i][0]
                records[i][0] = records[i+1][0]
                records[i+1][0] = tempScore

                tempName = records[i][1]
                records[i][1] = records[i+1][1]
                records[i+1][1] = tempName

            if i > 0:
                j = i
                while j > 0:
                    if int(records[j-1][0]) < int(records[j][0]) :
                        tempScore = records[j-1][0]
                        records[j-1][0] = records[j][0]
                        records[j][0] = tempScore

                        tempName = records[j-1][1]
                        records[j-1][1] = records[j][1]
                        records[j][1] = tempName
                    j -= 1
            i += 1
    return records
```

```
def addOneToScore(name, records):
    maxScoreReached = False
    count = int(len(records))
    maxScore = 999
    maxDigits = len(str(maxScore))
    if count > 0:
        i = 0
        while (i < count):
            currentName = records[i][1]
            if currentName == name:
                strScore = records[i][0]
                #cast score to int
                intScore = int(strScore)
                if intScore != maxScore:
                    intScore = intScore + 1
                    records[i][0] = formatScore(str(intScore), maxDigits)
                    break
                else:
                    maxScoreReached = True #max score is reached
                    break
            i += 1
    return records, maxScoreReached
```

```
def formatScore(strScore, maxDigits):
    zerosToAdd = maxDigits - len(strScore)
    if zerosToAdd > 0:
        while zerosToAdd > 0:
            strScore = "0" + strScore
            zerosToAdd = zerosToAdd - 1
    return strScore
```

```
98 def isNameInRecords(name, records):
99     nameInRecords = False
100    count = int(len(records))
101
102    if count > 0:
103        i = 0
104        while (i < count):
105            currentName = records[i][1]
106            if currentName.rstrip() == name.rstrip():
107                nameInRecords = True
108            i += 1
109    return nameInRecords
110
111 def getRecords(filename):
112    rowsCount = getCountRecords(filename)
113    readFile = open(str(filename),"r")
114    r = 0
115    records = [["0","none"] for _ in range(int(rowsCount))]
116    #first line
117    line = readFile.readline()
118    count = 0
119    while count < rowsCount:
120        line.rstrip() #get rid of whitespace
121        strings = line.split(" ")
122
123        for string in strings:
124            if string.isdigit():
125                records[r][0] = string.strip()
126            elif string:
127                records[r][1] = string.strip()
128
129        r += 1; #get the index for the new row
130        line = readFile.readline() #get new line
131        count += 1
132
133    readFile.close()
134    return records
```

```
136 def appendRecord(name, filename):
137    file = open(str(filename),'a')
138    file.write("{} {}\n".format("001", name))
139    file.close()
140
141 def isFileEmpty(filename):
142    return    0 == getCountRecords(filename)
143
144 def getCountRecords(filename):
145    return sum(1 for line in open(str(filename)))
146
147 def printScores(records):
148    for row in records:
149        print("{0} {1}".format(row[0],row[1]))
150
```

Appendix F (Kaloyan, code)

myI2C.h and myI2C.c below, were re-written(but uncomplete) from Chris Herring 2014 original files TWIlib.h and TWIlib.c and Nicholas Zambetti 2006 twi.h and twi.c files, this was done purely for educational purposes, some of the names and definitions might have changed.

myI2C.h

```
#ifndef MY_I2C__H__
#define MY_I2C__H__
// TWI bit rate
#define FREQ 100000L
#define F_CPU 16000000L   // in Hz

// Get TWI status
#define STATUS  (TWSR & 0xF8)

// Transmit buffer length
#define TX_BUFFER_LENGHT 20
// Receive buffer length
#define RX_BUFLEN_LENGHT 20
// Global transmit buffer
uint8_t transmitBuffer[TX_BUFFER_LENGHT];
// Global receive buffer
volatile uint8_t receiveBuffer[TX_BUFFER_LENGHT];
// Buffer indexes
volatile int bufferIndexTX; // Index of the transmit buffer. Is volatile, can change at any time.
int bufferIndexRX; // Current index in the receive buffer
// Buffer lengths
int bufferLenghtTX; // The total length of the transmit buffer
int bufferLenghtRX; // The total number of bytes to read (should be less than  RX_BUFLEN_LENGHT)

```

```
typedef enum {
  READY,
  INITIALIZING,
  REPEATED_START_SENT,
  MASTER_TRANSMITTER,
  MASTER_RECEIVER,
  SLAVE_TRANSMITTER,
  SLAVE_RECEIVER
}TWI_Mode;

typedef struct {
  TWI_Mode mode;
  uint8_t errorCode;
  uint8_t repStart;
}TWI_Status;
TWI_Status i2c_info;

//TWI control Macros to control the TWI hardware and
// for setting the control register TWCR
#define SEND_START()  (TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)|(1<<TWIE) )
#define SEND_STOP()  (TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)|(1<<TWIE) )
#define SEND_TRANSMIT()  (TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWIE) )
#define SEND_ACK()  (TWCR = (1<<TWINT)|(1<<TWEA)|(1<<TWEN)|(1<<TWIE) )
#define SEND_NACK()  (TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWIE) )

```

```
//Status codes
#define START_SENT      0x08 //start was sent
#define REP_START_SENT  0x10 //repeat start was sent
//Master Transmitter Mode
#define MT_SLAW_ACK     0x18 //SLA+W sent and ACK received
#define MT_SLAW_NACK    0x20 //SLA+W sent and NACK received
#define MT_DATA_ACK     0x28 //Data sent and ACK
#define MT_DATA_NACK    0x30 //Data sent and NACK
//Master Receiver  Mode
#define MR_SLAR_ACK     0x40 //SLA+R sent and ACK received
#define MR_SLAR_NACK    0x48 //SLA+R sent and NACK received
#define MR_DATA_ACK     0x50 //Data received and ACK
#define MR_DATA_NACK    0x58 //Data received and NACK
//Slave Receiver mode
#define SR_SLA_ACK            0x60 //SLA+W was addressed and ACk was returned
#define SR_GCALL_ACK           0x70 //addressed generally and ACK was returned
#define SR_ARB_LOST_SLA_ACK   0x68 //arbitration lost in SCL+R/W and own SLA+W addressed and ACK returned
#define SR_ARB_LOST_GCALL_ACK 0x78 //arbitration lost in SLA+R/W amd generral call was received and ACK returned

#define SR_DATA_ACK       0x80 //data received and ACK returned
#define SR_GCALL_DATA_ACK 0x90 //general call, data received and ACK returned
#define SR_STOP           0xA0 //stop or repeted start condition received

#define SR_DATA_NACK       0x88 //Data received and NACK returned
#define SR_GCALL_DATA_NACK 0x98 //Data received from general call and NACK returned

//Slave Transmitter
#define ST_SLA_ACK          0xA8 //SLA+R (master whats to read) addressed , Ack returned
#define ST_ARB_LOST_SLA_ACK 0xB0 //someone lost arbitration, SLA+R addressed, ACK returned

#define ST_DATA_ACK 0xB8 //one data byte was transmitted and got ACK from the receiver

#define ST_DATA_NACK 0xC0 //one data byte was transmitted and got NACK from the receiver(transmission is over)
#define ST_LAST_DATA 0xC8 //last data byte was transmitted and ACK was received
```

```
//function declarations
uint8_t transmitData(void * const dataToTransmit, uint8_t size, uint8_t isRepeatedStart);
void init(void);
uint8_t readData(uint8_t address, uint8_t bytesToread, uint8_t isRepeatedStart);
uint8_t isReady(void);

//different other states
#define LOST_ARBIT         0x38 //arbitration is lost
#define NO_RELEVANT_INFO   0xF8 //
#define ILLEGAL_START_STOP 0x00 //illegal start or stop has been detected
#define SUCCESS            0xFF //Successful transfer, this state is impossible from TWSR as bit2 is 0 and read only

//Master as Transmitter
#define TX_MAX_BUFFER_LENGHT 14
uint8_t TWITransmitBuffer[TX_MAX_BUFFER_LENGHT];
volatile int TXBufferIndex = 0;
int TXBufferLenght = 0;

//Master as Receiver
#define RX_MAX_BUFFER_LENGHT 14
volatile uint8_t TWIReceiveBuffer[RX_MAX_BUFFER_LENGHT];
int RXBufferIndex = 0;
int RXBufferLenght = 0;

#endif //MY_I2C__H__
```

myI2C.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include "myI2C.h"
#include "util/delay.h"

static void (* onSlaveTransmit)(void);//pointer to function with no params
static void (* onSlaveReceive)(uint8_t * receiveBuffer, int bufferIndexRX);//pointer to function

void attachSlaveTxEvent(void(*)(void) function);
void attachSlaveRxEvent(void(*)(uint8_t *, int) function);

void attachSlaveTxEvent(void(*)(void) function){
    onSlaveTransmit = function;
}
void attachSlaveRxEvent(void(*)(uint8_t *, int) function){
    onSlaveReceive = function;
}

//remember to pull SDA and SCL up f.eks.: digitalWrite(SDA, HIGH); digitalWrite(SCL, HIGH);
void i2c_init(){
    TWI_info.mode = READY;
    TWI_info.errorCode = 0xFF;
    TWI_info.repStart = 0;
    // Set pre-scalers (no pre-scaling)
    TWSR = 0;
    // Set bit rate
    TWBR = ((F_CPU / TWI_FREQ) - 16) / 2;
    // Enable TWI and interrupt
    TWCR = (1 << TWIE) | (1 << TWEN);
}
```

```c
uint8_t isReady(){
 return (i2c_info.mode == READY)|(i2c_info.mode == REPEATED_START_SENT) ? 1 : 0;
}
```

```
uint8_t transmitData(void * const dataToTransmit, uint8_t size, uint8_t isRepeatedStart){
    if(size <= TX_MAX_BUFFER_LENGHT){
       //wait to be ready
       while(!isReady()){
         ;
       }
      //set repeat status mode
      i2c_info.repStart = isRepeatedStart;
      //copy data in the transmit buffer
      uint8_t * dataPtr =  (uint8_t *)dataToTransmit;
      //make the data global
      for(int i = 0; i < size; i++){
          transmitBuffer[i] =  dataPtr[i];
      }
      bufferIndexTX = 0;
      bufferLenghtTX = size;
      //if a repeated start  was sent ,the i2c devises are already listening for address
      //no need for another start signal
      if(i2c_info.mode == REPEATED_START_SENT){
         i2c_info.mode = INITIALIZING;
         //load data to the transmit buffer
         TWDR = transmitBuffer[bufferIndexTX++];
         //call macro to sent current byte
         SEND_TRANSMIT();
      }else{ //else send the normal start signal to begin transmission
         i2c_info.mode = INITIALIZING;
         SEND_START();
      }
    }else{
        return 1; //this is error as data size is bigger than the buffer size
    }
    return 0; //transmission of the one byte in TWDR(the data register) was okay
}
```

```
//this will be called after each i2c hardware operation
ISR(i2c_vector){
    switch(STATUS){//this is the status register vale(TWSR & 0xF8)
        //Master is transmiter or writing address
        case MT_SLAW_ACK: //master transmitted -> slave address and write flag, and got ACK
            //set mode to master transmitter
            i2c_info.mode = MASTER_TRANSMITTER;
            //there is no break here so jump to the next case
        case START_SENT: //start signal was transmitted
        case MT_DATA_ACK: //one byte of data has been transmitted, and got ACK
            if(bufferIndexTX < bufferLenghtTX){
            //load data to the transmit buffer
                TWDR = transmitBuffer[bufferIndexTX++];
                SEND_TRANSMIT(); //macro -> sent current byte signal to the i2c hardware
            }else if(i2c_info.repStart){//this transmission is complete but do not relese the bus yet
                i2c_info.errorCode = 0xFF;
                SEND_START();//macro -> send start signal to the i2c hardware
            }else{ //all transmiffions are done , its time to exit
                i2c_info.mode = READY;
                i2c_info.errorCode = 0xFF;
                SEND_STOP();//macro -> send stop signal to the i2c hardware
            }
        break;
```

```
92      //Master is receiver
93      case MR_SLAR_ACK: //master receiver -> slave address and read flag, got ACK
94          //switch to master receiver mode
95              i2c_info.mode = MASTER_RECEIVER;
96          //if there is more than one byte to read, receive data byte and return ACK
97          if(bufferIndexRX < bufferLenghtTX-1){
98              i2c_info.errorCode = NO_RELEVANT_INFO;
99          SEND_ACK();//macro -> send ACK to the i2c hardware so that the next byte can be sent
100             }else{ //when the only data byte has been received, return NACK
101             i2c_info.errorCode = NO_RELEVANT_INFO;
102         SEND_NACK();//macro -> send NACK to the i2c hardware so that no more bytes need to be received
103             }
104         break;
105     case MR_DATA_ACK: // data byte was received and ACK was received meaning more bytes waiting to be received
106             //insert the byte from the data register to the received buffer array
107             receiveBuffer[bufferIndexRX++] = TWDR;
108
109             //if there is more than one byte to read , receive data byte and return ACK
110         if(bufferIndexRX < bufferLenghtRX - 1){
111             i2c_info.errorCode = NO_RELEVANT_INFO;
112             SEND_ACK();
113         }else{//when the last byte was received, set errorCode and return NACK
114             i2c_info.errorCode = NO_RELEVANT_INFO;
115             SEND_NACK();
116         }
117         break;
```

```
118     case MR_DATA_NACK:  //the last data byte is about to be received and NACK was transmitted. This is the end of the transmission.
119         //extract the last byte
120         receiveBuffer[bufferIndex++] = TWDR;
121         //the current transmission is complete, but do not relese the bus yet
122         if(i2c_info.repStart){
123             i2c_info.errorCode = 0xFF;
124                 SEND_START();
125             }else{//all transmissions are completed, send stop signal
126             i2c_info.mode = READY;
127         i2c_info.errorCode = 0xFF;
128         SEND_STOP(); //macro -> send stop to the i2c hardware
129         }
130         break;
131
132     //Master receiver and Master transmitter general cases
133     case MR_SLAR_NACK: //slave address transmitted with read flag, NACK received
134     case MT_SLAW_NACK: //slave address transmitted with write flag, NACK received
135     case MT_DATA_NACK: //arbitration has been lost
136         if(i2c_info.repStart){
137             i2c_info.errorCode = STATUS;
138         SEND_START(); //macro -> send start to the i2c hardware
139         }else{//all transmitions are complete, its time to exit
140             i2c_info.mode = READY;
141         i2c_info.errorCode = STATUS;
142         SEND_STOP(); //macro -> send stop to the i2c hardware
143             }
144         break;
```

```
145     //Slave is receiver
146     case SR_SCL_ACK://addressed, returned ACK
147         case SR_GCALL_ACK: //addressed generally, returned ACK
148         case SR_ARB_LOST_SLA_ACK://master lost arbitration, addressed and ACK returned
149         case SR_ARB_LOST_GCALL_ACK://general call, master lost arbitration, addressed ACK returned
150             //go into slave receiver mode
151         i2c_info.mode = SLAVE_RECEIVER;
152         //indicate that the  reciveBuffer can be overwritten and ACK
153         bufferIndexRX = 0;
154         SEND_ACK();
155         break;
156         case SR_DATA_ACK://data received, returned ack
157     case SR_GCALL_DATA_ACK: //data received generally, returned ack
158         if(bufferIndexRX < bufferLenghtRX){
159             receiveBuffer[bufferIndexRX++] = TWDR;
160         i2c_info.errorCode = NO_RELEVANT_INFO;
161         SEND_ACK();
162         }else{
163         i2c_info.errorCode = NO_RELEVANT_INFO;
164             SEND_NACK();
165             }
166         break;
167     case SR_STOP: //stop or repeated start condition received
168         //put end of string char in the end if there is spece
169         if(bufferIndexRX < bufferLenghtRX){
170             receiveBuffer[bufferIndex]='\0';
171         }
172             //callback to the user-defined callback function
173         onSlaveReceive(receiveBuffer, bufferIndex);
174             SEND_ACK();//ACK future responses
175         i2c_info.mode = READY;
176         break;
```

```
177     case SR_DATA_NACK: //data received, returned NACK
178     case SR_GCALL_DATA_NACK: //data recived generally, returned NACK
179         //NACK  than  back to master
180         SEND_NACK();
181         break;
182
183     //Slave transmitter
184         case ST_SLA_ACK://addressed and ACK returned
185     case ST_ARB_LOST_SLA_ACK: //master lost arbitration, ACK return
186             i2c_info.mode = SLAVE_TRANSMITTER;   //enter slave transmitter mode
187             bufferIndexTX = 0; //get the bufferIndex ready for iterations
188         bufferLenghtTX = 0; //let the user change the lenght
189             //user nust call i2c_transmit(bytes,lenght); to fill the buffer and the lenght
190         onSlaveTransmit();
191         //if user didn't change the buffer and lenght give them init values
192         if(bufferLenghtTX==0){
193             bufferLenght = 1;
194         transmitBuffer[0] = 0x00;
195             }
196     //transmit first byte from buffer, fall
197         case ST_DATA_ACK://byte was sent and ACK was returned
198         //copy data to output register
199         TWDR = transmitBuffer[bufferIndexTX++];
200         //if more to send give, ACK otherse give NACK
201             if(bufferIndexTX < bufferLenghtTX){
202             SEND_ACK();
203             }else{
204                 SEND_NACK();
205         }
206         break;
```

```
207     case ST_DATA_NACK: //received NACK, we are done
208     case ST_LAST_DATA: //received ACK but we are done already
209             //ACK for future responses
210             SEND_ACK();
211         //slave stays in receiver state
212             i2c_info.mode = READY;
213         break;
214     //different other states
215     case NO_RELEVANT_INFO://it is not possible but used here to jump between operations
216         break;
217     case ILLEGAL_START_STOP://illegal start or stop signal,abort and return with error
218         i2c_info.errorCode = ILLEGAL_START_STOP;
219         i2c_info.mode = READY;
220         SEND_STOP();//macro -> send stop signal to the i2c hardware
221         break;
222     }
223
224
225 }
```