Labyrint

EITA15 - Digitala System
Andersson Lukas, Fontes Arild, Grey Elofsson
Bo, Kajsjö Emil, Malm Alexander
2022-05-25

Handledare:
Bertil Lindvall
Lars-Göran Larsson

Sammanfattning

I denna rapport beskrivs framtagandet av ett labyrintspel som varit en del av kursen digitala system EITA15 på Lunds tekniska högskola. Utifrån tidigare teoretiska och praktiska kunskaper från kursen skapade denna projektgrupp ett handhållet labyrintspel vars huvudkomponenter består av en processor, en LCD-display samt en accelerometer. Man skulle kanske kunna tro att det största problemet borde vara att få ut lämpliga värden för styrningen från accelerometern, men det visade sig att uppritandet av kulan och labyrinten på LCD-skärmen samt kollisionsdetektering mellan kula och labyrintens väggar var än mer komplicerad. Till slut lyckades gruppen ta fram ett fungerande labyrintspel med startskärm, tidtagning och att spelet lagrar den snabbaste tiden genom labyrinten från det att strömmen var påslagen.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Projektet är ett avslutande moment på kursen digitala system. Under projektet tillämpas inhämtad kunskap kopplad till kursen digitala system samt att få erfarenhet av arbete i grupp och att leverera resultat under deadline. Förväntningar på projektet är att förankra kunskapen i ett praktiskt projekt och få en djupare förståelse för hela kedjan i utvecklingen från mjukvara till hårdvara och slutligen användaren.

1.2 Syfte

Syftet med projektet är att bygga ett labyrintspel som körs på en microcontroller. Spelet visas på en LCD-display och styr en kulas bana genom en labyrint. Kulans bana påverkas genom att luta en accelerometer.

1.3 Problembeskrivning

Nedan följer projektgruppens kravspecifikation, vilken får fungera som vår problembeskrivning. Under resultat beskrivs vilka delar av denna som en kunde eller valdes ej implementeras.

Syftet med det här projektet är att bygga ett slags labyrintspel, med ett klassiskt Brio Labyrintspel som förebild (sätt in ev bild). Istället för en trälabyrint ska den här varianten bestå av en Atmel ATmega1284, en GDM12864C (Red) LCD-display och en ADXL335 accelerometer. Displayen används för att rita upp labyrinten och kulan, medan accelerometern används för att styra kulan genom att man lutar på spelet. Spelet startas med hjälp av en startknapp och ifall något går fel finns det en reset-knapp, som gör att man kan nollställa spelet. Med hjälp av en tredje knapp kan man även välja mellan labyrinter av olika svårighetsgrad.

Materiel som behövs för att bygga hårdvaran

- Atmel ATmega1284
- GDM12864C (Red) LCD-display
- ADXL335 accelerometer
- Knappar

    - Startknapp/Ok
    - Uppåt
    - Nedåt
    - Reset/Cancel
- JTAG för kommunikation med utvecklingsmiljön
- Kondensatorer för att rensa accelerometersignalen från brus.
- Strömförsörjning både i form av batteri och som +5V DC.

Användarkrav

- Representationen av labyrinten och kulan skall ritas upp på en GDM12864C LCD-display.
- Vid uppstart eller reset visas en titelskärm med meny, med aktuell labyrint i bakgrunden.
- Menyn skall bestå av alternativen start och visa hi score.
- Menyalternativet start skall vara förvalt.
- När användaren valt start skall en ny meny, startmenyn, visas.
- Startmenyn består av alternativen svårighetsgrad och level.
- Svårighetsgrad består av två alternativ: n00b och 1337.
- Level väljer mellan flera olika labyrinter. När man bläddrar mellan alternativen skall labyrinten bakom menyn ändras till den valda labyrinten, så användaren kan se sitt val i realtid.
- Navigation i menyerna sker enligt följande:
    - Upp och ned flyttar markeringen av menyalternativ uppåt eller nedåt
    - Vanlig text skall ha fylld text och ej ifylld bakgrund.
    - Markerad text skall ha ifylld bakgrund och ej ifylld text.
    - Start/ok väljer menyalternativet
    - Cancel/reset backar ut ur nuvarande meny till föregående, eller nollställer spelet helt och hållet i den första menynivån.
- Användaren skall kunna starta spelet genom att trycka på startknappen.
- Användaren skall kunna nollställa spelet omedelbart genom att trycka på start- och nollställningsknappen.
- Användaren ska kunna välja mellan olika labyrinter genom att trycka upp eller ned.

- Under pågående spel skall det finnas en synlig timer, så användaren kan se hur lång tid som spelsessionen pågått.
- Timern skall sitta i det övre högra hörnet av LCD-displayen.
- Timern skall kunna visa minuter, sekunder och tiondelar.
- I det övre vänstra hörnet på displayen skall det stå vilken level man är på.
- Det skall finnas en highscore-lista med de tio snabbaste tiderna genom labyrinten.
- Efter att användaren har lyckats sätta en topp tio-tid skall användaren kunna mata in sin signatur i listan.
- Signaturen skall bestå av en till tre bokstäver.
- Spelet skall styras med hjälp av accelerometern som avläser hur snabbt och mycket spelet lutas i x- och y-led.
- Kulans fart och acceleration skall vara proportionell mot hur användaren lutar spelet.
- För varje labyrint skall det finnas en tydlig startpunkt och en tydlig mållinje.
- En labyrint består av
    - Väggar
    - Hål, som kulan kan trilla ned i.
    - Startlinje
    - Mållinje
- Spelet avslutas när ett av följande kriterier uppfylls/inträffar
    - När kulan “trillar ned” i ett hål
    - När kulan når mållinjen
    - 1337 mode: Kulan rör vid någon av väggarna.
- Användarens tid sparas endast om kulan har kommit till mållinjen.
- När spelet avslutas händer något av följande:
    - När kulan faller ned i ett hål eller träffar en vägg i 1337 mode visas en textruta med titeln “Pwnd! Retry (start) / Quit (cancel)”.
    - Trycker användaren på start/ok skall spelet starta om direkt.
    - Trycker användaren på reset/cancel skall spelet återgå till den första startmenyn.
    - När kulan mållinjen visas en textruta med titeln “#Winning” och highscore-listan visas. Ifall användaren har en tillräckligt bra tid skall användarens initialer matas in. Både start/ok och reset/cancel tar användaren till startmenyn.

- Spelet skall ha två svårighetsgrader n00b mode och 1337 mode.
- I n00b mode får kulan röra vid väggarna och i 1337 mode får kulan inte vidröra väggarna.
- Kulan skall representeras av fyra pixlar.
- Labyrintens väggar skall vara två pixlar breda.
- En vägg i labyrinten kan enbart gå i antingen x- eller y-led. Några vinklade eller sneda väggar existerar alltså inte.
- Spelet skall detektera kollision mellan kula och vägg.
- Kollision räknas som en elastisk stöt, där kulans ingångsvinkel blir dess utgångsvinkel.

1.3.1 Avgränsningar

De främsta avgränsningarna för projektet var inledningsvis utbudet av komponenter vi hade att tillgå. Under projektets senare delar var projektgruppen tvungna att avvara många av de punkter som är beskrivna nedan under kravspecifikation som följd av tidsbrist. För att få en rimlig arbetsbelastning valde vi att endast ha en labyrint och en svårighetsgrad, istället för en meny där man kunde välja mellan olika svårighetsgrader. Välkomstskärmen är ersatt av ett meddelande “Get Ready”, samt en nedräkning. Den snabbaste speltiden sedan strömmen slogs på lagras, men möjligheten att skriva in sina initialer saknas liksom en längre highscore-lista. Vi valde även bort att ha med hål, eftersom det blir lite för svårt att se skillnad på vad som är ett hål och vad som är en del av labyrintens väggar, eftersom grafiken är svartvit och allt består av relativt stora pixlar.

2 Metod

2.1 Arbetsgång

Projektet inleddes genom skapandet av en kravspecifikation för den labyrint vi skulle bygga. Med de hårdvarutekniska begränsningarna i åtanke sattes höga mål projektgruppen sedan jobbade emot. Med kravspecefikationen som bas satte gruppmedlemarna in sig i hårdvarans specifikationer varefter ett kretsschema producerades, se bilaga[1]. Detta gjordes med programvaran Autodesk Eagle. Utifrån kretsschemat kunde labyrinten sedan byggas ihop med de beskrivna komponenterna och sladdarna för el och dataöverföring kopplas. De flesta av kopplingarna mellan komponenterna virades medan strömförsörjningen löddes. Efter detta steg var påbörjades designen av mjukvaran. Med grund i projektgruppens kravspecifikation skapades en högnivådesign för de nödvändiga komponenterna som behövde konfigureras, deras övergripliga metoder för kommunikation samt den nödvändiga spellogiken.
All programkod har skrivits i C och vi har använt oss av utvecklingsmiljön Atmel Studio 7.

Mjukvaruutvecklingen har varit mycket iterativ där ny kod har skrivits och exekverats för att gruppen sedan kunnat avläsa resultatet på displayen: Ofta har det varit enstaka pixlar som hamnat fel vid olika kollisioner. Atmels debuggerfunktion och också varit ovärderlig vid dessa tillfällen för att stegvis kunna analysera resultatet från våra algoritmer.
I slutskedet av projektet skapades en hemsida där vi presenterar vårt arbete och allt material. Till sist sker en muntlig presentation där projektetarbetet opponeras av en annan grupp i kursen EITA15.

2.2 Hårdvara

- Processorn för projektet har varit en ATmega1284 [1] klockad till 1 MHz och 128 kb programmerbart flashminne.
- Accelerometer av modell ADXL335 [3] användes för att mäta nivåskillnader för att avgöra hur snabbt kulan skulle röra sig i de olika riktningarna. En spänningsregulator användes för att få korrekt spänning.
- JTAG ICE användes för att kommunikation mellan Atmel studio och programmering av processorn.

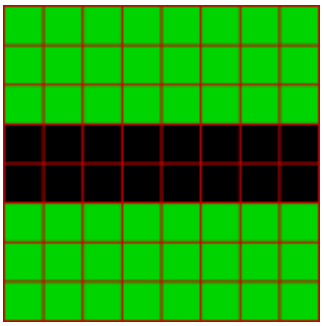
- Displayen som använts, ADM12864H [2], har en upplösning på 128✕64 pixlar. Det krävdes en lägre spänning för bakgrundsbelysningen än resterande vilket innebar att en potentiometer kopplades in.
- De fyra knapparna som beskrivs i kretsschemat, se bilaga 1, kopplades tillsammans med individuella resistorer. Dessa kom ej att implementeras vidare.

2.3 Komponenter och spellogik

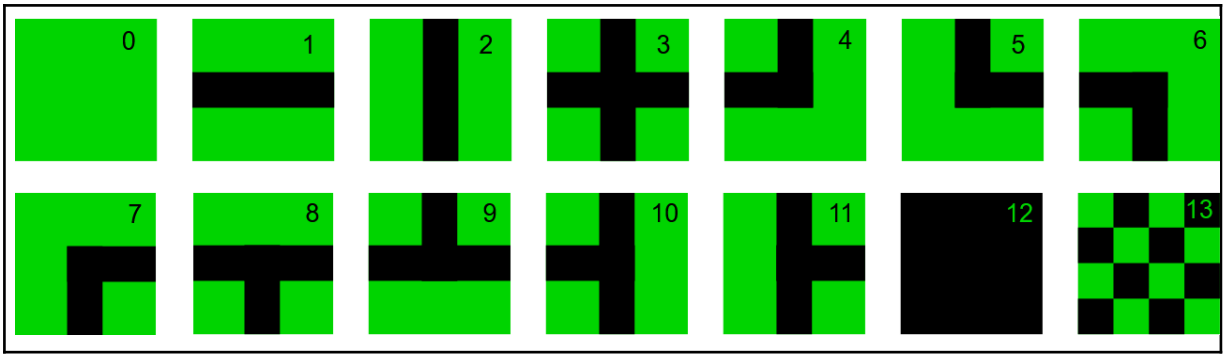
2.3.1 Display

Praktiskt är skärmen delad i två stycken 64✕64 segment sida vid sida [2]. För att skriva till displayen behöver en displayhalva väljas. I horisontellt led är displayen alltså indelad i två stycken sidor om 64 pixlar. I vertikalt led är displayen däremot indelad i åtta stycken segment om åtta pixlar. Dessa segment refereras till som “pages” (eng. sidor). För att skriva till displayen väljs alltså sidhalva, sedan position i horisontalled och sedan tar displayen emot ett helt åtta bitar långt ord för att fylla den nuvarande valda sidans alla åtta pixlar. För att underlätta användningen av displayen skrevs funktioner för att sätta specifika flaggor enligt databladet, till exempel för att välja displayhalva, ändra horisontalkoordinat och för att mata in data.

För att rita upp labyrinten användes ett rutnätssystem för att dela upp skärmen i 16✕8 rutor om 8✕8 pixlar. Dessa rutorna kallade vi internt för “tiles”, och rutnätet som innehöll dessa kallades “tile grid”. 14 stycken olika tiles ritades upp i ritprogrammet InkScape och presenteras i figur 1 och 2.

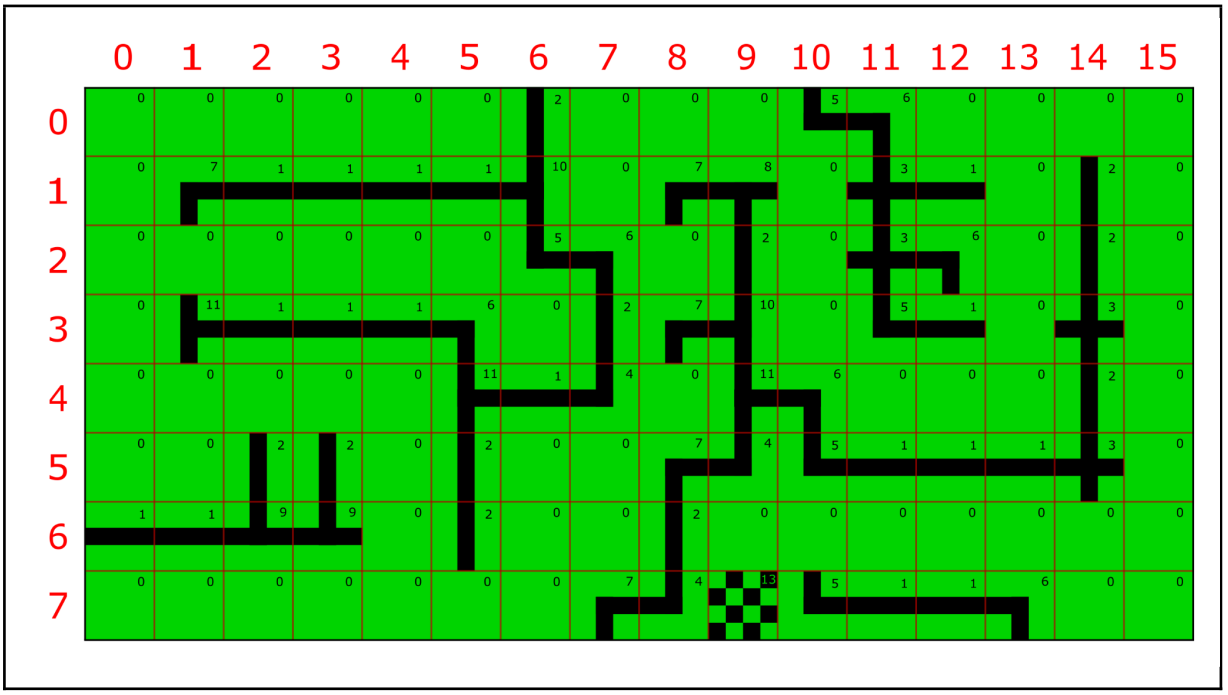


Figur 1: En tile som den ritades upp i ritprogrammet.



Figur 2: Samtliga tiles som de ritades upp i ritprogrammet.

Dessa tiles skrevs sedan manuellt in i en matris där siffran presenterad i övre högra hörnet motsvarar indexet i matrisen. Exempelvis är tile med index noll helt enkelt åtta upprepningar av bitarna 0b00000000, index ett är åtta upprepningar av bitarna 0b00011000 och så vidare. Dessa tiles användes för att manuellt designa labyrinten i det som kallades “tile grid”, vilket presenteras i figur x.



Figur 3: Hela labyrinten som den designades i ritprogrammet.

Efter att ha ritat upp labyrinten skrevs en 16×8 matris av tiles, vilket möjliggjorde uppslagningen av varje tile för varje position i rutnätet. Tack vare detta kunde en nestad for-loop skrivas för att iterera över hela matrisen och få rätt tile för varje position och således rita hela labyrinten till skärmen.

2.3.2 Accelerometer

Efter konfiguration av accelerometer och AD-omvandling kunde vi läsa av de värden som behövdes för att senare kunna styra kulan. Med reservation för mindre felmarginaler som vidare inte påverkade funktionaliteten gavs följande värden: i X- och Y-riktning fick avlästes 360 vid -90° tilt och 610 vid +90°, vid plant läge observerades 500. Inledningsvis rullade vår kula iväg vid start, vi misstänker att det beror på att accelerometern var aningen sned på kopplingsdäcket. Värdena räknades om så att de låg i intervallet [-1, 1] för att bli mer lättarbetade.

2.3.3 Kollisionsdetektering

Ett av de största problemen under projektets gång har handlat om kollisionsdetekteringen mellan kulan med displayens ändpunkter och labyrintens väggar. Displayen hade möjligheten att skicka information om tända pixlar, men vi valde att istället låta spellogiken kontrollera gränserna genom att hela tiden kolla av kulans relativa position med den hårdkodade labyrinten. Kulan kan kollidera i alla fyra riktningarna, och även två riktningar samtidigt som till exempel ett hörn, vilket kräver ett stort antal möjliga fall för spellogiken att fånga upp. Den tidigare beskrivna uppdelningen av displayen i två halvor försvårade arbetet ytterligare. Mycket tid behöver läggas för att fånga upp alla dessa fall.

För att upptäcka kollision skrevs kod som kollade upp vilken tile kulan befinner sig i, sedan görs den önskade förflyttningen enbart i kod, utan att uppdatera kulans position på skärmen. Därefter jämförs kulans position med nuvarande tiles basutseende med hjälp av en och-operator. Om och-operationen ger tillbaka en etta innebär detta att kulans position överlappar med en pixel på tilen, vilket innebär att förflyttningen inte var giltig. I sådant fall ångras förflyttningen och kulans hastighet i det relevanta ledet sätts till noll. Om och-operationen däremot ger tillbaka en nolla innebär detta att kulan enbart befinner sig på positioner där pixlarna redan var släckta, vilket innebär att kulan kan ritas på den beräknade positionen.

För uppdatering av kulans position krävdes det att programmet kodades så den vid varje loop raderar kulan, räknar ut den nya positionen för kulan och sedan ritar upp den. Precis som vid kollisionsdetekteringen finns det även vid radering och uppritning specialfall då kulan består av fyra pixlar. Kulan kan befinna sig på olika sidhalvor, i olika tiles, på olika pages av

skärmen och i värsta fall en kombination sådant att kulan befinner sig i fyra olika tiles samtidigt som den befinner sig på båda skärmhalvorna.

3 Resultat

Med mycket kod löstes de största problemen under arbetets gång. Skrivning till displayen löstes med hjälpmetoder, kulan kunde ritas på ett bra sätt. Tack vare många if-satser så kunde alla specialfall av kulans olika positioner med avseende på skärmens skarvar. För exempel på kod se bilagor 1 till 5. Accelerometervärdena kunde mätas på ett tillförlitligt sätt och användas med förnuft för att få ett rörelsemönster som känns naturligt. Slutresultatet blev ett välfungerande och responsivt labyrintspel som uppfyller de väsentliga kraven från kravspecifikationen. Bilder på spelet presenteras i bilagor 6 till 8.

Som tidigare nämnt i rapporten utelämnades en del funktioner då tiden inte tillät vidare utveckling. En så pass bra prototyp att gruppen kände att mer arbete skulle vara överflödigt för kursens krav. Till exempel monterades knappar i början för en eventuell meny-funktion med val av olika labyrinter och svårighetsgrader. I slutändan lämnades knapparna utan funktion. Ursprungligen var det även tänkt att labyrinten skulle ha “fallgropar” som skulle återställa kulans position till början av labyrinten, men dessa blev aldrig implementerade.

4 Diskussion

Gruppen fick under arbetets gång göra avkall på vissa funktioner i den väl tilltagna kravspecifikationen. Det fanns till exempel inte tid att implementera knapparna eller menysystemet. Processorns minneskapacitet innebar också att vi fick omvärdera tanken på att ha flera olika labyrinter och istället bara kida en. Inledningsvis då programkoden skulle skrivas fick vi inte kontakt med vår display, som dessutom blev väldigt varm. Vi såg att detta berodde på att kopplingarna var dragna fel vilket. Efter att ha gjort det korrekt enligt vår ritning kunde vi fortfarande inte få kontakt med displayen mer än bakgrundsbelysningen. Det visade sig slutligen att gruppen vid tidigare försök bränt ut displayen och vid byta fungerade allt som planerat.

Kulan kan kännas lite “klistrig” när man spelar. För att avhjälpa det borde vi kanske implementerat AD-omvandlingen med hjälp av avbrott, så både x- och y-värdena alltid är aktuella. Som det är nu riskerar värden i antingen x- eller y-led att gå förlorade eftersom vårt program väntar på en uträkning i varje ledd, vilket tar tid (se datablad [3]), för att till sist rita upp kulan. Detta Istället för att konstant rita kulan och ta nya värden direkt när de är färdigomvandlade, som det hade kunnat vara om vi använt oss av avbrott istället..

5 Källförteckning

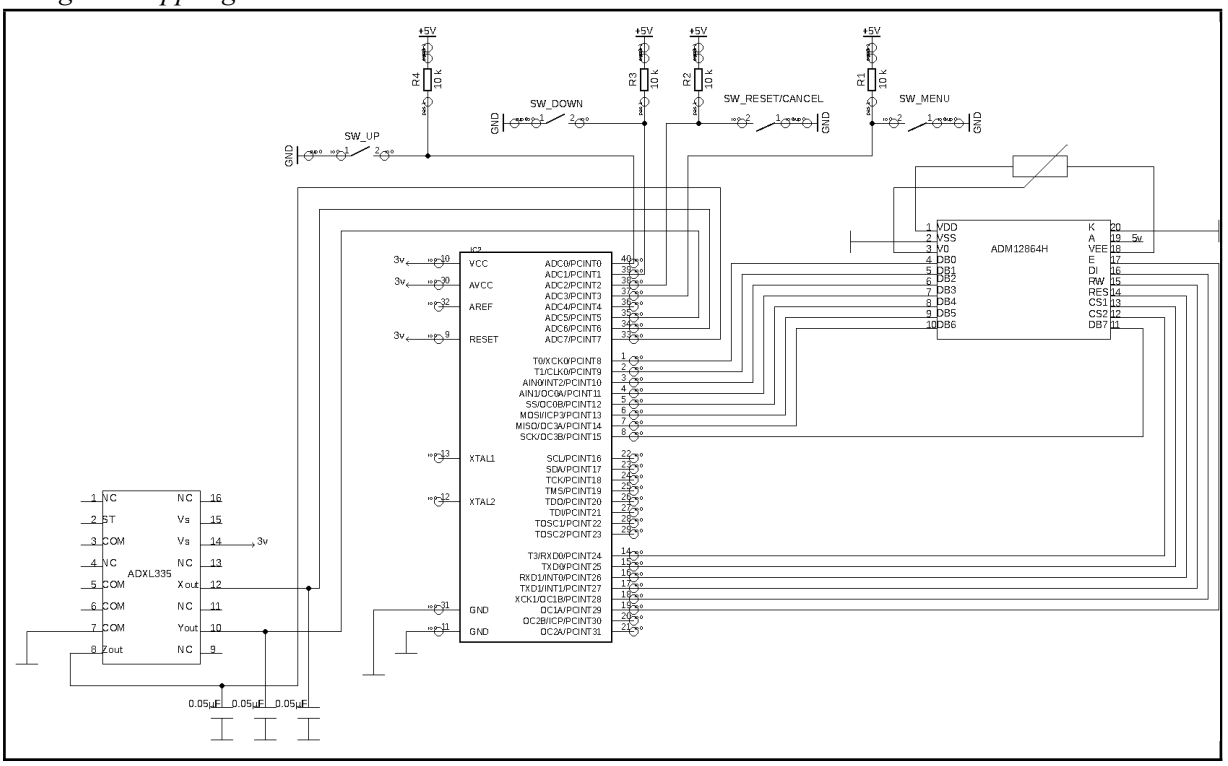
1. Atmel Corporation, “Atmel-42718C-ATmega1284_Datasheet_Complete-10/2016”, Datasheet, 2016
   https://www.eit.lth.se/fileadmin/eit/courses/datablad/Processors/ATmega1284.pdf

2. Amotec, “ADM12864H SPECIFICATIONS OF LCD MODULE”, Datasheet, 2008
   http://users.ece.utexas.edu/~valvano/Datasheets/ADM12864H.pdf

3. Analog Devices, “ADXL335 Small, Low Power, 3-Axis ±3 g Accelerometer”, Datasheet, 2009
   https://www.elfa.se/Web/Downloads/_m/an/SEN-09269_eng_man.pdf

Appendix A – Kopplingsschema

Bilaga 1: Kopplingsschema.



Appendix B – Källkod

B1 charset_header.h – Tiles och bokstäver

```
/*
 * charset_header.h
 *
 * Created: 2022-05-16 10:37:51
 *  Author: cid12bel
 */


#ifndef CHARSET_HEADER_H_
#define CHARSET_HEADER_H_

#define    TILE_WIDTH      8
#define  CHARSET_SIZE 58

typedef struct
{
  uint8_t is_start;
  uint8_t finish_line;
```

```
  uint8_t has_wall;
  uint8_t char_id;
} tile;

uint8_t charset[][8] = {
     {0b00000000, 0b11111000, 0b11111100, 0b11111110, 0b00100110,
0b00100110, 0b11111100, 0b11111000} // A                          0
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b10010010,
0b10010010, 0b11111110, 0b01101100} // B
    ,{0b00000000, 0b00111000, 0b01111100, 0b11111110, 0b10000010,
0b10000010, 0b11000110, 0b01000100} // C
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b10000010,
0b11000110, 0b01111100, 0b00111000} // D
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b10010010,
0b10010010, 0b10010010, 0b10000010} // E                          4
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b00010010,
0b00010010, 0b00010010, 0b00000010} // F
    ,{0b00000000, 0b00111000, 0b01111100, 0b11111110, 0b10000010,
0b10010010, 0b11110010, 0b11110000} // G
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b00010000,
0b00010000, 0b11111110, 0b11111110} // H
    ,{0b00000000, 0b10000010, 0b11111110, 0b11111110, 0b11111110,
0b10000010, 0b00000000, 0b00000000} // I
    ,{0b00000000, 0b01100000, 0b11100000, 0b11100000, 0b10000010,
0b10000010, 0b11111110, 0b01111110} // J                          9
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b00111000,
0b01101100, 0b11000110, 0b10000010} // K
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b10000000,
0b10000000, 0b10000000, 0b00000000} // L
    ,{0b00000000, 0b11111110, 0b11111110, 0b00011100, 0b00111000,
0b00011100, 0b11111110, 0b11111110} // M
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111100, 0b00011000,
0b00110000, 0b11111110, 0b11111110} // N
    ,{0b00000000, 0b01111100, 0b11111110, 0b11111110, 0b10000010,
0b10000010, 0b11111110, 0b01111100} // O                         14
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b00100010,
0b00100010, 0b00111110, 0b00011100} // P
    ,{0b00000000, 0b01111100, 0b11111110, 0b11111110, 0b10100010,
0b11100010, 0b01111110, 0b10111100} // Q
    ,{0b00000000, 0b11111110, 0b11111110, 0b11111110, 0b00100010,
0b01110010, 0b11011110, 0b10001100} // R
    ,{0b00000000, 0b01001100, 0b11011110, 0b11011110, 0b10010010,
0b10010010, 0b11110110, 0b01100100} // S
    ,{0b00000000, 0b00000010, 0b00000010, 0b11111110, 0b11111110,
0b11111110, 0b00000010, 0b00000010} // T                         19
    ,{0b00000000, 0b01111110, 0b11111110, 0b11111110, 0b10000000,
0b10000000, 0b11111110, 0b01111110} // U
```

```
    ,{0b00000000, 0b00011110, 0b00111110, 0b01111110, 0b11000000,
0b01100000, 0b00111110, 0b00011110} // V
    ,{0b00000000, 0b11111110, 0b11111110, 0b01100000, 0b00111000,
0b01100000, 0b11111110, 0b11111110} // W
    ,{0b00000000, 0b11000110, 0b11101110, 0b01111100, 0b00110000,
0b01111100, 0b00010001, 0b11000110} // X
    ,{0b00000000, 0b00001110, 0b00011110, 0b11111110, 0b11110000,
0b11110000, 0b00011110, 0b00001110} // Y                        24
    ,{0b00000000, 0b11000010, 0b11100010, 0b11110010, 0b10111010,
0b10011110, 0b10001110, 0b10000110} // Z
    ,{0b00000000, 0b11100000, 0b11110000, 0b11111100, 0b01011010,
0b01011100, 0b11110000, 0b11100000} // Å
    ,{0b00000000, 0b11100000, 0b11110010, 0b11111010, 0b01011000,
0b01011010, 0b11110010, 0b11100000} // Ä
    ,{0b00000000, 0b01110000, 0b11111010, 0b11111010, 0b10001000,
0b10001010, 0b11111010, 0b01110000} // Ö
    ,{0b00000000, 0b01111100, 0b11111110, 0b10000010, 0b10000010,
0b11111110, 0b11111110, 0b01111100} // 0                        29
    ,{0b00000000, 0b10000100, 0b11111110, 0b11111110, 0b11111110,
0b11111110, 0b10000000, 0b00000000} // 1
    ,{0b00000000, 0b11000100, 0b11100110, 0b11110010, 0b10110010,
0b10011110, 0b10011110, 0b10001100} // 2
    ,{0b00000000, 0b01000000, 0b11000010, 0b10001010, 0b10001010,
0b11111110, 0b11110110, 0b01100010} // 3
    ,{0b00000000, 0b00110000, 0b00111000, 0b00101100, 0b00100110,
0b11111110, 0b11111110, 0b00100000} // 4
    ,{0b00000000, 0b01001110, 0b11001110, 0b10001010, 0b10001010,
0b11111010, 0b11111010, 0b01110000} // 5                        34
    ,{0b00000000, 0b01111100, 0b11111110, 0b10001010, 0b10001010,
0b11111010, 0b11111010, 0b01110000} // 6
    ,{0b00000000, 0b00000110, 0b00000110, 0b11100010, 0b11110010,
0b00011110, 0b00001110, 0b00000110} // 7
    ,{0b00000000, 0b01101100, 0b11111110, 0b10010010, 0b10010010,
0b11111110, 0b11111110, 0b01101100} // 8
    ,{0b00000000, 0b00001100, 0b10011110, 0b10010010, 0b10010010,
0b11111110, 0b11111110, 0b01111100} // 9
    ,{0b00000000, 0b10000000, 0b11100000, 0b01100000, 0b00000000,
0b00000000, 0b00000000, 0b00000000} // ,                        39
    ,{0b00000000, 0b11000000, 0b11000000, 0b00000000, 0b00000000,
0b00000000, 0b00000000, 0b00000000} // .
    ,{0b00000000, 0b10000000, 0b11101100, 0b01101100, 0b00000000,
0b00000000, 0b00000000, 0b00000000} // ;
    ,{0b00000000, 0b11011000, 0b11011000, 0b00000000, 0b00000000,
0b00000000, 0b00000000, 0b00000000} // :
    ,{0b00000000, 0b00001100, 0b00001110, 0b10100010, 0b10110010,
0b00010010, 0b00011110, 0b00001100} // ?
    ,{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b10111110, 0b10111110, 0b00011110} // !                        44
```

```
  ,{0b00000000, 0b00000000, 0b00011000, 0b00011000, 0b00011000,
0b00011000, 0b00011000, 0b00000000} // -
};

uint8_t tile_set[][8]  = {
     {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000, 0b00000000, 0b00000000} // helt tom
  0
     ,{0b00011000, 0b00011000, 0b00011000, 0b00011000, 0b00011000,
0b00011000, 0b00011000, 0b00011000} // rak horisontell
  1
     ,{0b00000000, 0b00000000, 0b00000000, 0b11111111, 0b11111111,
0b00000000, 0b00000000, 0b00000000} // rak vertikal
  2
     ,{0b00011000, 0b00011000, 0b00011000, 0b11111111, 0b11111111,
0b00011000, 0b00011000, 0b00011000} // "plus"
     3
     ,{0b00011000, 0b00011000, 0b00011000, 0b00011111, 0b00011111,
0b00000000, 0b00000000, 0b00000000} // hörn upp-vänster
  4
     ,{0b00000000, 0b00000000, 0b00000000, 0b00011111, 0b00011111,
0b00011000, 0b00011000, 0b00011000} // hörn upp-höger
  5
     ,{0b00011000, 0b00011000, 0b00011000, 0b11111000, 0b11111000,
0b00000000, 0b00000000, 0b00000000} // hörn ner-vänster
  6
     ,{0b00000000, 0b00000000, 0b00000000, 0b11111000, 0b11111000,
0b00011000, 0b00011000, 0b00011000} // hörn ner-höger
  7
     ,{0b00011000, 0b00011000, 0b00011000, 0b11111000, 0b11111000,
0b00011000, 0b00011000, 0b00011000} // "T"
  8
     ,{0b00011000, 0b00011000, 0b00011000, 0b00011111, 0b00011111,
0b00011000, 0b00011000, 0b00011000} // "T" uppifrån
  9
     ,{0b00011000, 0b00011000, 0b00011000, 0b11111111, 0b11111111,
0b00000000, 0b00000000, 0b00000000} // "T" från vänster
  10
     ,{0b00000000, 0b00000000, 0b00000000, 0b11111111, 0b11111111,
0b00011000, 0b00011000, 0b00011000} // "T" från höger
  11
     ,{0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111} // helt fylld
     12
     ,{0b11001100, 0b11001100, 0b00110011, 0b00110011, 0b11001100,
0b11001100, 0b00110011, 0b00110011} // mållinje
  13
 };
```

```
#endif /* CHARSET_HEADER_H_ */
```

B2 game_logic.h

```
/*
 * game_logic.h
 *
 * Created: 2022-05-11 10:43:40
 *  Author: cid12bel
 */
#include <avr/io.h>

#ifndef GAME_LOGIC_H_
#define GAME_LOGIC_H_



/* Screen width max values for x and y */
#define    MAX_X 63
#define MAX_Y   127

//#define BALL_POSITIONS { 0, 0, 1, 3 }
uint8_t    BALL_POSITIONS[] = { 0b00000011, 0b00000110, 0b00001100,
0b00011000, 0b00110000, 0b01100000, 0b11000000, 0b10000000};

/*******************************************
 * this struct contains the ball used in the
 * labyrinth. The ball is 2x2 pixels in size.
 * When collision detecting you should check
 * posx + 1 when v_x is positive and posy + 1
 * when v_y is positive. For negative speeds
 * check posx and posy directly.
 *
 * posx - the X-position of the ball: 0 -- 63
 * posy - the Y-position of the ball: 0 -- 127
 * v_x - the ball's velocity in pixels direction x: -128 -- 127
 * v_y - the ball's velocity in pixels direction y: -128 -- 127
 *  */
typedef struct
{
  int8_t posx;
  int8_t posy;
  int8_t v_x;
  int8_t v_y;
  int8_t prevx;
  int8_t prevy;
} ball;
```

```
/* parameters helpful to keep track of the ball,
   manly for drawing */
int8_t curr_x = 0;
int8_t curr_y = 0;
int8_t curr_page = 0;
int8_t curr_tile_x = 0;
int8_t curr_tile_y = 0;
uint8_t ball_prev_x = 0;
uint8_t ball_prev_y = 0;

/* to keep score: game time and best time
 * 2^32 * 16us should be enough... */

uint32_t GAME_TIME = 0;
uint32_t BEST_TIME = 0;

typedef enum MODES
{
  GAME_START,
  GAME_RUN,
  GAME_END
} GAME_MODE;

/* just sets up the scaling for timer 1 . 256 is used.
 * roughly 1 second should be enough for the game loop
 * and then some.*/
void timer_setup();

/* ADC Initialization function */
void ADC_Init();

/*************************************************
 * Read a value from an input pin and do
 * A/D conversion
 *
 * @param channel - the number of the input you want to read
 *
 * @return  - the converted value as an int
 *************************************************/
int ADC_Read(char channel);

/*************************************************
 * Use this to get a horizontal ball acceleration
 * value from the accelerometer
 *
 *
 * @return  - acceleration in g
 *************************************************/
```

```
double getXValues();

/***********************************************
 * Use this to get a vertical ball acceleration
 * value from the accelerometer
 *
 *
 * @return  - acceleration in g
 ***********************************************/
double getYValues();

/*****************************************************
 * Use this to convert 16 us ticks to seconds
 *
 * @param time_ticks - the number of 16 us ticks you want to
convert
 *
 * @return - time in seconds
 ************************************/
double get_time_s(uint32_t time_ticks);

/*********************************************************
 * Convert a time in seconds into whole minutes.
 * NOTE: no more than 256 minutes.
 *
 * @param t_s - double time in seconds. Do not input more
 *              than 256 minutes if you want it to work.
 *
 * @return - time in whole minutes
*********************************************************/
uint8_t get_minutes(double t_s);

/*********************************************************
 * Convert a time in seconds into whole seconds.
 *
 * @param t_s - double time in seconds.
 *
 *
 * @return - input time modulo 60 to get the seconds
 *           part of the time
*********************************************************/
uint8_t get_seconds(double t_s);

/**
 * Convert a time in seconds into tenths.
 *
 * @param t_s - double time in seconds.
 *
```

```
 * @return - time tenths.
**********************/
uint8_t get_tenths(double t_s);

/**
 * Convert a time in seconds into hundredths.
 *
 * @param t_s - double time in seconds.
 *
 * @return - time hundredths. Note that >= 0.005 always gets
             rounded up.
**********************/
uint8_t get_hundredths(double t_s);

#endif /* GAME_LOGIC_H_ */
```

B2.1 game_logic.c

```
/*
 * game_logic.c
 *
 * Created: 2022-05-18 16:51:19
 *  Author: cid12bel
 */

#include <avr/io.h>
#include <util/delay.h>

/* values from the a/d converter */
double ADC_X_VALUE = 0;
double ADC_Y_VALUE = 0;
double TIME_TICK = 0.000064;

void timer_setup()
{
  TCCR1B = (1<<CS10) | (1<<CS11); /* scaler set to 64. 1 tick is
0.000064 s @ FCPU 1MHz. */
  TCNT1 = 0; /* reset counter */
}

void ADC_Init()            /* ADC Initialization function */
{
  DDRA = 0x00;        /* Make ADC port as input */
  ADCSRA = 0x87;           /* Enable ADC, with freq/128  */
  ADMUX = 0x40;       /* Vref: Avcc, ADC channel: 0 */
}

int ADC_Read(char channel) /* ADC Read function */
```

```
{
  ADMUX = 0x40 | (channel & 0x07);/* set input channel to read */
  ADCSRA |= (1<<ADSC);     /* Start ADC conversion */
  while (!(ADCSRA & (1<<ADIF)));     /* Wait until end of
conversion by polling ADC interrupt flag */
  ADCSRA |= (1<<ADIF);     /* Clear interrupt flag */
  _delay_ms(1);       /* Wait a little bit */
  return ADCW;        /* Return ADC word */
}

double getXValues(){

  double Axout;

  ADC_X_VALUE = ADC_Read(2);    /* Read X, Y axis ADC value */
  // Y  min-385 mid -500 max - 610

  /* Convert values in g unit */
  return Axout = (((double)((((ADC_X_VALUE-385.0)/225.0))*2))-1);
}

double getYValues(){

  double Ayout;

  /* Read X, Y axis ADC value */
  ADC_Y_VALUE = ADC_Read(1);    // Y  min-385 mid -500 max - 610

  /* Convert values in g unit */
  return Ayout = (((double)((((ADC_Y_VALUE-385.0)/225.0))*2))-1);

}

double get_time_s(uint32_t time_ticks)
{
  return time_ticks * TIME_TICK;
}

uint8_t get_minutes(double t_s)
{
  uint8_t temp;
  temp = (uint8_t) t_s/60;
  return temp;
}

uint8_t get_seconds(double t_s)
{
  uint8_t temp = (uint8_t)t_s;
```

```
  return temp % 60;
}

uint8_t get_tenths(double t_s)
{
  uint16_t temp_s = (int16_t)(100 * t_s);
  return temp_s % 10;
}

uint8_t get_hundredths(double t_s)
{
  uint16_t temp_s = 100*t_s / 1;
  if (t_s*10 - temp_s >= 0.5)
  {
    return (temp_s + 1) % 10;
  }
  else
  {
    return temp_s % 10;
  }
}
```

B4 LCD_display.h

```
/*
 * IncFile1.h
 *
 * Created: 2022-05-05 10:50:18
 *  Author: cid12bel
 */
#include <avr/io.h>


#ifndef LCD_display_H_
#define LCD_display_H_

/*************************************
* According to our blueprint IO-ports PB
* handles the data communication and PD
* handles the commands to the display.
*************************************/
#define COMMAND_PORT PORTD
#define COMMAND_DIRECTION DDRD
#define DATA_PORT PORTB
#define DATA_DIRECTION DDRB

/* Some control commands */

/* Reset is on pin 16 -> PORTD2*/
#define LCD_RST PORTD2
```

```
/* Read/Write is on pin 17 -> PORTD3 */
#define LCD_RW PORTD3

/* Shift register for internal data i/o is on port 18.
*  maybe rename to DI instead, since that's what's it's
*  called in the data sheet..
*  Set LCD_RS to high when you want to send commands
*  to the display. Set to low when you want to send
*  data (y'know to, like, draw stuff on the screen)..
*/
#define LCD_RS PORTD4

/* LCD enable is on pin 19 -> PORTD5 */
#define LCD_EN PORTD5

/* CS1 part of the screen is pin 15 -> PORTD1
*  and CS2 part is pin 14 -> PORTD0 */
#define CS1 PORTD1
#define CS2 PORTD0

/* The display has 8 different pages that can be displayed.
*  Usually you need to know which one you want to display. */
#define NBR_PAGES 8u


#define DISPLAY_OFF 0b00111110
#define DISPLAY_ON 0b00111111

/* Initializes the display */
void init_display();

/* Makes sure the screen is blank before we do anything and that
 * there is no garbage left in any of the 8 pages. */
void clear_display();

/* Use this function to send commands to the LCD-display. Shamelessly
"inspired" by a code example @
https://www.electronicwings.com/avr-atmega/graphical-lcd-128x64-interfacing-w
ith-atmega1632 */
void LCD_command(uint8_t command);

/* Use this function to send data to be drawn to the LCD. "Inspiration" as
above..
 * AFAIK the data represents an entire display column. Keep values between
 * 0 and 63 (max 0b0011111), since we only have 64 rows. */
void LCD_data(uint8_t data);

/* Use this function to get a the value that the display needs to
 * be able to set a new y-position
 *
 * @return LCD command value to be sent over the data pins. */
uint8_t disp_ypos(uint8_t new_pos);
```

```
/* Use this function to get a the value that the display needs to
 * be able to set a new render page (called "x")
 *
 * @return LCD command value to be sent over the data pins.*/
uint8_t disp_page(uint8_t new_line);

/* Use this function to get a the value that the display needs to
 * be able to set a new start line (called "z")
 *
 * @return LCD command value to be sent over the data pins. */
uint8_t disp_start_line(uint8_t new_pos);

/* Use this function to enable drawing on CS1 half of the LCD */
void set_CS1();

/* Use this function to enable drawing on CS2 half of the LCD */
void set_CS2();

#endif /* INCFILE1_H_ */
```

B4.1 LCD_display.c

```
/*
 * LCD_Test.c
 *
 * Created: 2022-05-05 10:43:03
 * Author : cid12bel
 */

#define F_CPU 16000000u

#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>
#include "LCD_display.h"

void init_display()
{
 DATA_DIRECTION = 0xFF;
 COMMAND_DIRECTION = 0xFF; /*set all to high*/

 /* Select both halves of the screen and set reset flag to high.
  * Note that all this happens simultaneously on both cs1 and cs2. */
 COMMAND_PORT |= (1 << CS1) | (1 << CS2) | (1 << LCD_RST);

 _delay_us(20); /* from code example. 20 ms should be safe */

 LCD_command(DISPLAY_OFF);
 LCD_command(disp_ypos(0));
 LCD_command(disp_page(0));
 LCD_command(disp_start_line(0));
 LCD_command(DISPLAY_ON);
```

```
}

void clear_display()
{
 int i, j;

 /* Write to both halves of the screen at the same time */
 COMMAND_PORT |= (1 << CS1) | (1 << CS2);

 for (i = 0; i < NBR_PAGES; ++i)
 {
     LCD_command(disp_page(i)); /* each page needs to be cleared */

     for (j = 0; j < 64; ++j)
     {
      /* Write zeros to each column. Note that the column
      value increments automatically each time data is
      written to the LCD display. That's why we don't have
      to set a new start line for each iteration of the loop.*/
      LCD_data(0);
     }
 }
 LCD_command(disp_ypos(0)); /* Make sure the display will start drawing at
pos 0 again */
 LCD_command(disp_page(0)); /* Start drawing at page 0 again */

}

void LCD_command(uint8_t command)
{
 DATA_PORT = command; /* Send the command you want to execute to the data
pins */
 COMMAND_PORT &= ~(1 << LCD_RS);  /* set shift register to low to send
commands*/
 COMMAND_PORT &= ~(1 << LCD_RW); /* set RW low to enable write */
 COMMAND_PORT |= (1 << LCD_EN); /* set enable to high */
 _delay_us(5); /* allow some time to pass. 5us in code example. Hopefully
enough..*/
 COMMAND_PORT &= ~(1 << LCD_EN); /* set enable to low, to stop sending
commands */
 _delay_us(5); /* wait, see above */
}

void LCD_data(uint8_t data)
{
 DATA_PORT = data;
 COMMAND_PORT |= (1 << LCD_RS); /* set shift register high to send data */
 COMMAND_PORT &= ~(1 << LCD_RW); /* set RW low to enable write */
 COMMAND_PORT |= (1 << LCD_EN); /* set enable to high */
 _delay_us(5); /* allow some time to pass. 5us in code example. Hopefully
enough..*/
 COMMAND_PORT &= ~(1 << LCD_EN); /* set enable to low, to stop sending
commands */
```

```
 _delay_us(5); /* wait, see above */
}

uint8_t disp_ypos(uint8_t new_pos)
{
 return 0b01000000 + new_pos;
}

uint8_t disp_page(uint8_t new_page)
{
 return 0b10111000 + new_page;
}

uint8_t disp_start_line(uint8_t new_line)
{
 return 0b11000000 + new_line;
}


void set_CS1()
{
 COMMAND_PORT |= (1 << CS1);
 COMMAND_PORT &= ~(1 << CS2);
}

void set_CS2()
{
 COMMAND_PORT |= (1 << CS2);
 COMMAND_PORT &= ~(1 << CS1);
}
```

B5 main.c

```
/*
 * LCD_Test.c
 *
 * Created: 2022-05-05 10:43:03
 * Author : cid12bel
 */

#define F_CPU 16000000u

#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>
#include <stdlib.h>
#include "LCD_display.h"
#include "game_logic.h"
#include "charset_header.h"
#include <stdio.h>


#include <avr/interrupt.h>
```

```
/* the ball used in the game */
ball a_ball;



/* iterator parameters */
uint8_t i = 0;
uint8_t j = 0;
uint8_t k = 0;

/* the representation of the maze */
tile maze[16][8];

/* a byte to be filled with pixel data for drawing
   one 8 pixel column on a display page */
uint8_t temp_draw = 0;


uint8_t detect_collision(uint8_t move_x, uint8_t move_y)
{
 uint8_t hit = 0;

 /* save the old position so we can erase the old image if the ball moves */
 a_ball.prevx = a_ball.posx;
 a_ball.prevy = a_ball.posy;

 while (move_x > 0 || move_y > 0)
 {
  if (a_ball.v_y > 0 && move_y > 0)
  {
   /* right side of ball is front */

   if (a_ball.posy > 125)
   {
    a_ball.v_y = 0;
    hit = 1;
    move_y = 0;
   }
   else
   {
    ++a_ball.posy;
    curr_tile_y = (a_ball.posy + 1) / 8;
    curr_y = (a_ball.posy + 1) % 8;
    curr_page = a_ball.posx / 8;
    curr_x = a_ball.posx % 8;

    if (maze[curr_tile_y][curr_page].has_wall > 0)
     hit = tile_set[maze[curr_tile_y][curr_page].char_id][curr_y] &
BALL_POSITIONS[curr_x];
    if (curr_x == 7 && hit < 1 && maze[curr_tile_y][curr_page + 1].has_wall
> 0)
```

```
      hit = tile_set[maze[curr_tile_y][curr_page + 1].char_id][curr_y] &
0b00000001;

     if (hit > 0)
     {
      a_ball.v_y = 0;
      --a_ball.posy;
      move_y = 0;
     }
     else --move_y;
    }
   }
   else if (a_ball.v_y < 0 && move_y > 0)
   {

    if (a_ball.posy == 0)
    {
     a_ball.v_y = 0;
     hit = 1;
     move_y = 0;
    }
    else
    {
     --a_ball.posy;
     curr_tile_y = (a_ball.posy) / 8;
     curr_y = (a_ball.posy) % 8;
     curr_page = a_ball.posx / 8;
     curr_x = a_ball.posx % 8;

     if(maze[curr_tile_y][curr_page].has_wall > 0)
      hit = tile_set[maze[curr_tile_y][curr_page].char_id][curr_y] &
BALL_POSITIONS[curr_x];
     if (curr_x == 7 && hit < 1 && maze[curr_tile_y][curr_page + 1].has_wall
> 0 )
     {
      hit = tile_set[maze[curr_tile_y][curr_page + 1].char_id][curr_y] &
BALL_POSITIONS[0b00000001];
     }

     if (hit > 0)
     {
      a_ball.v_y = 0;
      ++a_ball.posy;
      move_y = 0;
     }
     else --move_y;
    }
   }
   if (a_ball.v_x > 0 && move_x > 0)
   {
    if(a_ball.posx == 62)
    {
     a_ball.v_x = 0;
```

```
     hit = 1;
     move_x = 0;
    }
    else
    {
     ++a_ball.posx;
     curr_tile_y = (a_ball.posy) / 8;
     curr_y = (a_ball.posy) % 8;
     curr_page = (a_ball.posx + 1) / 8;
     curr_x = (a_ball.posx + 1) % 8;

     if (curr_x == 0)
     {
      if (maze[curr_tile_y][curr_page].has_wall > 0)
       hit = tile_set[maze[curr_tile_y][curr_page].char_id][curr_y] &
0b00000001;
      if(curr_y == 7 && hit < 1 && maze[curr_tile_y + 1][curr_page].has_wall
> 0)
      {
       hit = tile_set[maze[curr_tile_y + 1][curr_page].char_id][0] &
0b00000001;
      }
      else if (maze[curr_tile_y][curr_page].has_wall > 0 && hit < 1)
      {
       hit = tile_set[maze[curr_tile_y][curr_page].char_id][curr_y + 1] &
0b00000001;
      }
     }
     else
     {
      if (maze[curr_tile_y][curr_page].has_wall > 0)
       hit = tile_set[maze[curr_tile_y][curr_page].char_id][curr_y] &
BALL_POSITIONS[curr_x - 1];
      if(curr_y == 7 && hit < 1 && maze[curr_tile_y + 1][curr_page].has_wall
> 0)
      {
       hit = tile_set[maze[curr_tile_y + 1][curr_page].char_id][0] &
BALL_POSITIONS[curr_x - 1];
      }
      else if (maze[curr_tile_y][curr_page].has_wall > 0 && hit < 1)
      {
       hit = tile_set[maze[curr_tile_y][curr_page].char_id][curr_y + 1] &
BALL_POSITIONS[curr_x - 1];
      }
     }
     if (hit > 0)
     {
      a_ball.v_x = 0;
      --a_ball.posx;
      move_x = 0;
     }
     else --move_x;
```

```
    }
   }
   else if (a_ball.v_x < 0 && move_x > 0)
   {
    if (a_ball.posx == 0)
    {
     a_ball.v_x = 0;
     hit = 1;
     move_x = 0;
    }
    else
    {
     --a_ball.posx;
     curr_tile_y = (a_ball.posy) / 8;
     curr_y = (a_ball.posy) % 8;
     curr_page = a_ball.posx / 8;
     curr_x = a_ball.posx % 8;

     if (maze[curr_tile_y][curr_page].has_wall > 0)
      hit = tile_set[maze[curr_tile_y][curr_page].char_id][curr_y] &
BALL_POSITIONS[curr_x];
     if (curr_y == 7 && hit < 1 && maze[curr_tile_y + 1][curr_page].has_wall
> 0)
     {
      hit = tile_set[maze[curr_tile_y + 1][curr_page].char_id][0] &
BALL_POSITIONS[curr_x];
     }
     else if (maze[curr_tile_y][curr_page].has_wall > 0 && hit < 1)
     {
      hit = tile_set[maze[curr_tile_y][curr_page].char_id][curr_y + 1] &
BALL_POSITIONS[curr_x];
     }
     if (hit > 0)
     {
      a_ball.v_x = 0;
      ++a_ball.posx;
      move_x = 0;
     }
     else --move_x;
    }
   }
  }
  return hit;
}

void draw_ball()
{
 /** TRY TO MOVE THE BALL INCLUDING COLLISION DETECTION **/

 detect_collision(abs(a_ball.v_x), abs(a_ball.v_y));

 /*********************************************************
  * First, erase the ball's previous position on the screen *
```

```
  ***********************************************************/
 curr_page = a_ball.prevx / NBR_PAGES; /* To know what page we are on */
 curr_x = a_ball.prevx % 8; /* Each page is 8 px high. */
 curr_y = a_ball.prevy % 64;
 curr_tile_y = a_ball.prevy / 8; /* integer division by 8 gives the tile's
horizontal position */

 /* set which part of the screen to draw on */
 if (curr_tile_y < 8)
 {
  set_CS2(); /* left side */
 }
 else
 {
  set_CS1(); /* right side */
 }

 LCD_command(disp_ypos(curr_y));
 LCD_command(disp_page(curr_page));

 /*** B A L L   L E F T   H A L F  ***/

 /* slightly complicated.. the position in the maze contains a reference to
the 8x8 image that
     corresponds to what should be drawn on the current tile. curr_x gives
which of the 8 columns
   that should be drawn. maze[curr_tile_y][curr_tile_x].char_id gives the int
value for
   the character index row */
 LCD_data(tile_set[maze[curr_tile_y][curr_page].char_id][curr_y % 8]);

 if (curr_x == 7 && curr_page < NBR_PAGES - 1 ) /* only on pages 1--6 */
 {
  /* In case we are at the bottom border between pages,
     except the bottom page. */
  LCD_command(disp_ypos(curr_y)); /* reset y */
  LCD_command(disp_page(curr_page + 1)); /* move 1 page down */
  LCD_data(tile_set[maze[curr_tile_y][curr_page + 1].char_id][curr_y % 8]);
  LCD_command(disp_page(curr_page)); /* we need to reset page position again
*/
 }

/*** B A L L   R I G H T   H A L F  ***/
 if ((a_ball.prevy % 8) == 7 && curr_tile_y < 15)
 {
  if (a_ball.prevy == 63)
  {
   set_CS1();
   LCD_command(disp_ypos(0));
  }
  else
  {
   LCD_command(disp_ypos(curr_y + 1));
```

```
   }
   LCD_data(tile_set[maze[curr_tile_y + 1][curr_page].char_id][0]);
   if (curr_x == 7 && curr_page < NBR_PAGES - 1 ) /* only on pages 1--6 */
   {
   /* In case we are at the bottom border between pages,
      except the bottom page. */
    if (a_ball.posy == 63)
    {
     LCD_command(disp_ypos(0));
    }
    else
    {
     LCD_command(disp_ypos(curr_y + 1));
    }
    LCD_command(disp_page(curr_page + 1)); /* move 1 page down */
    LCD_data(tile_set[maze[curr_tile_y + 1][curr_page + 1].char_id][0]);
    LCD_command(disp_page(curr_page)); /* we need to reset page position
again */
   }
   /* make sure we draw on the left half again when moving the other way */
   if (a_ball.prevy == 63/* && a_ball.v_y < 0 */)
   {
    set_CS2();
   }

 }
 else
 {
  LCD_data(tile_set[maze[curr_tile_y][curr_page].char_id][(curr_y+1) % 8]);
  if (curr_x == 7 && curr_page < NBR_PAGES - 1 ) /* only on pages 1--6 */
  {
  /* In case we are at the bottom border between pages,
   except the bottom page. */
   LCD_command(disp_ypos(curr_y + 1)); /* reset y */
   LCD_command(disp_page(curr_page + 1)); /* move 1 page down */
   LCD_data(tile_set[maze[curr_tile_y][curr_page + 1].char_id][(curr_y + 1)
% 8]);
   LCD_command(disp_page(curr_page)); /* we need to reset page position
again */
  }
 }




 curr_page = a_ball.posx / 8;
 curr_x = a_ball.posx % 8;
 curr_y = a_ball.posy % 64;
 curr_tile_y = a_ball.posy / 8; /* integer division by 8 gives the tile's
horizontal position */

 if (a_ball.posy < 64)
 {
```

```
  set_CS2(); /* left side */
 }
 else
 {
  set_CS1(); /* right side */
 }

 /*****************************************************************
  *    Time to draw the ball                                      *
  *****************************************************************/


 /*** B A L L   L E F T   H A L F  ***/
 LCD_command(disp_ypos(curr_y));
 LCD_command(disp_page(curr_page));


 /*** B A L L   L E F T   H A L F  ***/

 /* slightly complicated.. the position in the maze contains a reference to
the 8x8 image that
     corresponds to what should be drawn on the current tile. curr_x gives
which of the 8 columns
   that should be drawn. maze[curr_tile_y][curr_tile_x].char_id gives the int
value for
   the character index row */
 LCD_data(tile_set[maze[curr_tile_y][curr_page].char_id][curr_y % 8] +
BALL_POSITIONS[curr_x]);

 if (curr_x == 7 && curr_page < NBR_PAGES - 1 ) /* only on pages 1--6 */
 {
  /* In case we are at the bottom border between pages,
     except the bottom page. */
  LCD_command(disp_ypos(curr_y)); /* reset y */
  LCD_command(disp_page(curr_page + 1)); /* move 1 page down */
  LCD_data(tile_set[maze[curr_tile_y][curr_page + 1].char_id][curr_y % 8] +
1);
  LCD_command(disp_page(curr_page)); /* we need to reset page position again
*/
 }

    /*** B A L L   R I G H T   H A L F  ***/
 if ((a_ball.posy % 8) == 7 && curr_tile_y < 15)
 {
  if (a_ball.posy == 63)
  {
   set_CS1();
   LCD_command(disp_ypos(0));
  }
  else
  {
   LCD_command(disp_ypos(curr_y + 1));
  }
```

```
   LCD_data(tile_set[maze[curr_tile_y + 1][curr_page].char_id][0] +
BALL_POSITIONS[curr_x]);
   if (curr_x == 7 && curr_page < NBR_PAGES - 1 ) /* only on pages 1--6 */
   {
   /* In case we are at the bottom border between pages,
      except the bottom page. */
    if (a_ball.posy == 63)
    {
     LCD_command(disp_ypos(0));
    }
    else
    {
     LCD_command(disp_ypos(curr_y + 1));
    }
    LCD_command(disp_page(curr_page + 1)); /* move 1 page down */
    LCD_data(tile_set[maze[curr_tile_y + 1][curr_page + 1].char_id][0] + 1);
    LCD_command(disp_page(curr_page)); /* we need to reset page position
again */
   }
   /* make sure we draw on the left half again when moving the other way */
   if (a_ball.posy == 63 && a_ball.v_y < 0)
   {
    set_CS2();
   }

 }
 else
 {
   LCD_data(tile_set[maze[curr_tile_y][curr_page].char_id][(curr_y+1) % 8] +
BALL_POSITIONS[curr_x]);
   if (curr_x == 7 && curr_page < NBR_PAGES - 1 ) /* only on pages 1--6 */
   {
   /* In case we are at the bottom border between pages,
    except the bottom page. */
    LCD_command(disp_ypos(curr_y + 1)); /* reset y */
    LCD_command(disp_page(curr_page + 1)); /* move 1 page down */
    LCD_data(tile_set[maze[curr_tile_y][curr_page + 1].char_id][(curr_y + 1)
% 8] + 1);
    LCD_command(disp_page(curr_page)); /* we need to reset page position
again */
   }
 }
}

void setup_labyrinth()
{
 uint8_t left = 1;
 int col = 0;
 int row = 0;
 int k = 0;
 int l = 0;
 LCD_command(disp_page(row)); /* replace with proper page calculation !! */
 LCD_command(disp_ypos(0));
```

```
 set_CS2(); /* this also needs to be calculated */
 for (k = 0; k < CHARSET_SIZE; ++k)
 {
  LCD_command(disp_ypos(col*8));
  for (l = 0; l < 8; ++l)
  {
   LCD_data(charset[k][l]);
  }
  ++col;
  if (col == 8 && left > 0)
  {
   set_CS1();
   left = 0;
   col = 0;
   LCD_command(disp_page(row));
  }

  else if (col == 8 && left < 1)
  {
   set_CS2();
   left = 1;
   col = 0;
   ++row;
   LCD_command(disp_page(row));
  }
 }
}

void msg_game_over()
{
 set_CS2();
 LCD_command(disp_page(3));
 LCD_command(disp_ypos(24));

 /* Left half; write GAME */
 /* G */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[6][i]);
 }
 /* A */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[0][i]);
 }
 /* M */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[12][i]);
 }
 /* E */
 for (i = 0; i < 8; ++i)
 {
```

```
   LCD_data(charset[4][i]);
  }
  for (i = 0; i < 8; ++i)
  {
   LCD_data(0);
  }

  /* Right half; write OVER */
  set_CS1();
  LCD_command(disp_page(3)); /* Note that both halves needs to be set to the
correct page */
  LCD_command(disp_ypos(0)); /* Leave 1 tile for space */

  for (i = 0; i < 8; ++i)
  {
   LCD_data(0);
  }
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[14][i]);
  }
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[21][i]);
  }
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[4][i]);
  }
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[17][i]);
  }
}

void make_maze()
{
 maze[0][0].char_id = 0;
 maze[0][1].char_id = 0;
 maze[0][2].char_id = 0;
 maze[0][3].char_id = 0;
 maze[0][4].char_id = 0;
 maze[0][5].char_id = 0;
 maze[0][6].char_id = 1;
 maze[0][6].has_wall = 1;
 maze[0][7].char_id = 0;

 maze[1][0].char_id = 0;
 maze[1][1].char_id = 7;
 maze[1][1].has_wall = 1;
 maze[1][2].char_id = 0;
 maze[1][3].char_id = 11;
 maze[1][3].has_wall = 1;
```

```
maze[1][4].char_id = 0;
maze[1][5].char_id = 0;
maze[1][6].char_id = 1;
maze[1][6].has_wall = 1;
maze[1][7].char_id = 0;

maze[2][0].char_id = 0;
maze[2][1].char_id = 1;
maze[2][1].has_wall = 1;
maze[2][2].char_id = 0;
maze[2][3].char_id = 1;
maze[2][3].has_wall = 1;
maze[2][4].char_id = 0;
maze[2][5].char_id = 2;
maze[2][5].has_wall = 1;
maze[2][6].char_id = 9;
maze[2][6].has_wall = 1;
maze[2][7].char_id = 0;

maze[3][0].char_id = 0;
maze[3][1].char_id = 1;
maze[3][1].has_wall = 1;
maze[3][2].char_id = 0;
maze[3][3].char_id = 1;
maze[3][3].has_wall = 1;
maze[3][4].char_id = 0;
maze[3][5].char_id = 2;
maze[3][5].has_wall = 1;
maze[3][6].char_id = 9;
maze[3][6].has_wall = 1;
maze[3][7].char_id = 0;

maze[4][0].char_id = 0;
maze[4][1].char_id = 1;
maze[4][1].has_wall = 1;
maze[4][2].char_id = 0;
maze[4][3].char_id = 1;
maze[4][3].has_wall = 1;
maze[4][4].char_id = 0;
maze[4][5].char_id = 0;
maze[4][6].char_id = 0;
maze[4][7].char_id = 0;

maze[5][0].char_id = 0;
maze[5][0].is_start =1;
maze[5][1].char_id = 1;
maze[5][1].has_wall = 1;
maze[5][2].char_id = 0;
maze[5][3].char_id = 6;
maze[5][3].has_wall = 1;
maze[5][4].char_id = 11;
maze[5][4].has_wall = 1;
maze[5][5].char_id = 2;
```

```
maze[5][5].has_wall = 1;
maze[5][6].char_id = 2;
maze[5][6].has_wall = 1;
maze[5][7].char_id = 0;

maze[6][0].char_id = 2;
maze[6][0].has_wall = 1;
maze[6][1].char_id = 10;
maze[6][1].has_wall = 1;
maze[6][2].char_id = 5;
maze[6][2].has_wall = 1;
maze[6][3].char_id = 0;
maze[6][4].char_id = 1;
maze[6][4].has_wall = 1;
maze[6][5].char_id = 0;
maze[6][6].char_id = 0;
maze[6][7].char_id = 0;

maze[7][0].char_id = 0;
maze[7][1].char_id = 0;
maze[7][2].char_id = 6;
maze[7][2].has_wall = 1;
maze[7][3].char_id = 2;
maze[7][3].has_wall = 1;
maze[7][4].char_id = 4;
maze[7][4].has_wall = 1;
maze[7][5].char_id = 0;
maze[7][6].char_id = 0;
maze[7][7].char_id = 7;
maze[7][7].has_wall = 1;

maze[8][0].char_id = 0;
maze[8][1].char_id = 7;
maze[8][1].has_wall = 1;
maze[8][2].char_id = 0;
maze[8][3].char_id = 7;
maze[8][3].has_wall = 1;
maze[8][4].char_id = 0;
maze[8][5].char_id = 7;
maze[8][5].has_wall = 1;
maze[8][6].char_id = 2;
maze[8][6].has_wall = 1;
maze[8][7].char_id = 4;
maze[8][7].has_wall = 1;

maze[9][0].char_id = 0;
maze[9][1].char_id = 8;
maze[9][1].has_wall = 1;
maze[9][2].char_id = 2;
maze[9][2].has_wall = 1;
maze[9][3].char_id = 10;
maze[9][3].has_wall = 1;
maze[9][4].char_id = 11;
```

```
maze[9][4].has_wall = 1;
maze[9][5].char_id = 4;
maze[9][5].has_wall = 1;
maze[9][6].char_id = 0;
maze[9][7].char_id = 13;
maze[9][7].finish_line = 1;

maze[10][0].char_id = 5;
maze[10][0].has_wall = 1;
maze[10][1].char_id = 0;
maze[10][2].char_id = 0;
maze[10][3].char_id = 0;
maze[10][4].char_id = 6;
maze[10][4].has_wall = 1;
maze[10][5].char_id = 5;
maze[10][5].has_wall = 1;
maze[10][6].char_id = 0;
maze[10][7].char_id = 5;
maze[10][7].has_wall = 1;

maze[11][0].char_id = 6;
maze[11][0].has_wall = 1;
maze[11][1].char_id = 3;
maze[11][1].has_wall = 1;
maze[11][2].char_id = 3;
maze[11][2].has_wall = 1;
maze[11][3].char_id = 5;
maze[11][3].has_wall = 1;
maze[11][4].char_id = 0;
maze[11][5].char_id = 1;
maze[11][5].has_wall = 1;
maze[11][6].char_id = 0;
maze[11][7].char_id = 1;
maze[11][7].has_wall = 1;

maze[12][0].char_id = 0;
maze[12][1].char_id = 1;
maze[12][1].has_wall = 1;
maze[12][2].char_id = 6;
maze[12][2].has_wall = 1;
maze[12][3].char_id = 1;
maze[12][3].has_wall = 1;
maze[12][4].char_id = 0;
maze[12][5].char_id = 1;
maze[12][5].has_wall = 1;
maze[12][6].char_id = 0;
maze[12][7].char_id = 1;
maze[12][7].has_wall = 1;

maze[13][0].char_id = 0;
maze[13][1].char_id = 0;
maze[13][2].char_id = 0;
maze[13][3].char_id = 0;
```

```
 maze[13][4].char_id = 0;
 maze[13][5].char_id = 1;
 maze[13][5].has_wall = 1;
 maze[13][6].char_id = 0;
 maze[13][7].char_id = 6;
 maze[13][7].has_wall = 1;

 maze[14][0].char_id = 0;
 maze[14][1].char_id = 2;
 maze[14][1].has_wall = 1;
 maze[14][2].char_id = 2;
 maze[14][2].has_wall = 1;
 maze[14][3].char_id = 3;
 maze[14][3].has_wall = 1;
 maze[14][4].char_id = 2;
 maze[14][4].has_wall = 1;
 maze[14][5].char_id = 3;
 maze[14][5].has_wall = 1;
 maze[14][6].char_id = 0;
 maze[14][7].char_id = 0;

 maze[15][0].char_id = 0;
 maze[15][1].char_id = 0;
 maze[15][2].char_id = 0;
 maze[15][3].char_id = 0;
 maze[15][4].char_id = 0;
 maze[15][5].char_id = 0;
 maze[15][6].char_id = 0;
 maze[15][7].char_id = 0;

}

void draw_maze()
{
 for (i = 0; i < 8; ++i)
 {

   for(j = 0; j < 8; ++j)
   {
    /* display right half */
    set_CS1();
    LCD_command(disp_page(i));
    LCD_command(disp_ypos(j*8));
    for (k = 0; k < 8; ++k) LCD_data(tile_set[maze[j + 8][i].char_id][k]);

    /* display left half */
    set_CS2();
    LCD_command(disp_page(i));
    LCD_command(disp_ypos(j*8));
    for (k = 0; k < 8; ++k) LCD_data(tile_set[maze[j][i].char_id][k]);
   }
 }
}
```

```
void msg_well_done(uint8_t a_line)
{
 set_CS2();
 LCD_command(disp_page(a_line));
 LCD_command(disp_ypos(24));

 /* Left half; write WELL */
 /* G */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[22][i]);
 }
 /* E */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[4][i]);
 }
 /* L */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[11][i]);
 }
 /* L */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[11][i]);
 }

 /* ' ' */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(0);
 }

 /* Right half; write DONE  ! */
 set_CS1();
 LCD_command(disp_page(a_line)); /* Note that both halves needs to be set to
the correct page */
 LCD_command(disp_ypos(0)); /* Leave 1 tile for space */
  /* ' ' */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(0);
 }

 /* D */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[3][i]);
 }
 /* O */
 for (i = 0; i < 8; ++i)
```

```
 {
  LCD_data(charset[14][i]);
 }
 /* N */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[13][i]);
 }
 /* E */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[4][i]);
 }

 /* ! */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[44][i]);
 }

}

void msg_get_ready()
{
 set_CS2();
 LCD_command(disp_page(3));
 LCD_command(disp_ypos(24));

 /* Left half; write GET */
 /* G */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[6][i]);
 }
 /* E */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[4][i]);
 }
 /* T */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[19][i]);
 }

 /* ' ' */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(0);
 }
 /* ' ' */
 for (i = 0; i < 8; ++i)
 {
```

```
   LCD_data(0);
  }

  /* Right half; write READY ! */
  set_CS1();
  LCD_command(disp_page(3)); /* Note that both halves needs to be set to the
correct page */
  LCD_command(disp_ypos(0)); /* Leave 1 tile for space */

  /* R */
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[17][i]);
  }
  /* E */
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[4][i]);
  }
  /* A */
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[0][i]);
  }
  /* D */
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[3][i]);
  }
  /* Y */
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[24][i]);
  }
//  /* ' ' */
//  for (i = 0; i < 8; ++i)
//  {
//   LCD_data(0);
//  }
//   /* ' ' */
//  for (i = 0; i < 8; ++i)
//  {
//   LCD_data(0);
//  }
  /* ! */
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[44][i]);
  }
}

void msg_your_time(uint8_t line_nbr)
{
```

```
 set_CS2();
 LCD_command(disp_page(line_nbr));
 LCD_command(disp_ypos(24));

 /* Left half; write YOUR */
 /* Y */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[24][i]);
 }
 /* O */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[14][i]);
 }
 /* U */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[20][i]);
 }

 /* R */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[17][i]);
 }
 /* ' ' */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(0);
 }

 /* Right half; write TIME: */
 set_CS1();
 LCD_command(disp_page(line_nbr)); /* Note that both halves needs to be set
to the correct page */
 LCD_command(disp_ypos(0)); /* Leave 1 tile for space */

 /* ' ' */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(0);
 }
 /* T */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[19][i]);
 }
 /* I */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[8][i]);
 }
```

```
 /* M */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[12][i]);
 }
 /* E */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[4][i]);
 }
 /* : */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[42][i]);
 }
}

void msg_kycklingmiddag(uint8_t line_nbr)
{
 set_CS2();
 LCD_command(disp_page(line_nbr));
 LCD_command(disp_ypos(0));

 /* Left half; write KYCKLING */
 /* K */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[10][i]);
 }
 /* Y */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[24][i]);
 }
 /* C */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[2][i]);
 }
 /* K */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[10][i]);
 }
 /* L */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[11][i]);
 }
 /* I */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[8][i]);
```

```
 }
 /* N */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[13][i]);
 }
 /* G */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[6][i]);
 }

 /* Right half; write MIDDAG!! */
 set_CS1();
 LCD_command(disp_page(line_nbr)); /* Note that both halves needs to be set
to the correct page */
 LCD_command(disp_ypos(0)); /* Leave 1 tile for space */

 /* M */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[12][i]);
 }
 /* I */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[8][i]);
 }
 /* D */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[3][i]);
 }
 /* D */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[3][i]);
 }
 /* A */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[0][i]);
 }
 /* G */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[6][i]);
 }
 /* ! */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[44][i]);
 }
```

```
 /* ! */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[44][i]);
 }
}

void msg_best_time(uint8_t line_nbr)
{
 set_CS2();
 LCD_command(disp_page(line_nbr));
 LCD_command(disp_ypos(24));

 /* Left half; write YOUR */
 /* B */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[1][i]);
 }
 /* E */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[4][i]);
 }
 /* S */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[18][i]);
 }
 /* T */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[19][i]);
 }
 /* ' ' */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(0);
 }

 /* Right half; write TIME: */
 set_CS1();
 LCD_command(disp_page(line_nbr)); /* Note that both halves needs to be set
to the correct page */
 LCD_command(disp_ypos(0)); /* Leave 1 tile for space */

 /* ' ' */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(0);
 }
 /* T */
 for (i = 0; i < 8; ++i)
```

```
 {
  LCD_data(charset[19][i]);
 }
 /* I */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[8][i]);
 }
 /* M */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[12][i]);
 }
 /* E */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[4][i]);
 }
 /* : */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[42][i]);
 }
}

void msg_empty_record(uint8_t line_nbr)
{
 /* Left half */
 set_CS2();
 LCD_command(disp_page(line_nbr));
 LCD_command(disp_ypos(32));
 /* - */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[45][i]);
 }
 /* - */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[45][i]);
 }
 /* : */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[42][i]);
 }
 /* - */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[45][i]);
 }
 /* Right half */
 set_CS1();
```

```
 LCD_command(disp_page(line_nbr)); /* Note that both halves needs to be set
to the correct page */
 LCD_command(disp_ypos(0)); /* Leave 1 tile for space */
 /* - */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[45][i]);
 }
 /* : */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[42][i]);
 }
 /* - */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[45][i]);
 }
 /* - */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[45][i]);
 }
}

void msg_a_time(uint8_t line_nbr, double a_time)
{
 uint8_t tempv = 0;
 /* Left half */
 set_CS2();
 LCD_command(disp_page(line_nbr));
 LCD_command(disp_ypos(32));

 /* minutes */
 tempv = get_minutes(a_time); /* get the time in whole minutes*/
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[29 + tempv / 10][i]); /* 10 minutes */
 }
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[29 + tempv % 10][i]); /* 1 minute */
 }
 /* : */
 for (i = 0; i < 8; ++i)
 {
  LCD_data(charset[42][i]);
 }

 /* seconds */
 tempv = get_seconds(a_time);
 for (i = 0; i < 8; ++i)
 {
```

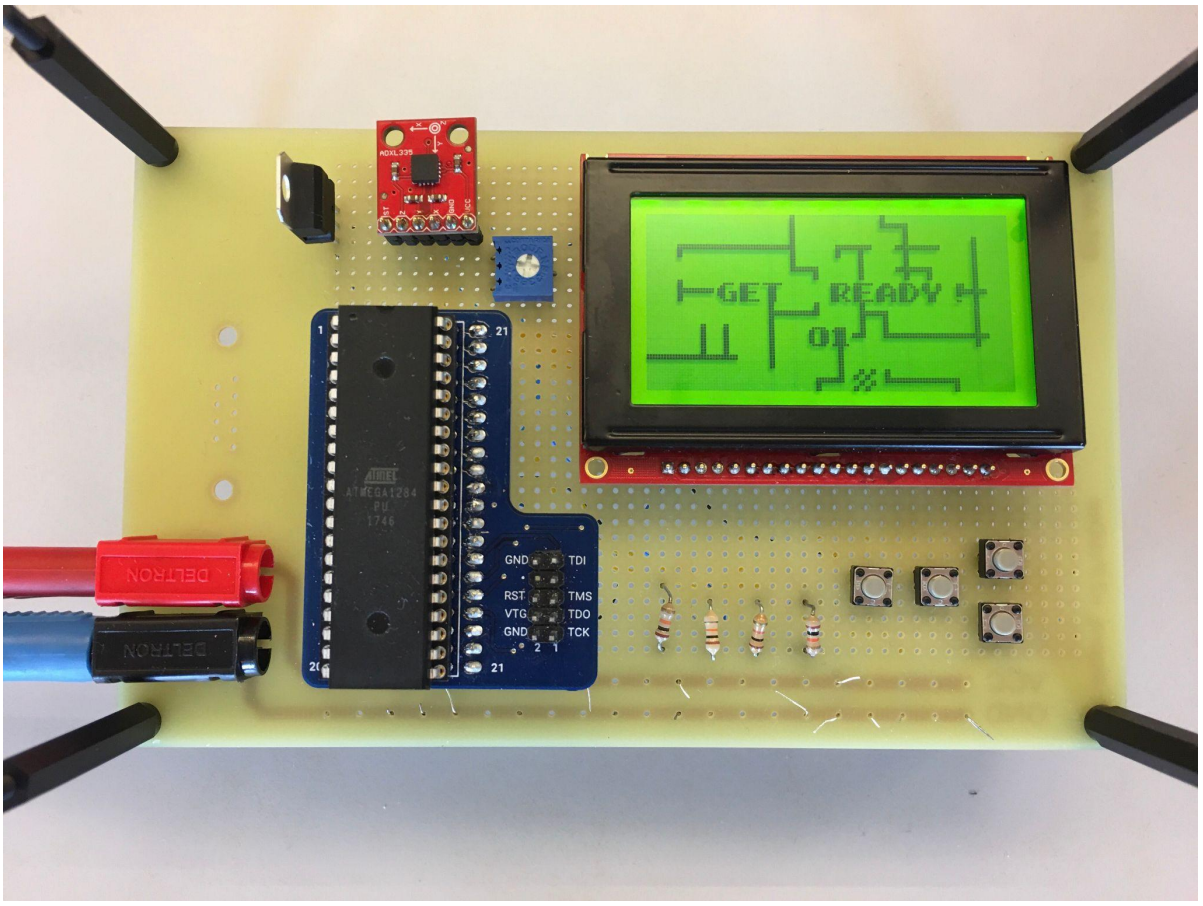
```
   LCD_data(charset[29 + tempv / 10][i]); /* 10s of seconds*/
  }
  /* Right half */
  set_CS1();
  LCD_command(disp_page(line_nbr)); /* Note that both halves needs to be set
to the correct page */
  LCD_command(disp_ypos(0));
  /* seconds continued */
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[29 + tempv % 10][i]);
  }
  /* : */
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[42][i]);
  }
  /* tenths */
  tempv = get_tenths(a_time);
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[29 + tempv][i]);
  }
  /* maybe implement a get_hundredth ... */
  tempv = get_hundredths(a_time);
  for (i = 0; i < 8; ++i)
  {
   LCD_data(charset[29 + tempv][i]);
  }
}

void do_countdown()
{
  for (j = 5; j > 0; --j)
  {
     /* left half */
     set_CS2();
     LCD_command(disp_page(5)); /* Note that both halves needs to be set to
the correct page */
     LCD_command(disp_ypos(56));
     for (i = 0; i < 8; ++i)
     {
      LCD_data(charset[29][i]);
     }
     /* Right half */
     set_CS1();
     LCD_command(disp_page(5)); /* Note that both halves needs to be set to
the correct page */
     LCD_command(disp_ypos(0));
     for (i = 0; i < 8; ++i)
     {
      LCD_data(charset[29 + j][i]);
     }
```

```
    _delay_ms(100);
  }
}

int main(void)
{
 init_display();
 ADC_Init();
 clear_display();
 make_maze();
 draw_maze();
 timer_setup();
 uint8_t current_mode = GAME_START;
 BEST_TIME = UINT32_MAX;
 int bool_v = 0;


 //msg_game_over();


 /* Replace with your application code */
 while (1)
 {

   switch(current_mode)
   {
    case GAME_START:
     current_mode = GAME_RUN;
     GAME_TIME = 0;
     msg_get_ready();
     do_countdown();
     draw_maze();
     /* in a proper c program the start position
           of the ball should be calculated, but alas... */
     a_ball.posx = 4;
     a_ball.posy = 30;
     a_ball.v_x = 0;
     a_ball.v_y = 12;
     TCNT1 = 0;
     break;
    case GAME_RUN:
     a_ball.v_y += (getXValues()*3); /* maybe we should make the game less
sensitive */
     a_ball.v_x -= (getYValues()*3);
     draw_ball();
     GAME_TIME += TCNT1; // read timer 1 counter
     TCNT1 = 0; // reset it for next read

     if (maze[a_ball.posy/8][a_ball.posx/8].finish_line)
      current_mode = GAME_END;
     _delay_ms(5);
     break;
    case GAME_END:
```

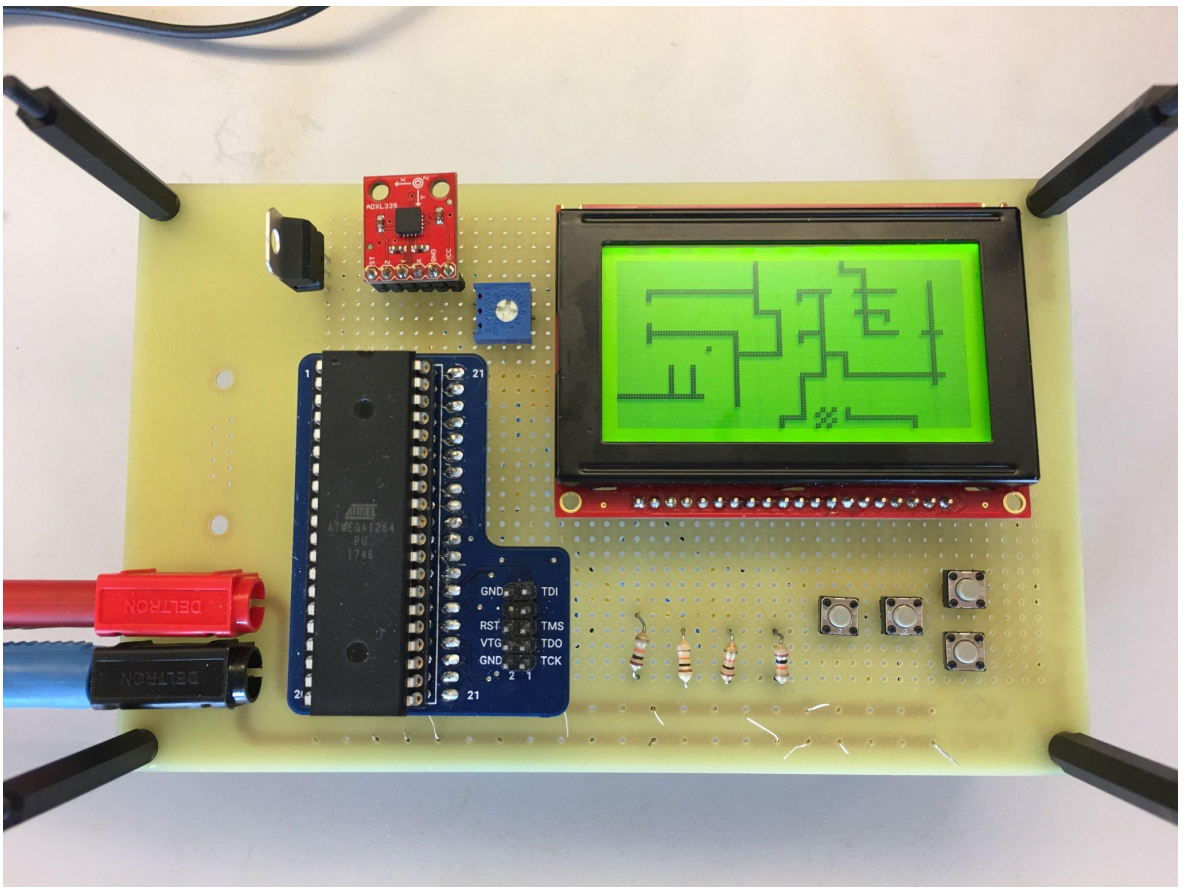
```
      if (GAME_TIME < BEST_TIME)
      {
       BEST_TIME = GAME_TIME;
       msg_kycklingmiddag(1);
      }
      else
      {
       msg_well_done(1);
      }
      msg_your_time(3);
      msg_a_time(4, get_time_s(GAME_TIME));
      msg_best_time(5);
      msg_a_time(6, get_time_s(BEST_TIME));
      _delay_ms(1000);
      draw_maze();
      current_mode = GAME_START;
      break;
    }
  }
}
```

Appendix C – Bilder

Bilaga 6: Start av spelrunda.



Bilaga 7: Pågående spelrunda



Bilaga 8: Slut av spelrunda.