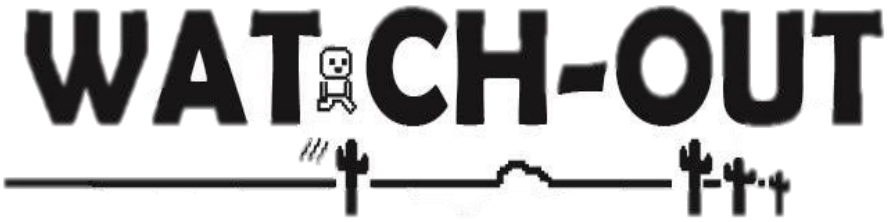


Projektrapport
Mikrodatorprojekt, TSIU51

Grupp 5

Emil Pihl
Kebba Jeng
Aidin Jamshidi
Martin Castro Bildhjerd

Innehållsförteckning

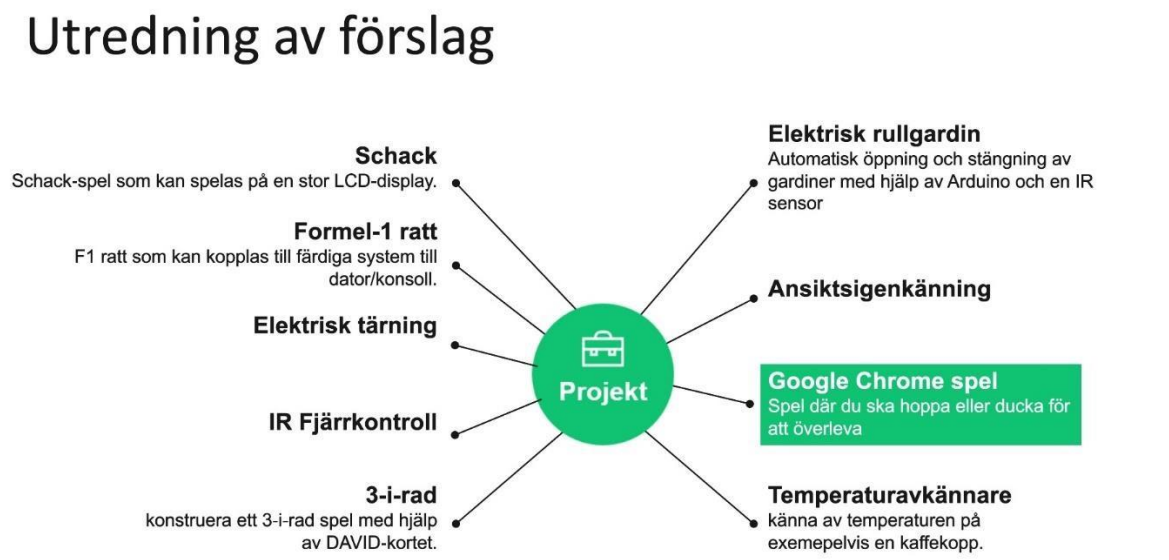
Figurförteckning

1.0 - Inledning

Watch-Out Är ett spel framtaget för kursen mikrodatorprojekt, *TSIU51*. Syftet med kursen är att i projektform konstruera något av teknisk natur, exempelvis ett spel. Spelet är inspirerat av *Googles Dinosaur Game*, som går ut på att undvika hinder för att samla poäng.

1.1 – Projektbakgrund

Under projektets inledning togs det fram olika projektförslag. Därefter utreddes förslagen med hänsyn till svårighetsgrad och arbetsbörda, detta för att färdigställa projektet inom kursens tidsram. *Figur 1.1* visar de förslag som togs fram, och det slutgiltiga förslaget.

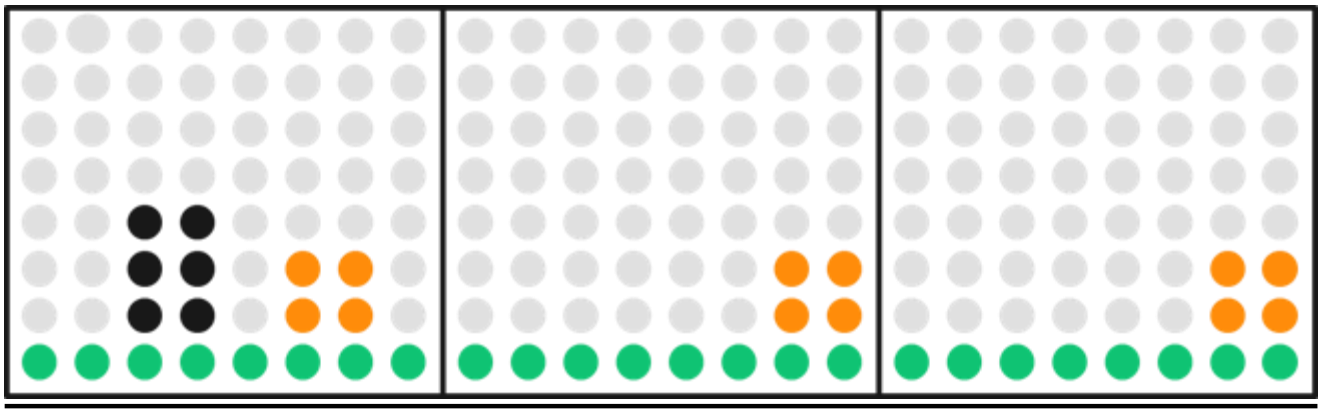


Figur 1.1 - Förslag: *Exempel på projektidéer och det slutgiltiga markerat med grön färg*

1.2 – Spelbeskrivning

Spelet går ut på att en spelare med hjälp av en spelkaraktär ska undvika objekt för att försöka att samla poäng. Spelet *Watch-Out* spelas på hårdvaruplattformen DAvid, med hjälp av de komponenter som finns på kortet.

Spelkaraktären visar sig på vänster sida av spelplan, från höger sida av spelplan dyker objekt upp som karaktären måste undvika för att överleva och fortsätta spelet. Detta illustreras i *figur 1.2*. Misslyckas spelaren med att undvika ett hinder, resulterar det i förlust.



Figur 1.2 - Spelexempel: *På bilden syns ett exempel på hur det ser ut när tre objekt närmar sig karaktären som styrs av spelaren. Karaktären är i färgen svart medan objekten är i färgen orange.*

1.3 - Kravspecifikation

Projektet är utformat utefter en kravspecifikation som består av skall- och bör-krav. Dessa krav ska uppfyllas för att projektet ska färdigställas och för att kunna leverera en fullständig slutprodukt.

Skall-krav:

Listan innehåller de funktioner som tillsammans utgör grunden för spelet.

1. Spelaren **skall** vara en figur som liknar en dinosaurie.
2. Spelaren **skall** kunna hoppa med figuren.
3. Spelaren **skall** röra sig åt höger automatiskt med figuren.
4. Spelaren **skall** kunna samla poäng.
5. Spelaren **skall** kunna förlora spelet igenom att förlora alla hjärtan.
6. Spelaren **skall** kunna se hur mycket poäng den har samlat ihop.
7. Spelaren **skall** kunna förlora spelet genom att förlora alla hjärtan.
8. Spelaren **skall** kunna förlora spelomgången genom att bli träffad av ett objekt.
9. Spelaren **skall** kunna spela med tryckknappar på hårdvarukortet DAvid.
10. Det **skall** finnas en startsekvens när spelaren startar spelet.
11. Det **skall** finnas en slutsekvens när spelaren förlorar.
12. Det **skall** finnas en huvudmeny.
13. Det **skall** finnas tre pixel-skärmar av typen DAmatrix.

Bör-krav:

Listan innehåller de funktioner som förbättrar spelupplevelsen utöver spelet grundfunktioner.

1. Spelaren **bör** kunna ducka med figuren.
2. Spelaren **bör** kunna samla poäng genom olika metoder.
3. Spelaren **bör** kunna samla på sig upp till tre hjärtan.
4. Spelaren **bör** kunna spela med spelplattformens *joystick*.
5. Det **bör** finnas flygande föremål som spelare måste undvika.
6. Det **bör** spelas upp ett ljud när man samlar på sig poäng.
7. Det **bör** spelas upp ett ljud när man samlar på sig ett hjärta.
8. Det **bör** spelas upp ett ljud när förlorar ett hjärta.
9. Det **bör** gå att starta om spelet från "förlora-bilden" utan att starta om hårdvaruplattformen DAvid.
10. Spelets hastighet **bör** öka för en ökad svårighetgrad.

2.0 – Översikt

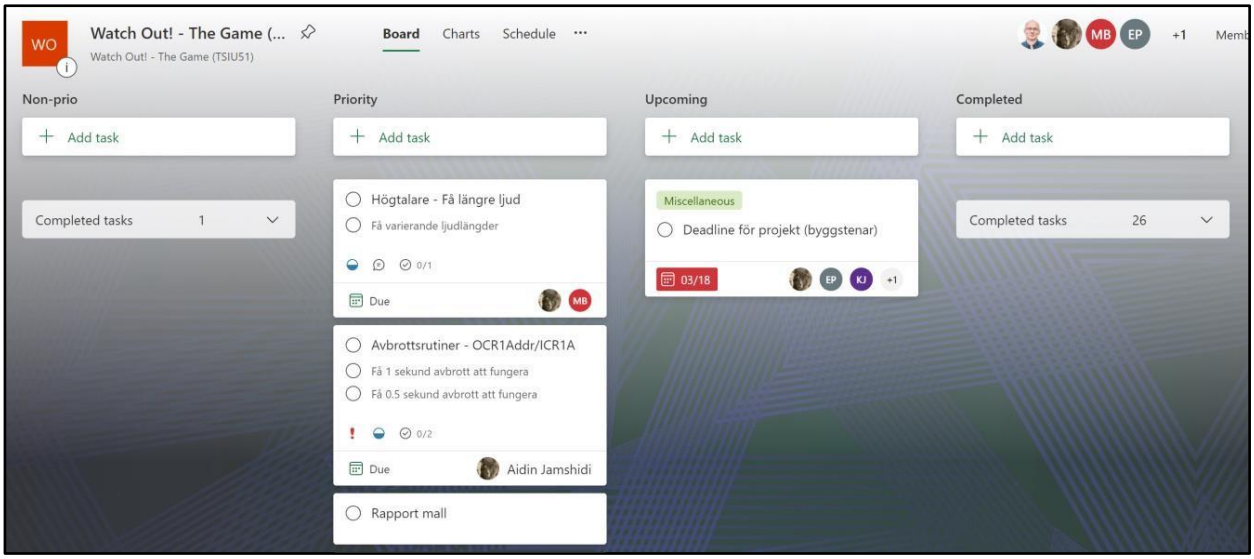
När spelaren påbörjat ett nytt spel ska en melodi spelas upp för att meddela spelaren att spelomgången startar. Detta förmedlas även genom LCD-displayen, i form av ett textmeddelande, "*New Game Started* ".

Då ett nytt spel påbörjas är spelkaraktären stillastående på startpositionen och en tom spelplan uppenbarar sig. Därefter laddas hindret in på DAmatrix-skärmen längst till höger som rör sig mot karaktären som spelaren styr. När ett hinder närmar sig spelaren måste spelaren hoppa för att undvika objektet och samla poäng.

När spelaren förlorar ska en annan typ av melodi spelas och förlusttext på LCD-displayen ska framföras till användaren. Poängsamlingen sammanställs och spelaren noteras om hur många poäng som samlades under spelets gång. Desto längre spelaren klarar sig utan att kollidera desto mer poäng samlas ihop.

2.1 - Organisering av projektarbete

Arbetet var organiserat på sådant sätt att det ingick två möten i veckan. Mötena innefattades av uppföljning av projektets utvecklingsstadier för att säkerställa att projektet fortlöpte. Utöver möten användes planeringsverktyget *Microsoft Planner* för att visualisera och planera arbetsflödet. *Figur 2.1* visar exempel.



Figur 2.1 – Planner: *Olika uppgifter i Microsoft planner där de rankas under olika kategorier med specifika medlemmar tilldelade.*

2.2 – Komponenter

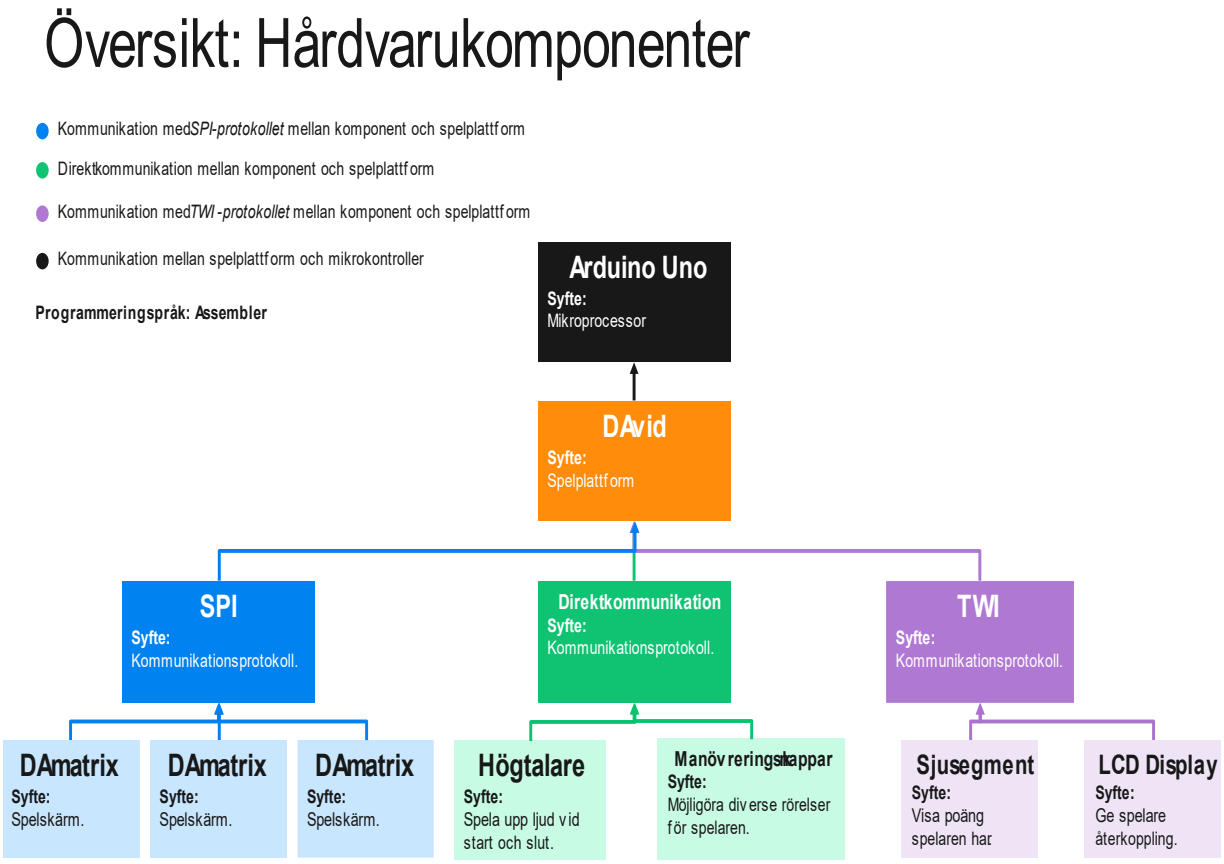
För spelet valdes ett antal komponenter på hårdvarukortet som ansågs lämpliga för att uppnå bör- och skall-kraven i kravspecifikationen. Spelet använder följande komponenter som listas i *figur 2.2*.

Komponenter			
Komponent	**Antal**	**Typ**	**Beskrivning**
DAvid hårdvarukort	1	Hårdvarukort	Hårdvarukort framtaget för kursen TSIU51
Arduino Uno	1	Mikrokontroller	Mikrokontroller baserad på processorn ATMega328P
DAmatrix	3	Skärm 8x8 pixlar	Tre skärmar som kopplas ihop för att få en 8x24 spelplan
Piezoelektrisk högtalare	1	Högtalare	Högtalare som används för spela upp ljud under start-meny och förlust-meny
Tryckknapp	2	L och R tryckknapp	L-knapp används för att ge spelaren möjlighet att hoppa. R-key används för att starta spelet
LCD-display	1	Skärm	Skärm som används för att förmedla information till spelaren
3D-printad stativ	1	Stativ för skärm	Stativ för att koppla ihop tre DAmatrix-skärmar
7-segment	2	Skärm	Skärm som visar hur många poäng spelaren har samlat ihop

Figur 2.2 - Komponenter: *Lista med komponenter, antal och beskrivningar om användningsområde. Dessa komponenter tillsammans skapar slutprodukten.*

2.3 - Blockschema

I *figur 2.3* visar blockschemat den hårdvara och de tillhörande komponenterna som används i projektet. Kommunikationsprotokollen *TWI*, och *SPI* möjliggör kommunikation mellan dessa komponenter och *DAvid-kortet.*



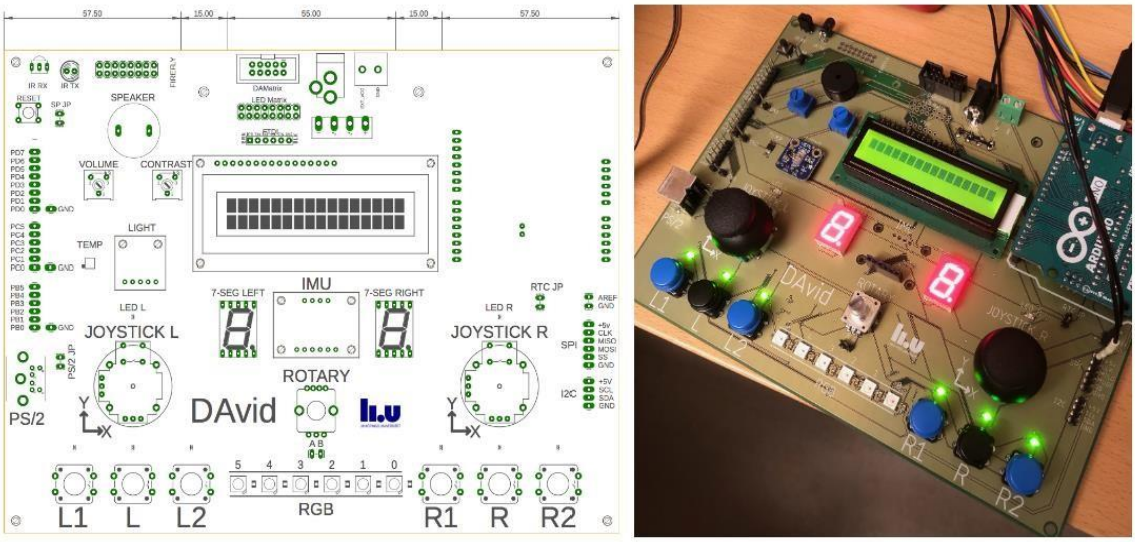
Figur 2.3 - Blockschema: *Hårdvarukortet DAvid. Framtaget för projektkurser i Mikrodatorteknik. Hårdvarumiljön består utav komponenter som LCD-display, LEDs, tryckknappar med mera. Även syfte för komponenterna är inkluderat i figuren.*

2.4.0 - Beskrivning av hårdvara och protokoll

Nedan följer en ingående beskrivning av de komponenter och protokoll som används för spelet *Watch-Out*. Dessa protokoll är viktiga för kommunikationen mellan komponenterna.

2.4.1 - DAvid-kort

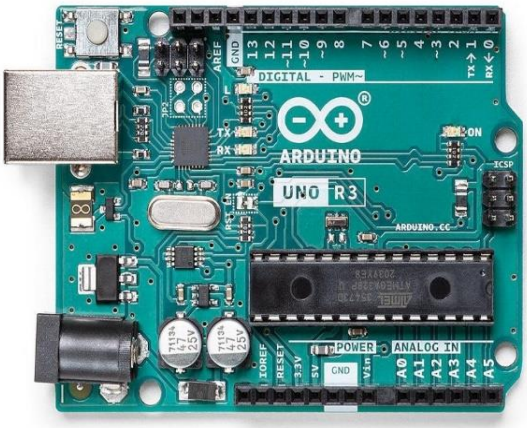
DAvid-kortet är ett hårdvarukort utvecklat och tillverkat för kursen Mikrodatorprojekt (TSIU51). Det är en hårdvarumiljö med komponenter för att möjliggöra strukturerad mjukvaruutveckling, vars hårdvarumiljö lämpar sig för exempelvis spel. Kortet styrs av mikronkontrollerkortet *Arduino UNO* med processorn *ATMega328p* [1].



Figur 2.4.1 – DAvid-kort: *Hårdvarukortet DAvid. Framtaget för projekt i kursen Mikrodatorprojekt. Hårdvarumiljön består utav diverse komponenter som LCD-display, LEDs, tryckknappar, med mera.*

2.4.2 - Arduino Uno

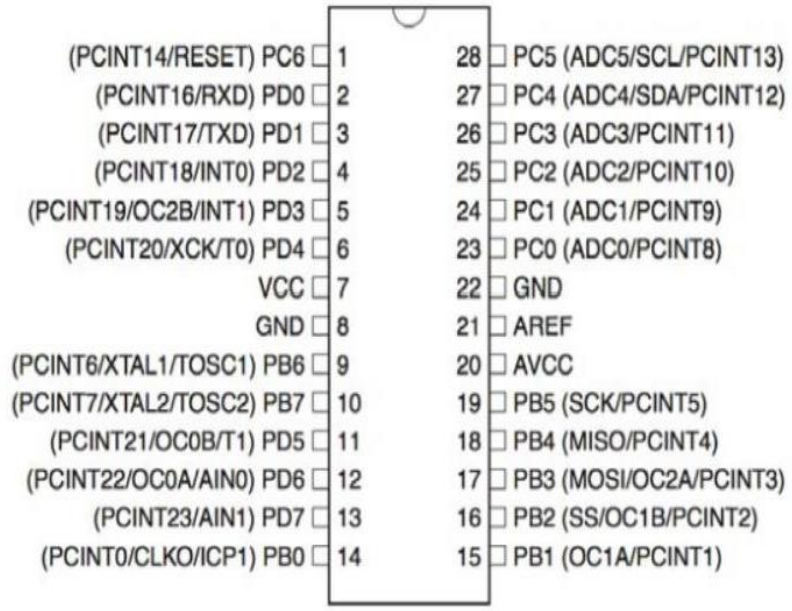
Arduino Uno är ett mikrokontrollerkort baserat på mikrokontroller ATMega328p. En mikrokontroller är en mindre dator med processor, arbetsminne, och programminne integrerat komplett med stödfunktioner på en enda elektronisk krets.



Figur 2.4.2 – Arduino Uno: *Överblick av mikrokontrollerkortet Arduino Uno. På bilden finnes ett kort och diverse elektroniska komponenter som tillsammans utgör Arduino Uno.*

Mikrokontrollerkortet är utrustat med digitala och analoga *I/O pinnar* som kan användas för att kommunicera med andra komponenter och kretsar [2].

ATMega328p är en mikrokontroller från *Atmel*. ATMega328p har ett 32kb minne och en 16 MHz klockfrekvens. Detta gör att en mikrokontroller kan utföra upp till 16 miljoner instruktioner per sekund. Det finns ett flertal *I/O-pinnar*[1] på ATMega328p. Detta ger möjlighet att ansluta extern hårdvara som inte finns i DAvid-kortets hårdvarumiljö, exempelvis ljusdiodmatrisen DAmatrix.



Figur 2.4.2 – I/O-portar: *Överblick på tillgängliga I/O portar på Arduino Uno. I/O portarna möjliggör kommunikation med extern hårdvara till processorn ATMega328P.*

2.4.3 – TWI

Two-Wire Interface, förkortat *TWI*, är ett synkront seriellt kommunikationsprotokoll. TWI ger hårdvara, system, och processer ett kommunikationsgränssnitt med en buss som består av två ledningar: *Serial Data* (SDA), och *Serial Clock* (SCL). SDA är dataledningen, där data överförs mellan en *Master* och *Slave-nod*. SCL är klockledningen som skickar pulser.

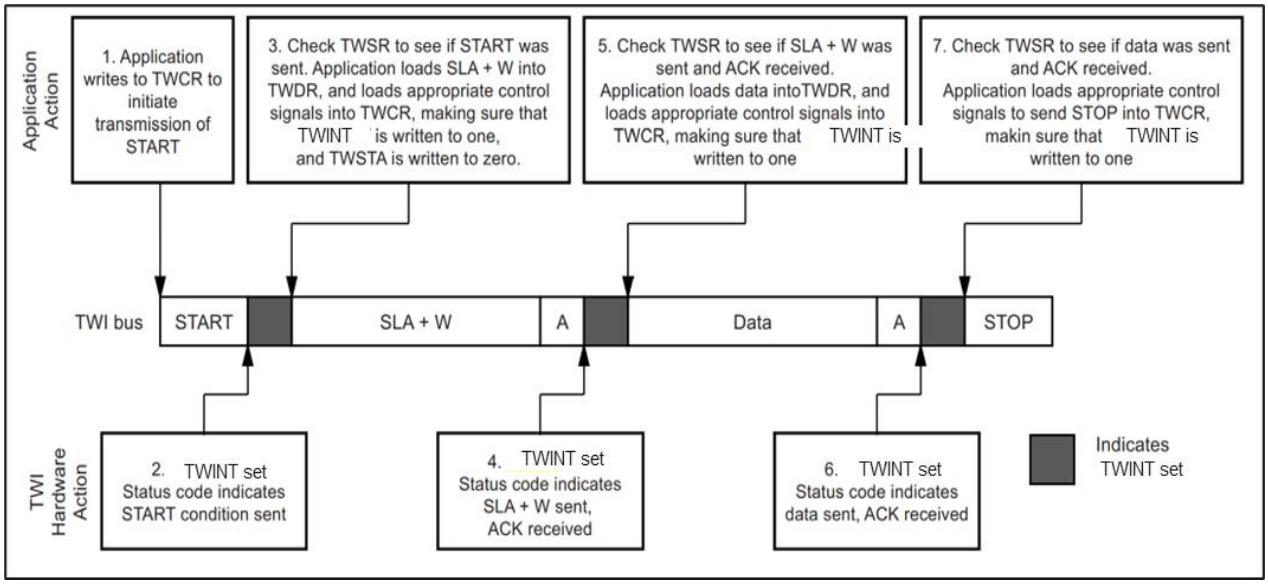
TWI kommunicerar i duplex med en *master-slave-arkitektur*, vilket innebär att kommunikation sker samtidigt i två riktningar. För att överföra data och kommunicera mellan noder[2] [4] så behöver protokollet konfigureras.

[1] En Input/Output pin, eller I/O pin, är gränssnittet mellan en mikrokontroller och en annan krets.
[2] En nod är antingen en slutpunkt eller förgrening i ett datornätverk. Varje aktiv enhet som kan sända, ta emot eller vidareförmedla data är en nod.

I detta projekt konfigurerades följande register:

- *TWI Bit Rate Register*. Kontrollerar klockledningens period.
- *TWI Control Register*. Kontrollerar TWI- operationer, exempelvis avbrott. Används även för att generera START, STOP, och ACK-pulser.
- *TWI Data Register*. Kan sättas i olika lägen. I överförningsläge skickar den data. I mottagningsläge, sparas data som tagits emot.

En dataöverföring med TWI består av ett starttillstånd. En adress för information om det ska vara en avläsning eller en skrivning (*Read/Write*)[3] [5] samt *Slave Acknowledge*. Data med utförande information. Det måste även ingå ett stopptillstånd. *Figur 2.4.3* illustrerar TWI-processen.



Figur 2.4.3 – TWI: *Exempel på hur information måste skickas med hjälp av protokollet TWI.*

2.4.4 - SPI

Serial Peripheral Interface, förkortat *SPI* är ett synkront seriellt kommunikationsprotokoll som huvudsakligen används för kort-distanskommunikation i inbyggda system.

SPI förbinds av en buss som består av fyra gemensamma ledningar. Serial Clock Output (SCLK), Master Out Slave In (MOSI), Master in Slave Out (MISO), och Slave Select (SS).

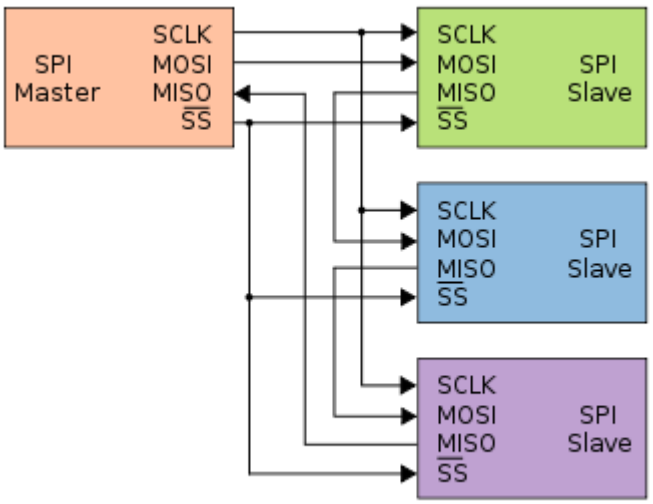
- SCLK, *Serial Clock Output*. Är som namnet indikerar klockledningen. Data som skickas genom MOSI och MISO synkroniseras till klockan.

[3] Read/Write memory, är ett minne som kan både skrivas och läsas.

- MOSI och MISO är dataledningarna. MOSI och MISO överför data mellan noder. Protokollet kan göra detta simultant. MOSI innebär att master enheten har full kontroll över slavenheten och kan skicka data till vald enhet. MISO innebär att slavenheten agerar masterenhet och data slavenheten skickar tas upp av ATmega328P.

- SS, Slave Select. Dikterar vilken enhet masternenheten kommunicerar med.

SPI kommunicerar i full-duplex med en *Master-slave-arkitektur*. Det vill säga att kommunikation sker samtidigt i två eller fler riktningar. *Master-slave-arkitekturen* möjliggör att en enhet kan kontrollera en annan enhet. Detta åskådliggörs i *figur 2.4.4*. För att överföra data och kommunicera mellan noder behöver protokollet konfigureras. Detta görs genom att modifiera bitar i SPI:s kontrollregister (SPCR). Därefter anges riktningen, det vill säga vart informationen ska skickas.



Figur 2.4.4 – SPI: *Schema som visar kommunikationsflödet mellan SCLK (Serial Clock Output), MOSI (Master Out Slave In), MISO (Master In Slave Out) och SS (Slave Select). Genom denna process sker informationsöverföring mellan spelplattformen DAvid och de komponenter som använder SPI.*

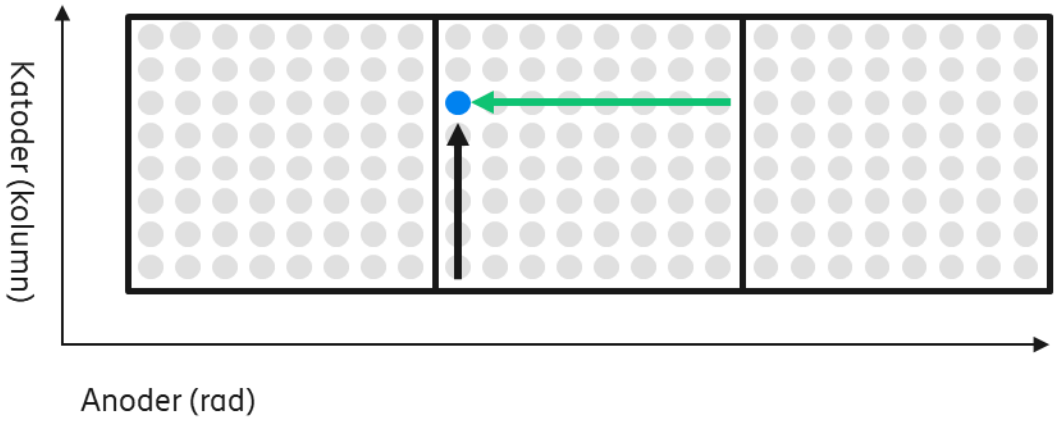
2.4.5 - DAmatrix

DAmatrix är en seriell[4] [6] LED-lysdiodmatris. Lysdiodmatrisen består utav åtta rader, och åtta kolumner. Sammanlagt 64 lysdioder (8x8 lysdioder). Genom att använda tre *DAmatrix* kan en större spelplan skapas (8x24 lysdioder). För att driva *DAmatrix* används seriella

[4] Seriellkommunikation, är en kommunikationsmetod som använder två dataledningar för att skicka och ta emot data. Data skickas och tas emot kontinuerligt, en bit åt gången.

gränssnittet *SPI* [3] via *Arduino UNO:s I/O-Pinnar*. Flödesdiagrammet i *figur 2.4.5* illustrerar hur *SPI* används i detta projekt.

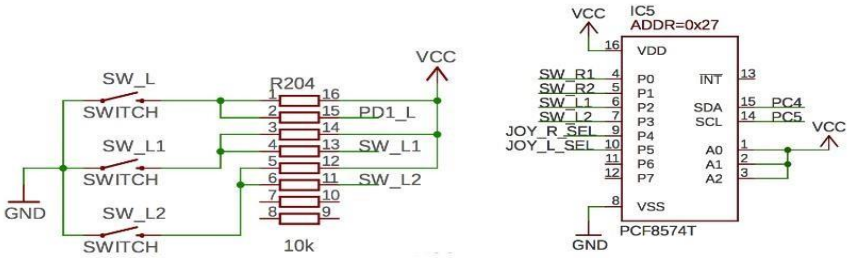
DAmatrix är konstruerat så att dioder i varje rad i matrisen hänger ihop i sina anoder, och dioder av samma sort och kolumn har sina katoder sammankopplade. För att tända en diod tillförses diodens anod med positiv spänning, samtidigt som diodens katod jordas [4].



Figur 2.4.5 – Pixel position: *Exempelbild på hur en diod kan tändas på DAmatrix genom beskrivningen ovan. Gröna pilen visar hur diodens anod tillförses med spänning. Svarta pilen visar hur lysdiodens katod jordas. Processen resulterar i att en lysdiod tänds. I figuren är lysdioden representerad av en blå punkt.*

2.4.6 – Tryckknappar

Tryckknapparna används för den animationen i spelet som resulterar i att spelaren undviker hinder. Tryckknapparna är kopplad till specifika portar på ATMega328P. Porten behöver i sitt *standby-läge* en positiv insignal för att fungera. För att adressera knapparna behövs därför en positiv insignal.



Figur 2.4.6 – Tryckknappar: *Kopplingsschema för tryckknappen L och R. Knappen L används tills att hoppa med karaktären medan R används till att starta spelet.*

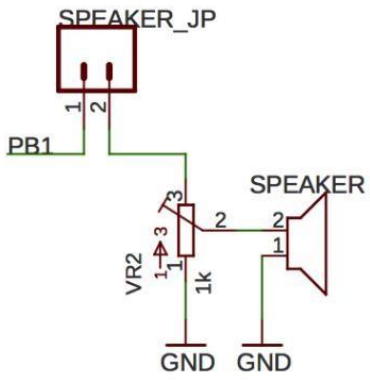
2.4.7 - LCD-display

En LCD-display (*Liquid Crystal Display*) är en elektronisk skärmmodul. För detta projekt är displayens syfte att ge spelaren återkoppling. Detta illustreras i *figur 2.4.7*. Det är en 16x2 display, det innebär att den kan visa 16 tecken per rad och att skärmen har 2 rader.

Figur 2.4.7 – New game: *Skärmen som visas för spelaren när spelet har startat en ny omgång. Texten ”NEW GAME STARTED” visas till spelaren förlorar, då texten ändras till ”GAME OVER!”.*

2.4.8 - Piezoelektrisk högtalare

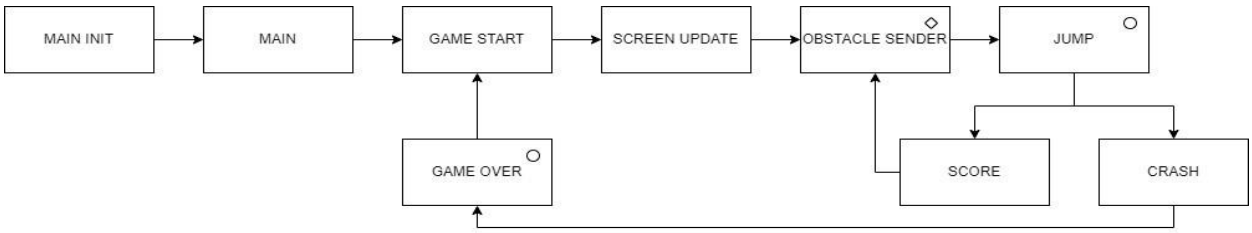
Piezoelektriska högtalarens syfte är att ge spelaren återkoppling under spelet. Högtalaren är *passiv*, och behöver en förstärkt insignal för att fungera. För att adressera högtalaren skickas signaler till enheten i form av kortvariga spänningspulser. Melodier och ljud för spelet skapas genom att skicka olika frekvenser till högtalaren vilket ger olika noter.



Figur 2.4.8 – Högtalare: *Kopplingsschema för den piezoelektriska högtalaren.*

3.0.0 – Kodbeskrivning

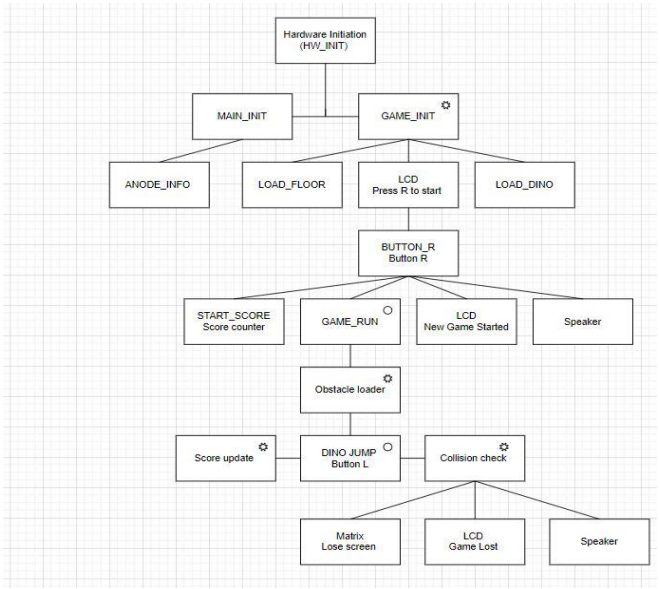
Figur 3.0.0 visar en förenklad version av kodflödet. Strukturdiagrammet illustrerar olika stadier i spelet och den generella loopen som driver spelet.



Figur 3.0.0 – Game-loop: *Strukturdiagram som visar hur koden fungerar. I detta fall sker några viktiga rutiner innan spelrutinerna börjar. Cirkel indikerar valbarhet, rubin indikerar iteration.*

3.0.1 – JSP

Figur 3.0.1 illustrerar kodflödet för *Watch-Out* strukturerat enligt JSP-metoden[5]. Vid start av hårdvarukortet börjar rutinen *HW_INIT* för att sedan förbereda komponenter med *MAIN_INIT* och *GAME_INIT*.



Figur 3.0.1 - JSP: *JSP-diagram som visar strukturen av koden med hjälp av strukturerar som sekvens, iteration och selektion.*

[5] En metod för strukturerad programmering, utvecklad av Michael A. Jackson.

3.0.2 – Videominne

Videominne är 92 bytes som är reserverade i SRAM[6], där information som karaktär och golv kan sparas. Informationen kan sedan plockas ut och skrivas ut på DAmatrix. Med hjälp av videminnet möjliggörs rutinerna *LOAD_FLOOR* och *LOAD_DINO* som visas i *figur 3.0.1*.

3.0.3 – Anod-information

Rutinen *ANOD_INFO* ser till att rätt anoddata lagras i videominnet. Utan rutinen fungerar inte utskriften till DAmatrix. Sekvensen ser till att var fjärde byte i videominnet lagrar rätt information som representerar vilken skärm och rad. Denna rutin körs efter *HW_INIT*, se *figur 3.0.1*, och är en sekvens som fungerar självständigt.

3.0.4 – Avbrott

Spelet använder sig av två avbrott[7], *avbrottsrutin*, och *FPS_RUTIN*. *FPS_RUTIN* ser till att den informationen som finns i videominnet skrivs ut till skärmen. Utskriften görs i iterationer vilket bidrar till formen på hinder och karaktär. *FPS-rutin* möjliggör utskriften av karaktären och hindret i spelet.

Avbrottsrutin är avbrottet som uppdaterar positionen på hindret och ser till att den rör sig från höger till vänster på DAmatrix. Denna rutin är avbrottsstyrd för att hindret alltid färdas höger till vänster oavsett vilket stadie kodflödet befinner sig.

3.0.5 – *Collision*

Rutinen *Collision* använder sig av en byte i *SRAM* som laddas med information om vart karaktären befinner sig. För att ta fram informationen hämtas karaktärens pixeldata från *videominnet.* Pixeldata består av 3 bytes där karaktären utgör en byte.

Befinner sig endast karaktären på den pixeln blir informationen som sparas binär nolla. Skulle karaktären och hindret befinna sig på samma pixel kommer information i stället bli en binär etta som räknas som en träff och spelet avslutas.

[6] *Static random-access memory* (*static RAM* eller *SRAM*) är en typ av läsbart och skrivbart datorminne. *SRAM* är ett volatilt datorminne. Detta innbär att data förloras när ström inte tillförs.

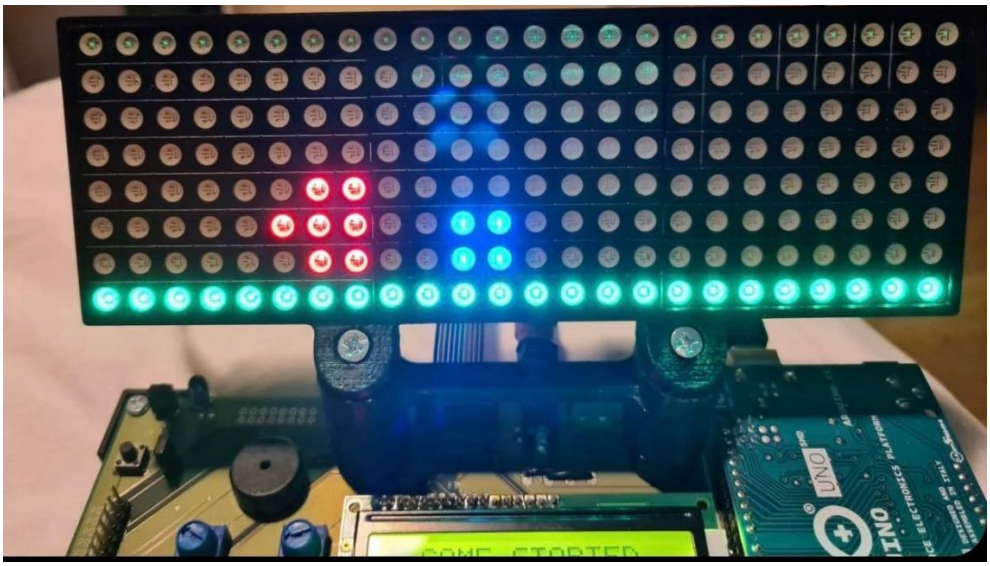
[7] Ett avbrott är en signal skickad från en enhet eller process som när en process eller händelse behöver omedelbar uppmärksamhet.

```
COLLISION:
        push    r28
        push    YL
        push    YH
        push    r16
        push    r19
        push    r18

        ldi     r28, $94
        ld      r19, Y+
        ld      r18, Y+
        ld      r16, Y

        andi    r18, 0b00000011
        andi    r19, 0b00000011
        or      r19, r18
        and     r16, r19

        cpi     r16,1
        brlo    NO_HIT
        sts     HIT, r16

NO_HIT:
```

Figur 3.0.5 - Collision: *Assemblerkod på spelets kollision-rutin. Pixelinformation laddas in i dataregister genom Y-pekaren. Därefter utförs instruktioner, i syfte att jämföra värdet i dataregistret med värdet 1. Om sådant är fallet har karaktären kolliderat med hindret och därmed förlorat spelet.*

4.0 – Slutprodukt

Slutprodukten består av tre DAmatrix-skärmar som utgör spelplanen. Spelarens karaktär är i färgen röd på vänster sida och höger sida är ett objekt i färgen blå, som närmar sig spelaren. Slutprodukten uppnådde majoriteten av skall- och bör-kraven. Det som inte uppnåddes var skall-krav fem, vilket innebär att möjligheten att samla hjärtan inte finns med i spelet. För att kompensera för det uteblivna skall-kravet uppfylldes bland annat bör-krav fem. Detta resulterade i användandet av tre DAmatrix-skärmar i stället för två. Bör-krav tio, att det går att starta om spelet utan att starta om spelplattformen, uppfylldes vilket bidrar till en bättre användarupplevelse. *Figur 4.0* visar slutprodukten.



Figur 4.0 – Slutprodukt: *På spelplan syns ett objekt i färgen blå och spelarens karaktär i färgen röd. I detta fall börjar objektet närma sig spelaren och dess uppgift är att hoppa.*

4.1.0 - Diskussion

Under utvecklingen av projektet har flertal problem uppkommit, dessa har bidragit till att vissa skall-krav saknas från spelet. Problemen kommer huvudsakligen från ofokuserad arbetsuppdelning och bristfällig kommunikation, men även bristande programmeringskunskaper.

4.1.1 – Vad gick som planerat?

Utformningen av kommunikationsprotokollen, TWI samt SPI färdigställdes tidigt under projektet. Detta fördelaktigt då det var en grundförutsättning för att kommunicera med DAvid-kortet. Arbetet med adressering av de individuella komponenterna som används, färdigställdes snabbt efter kommunikationsprotokollen. Detta möjliggjorde resursfördelning på mer invecklade utvecklingsområden.

Tidigt i utvecklingen bestämdes det att tre dagar i veckan skulle spenderas på utvecklingen av *Watch-Out.* Strukturering och organisering av projektet krävdes för att färdigställa de olika delmomenten i spelet. När detta arbetssätt kombinerades så gav optimala förutsättningar att utveckla projektet kontinuerligt.

4.1.2 – Motgångar

Under arbetet med *videominnet*, det vill säga processen för informationslagring och adressering av individuella pixlar på lysdiodmatrisen, DAmatrix, uppstod många flaskhalsar. Detta på grund av bland annat bristfällig kommunikation från gruppen, både internt och med kurshandledare. Som följd spenderades omfattande tid av projektet på att felsöka och skriva om programmeringskod.

Ett misstag var att inte tillämpa mer strukturerat tillvägagångs för programmeringskoden. Strukturering enligt *JSP-metoden (Jackson Struktured Programming)* hade troligtvis resulterat i kod med mindre felkällor, och *buggar.* Ett annat misstag var att inte rita upp hur rutiner och funktioner skulle formas. Detta hade gett en bättre överblick, och förenklat arbetet mot ett gemensamt mål. Följden av detta blev att det spenderades tid på att skriva kod som i slutändan inte bidrog till slutprodukten.

4.1.3 – Slutsats

Efter att projektet genomförts kan det konstateras att slutprodukten uppfyller majoriteten av kravspecifikationerna. Tillvägagångsättet, det vill säga hur arbetet fortlöpte och problem löstes är den största faktorn bakom de problem som uppstod, och konsekvent att projektet inte slutfördes inom tid.

Vi är efter omständigheterna nöjda med slutresultatet. De erfarenheter vi har fått under projektarbetet har vart givande och lärdomar från projektet kommer tillämpas i framtida arbeten.

Referenser och information

Referenser

[1] M. Josefsson, "Microsoft Teams," 17 01 2022. [Online]. Available: https://teams.microsoft.com/_#/pdf/viewer/teamsSdk/https:~2F~2Fliuonline.sharepoint.com~2Fsites~2FLisam_TSIU51_2022VT_SP~2FDelade%20dokument~2FGeneral~2FDavid_hardvarubeskrivning.pdf?threadId=19:P7uRHZyBcChbuIiSdQhbOrWzM1xBRpg9u6L_WzCjAB01@thread.tacv2&fi. [Accessed 20 04 2022].

[2] Atmel Corporation, "Microsoft Teams," 2009. [Online]. Available: https://teams.microsoft.com/_#/pdf/viewer/teams/https:~2F~2Fliuonline.sharepoint.com~2Fsites~2FLisam_TSIU51_2022VT_SP~2FDelade%20dokument~2FGeneral~2FKursdokument~2FDatablad_ATMega328.pdf?threadId=19:P7uRHZyBcChbuIiSdQhbOrWzM1xBRpg9u6L_WzCjAB01@thread.tac. [Accessed 20 04 2022].

[3] M. Josefsson, "Microsoft Teams," 2022. [Online]. Available: https://teams.microsoft.com/_#/pdf/viewer/teamsSdk/https:~2F~2Fliuonline.sharepoint.com~2Fsites~2FLisam_TSIU51_2022VT_SP~2FDelade%20dokument~2FGeneral~2FDavid_hardvarubeskrivning.pdf?threadId=19:P7uRHZyBcChbuIiSdQhbOrWzM1xBRpg9u6L_WzCjAB01@thread.tacv2&fi. [Accessed 26 04 2022].

[4] M. Josefsson, "Microsoft Teams," 2022. [Online]. Available: https://teams.microsoft.com/_#/pdf/viewer/teamsSdk/https:~2F~2Fliuonline.sharepoint.com~2Fsites~2FLisam_TSIU51_2022VT_SP~2FDelade%20dokument~2FGeneral~2FDAvid%20datasheets~2FDAmatrix%20-%208x8%20RGBLED.pdf?threadId=19:P7uRHZyBcChbuIiSdQhbOrWzM1xBRpg9u6L_W. [Accessed 26 04 2022].

Kod

Kod är uppdelad i ett *main-program* och tio inkluderade *sub-program*, där *main* innehåller grunden för spelet och *sub* innehåller byggstenarna. Grunden i detta fall blir huvudrutinerna för att spelet ska fungera. Det som blir byggstenarna är rutinerna för knapparna, högtalare, LCD och diverse andra komponenter.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Main
                    jmp         HW_INIT
                    .org        OC2Baddr
                    jmp         AVBROTTSRUTIN
                    .org        0x0020
                    jmp         FPS_RUTIN
                    .dseg

                    .include    "SPI.inc"
                    .include    "AVBROTT_FPS.inc"
                    .include    "AVBROTT_ROCK.inc"
                    .include    "CHARACTER.inc"
                    .include    "VIDEOMINNE.inc"
                    .include    "LCD.inc"
                    .include    "TWI.inc"
                    .include    "SPEAKER.inc"
                    .include    "NEW_GAME_OVER.inc"
                    .include    "7_SEG.INC"

                    .equ        IM_SPEED = 2
                    .equ        ADDR_LCD = $20
                    .equ        SLA_LCD_W = (ADDR_LCD << 1) | 0
                    .equ        SLA_LCD_R = (ADDR_LCD << 1) | 1
                    .equ        FN_SET = $2B
                    .equ        E_MODE = $06
                    .equ        DISP_ON = $0F
                    .equ        LCD_CLR = $01
                    .equ        E = $01
                    .equ        HOME = $02
                    .equ        RS = PB0
                    .equ        ADDRESS_1 = $25
                    .equ        ADDRESS_2 = $24
TEXTONE:
                    .db         " GAME STARTED", $00
TEXTTWO:
                    .db         "GAME OVER!", $00
TEXTTHREE:
                    .db         "PRESS R TO START", $00
                    .cseg
HW_INIT:

                    call        TWI_INIT
                    clr         r16
                    call        SPI_MASTER_INIT
                    clr         r16
                    call        AVBROTTSRUTIN_INIT
                    clr         r16
                    call        FPS_INIT
                    clr         r16
                    clr         r17
                    call        SRAM_CLEAR
                    call        CLEAR_SCREEN
MAIN_INIT:
                    call        ANOD_INFO
GAME_INIT:
                    call        START_SCORE
                    call        CLEAR_LOCATION
```

```
                call    LCD_READY
                call    BUTTON_R
                call    LCD_GAMESTART
                call    NEW_GAME
                call    LOAD_FLOOR
                call    LOAD_DINO
                call    ROCK
                sei
GAME_RUN:
                sbis    PIND, PD1
                call    DINO_JUMP
                lds     r16, HIT
                cpi     r16, 1
                sbrs    r16,0b00000001
                jmp     GAME_RUN
END:
                push    r16
                ldi     r16,10
                call    SRAM_CLEAR
                sts     STORE_SPEED, r16
LOST_SCREEN:
                dec     r16
                call    LOAD_DINO
                call    DELAY_HALFHALF
                call    GAME_OVER
                call    DELAY_HALFHALF
                cpi     r16,0
                brne    LOST_SCREEN
                pop     r16
                call    SRAM_CLEAR
                cli
                ldi     r16,0
                sts     HIT, r16
                sts     ROW, r16
                sts     STORE_SPEED, r16
                call    LCD_GAMEOVER
                call    LOST_GAME
                ldi     r18,3
DELAY_LOOP:
                call    DELAY_HALFHALF
                dec     r18
                brne    DELAY_LOOP
                jmp     GAME_INIT
RESET_Z:
                ldi     r30, 0x50
                ldi     r31, 0x01
                ret
RESET_Y:
                ldi     r29, 0x01
                ldi     r28, 0x50
                ret
ANOD_INFO:
                push    YL
                push    YH
                push    r23
                ldi     YH, HIGH(VMEM)
                ldi     YL, LOW(VMEM)
                call    RESET_Y
                dec     YL
                ldi     r23, 0b11111110
                call    FOUR_STEP
                pop     r23
                pop     YH
                pop     YL
                ret

DELAY_HALFHALF:
                push    r20
                push    r21
                push    r22
```

```
                ldi     r20,5
DELAY1_THREE:
                ldi     r21,245
DELAY2_THREE:
                ldi     r22,255
DELAY3_THREE:
                dec     r22
                nop
                brne    Delay3_THREE
                dec     r21
                brne    Delay2_THREE
                dec     r20
                brne    Delay1_THREE
                pop     r22
                pop     r21
                pop     r20
                ret
SPEAKER_HALFHALF:
                push    r16
                push    r17
                push    r18
                ldi     r16,16
SPEAKER1_THREE:
                ldi     r17,245
SPEAKER2_THREE:
                ldi     r18,255
SPEAKER3_THREE:
                dec     r18
                brne    SPEAKER3_THREE
                dec     r17
                brne    SPEAKER2_THREE
                dec     r16
                brne    SPEAKER1_THREE
                pop     r18
                pop     r17
                pop     r16
                ret
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//SPI
                .dseg
                .equ    MOSI = PB3
                .equ    MISO = PB4
                .equ    SCLK = PB5
                .equ    SPI_DDR = DDRB
                .equ    CS = PB2
                .cseg

SPI_MASTER_INIT:
                ldi     r17, (1<<MOSI)|(1<<SCLK)|(1<<PB2)
                out     SPI_DDR, r17
                ldi     r17, (1<<SPE) | (1<<MSTR) | (1<<SPR0)
                out     SPCR, r17
                ret
RETREIVE_PIXEL:
                ldi     r19,12
MATRIX_LOOP:
                ld      r16, Z+
                call    SPI_TRANSMIT
                clr     r16
                dec     r19
                cpi     r19, 0
                brne    MATRIX_LOOP
SPI_SEND:
                sbi     PORTB, PB2
                nop
                cbi     PORTB, PB2
                ret
SPI_TRANSMIT:
                out     SPDR, r16
WAIT_TRANSMIT:
```

```
                    in      r16, SPSR
                    sbrs    r16, SPIF
                    rjmp    WAIT_TRANSMIT
                    ret
SHORT_WAIT:
                    adiw    r24,8
                    brne    SHORT_WAIT
                    ret
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//AVBROTT FPS
                    .equ    TIMER2_TICK = 60
FPS_INIT:

                    ldi     r16, (1<<WGM12) | (1<<CS12)
                    sts     TCCR1B, r16
                    ldi     r16, HIGH(TIMER2_TICK)
                    sts     OCR1AH, r16
                    ldi     r16, LOW(TIMER2_TICK)
                    sts     OCR1AL, r16
                    ldi     r16, (1<<OCIE1A)
                    sts     TIMSK1, r16
                    ret
FPS_RUTIN:
                    push    r16
                    in      r16, sreg
                    push    r16
                    push    ZH
                    push    ZL
                    ldi     ZH, HIGH(VMEM)
                    ldi     ZL, LOW(VMEM)
                    call    RESET_Z
                    call    ROW_PRINT
                    call    RETREIVE_PIXEL
COLLISION:
                    push    r28
                    push    YL
                    push    YH
                    push    r16
                    push    r19
                    push    r18
                    ldi     r28, $94
                    ld      r19, Y+
                    ld      r18, Y+
                    ld      r16, Y
                    andi    r18, 0b00000011
                    andi    r19, 0b00000011
                    or      r19, r18
                    and     r16, r19
                    cpi     r16,1
                    brlo    NO_HIT
                    sts     HIT, r16
NO_HIT:
                    pop     r18
                    pop     r19
                    pop     r16
                    pop     ZH
                    pop     ZL
                    pop     r28
                    pop     ZL
                    pop     ZH
                    pop     r16
                    out     sreg, r16
                    pop     r16
                    reti
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//AVBROTT ROCK
                    .equ    TIMER1_TICK =  62500 - 1
AVBROTTSRUTIN_INIT:
                    clr     r16
```

```
                ldi     r16, (1<<WGM21)
                sts     TCCR2A, r16
                ldi     r16, (1<<CS22) | (1<<CS21) | (1<<CS20)
                sts     TCCR2B, r16
                ldi     r16, 251
                sts     OCR2A, r16
                ldi     r16, (1<<OCIE2B)
                sts     TIMSK2, r16
                ret
AVBROTTSRUTIN:
                push    r22
                push    r16
ROCK_SPEED:
                lds     r22, STORE_SPEED
                inc     r22
                sts     STORE_SPEED, r22
                cpi     r22, IM_SPEED
                brne    NO_ROCK
                call    ROCK_MOVE
                clr     r22
                sts     STORE_SPEED, r22
NO_ROCK:
                pop     r16
                pop     r22
                reti
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//CHARACTER
DINO_JUMP:
                push    YL
                ldi     YL, LOW(VMEM)
                ldi     YL, $7C
                call    MOVE
                call    DELAY_HALFHALF
                ldi     YL, $70
                call    MOVE
                call    DELAY_HALFHALF
                ldi     YL, $64
                call    MOVE
                call    DELAY_HALFHALF
                ldi     YL, $64
                call    DOWN
                call    DELAY_HALFHALF
                ldi     YL, $70
                call    DOWN
                call    DELAY_HALFHALF
                ldi     YL, $7C
                call    DOWN
                call    DELAY_HALFHALF
                pop     YL
                call    SEG_7
                ret
MOVE:
                call    INC_BYTES
                call    JUMP_10_BYTES
                call    INC_TAIL
                clr     r24
                call    JUMP_10_BYTES
                call    INC_BYTES
                clr     r24
                call    JUMP_10_BYTES
                call    REMOVE_BYTES
                clr     r24
                ret
JUMP_10_BYTES:
                inc     r28
                inc     r24
                cpi     r24,10
                brne    JUMP_10_BYTES
                ret
DOWN:
```

```
                call    REMOVE_BYTES

DOWN_LOOP:
                inc     r28
                inc     r24
                cpi     r24,10
                brlo    DOWN_LOOP
                call    INC_BYTES
                clr     r24
PUT_TAIL_D:
                inc     r28
                inc     r24
                cpi     r24,10
                brne    PUT_TAIL_D
                call    INC_TAIL
                clr     r24
ADD_FEET:
                inc     r28
                inc     r24
                cpi     r24,10
                brne    ADD_FEET
                call    INC_BYTES
                call    RESET_Z
                clr     r24
                ret
INC_BYTES:
                push    r20
                ldi     r20,0b00000011
                inc     r28
                inc     r28
                st      Y, r20
                pop     r20
                ret
REMOVE_BYTES:
                push    r19
                inc     r28
                inc     r28
                st      Y, r19
                pop     r19
                ret
INC_TAIL:
                push    r22
                ldi     r22, 0b00000111
                inc     r28
                inc     r28
                st      Y, r22
                pop     r22
                ret
LOAD_FLOOR:
                push    r16
                push    r17
                ldi     r16, $00
                ldi     r17, $FF
                sts     vmem+92, r16
                sts     vmem+93, r17
                sts     vmem+94, r16
                sts     vmem+88, r16
                sts     vmem+89, r17
                sts     vmem+90, r16
                sts     vmem+86, r16
                sts     vmem+85, r17
                sts     vmem+84, r16
                pop     r17
                pop     r16
                ret
LOAD_DINO:
                push    r16
                push    r17
                push    r18
```

```
                ldi     r16, 0b00000111
                ldi     r17, 0b00000011
                ldi     r18, 0b00000000
                sts     vmem+56, r18
                sts     vmem+57, r18
                sts     vmem+58, r17
                sts     vmem+68, r18
                sts     vmem+69, r18
                sts     vmem+70, r16
                sts     vmem+80, r18
                sts     vmem+81, r18
                sts     vmem+82, r17
                pop     r18
                pop     r17
                pop     r16
                ret
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//VIDEOMINNE
                .dseg
                .org    0x150
VMEM:           .byte 96
                .org    0x200
CHECK_JUMP:     .byte 1
CHECK_ROCK:     .byte 1
STORE_SPEED:    .byte 1
DATA:           .byte 1
LCD_DATA:       .byte 1
LCD_DATA2:      .byte 1
HIT:            .byte 1
ENTAL:          .byte 1
TIOTAL:         .byte 1
ROW:            .byte 1
                .cseg
SRAM_CLEAR:
                push    r17
                push    r18
                push    r19
                push    ZH
                push    ZL
                ldi     r18, $00
                ldi     r19, 30
                ldi     ZH, HIGH(VMEM)
                ldi     ZL, LOW(VMEM)
SRAMCLEARLOOP:
                ldi     r17,3
LOAD_:
                dec     r17
                st      Z+, r18
                cpi     r17, 0
                brne    LOAD_
                inc     ZL
                dec     r19
                cpi     r19, 0
                brne    SRAMCLEARLOOP
                pop     ZL
                pop     ZH
                pop     r19
                pop     r18
                pop     r17
                ret
GAME_OVER:
                push    r16
                push    r17
                push    r18
                ldi     r16, 0b00000000
                ldi     r17, 0b00000000
                ldi     r18, 0b00000000
                sts     vmem+56, r18
                sts     vmem+57, r18
                sts     vmem+58, r17
```

```
                    sts     vmem+68, r18
                    sts     vmem+69, r18
                    sts     vmem+70, r16
                    sts     vmem+80, r18
                    sts     vmem+81, r18
                    sts     vmem+82, r17
                    pop     r18
                    pop     r17
                    pop     r16
                    ret
CLEAR_SCREEN:
                    push    r18
                    push    r16
                    ldi     r18, 72
SCREEN_LOOP:
                    ldi     r16, $00
                    dec     r18
                    cpi     r18,0
                    call    SPI_TRANSMIT
                    brne    SCREEN_LOOP
                    pop     r16
                    pop     r18
                    ret
FOUR_STEP:
                    inc     r28
                    inc     r24
                    cpi     r24,4
                    brne    FOUR_STEP
                    clr     r24
FOUR_STEP_LOOP:
                    st      Y, r23
                    inc     r17
                    cpi     r17,3
                    brne    FOUR_STEP
                    clr     r17
                    lsl     r23
                    brcc    SKIP_ORI
                    ori     r23,0b00000001
SKIP_ORI:
                    cpi     r28, $AF
                    brlo    FOUR_STEP
                    ret
ROW_COUNTER:
                    push    r17
                    lds     r17, ROW
                    inc     r17
                    sts     ROW, r17
                    pop     r17
                    ret
ROW_PRINT:
                    push    r17
                    push    r18
                    call    ROW_COUNTER
                    lds     r17, ROW

ROW_LOOP:
                    dec     r17
                    cpi     r17,0
                    breq    ROW_END
                    ldi     r18, 12
                    add     ZL, r18
                    jmp     ROW_LOOP
ROW_END:
                    lds     r17,ROW
                    cpi     r17,8
                    brne    ROW_DONE
                    clr     r17
                    sts     ROW, r17

ROW_DONE:
```

```
            pop     r18
            pop     r17
            ret

ROCK:
            ldi     r25, 0b00000001
            push    r16
            ldi     r16,$8C
            sts     CHECK_ROCK, r16
            pop     r16
            ret
ROCK_MOVE:
            lds     YL, CHECK_ROCK
            call    LOAD_ROCK
ROCK_STEP:
            inc     r28
            inc     r24
            cpi     r24,10
            brne    ROCK_STEP
            call    LOAD_ROCK
            clr     r24
            lsl     r25
            inc     r17
            cpi     r17,9
            brne    SECOND_SCREEN
            ldi     r16, $90
            sts     CHECK_ROCK, r16
            ori     r25,0b00000001
SECOND_SCREEN:
            cpi     r17,18
            brne    THIRD_SCREEN
            ldi     r16, $94
            sts     CHECK_ROCK, r16
            ori     r25,0b00000001
THIRD_SCREEN:
            cpi     r17,27
            brne    DONE
            ldi     r16, $8C
            sts     CHECK_ROCK, r16
            ori     r25, 0b00000001
            clr     r17
DONE:
            ret
LOAD_ROCK:
            st      Y, r25
            inc     r28
            st      Y,r25
            inc     r28
            ret
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//LCD
LCD_INIT:

            call    DELAY_HALFHALF

            ldi     r16,$30

            call    LCD_WRITE4
            call    LCD_WRITE4
            call    LCD_WRITE4
            ldi     r16,$20

            call    LCD_WRITE4
            ldi     r16,FN_SET

            call    LCD_COMMAND

            ldi     r16,DISP_ON
```

```
                call    LCD_COMMAND

                ldi     r16,LCD_CLR

                call    LCD_COMMAND

                ldi     r16,E_MODE

                call    LCD_COMMAND

                clr     r16

                clr     r17
                clr     r20
                ret
LCD_GAMESTART:
                push    ZH
                push    ZL
                ldi     ZH, HIGH(TEXTONE*2)
                ldi     ZL, LOW(TEXTONE*2)
                call    LCD_LINE_PRINT
                pop     ZL
                pop     ZH
                ret
LCD_GAMEOVER:
                push    ZH
                push    ZL
                ldi     ZH, HIGH(TEXTTWO*2)
                ldi     ZL, LOW(TEXTTWO*2)
                call    LCD_LINE_PRINT
                pop     ZL
                pop     ZH
                ret
LCD_READY:
                push    ZH
                push    ZL
                ldi     ZH,HIGH(TEXTTHREE*2)
                ldi     ZL,LOW(TEXTTHREE*2)
                call    LCD_LINE_PRINT
                pop     ZL
                pop     ZH
                ret
LCD_LINE_PRINT:
                 call   LCD_HOME
                 call   LCD_PRINT
                 ret
LCD_PRINT:
                lpm     r16, Z+
                cpi     r16, $00
                breq    LCD_PRINT_DONE
                call    LCD_ASCH
                rjmp    LCD_PRINT
LCD_PRINT_DONE:
                ret

LCD_ASCH:
                call    LCD_SEND_PREP
                lds     r16, LCD_DATA
                ori     r16, $01
                sts     LCD_DATA, r16
                lds     r16, LCD_DATA2
                ori     r16, $01
                sts     LCD_DATA2, r16
                call    LCD_SEND
                ret
LCD_COMMAND:
                call    LCD_SEND_PREP
                lds     r16, LCD_DATA
                andi    r16, $FE
                sts     LCD_DATA, r16
```

```
                    lds     r16, LCD_DATA2
                    andi    r16, $FE
                    sts     LCD_DATA2, r16
                    call    LCD_SEND
                    ret
LCD_WRITE:
                    sts     DATA, r16
                    ldi     r20, $20
                    call    TWI_SEND
                    ret
LCD_WAIT:
                    push    r24
                    push    r25
                    ldi     r24, $FF
                    ldi     r25, $CF
LCD_W1:
                    sbiw    r24, 1
                    brne    LCD_W1
                    pop     r25
                    pop     r24
                    ret
LCD_CLEAR:
                    ldi     r16, LCD_CLR
                    call    LCD_COMMAND
                    call    LCD_WAIT
                    ret
LCD_HOME:
                    ldi     r16, HOME
                    call    LCD_COMMAND
                    ret
LCD_SEND_PREP:
                    mov     r17, r16
                    andi    r16, $FD
                    ori     r16, $08
                    sts     LCD_DATA, r16
                    swap    r17
                    andi    r17, $FD
                    ori     r17, $08
                    sts     LCD_DATA2, r17
                    ret
LCD_SEND:
                    call    LCD_WRITE4
                    lds     r16, LCD_DATA2
                    sts     LCD_DATA, r16
                    call     LCD_WRITE4
                    ret
LCD_WRITE4:
                    call    LCD_E_LOW
                    call    LCD_E_HIGH
                    call    LCD_E_LOW
                    ret
LCD_E_LOW:
                    lds     r16, LCD_DATA
                    andi    r16, $FB
                    call    LCD_WRITE
                    ret
LCD_E_HIGH:
                    lds     r16, LCD_DATA
                    ori     r16, $0C
                    call    LCD_WRITE
                    ret
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//TWI
TWI_INIT:
                    ldi     r16, 100
                    sts     TWBR, r16
                    ldi     r16, (0<<TWINT) | (1<<TWEN)
                    sts     TWCR,r16
                    ret
TWI_SEND:
```

```
                    ldi     r16,(1<<TWINT) | (1<<TWSTA) | (1<<TWEN)
                    sts     TWCR, r16
                    call    W_WAIT2
LOAD_SLA_W:
                    lsl     r20
                    sts     TWDR, r20
                    call    TX

LOAD_DATA:
                    lds     r16, DATA
                    sts     TWDR, r16
                    call    TX
W_STOP:
                    ldi     r16, (1<<TWINT) | (1<<TWEN) | (1<<TWSTO)
                    sts     TWCR, r16
                    ret
TX:
                    ldi     r16, (1<<TWINT) | (1<<TWEN)
                    sts     TWCR, r16
W_WAIT2:
                    lds     r24,TWCR
                    sbrs    r24,TWINT
                    rjmp    W_WAIT2
                    ret
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//SPEAKER
BEEP_LOOP1:
                    ldi     r17,70
BEEP_READY1:
                    dec     r17
                    cpi     r17,0
                    breq    NOBEEP
BEEP1:
                    call    FREQUENCY_LOAD2
                    sbi     PORTB,PB1
                    sbi     DDRB,1
                    call    WAIT
                    cbi     PORTB,PB1
                    jmp     BEEP_READY1
NOBEEP:
                    call    SPEAKER_HALFHALF
                    cbi     PORTB,PB1
                    cbi     DDRB,1
                    call    SPEAKER_HALFHALF
                    clr     r17
                    ret
FREQUENCY_LOAD1:
                    push    r16

                    ldi     r16, 255
FREQUENCY_1:
                    dec     r16
                    cpi     r16,0
                    brne    FREQUENCY_1
                    pop     r16
                    ret
FREQUENCY_LOAD2:
                    push    r16

                    ldi                 r16,100
FREQUENCY_2:
                    dec     r16
                    cpi     r16,0
                    brne    FREQUENCY_2
                    pop     r16
                    ret
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//NEW GAME OVER
NEW_GAME:
                    sbi     PORTD,PD4
```

```
                sbi     PORTD,PD5
                call    BEEP_LOOP1
                call    WAIT
                cbi     PORTD,PD4
                cbi     PORTD,PD5
                brne    NEW_GAME
                ret
LOST_GAME:
                push    r18
                ldi     r18,0
LOST_GAME_LOOP:
                sbi     PORTD,PD5
                cbi     PORTD,PD4
                call    WAIT
                sbi     PORTD,PD4
                cbi     PORTD,PD5
                call    BEEP_LOOP1
                cbi     PORTD,PD5
                cbi     PORTD,PD4
                inc     r18
                cpi     r18,2
                brne    LOST_GAME_LOOP
                pop     r18
                ret
BUTTON_R:
                sbic    PIND, PD0
                jmp     BUTTON_R
                ret
WAIT:
                sbiw    r24,4

                brne    WAIT
                ret
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///7 SEG
TABLE:
                .db $3F, $6, $5B, $4F, $66, $6D, $7D, $7, $7F, $67
SEG_7:
                push    ZL
                push    ZH
CONTINUE:
                call    COUNT_ENTAL
                lds     r21, ENTAL
                cpi     r21, 10
                brne    SKIP_TIO
                CALL    RESET_RIGHT
                CALL    COUNT_TIOTAL
                call    RESET_POINTER
                inc                     r22
SKIP_TIO:
                pop     ZH
                pop     ZL
                ret
COUNT_ENTAL:
                call    RESET_POINTER
                lds     r21, ENTAL
                inc     r21
                add     ZL, r21
                sts     ENTAL, r21
                lpm     r23, Z+
                call    SEND_RIGHT
                ret
COUNT_TIOTAL:
                call    RESET_POINTER
                lds     r21, TIOTAL
                inc     r21
                add     ZL, r21
                sts     TIOTAL, r21
                lpm     r23, Z+
                call    SEND_LEFT
```

```
            clr     r21
            sts     ENTAL, r21
            lds     r21, ENTAL
            ret
SEND_LEFT:
            ldi     r20, ADDRESS_2
            sts     DATA, r23
            call    TWI_SEND
            ret
SEND_RIGHT:
            ldi     r20, ADDRESS_1
            sts     DATA, r23
            call    TWI_SEND
            ret
START_SCORE:
            ldi     r23, $3F
            call    SEND_LEFT
            call    SEND_RIGHT
            ret
RESET_RIGHT:
            ldi     r23, $3F
            call    SEND_RIGHT
            ret
RESET_POINTER:
            ldi     ZH, HIGH(TABLE*2)
            ldi     ZL, LOW(TABLE*2)
            ret
CLEAR_LOCATION:
            clr     r22
            clr     r21
            sts     ENTAL,r21
            sts     TIOTAL, r21
            ret
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```