
30010 Programmeringsprojekt



Mads Friis Bornebusch (s123627)



Tobias Tuxen (s120213)



Kristian Sloth Lauszus (s123808)

Reflex Ball
Gruppe Nummer: 15
DTU Space
June 28, 2013

Abstract

In this programming project we have written and designed all the code to the game Reflex Ball. The code has been written in C in the compiler Z8Encore! and has been implemented on a Zilog 6403 Microcontroller.

We have concluded that the written code implements the desired circuit on the Microcontroller as all different facilities of the game has been thoroughly tested and is in compliance with the expected result. In the end we have a fully functional Reflex Ball Game with many expansions, which amongst others, include different types of bricks (Arkanoid style), difficulties, different levels and steering-wheel game controller. So far no bugs has been found in the final version of the game.

Resumé

I dette programmeringsprojekt har vi skrevet og designet al koden til spillet Reflex Ball. Koden er skrevet i C i compileren Z8Encore! og implementeret på en Zilog 6403 mircocoontroller.

Vi har konkluderet at den skrevne kode implementerer det ønskede kredsløb på Mircocoontrolleren, da alle spillets facetter er blevet gennemtestet og giver det ønskede output. Som slutresultat har vi et fuldt funktionelt ReflexBall-spil med mange udvidelser der bl.a. inkluderer forskellige typer brikker (Arkanoid-stil), sværhedsgrader, forskellige levels og styring med ret. Der er indtil videre ikke fundet nogle bugs i slutversionen af spillet.

Forord

Denne rapport er skrevet som en del af eksaminationen i DTU kursus 30010 - Programmeringsprojekt. Alle tre, på forsiden nævnte, gruppemedlemmer har bidraget til rapporten på lige vis. Vi har lavet alle forberedelsesøvelser, skrevet koden og afsnittene i rapporten i fællesskab. Denne rapport beskriver vores arbejde og resultater.

Vi har valgt at kommentere hele koden på engelsk idet vi har gjort den tilgængelig Open Source på følgende side: <https://github.com/Lauszus/ReflexBall>.

Derudover kan dokumentation af kildekoden findes på siden: <http://lauszus.github.io/ReflexBall/>.



Figure 1: Et igangværende spil ReflexBall RALLY

Contents

Abstract	ii
Resumé	ii
Forord	iii
1 Introduktion	1
2 Specifikationer	1
3 Tidsplan	3
4 Struktur	3
4.1 Basics	3
4.2 Advanced	6
4.3 Brikker	9
4.4 Helhedsindtryk	15
4.5 Styring med rat	18
4.6 Rettelser og fintuning	20
4.7 Blokdiagram	21
4.8 Generelt flowchart	23
5 Verifikation	24
6 Brugermanual	26
7 Diskussion	29
8 Konklusion	31
A Introduktionsopgaver	32
A.1 Håndregningsopgaver	32
A.2 Programmeringsopgaver	33
A.3 ansi	34
A.4 time	35
B Java kode	37
B.1 BackslashEscapes	37
C C kode	39
C.1 ansi	39
C.2 ascii	42
C.3 asciidisplay	49
C.4 buttons	55
C.5 charset	55
C.6 gameport	58

C.7 LED	59
C.8 levels	62
C.9 lut	65
C.10 main	67
C.11 math	68
C.12 reflexball	69
C.13 time	81

1 Introduktion

Denne rapport dokumenterer de overvejelser vi har gjort os i forhold til design og struktur af Reflex Ball spillet.

Programmeringsprojektet har overordnet set været inddelt i to faser.

Første del, der inkluderede de fire første arbejdssage, blev brugt på at lave håndregningsøvelser omhandlende manipulation af hexadecimal- og bitrepræsenterede tal og flere programmeringsøvelser i C omhandlende hexadecimal- og bitmanipulation samt ansi-koder, driverhåndtering, vektorregning og styring af LED-display. Formålet med disse øvelser var samlet set at blive bedre til at håndtere hexadecimal- og bitrepræsenterede tal, og desuden at lære om/genopfriske pointers, headers og opsætning af hardware-drivere til microcontrolleren i C.

Anden del af projektet, der inkluderede de sidste ni arbejdssage, har vi brugt på først at designe Reflex Ball-spillet ud fra de i opgaven specificerede krav og derefter implementere udvidelser både på hardware- og softwareniveau, sideløbende med vi hele tiden har videreudviklet spillets 'engine', så spillet bruger mindre RAM og kører så fejlfrit som muligt, samtidig med at koden gradvis er blevet gjort mere fleksibel (ingen hardcode-stil) og mere overskuelig.

Som sidste led af anden del af projektet, har vi skrevet denne rapport.

2 Specifikationer

Nedenfor er vores forskellige delmål i projektet. Spillet skal have de nedenstående funktioner. Listen kan derfor ses som kravene til hvad vores spil skal kunne og er samtidig opdelt på en logisk måde så man kan starte fra toppen af og arbejde sig igennem delmålene indtil man har et færdigt spil der fungerer og ser ud efter vores hensigt. Delmålene er samtidig bygget op på en sådan måde, at skulle der ikke være tid til at nå det hele, har man stadigvæk et fuldt funktionelt spil. Det vil altså sige at vi arbejder fra basale funktionaliteter ud mod mere specifikke og detaljerede elementer.

Delmål Basics

- Detektere tastatur-input så keyboardet kan bruges til at styre striker
- Tegne banen
- Bevæge strikeren
- Bevægelig bold
- Bolden reflekteres af striker, loft og vægge
- Det skal detekteres når bolden er under strikeren (man er død)

Delmål advanced

- Liv-system, så man har 3 liv inden man bliver Game Over
- Pointsystem

- Ændring i boldhastighed som ens score stiger
- Internt 18.14 koordinatsystem, så boldens bevægelse kan laves som en enhedsvektor der roteres rundt
- Bolden skal starte i en tilfældig vinkel
- Striker-zoner, så boldens afbøjningsvinkel afhænger af hvor den rammes
- Stor bold, der fylder 4x2 monospace-karakterer.
- Forskellige sværhedsgrader
- Brikker og forskellige baner
- Achievements: Power-Up's og Power-Down's

Brikker

- Banen skal indeholde brikker
- Der skal være forskellige typer af brikker (forskelligt antal liv)
- Der skal være forskellige levels med brikker i forskellige mønstre
- Når bolden rammer brikker og kanter skal den reflekteres på en logisk måde så fysikken i spillet ser realistisk ud

Helhedsindtryk

- LED display med score, liv og levels
- Menu hvor man kan vælge sværhedsgrad
- Beskeder ved Game Over
- Besked hvis spillet vindes

Hardware

- Styring med intern hardware (knapper på microcontrolleren)
- Tilslutning af ekstern hardware (DEXXA Steering Wheel)

Vi har lagt en del vægt på at spillet skal se realistisk ud og at bolden, strikeren og brikkerne skal flytte sig på en realistisk måde. Vi vil forsøge at lave spillets fysik så dette er opfyldt, så bolden bevæger sig så naturligt og logisk som muligt.

3 Tidsplan

Fra starten af lagde vi en plan over hvornår vi ville udføre de delmål vi havde. Denne plan fremgår af figur 2. Som det kan ses var planen at starte med basic-delmålene og derfra bevæge os ud mod de mere avancerede som liv, hastighed, score, internt koordinatsystem, sværhedsgrader osv. Efterfølgende ville vi implementere et DEXXA Steering Wheel som controller til sjovere styring af spillet.

Efter dette ville vi begynde på hvad man kan kalde en finpudsning af spillet, dvs. ordentlig start menu og beskeder hvis spillet vindes eller tabes.

Efterfølgende ville vi lave achievements, altså power-up's og power-down's til spillet. Vi fandt lynhurtigt masse potentielle af disse, såsom at strikeren skulle kunne blive bredere/smallere, bolden skulle kunne klister til strikeren, bolden skulle kunne skydes direkte igennem brikkerne uden at blive reflekteret og størrelsen og hastigheden af bolden skulle kunne ændres osv. Vi havde rent faktisk så mange idéer, at kun tiden satte grænser for hvor mange achievements vi ville implementere. Vi valgte at sætte dette som det tredje sidste punkt i planen, da achievements skulle være en bonus til en allerede spilbart og udfordrende gameplay.

Oprindelige mål og plan

Mål	Fredag d. 14.	Mandag d. 17.	Tirsdag d. 18.	Onsdag d. 19.	Torsdag d. 20.	Fredag d. 21.	Mandag d. 24.	Tirsdag d. 25.	Onsdag d. 26.	Torsdag d. 27.	Fredag d. 28.
Delmål Basics											
Advanced (liv, hastighed, startvinkel)											
Brikker											
Display											
DEXXA Steering Wheel											
Start Menu, Game Over, Game Won											
Achievements (Power Up's og Down's)											
Retabler, test og fintuning af koden											
Rapport											

Figure 2: Tidsplan over de forskellige delmål og hvornår de skulle være færdige

4 Struktur

4.1 Basics

Da vi skulle planlægge projektet valgte vi efter en brainstorm at dele projektet op i en mængde delmål vi ville have implementeret. Delmålene i kategorien basics var dem vi skulle implementere for at have et fungerende spil. Vi fandt frem til at det skulle være følgende funktionalteter.

Keyboard input

Vi har valgt at man i basic implementeringen af spillet laver al styring vha. keyboardet, da knapperne på udviklings boardet var al for slidte og derfor virkede dårligt. Vi implementerede det sådan så man bevæger strikeren ved hjælp af piletasterne på tastaturet og sender bolden af sted ved space-tasten. Hvis man dør og bliver Game Over, startes spillet forfra ved tryk på space-tasten. Der er dog et 1000ms delay fra man dør til spillet kan startes forfra. På den måde kommer man ikke til at starte spillet forfra uden at opdage at man døde.

Banen

Banen har vi implementeret i et koordinatsystem på den måde at vi har x_1 - og x_2 -koordinater, svarende til banens bredde (x_1 = banens venstre væg, x_2 = banens højre væg) og y_1 - og y_2 -koordinater, svarende til banens højde (y_1 = loft, y_2 = 'gulv'). Vi arbejder i vores implementering med et venstredrejet koordinatsystem, hvor y -aksen er inverteret i forhold til et almindeligt koordinatsystem, så y_2 har altid en større værdi end y_1 .

Udnyttelse af Timer

Da vi ofte har brug for at kunne bestemme tiden siden sidste gang en bestemt funktion blev kaldt, opsatte vi Timer1 til at interrupte hvert 1ms. I dette interrupt forøgede vi en variabel `mscounter`. Denne er således et udtryk for hvor lang tid der er gået siden programmets start i millisekunder. Ved at kalde funktionen `millis()` der returner `mscounter` kunne vi således let bestemme hvor langt tid der var gået siden sidst funktionen blev kaldt hvis vi havde gemt den sidste tid i en variabel.

Bevægelig striker

Til strikeren har vi lavet et `struct`, der er givet ved strikerens x-værdi (yderst venstre karakter af strikeren), y-værdi, samt sin bredde. Strikeren er blevet implementeret på den måde at når den rykkes til venstre, tjekkes der om strikerens x-værdi, minus det antal felter den skal rykkes til venstre, er mindre end eller lig med banens venstre side (dvs. banens x_1 -værdi). Hvis dette er tilfældet befinder strikens sig ved dens venstre yderposition og sættes dermed til at være helt ude til venstre med venstre siden mod væggen. Hvis strikeren er ude i sin yderste venstre position er strikerens x-værdi nemlig 1 større end banens x_1 -værdi, strikeren x-værdi må således aldrig blive lig eller mindre end banens x_1 -værdi, da den således ville befinde sig uden for banens vægge.

Hvis strikerens x-værdi derimod er større end banens x_1 -værdi, trækkes der det antal felter den skal rykkes fra strikerens x-værdi. Derefter fjernes de felter strikeren har rykket væk fra og strikeren tegnes på ny.

Når strikeren bevæger sig til højre fungerer samme princip, bare hvor der tjekkes om `striker.x + striker.width-1+antal_ryk` er større eller lig med banens x_2 -værdi.

Hvis testen er positiv rykkes strikeren helt ud til væggen, hvis testen er negativ overskrives felterne det antal felter den skal rykke med mellemrum i venstre side. Derefter tegnes strikeren igen.

I denne første implementation af spillet blev strikeren bygget til først at rykke sig et monospace ad gangen, derefter satte vi det op til at strikeren bevægede sig to monopaces ad gangen. Dette var vi nødt til at gøre eftersom vi styrede strikeren med keyboardet og der i keyboardets hardware er en lavere grænse for maksimumsfrekvensen hvormed en knap bliver genaktiveret når knappen på keyboardet holdes nede, i forhold til microcontrolleren der har en højere maksimumsfrekvens. Her i Basics-implementationen byggede vi banen sådan så det passede med at strikeren altid kun rykkede to felter ad gangen indtil den kom ud i banens yderpositioner.

Bevægelig bold

Ved at aflæse timer funktionen `millis()` hver gang vi kalder funktionen `updateGame()`, kan vi bestemme hvor lang tid der er gået siden sidste gang vi opdaterede spillet. Hvis der således er gået længere end en bestemt værdi bevægede vi bolden og opdaterede resten af spil-logikken.

Her i basics implementationen hardcodede vi dette til at boldens x-værdi ændrede sig med plus 1 for hvert tick og boldens y-værdi ændrede sig med minus 1 for hvert tick, når bolden blev skudt afsted fra strikeren. Dette skyldes at vi jo arbejder med en inverteret y-akse i forhold til et almindeligt koordinatsystem. Det fungerer på den måt at vi har et `Ball struct` sådan så bolden hele tiden kender sin egen position i form af x- og y-koordinat, samt en vektor (den retning og hastighed den bevæger sig i). Så når bolden rykker sig, overskrives feltet som er boldens (x,y)-koordinat med et mellemrum (da bolden i basic implementationen kun fylder et enkelt felt) og derefter tillægges boldens vektor til boldens (x,y)-koordinat og den gentegnes.

Reflex-logik på kanter og hjørner

Hver gang bolden har rykket sig, inden den tegnes, laves en række tjek for at se om bolden har ramt en væg, et hjørne eller loftet. Bl.a. tjekkes om boldens x-koordinat bliver større end eller lig med banens x_2 -koordinat. Hvis dette er tilfældet ved vi at bolden har ramt højre væg af banen og bolden tegnes ikke umiddelbart. Det der i stedet skal ske er at boldens vektor skal trækkes fra boldens nye (x,y)-koordinat, sådan så bolden er tilbage på den plads den stod på inden den ramte ind i væggen, derefter skal boldens vektors x-komposant inverteres og vektoren skal lægges til boldens position inden bolden igen bliver tegnet på næste tick. Det var netop sådan vi implementerede kant-logikken til at starte med. Efterfølgende ændrede vi dette til at hvis bolden rammer højre væg, så trækkes 2 fra boldens x-koordinat og boldens vektors x-komposant inverteres. Effekten af de to implementeringer er den samme, forskellen er bare at den sidste kræver lidt færre beregninger og den er derfor mere effektiv.

Når bolden rammer venstre væg fungerer der på præcis samme måde, bortset fra at tjekket er om boldens x-koordinat bliver mindre eller lig med banens x_1 -koordinat og hvis dette er tilfældet inverteres vektorens x-komposant og der ligges 2 til boldens x-koordinat.

Loftet fungerer også mere eller mindre på samme måde. Her tjekkes bare om boldens y-koordinat kommer under banens y_1 -koordinat (husk y-aksen er inverteret). Hvis dette er tilfældet inverteres vektorens y-komposant og der ligges to til boldens y-koordinat inden den tegnes igen.

Al refleks-logikken tjekkes med if-sætninger, sådan så hver gang bolden har bevæget sig tjekkes først om den har ramt banens højre side, derefter banens venstre side og derefter loftet, inden den gentegnes. Så hvis fx bolden rammer højre øvre hjørne sker der det at refleks-logikken først detekterer at banens højre side er overskredet og derfor inverteres vektorens x-komposant og boldens x-koordinat formindskes med 2. Efterfølgende detekteres at bolden har ramt loftet og derfor tillægges to til boldens y-koordinat, samtidig med at vektorens y-komposant inverteres. Når bolden tegnes igen er den tilbage på feltet den var på 2 gange forinden den ramte væggen, med både x- og y-komposant af vektoren inverteret, hvilket er det ønskede resultat.

Game Start og Game Over

Når spillet starter kan man rykke strikeren rundt som normalt. Her følger bolden med sådan så den bliver ved med at være på midten af strikeren. Det fungerer på den måde at når variablen `gameStarted` er 0 så gentegnes både strikeren og bolden hver gang man rykker strikeren. Både boldens og strikerens x-koordinat sættes herefter. På samme måde som når strikeren rykkes normalt laves de samme check for om strikeren har bevæget sig ud over banens vægge. Når der så trykkes på space-tasten på tastaturet skydes bolden afsted og `gameStarted` sættes til 1, og når strikeren nu rykkes er det kun strikeren der gentegnes.

På samme måde som bolden hver gang den har bevæget sig, inden den tegnes, testes for om den har ramt vægge eller loft, tjekkes også om boldens y-koordinat lig strikerens y_2 -koordinat. Hvis denne test er sand betyder det at bolden ligger på feltet lige i strikeren. Her testes så om boldens x-koordinat ligger indenfor strikeren, dvs. imellem `striker.x` og `striker.x + striker.width`. Hvis dette er tilfældet bliver bolden skudt op igen, hvilket fungerer ved at boldens vektors y-komposant bliver inverteret, på samme måde som når bolden rammer loftet. Hvis bolden er uden for strikeren bliver variablen `alive` sat til 0 og der vises `Game Over!` midt på skærmen. I denne tilstand kan man ikke bevæge strikeren. Dette har vi implementeret fordi man ellers kan 'køre bolden over', da bolden jo ligger nede på gulvet, hvilket ikke ser så grafisk pænt ud. Når der trykkes space igen gentegnes hele spillet banen og og variablerne `gameStarted` sættes til 0 og `alive` sættes til 1.

4.2 Advanced

Efter alle basics var færdig-implementeret, havde vi planlagt at lave en række udvidelser til spillet. Disse omfatter dels implementering af liv og pointsystem, men også videreudviklinger af basic-funktionaliteterne, sådan så deres bagvedliggende virkemåde gøres mere avanceret. Advanced inkluderer følgende.

Liv

Som i ethvert andet arkade-spil har vi implementeret liv, sådan så man starter med tre og bliver Game Over, når der er nul liv tilbage. Som en ekstra feature har vi implementeret at man får 2 ekstra liv, når en bane er gennemført.

Pointsystem

Scoren er opbygget sådan så det giver 1 point hver gang man rammer en brik. Vi har valgt at det ikke skal give nogle point når bolden rammer strikeren, da vi ikke synes det skal give point 'ikke' at ramme brikkerne.

Internt 18.14 koordinat-system

I basic-implementeringen af spillet satte vi bolden til at rykke sig et bestem antal monospaces hver gang den bevægede sig, altså en simpel vektor-implementering. Men efter denne udvidelse er hele banens indre struktur blevet gentænkt sådan så bolden bevæger sig som en enhedsvektor i et internt koordinatsystem. Som nævnt i basics har vi lavet et `Ball struct` sådan så bolden hele tiden kender sine egne (x,y)-koordinater, samt nu dens egen enhedsvektor (altså

retning hvori den bevæger sig). I denne udvidelse valgte vi dog at angive disse værdier i et 18.14 format, så vi får en meget højere opløsning for boldens bane.

Når bolden bevæger sig sker der det at dens vektors (x,y)-koordinat lægges til boldens (x,y)-koordinat. Derefter tjekkes om bolden er død eller har ramt en brik, en væg, et hjørne, strikeren eller loftet. Hvis intet af dette er tilfældet, slettes boldens gamle koordinater ved at der på disse felter tegnes blanke mellemrum. Derefter afrundes boldens (x,y)-koordinater til nærmeste heltal ved at kigge på 1. bit efter kommaet, det vil sige boldens x- hhv. y-koordinats 13. bit (da det tænkte komma er sat mellem 14. og 13. bit). Hvis denne er 0 rundes ned og ellers rundes op. Nu kendes boldens (x,y)-koordinat i heltal og den kan derfor indtegnes på banen.

Vilkårlig startvinkel

Når bolden skydes af bliver den sendt afsted i en vilkårlig vinkel på mellem 45 og 135 grader. Det vil sige lodret op fra strikeren plus minus op til 45 grader. Dette er implementeret for at man ikke kan time startvinklen så man er sikker på den altid rammer et bestemt sted.

Striker-zoner

Strikeren er opbygget sådan så den altid skal bestå af et lige antal felter. Dens midterste venstre felt har en afbøjningsvinkel på indgangsvinkel plus 0 grader, og dens yderste venstre felt har en afbøjningsvinkel på indgangsvinkel plus ca. 45 grader. Hvert felt mellem det midterste venstre til det yderste venstre felt, har så en stigende afbøjningsvinkel, hvor den vinkel der bliver lagt til hvert felt er 45 grader divideret med antallet af felter fra det midterste venstre (eksklusiv) til det yderste venstre (eksklusiv). Højre-siden af strikeren fungerer på præcis samme måde, bare med omvendt fortegn, sådan så afbøjningsvinklen på midterste højre felt er lig indgangsvinklen og afbøjningsvinklen på yderste højre felt er lig indgangsvinklen minus 45 grader.

Alt dette er lavet ved at boldens reflekslogik er gentænkt sådan så dens vektors x- og y-komposant nu bare er lig vektorens sinus hhv. cosinus-værdi, da boldens vektor nu hele tiden er en enhedsvektor. Når bolden skal reflekteres kan vi så bruge den sinus look-up table, som vi fik givet som led i en af introduktionsopgaverne.

Strikeren er lavet på denne måde så dens bredde er meget fleksibel. Fx bruges der forskellige striker-størrelser på spillets forskellige sværhedsgrader og dette er så bare implementeret ved at ændre på størrelsen af **striker.width**. Så sørger spillet selv for at inddelte Striker-zonerne.

Figur 3 viser hvordan strikeren er opdelt i zoner for en striker med længden 10 og en bold med længden 4. Det ses hvis bare boldens ene ende er inde over midten, så afbøjes den ikke. Hvis bolden derimod kun rører med den ene kant, så afbøjes den med den maksimale vinkel ± 42.2 grader.

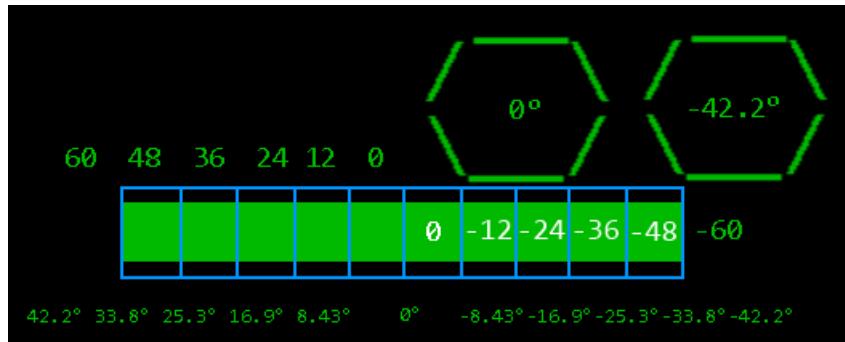


Figure 3: Oversigt over striker zoner for en striker der er med længden 10 og en bold med længden 4

Der er desuden implementeret en minimums-afbøjningsvinkel på 15 grader og en maksimums-afbøjningsvinkel på 165 grader. På samme måde som når bolden rammer noget andet, bitshiftes boldens vektors x-komposant 1 til højre når bolden rammer strikeren, sådan så boldens vektor igen er en enhedsvektor. Derefter beregnes boldens afbøjningsvinkel. Hvis dennes y-værdi er mindre end 0,25, svarende til sinus til ca. $15/165$ grader, ved vi at boldens vinkel er enten under 15 grader eller over 165 grader efter afbøjning og afbøjningsvinklen bliver derfor sat til 15 grader hvis `ball.vector.x` er positiv og til 165 grader hvis `ball.vector.x` er negativ.

Stor bold

I `Ball struct` er der ud over x- og y-koordinater og en vektor to andre variable som er bredde (`width`), højde (`height`). Den bold vi bruger i spillet har en højde på 2 og en bredde på 4. Dette komplicerer en del ting eftersom bolden nu både kan ramme alle objekter (strikere, vægge, loft, brikker) på mange flere måder. Desuden er der en grænse for hvor hurtigt uarten kan arbejde, selvom vi har sat baud-raten op til maksimum (115200), dette kan godt gå hen og blive lidt problematisk, da uarten ved en vis hastighed af bolden ikke kan nå at slette og gentegne den ordentligt. Som løsning på dette, har vi sørget for at bolden ved en vis hastighed kun tegnes hver anden gang, hvilket letter UART'ens arbejde en smule. Læs mere om dette under afsnittet om **Rettelser og fintuning**.

Sværhedsgrader og ændring i boldens hastighed

I spillet er der implementeret fire forskellige sværhedsgrader. Disse er Easy, Medium, Hard og Chuck Norris. Forskellen på sværhedsgraderne er hvor bred strikeren er, hvor hurtig starthastighed bolden har og hvor mange point man skal have før at boldens hastighed bliver sat op. På Easy er `striker.width = 30` og boldens starthastighed er 40ms. Det vil sige der går 40ms fra bolden bliver tegnet, til den tegnes igen. På Medium er `striker.width = 20` og boldens starthastighed 40ms, på Hard er `striker.width = 10` og boldens starthastighed 40ms og på Chuck Norris er `striker.width = 4` og boldens starthastighed er 10ms.

Som man får flere point sættes boldens hastighed op. På Easy bliver delayet mellem to på hinanden følgende gange hvor bolden tegnes, sat 1ms ned for hvert 10 point man får. På Medium ved hvert 5. point man får og på Hard ved hvert 2. point man får. Når delayet mellem bold-aftegningerne kommer under 20 ms begynder bolden kun at blive tegnet hver

anden gang for at UART'en kan følge med. Minimumshastigheden som bolden kan bevæge sig i er 10ms delay mellem aftegningerne. Grænsen er sat her fordi spillet ellers bliver for umuligt at gennemføre. Da Chuck Norris-sværhedsgraden starter på 10ms delay øges boldhastigheden altså ikke uanset hvor mange point man får.

Når der skiftes bane bliver boldhastigheden resat til starthastigheden ved den givne sværhedsgrad. Dvs. 40ms delay på Easy, Medium og Hard og 10ms på Chuck Norris.

Brikker og baner

Som en del af de avancerede mål havde vi at lave brikker, så gameplayet bliver sjovere. Se mere om implementering af brikker, forskellige brik-typer og forskellige baner i afsnit **3.3**.

4.3 Brikker

Formålet med at implementere brikker i spillet var at give gameplayet en helt ny dimension.

Tegne Brikker.

For lettest at kunne holde styr på brikkerne har vi lavet en struct, `Brick` der repræsenterer en brik. Den indeholder position i x- og y-koordinater, bredde, højde og brikkens liv. Brikkerne i en bane er gemt i et array:

`Brick bricks[BRICK_TABLE_HEIGHT][BRICK_TABLE_WIDTH]`. Når banen initialiseres gennemløbes dette array og hver brik tildeles koordinater, bredde, højde og liv. Efter dette tegnes brikken. Hvis brikkens har 0 liv svarer det til at der ikke er en brik. Ved at give brikkerne i array'et forskellige antal liv kan man lave mønstre med brikkerne så man på den måde har flere forskellige baner i spillet. På Figur 4 er der vist hvordan brikker med forskelligt antal liv er tegnet. Jo flere liv de har, des mere solide tegnes de. Som et lille twist i gameplayet har vi valgt at gøre brikker med mere end fire liv usynlige så de pludseligt kan dukke op når man rammer dem.



Figure 4: Brikker med forskelligt antal liv tegnes forskelligt

Baner gemt i et array i ROM'en.

Vi har valgt at hardcode brikkernes bredde og højde for at spare plads når vi gemmer banerne i ROM'en. På denne måde kan en bane i ROM'en gemmes i arrayet

`unsigned char rom levels[6][BRICK_TABLE_HEIGHT][BRICK_TABLE_WIDTH]`. Dette array er et tredimensionalt array af chars der er gemt i ROM'en. Det indeholder 6 "lag" der hver indeholder de todimensionale data til en bane. Værdien af hver char er antallet af liv den tilsvarende brik får. Når banen initialiseres, løbes dette array og arrayet `bricks[BRICK_TABLE_HEIGHT][BRICK_TABLE_WIDTH]` igennem og brikkerne i `bricks` får det antal liv der står i `levels` arrayet. Den dynamiske hukommelse i mikroprocessoren indeholder således kun et array af brikker der svarer til den aktuelle bane. Et eksempel på hvordan en bane er gemt i arrayet `levels` er vist nedenfor.

```

9  {
10   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
11   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
12   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
13   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
14   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
15   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
16   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
17   { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1 },
18   { 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 2, 0, 3 },
19   { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1 },
20   { 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 2, 0, 3 },
21   { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1 },
22   { 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 2, 0, 3 },
23   { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1 },
24   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
25   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
26   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
27   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
28   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
29   { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
30 },

```

Banen der svarer til det ovenstående array, er vist på Figur 5.

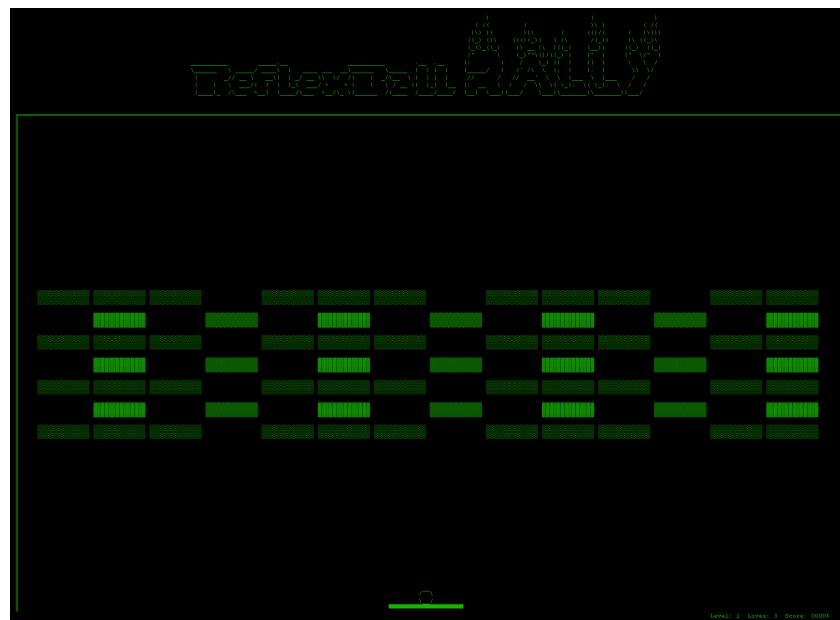


Figure 5: Level 1 i spillet

Tjekke om man rammer brikker.

Den helt store udfordring med brikkerne var at tjekke om bolden ramte dem. Ved hver eneste iteration af spillet gennemløber vi arrayet `bricks[][]` og for alle brikker med mere end 0 liv tjekker vi om bolden har ramt og i givet fald hvordan.

Når spillet itereres og bolden flyttes tjekker vi først om bolden rammer brikkerne før vi tegner den. Hvis bolden har ramt en brik reflekteres den og får sin nye position før den tegnes. Hvis bolden har ramt en brik, er boldens koordinater altså inde i brikken når vi tjekker.

For at bolden skulle bevæge sig lige hurtigt i begge retninger på skærmen har vi lavet x-komposanten af boldens vektor dobbelt så stor, da der var næsten præcis dobbelt så stor oplosning i x-aksen i forhold til y-aksen. Det giver nogle ekstra udfordringer når bolden rammer brikkerne. Når bolden kan bevæge sig 2 karakterer i x-retningen kan den ramme brikkerne fra siden på en række forskellige måder. Disse er vist for højre side af brikken på Figur 6. Når vi har itereret spillet og tjekker boldens position kan den på x-aksen befinde sig både en eller to karakterer inde i brikken. Den kan således ramme brikken på seks forskellige måder som vist på ovennævnte figur. Det er vigtigt at tage højde for at bolden kan ramme to karakterer ind i brikken, når man tjekker om bolden har ramt brikken midtpå fra siden. Hvis man ikke tager højde for dette, vil bolden bevæge sig igennem brikken mens den reflekteres af toppe og bunden.

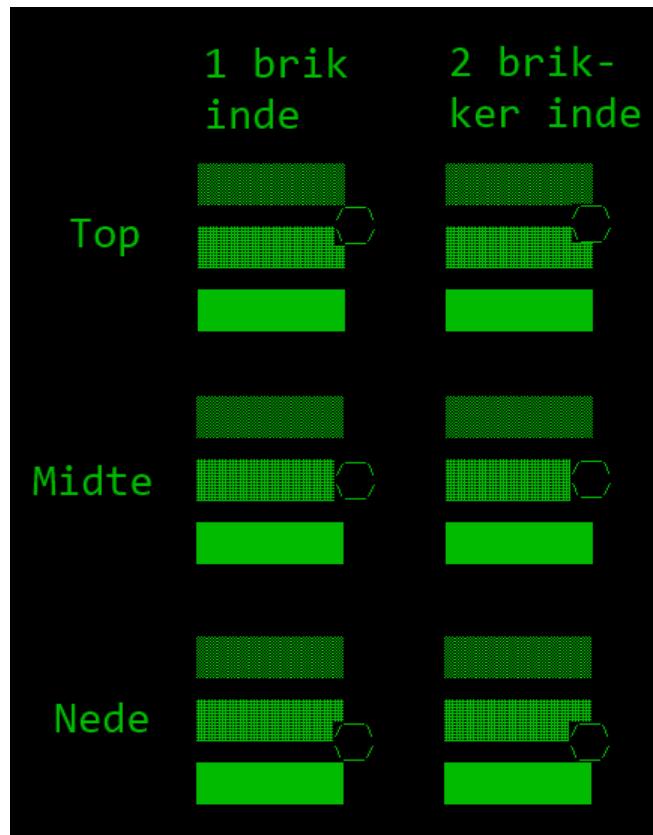


Figure 6: Forskellige måder bolden kan ramme fra siden

For at give et godt gameplay har vi besluttet at bolden fortrinsvis skal reflekteres op og ned så brugeren kan få den. Dette er vist på Figur 7. Når bolden rammer brikken fra toppen og er 2 karakterer inde i brikken vil den blive reflekteret som vist på figuren. Hvis bolden havde ramt på samme måde, men var kommet nedefra var den blevet reflekteret som om den havde ramt siden af brikken.

Det der er vist på denne figur er, borset fra de omgivende brikker, samme situation som på Figur 6, Top, 2 brikker inde.

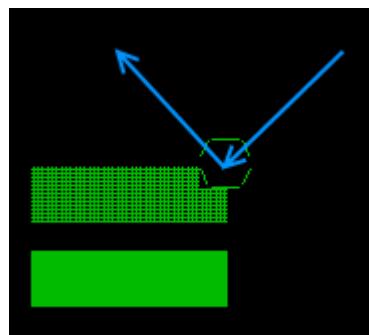


Figure 7: Forskellige måder bolden kan ramme fra siden

Hjørnerne.

Et specialtilfælde er når en brik bliver ramt lige på hjørnet, en karakter inde. Her er tre mulige situationer som vist på Figur 8. Den eneste forskel på de tre tilfælde er hvor de omgivende brikker er positioneret. Hvis bolden rammer højre hjørne skråt oppefra og der er en brik til højre for skal bolden reflekteres op igen som vist øverst på figuren. Hvis der derimod er en brik ovenover vil det ikke give nogen mening at reflektere brikken op og den skal derfor reflekteres til siden som vist midt på figuren. Sidste situation er der ikke er brikker hverken over eller ved siden af og her skal brikken reflekteres lige tilbage.

Fler brikker ramt.

Som det kan ses på Figur 8, øverst, kan bolden ramme flere brikker af gangen. Når dette sker, skal man tjekke om der bare findes én brik på banen, hvor der er en konflikt med at reflektere bolden i en bestemt retning. Hvis der er det må brikken ikke reflekteres i denne retning.

Hvis man så på en situation hvor bolden kom oppefra og ramte øverste højre hjørne af en brik der havde en brik både til højre og ovenover den (en kombination af det øverste og midterste billede på Figur 8) vil der både være en konflikt både i forhold til at reflektere bolden op og til siden. I denne specialsituation reflekteres bolden i begge retninger ligesom på det nederste billede i figuren.

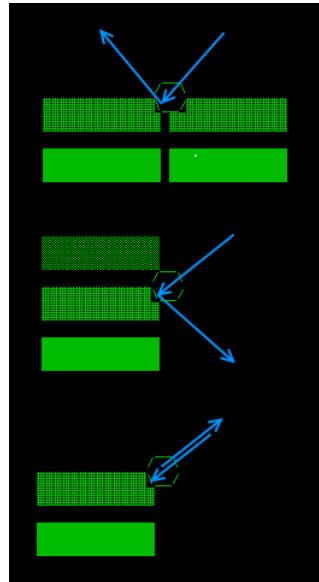


Figure 8: Boldens bevægelse når den rammer et hjørne

Briklogikken.

Briklogikken er selve spillets game engine der styrer hvordan det opfører sig. Det er her alle reglerne for interaktion mellem bolden og brikkerne ligger. Når spillet itereres tjekker briklogikken om iterationen er gyldig. Et flowchart for briklogikken er vist på Figur 9. De blå bokse på figuren repræsenterer funktionskald eller operationer på boldens vektors x- og y-koordinater. De grønne bokse er variable der bliver forøget eller sat (flagene dontDeflectX og dontDeflectY der bliver sat). De orange bokse er tests i form af if-statements. Med de grå bokse har vi vist alle indgange til, og udgange fra, det indlejrede for-loop der tjekker alle brikkerne¹. De to gule bokse repræsenterer de samme linjer kode. Denne kode er en blanding af variable der bliver sat og funktioner der bliver kaldt hver gang logikken har gennemløbet en brik.

På Figur 9 kan det ses at der er adskillige steder hvor flagene `dontDeflectX` og `dontDeflectY` kan blive sat. Disse flag sættes hvis bolden ikke må blive reflekteret i en retning. De test der går forud for at et flag sættes, er altså reglerne for hvornår bolden ikke må reflekteres i en bestemt retning. Når bolden er itereret (øverste venstre hjørne på Figuren) kaldes briklogikken. Den funktion der itererer boldens position er `iterate()` der kan findes på linje 375 i `reflexball.c` der er vedlagt i Appendix C. Denne funktion kalder `checkIteration(x,y)` på linje 145 i samme fil som er selve briklogikken. Briklogikken tester om iterationen er gyldig. Hvis den er det, tegnes bolden med `drawBigBall()`. Er den ikke det, reflekteres bolden hvorefter den tegnes.

¹Da et for-loop egentlig bare er noget kode der køres så længe en test er sand, kunne det tegnes med en orange boks og en pil der fører tilbage til den. For at forøge overskueligheden og holde antallet af test-bokse og lange pile i diagrammet nede, har vi dog valgt, at undlade at lave det på denne måde.

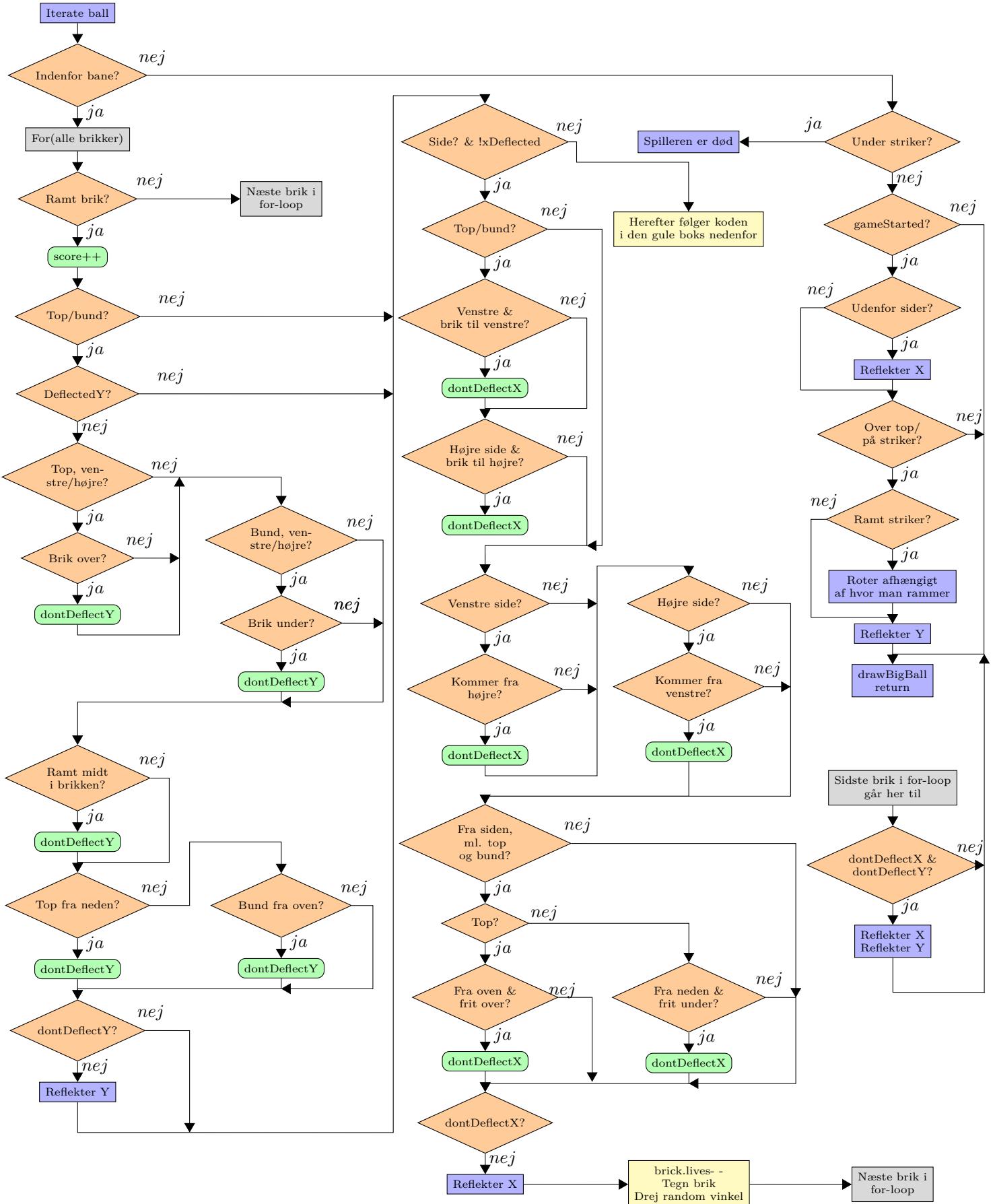


Figure 9: Flowchart for briklogikken der styrer interaktionen mellem brikkerne og bolden

4.4 Helhedsindtryk

LED display med score, liv og levels

Udover at vise brugerens score, liv og nuværende level på skærmen, valgte vi også at bruge boardets indbyggede LED display til at vise dette. På den måde kan brugeren se hans score når spillet kører og hans liv hver gang han dør eller går et level up.

For at vise de forskellige karakterer på displayet opsatte vi Timer2 til at inerrupte hver 500us, den kalder derefter en funktion der sørger for at multiplekse LED displayet, så hurtigt at det virker som om alle LED'er hele tiden er tændt. Oprindeligt kaldte vi opdaterings funktionen fra vores main-loop, men det bevirkede i at displayet blinkede, da der var forholdsvis lange pauser hver gang den skulle opdaterer spillet.

Da funktionen bliver kaldt inde i et interrupt er det nødvendigt at sikre sig at denne er så kort som mulig, så programmet ikke bruger for lang tid på at eksekvere interruptet. Dette sikrede vi ved kun at gemme de første 5 karakterer i en video-buffer og derefter læse fra den ved at læse direkte fra en specifik adresse i microcontrollerens memory. Derved undgår man at skulle flytte rundt på video-bufferen hele tiden, der ville tage meget lang tid at eksekverer. Man skal dog selv holde styr på at man ikke læser fra et andet sted i hukommelsen.

Koden til opdaterings-funktionen kan ses nedenfor.

```

117 PGOUT = (PGOUT & (1 << 7)) | *(&videoBuffer[0][0] + digit*6 + column + index);
118 PEOUT |= 0x1F; // Set all cathodes high
119 PEOUT &= ~(1 << (4-column)); // Set one cathodes low decided by column
120
121 clockLed(digit);
122 if (++digit == 4) {
123     digit = 0;
124     if (++column == 5) {
125         column = 0;
126     if (++delayCounter == SCROLL_SPEED && stringLength > 4) { // We don't have to scroll the text if there is less
127         than five characters
128         delayCounter = 0;
129         if (++index > 5) {
130             index = 0;
131             moveVideoBuffer();
132         }
133     }
134 }
```

Det ses i linje 1, at vi blot lægger et tal til video-bufferens adresse, dette tager således utroligt lidt tid og dermed opstår der ingen problemer ved at kalde den i et interrupt. Som det ses i linje 14 skal den dog stadig flytte bufferen hver gang den kommer til en ny karakter. Denne kode kan ses i afsnittet LED i appendiks.

Udover at kunne scolle den samme tekst havde vi også brug for at kunne scolle en besked én gang på displayet - dette bruger vi fx hver gang man dør eller går til næste level.

Funktionen `LEDRunOnce()` kan ses nedenfor.

```

89 void LEDRunOnce(char *firstString, char* secondString) { // Used to scroll the first string once and then show the
  second string afterwards
90   LEDsetString(firstString);
91   runOnce = 1;
92   pSecondString = secondString; // We will save the location of the second string
93 }
```

Den virker på den måde at den sætter den første streng på normal vis, den sætter dog flaget `runOnce()` til 1 og derefter sætter den en pointer til den anden streng.

Et udsnit af funktionen `moveVideoBuffer()` kan ses nedenfor.

```

106 pString++;
107 if (*pString == '\0') { // Check if we have reached the end of the string
108   if (runOnce) { // This wil actually abort when it loads the last character in the 5th digit, so you have to put a space
    in end of the sentence
109   runOnce = 0;
110   LEDsetString(pSecondString);
111 } else
112   pString -= stringLength;
113 }
```

Når den kommer til null-terminatoren i strengen tjekker den om det var funktionen `LEDRunOnce()` der blev kaldt. Hvis dette var tilfældet kalder den `LEDsetString()` med pointen til den anden streng som argument.

Menu hvor man kan vælge sværhedsgrad

For at kunne variere spillets sværhedsgrad implementerede vi et menu system i starten af spillet hvor brugeren kan vælge den ønskede sværhedsgrad. Et screenshot af menuen kan ses i se figur 10. Menuen står og kører i et while-loop indtil brugeren har valgt en sværhedsgrad.



Figure 10: Screenshot af menuen

Beskeder ved Game Over

Når spilleren dør vises en fast tekst "GameOver", samt et vilkårligt citat. I alt er der 8 forskellige citater som kan vises når man dør. En eksempel på dette kan ses i figur 11.



Figure 11: Eksempel på screenshot game over tekst

For at kunne vælge en tilfældig tekst benytter vi os af at funktionen `millis()` nederste bits ændrer sig meget hurtigt. Ved at læse de tre nederste bits kan vi således få et "tilfældigt" tal mellem 0-7. Den er dog ikke helt tilfældig, men i stedet det man kalder pseudo-tilfældig - dette er dog tilstrækkeligt i dette tilfælde.

Det udsnit af funktionen `showGameOver()` der læser `millis()` kan ses nedenfor.

```
35 switch (millis() & 0x7) { // Pseudo random number from 0-7
```

Besked hvis spillet vindes

Hvis man er så heldig at vinde spillet vises der en lykønskning samt en opkvikker i form af en flot præmiepige, da dette er ægte klassisk japansk arkade coutume.

Figur 12 viser et screenshot ved gennemførsel af spillet.

Implementering af ASCII-Art

For let at kunne generere et array der kan sættes direkte ind i C-koden, lavede vi et Java-program der kan åbne et billede i ASCII-art og indsætte de nødvendige escape backslashes i filen samt bestemme længden og højden af array'et. Koden til programmet kan ses i afsnittet BackslashEscapes i appendiks.

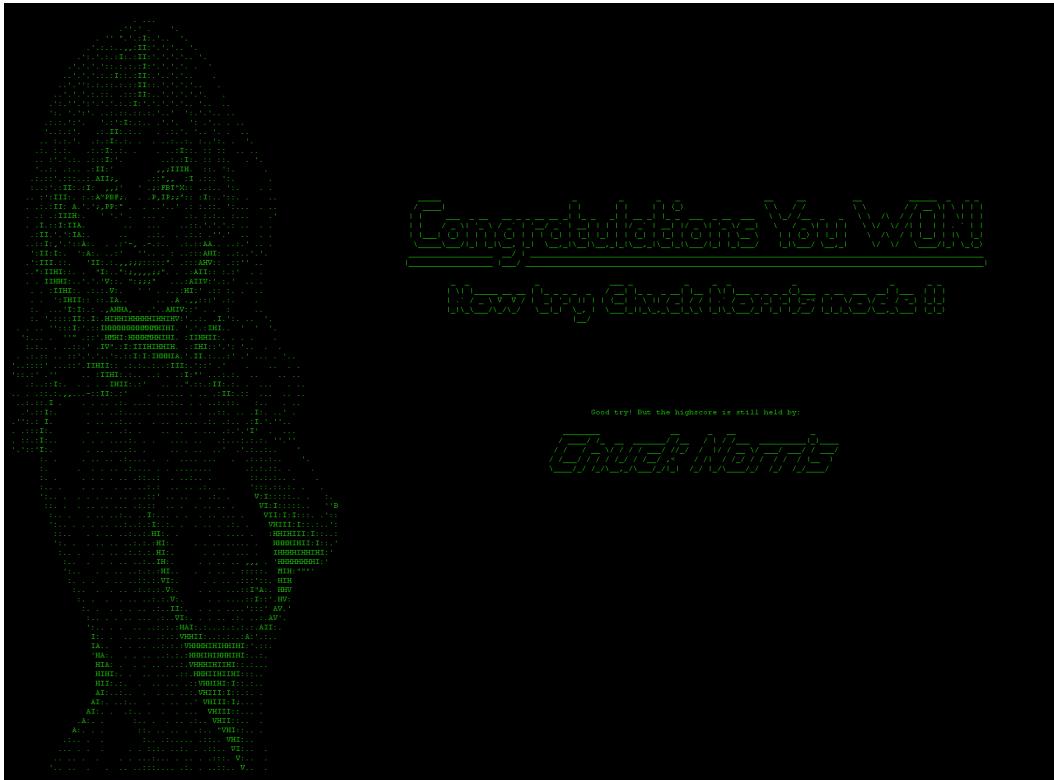


Figure 12: Screenshot ved gennemførsel af spillet

4.5 Styring med rat

Som et delmål fandt vi hurtigt ud af vi ville have et DEXXA Steering Wheel sluttet til microcontrolleren, sådan så styringen af strikeren var væsentlig bedre i forhold til keyboardet eller udviklings boardets indbyggede knapper.

Gameport adapter

For at kunne koble DEXXA Steering Wheel'et til vores udviklingsboard var det dog nødvendigt at lave en lille adapter. På den måde kan vi aflæse alle digitale knapper samt de to potentiometre inde i rattet og pedalerne - vi bruger dog ikke pedalerne i spillet.

De digitale knapper fungerer ved at de er forbundet til en række knapper der svæver når de ikke holdes i bund. Hvis knappen holdes nede kortsluttes den til GND. Ved at forbinde indgangen til knappen til en pullup modstand og derefter forbinde indgangen til en indgang på microcontrolleren kan man således aflæse knapperne. Udover at forbinde en pullup modstand til indgangen forbandt vi også en 100nF kondensator for at modvirke debouncing.

Rattet og pedalerne fungerer en smule anderledes i det de er forbundet til et $100\text{k}\Omega$ potentiometer. Potentiometerets ene ben er forbundet til VCC, mens det andet er forbundet til en udgang på Gameporten. Ved at forbinde denne udgang til en $100\text{k}\Omega$ modstand der er forbundet til GND har man således dannet en spændingsdeler vha. de to modstande. Ved at måle

spændingen vha. ADC'en i microcontrolleren kan man således bestemme positionen af rattet og pedalerne.

Koden til at læse en analog indgang kan ses nedenfor.

```

6 unsigned int readADC(unsigned char channel) { // Read a specific analog channel
7     unsigned char inHigh, inLow;
8     unsigned int ADC_data;
9
10    ADCCTL = 0x80 | 0x20 | (channel & 0x0F); // Enable ADC on the selected channel and use external voltage (3.3V) as
11    VREF
12    while (ADCCTL & 0x80); // Wait for conversion to be completed
13    inHigh = ADCCD_H; // ADC high byte
14    inLow = ADCCD_L; // ADC low low byte
15    ADC_data = ((unsigned int)inHigh << 2) | inLow >> 6; // ADC output word
16
17    return ADC_data;
}
```

På linje 5 opsættes ADC'en til at starte målingen på den pågældende input. Derudover benytter vi os af en ekstern spændings reference på 3.3V. I linje 6 venter koden på at ADC konverteringen er færdig, når dette sker cleares bit 7 og koden aflæser nu de to registre og returnerer spændingen som en 10-bit værdi mellem 0-1023.

Først lavede vi lineær model for rattet, men det viste sig hurtigt at den ikke var lineær, derfor lavede vi blot inddelingen manuelt vha. en række if-sætning.

Den samlede koden til Gameport adapteren kan ses i gameport i appendiks.

Et pinout til Gameport'en samt dybere forklaring kan ses på følgende side ².

Figur 13 nedenfor viser hvordan Gameport adapteren er forbundet til udviklingsboardet.

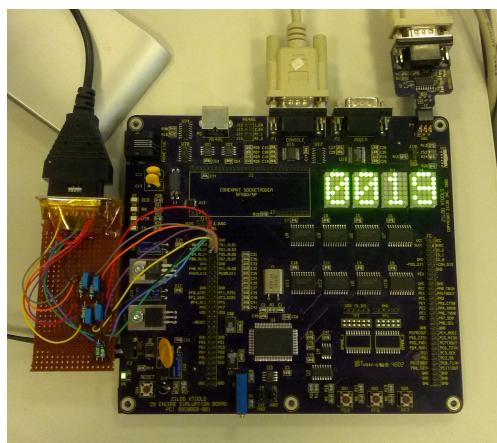


Figure 13: Billede af Gameport adapteren forbundet til udviklings boardet

²http://pinouts.ru/Inputs/GameportPC_pinout.shtml

Styring med intern hardware og keyboard

Udover at kunne styre spillet vha. rattet kan man også styre spillet vha. computerens keyboard eller udviklings boardets knapper. Spillet er dog tiltænkt at skulle styres med rattet, da det giver en meget bedre styringsfølsomhed og dermed en bedre spilleoplevelse.

4.6 Rettelser og fintuning

Bolden bevæger sig med samme hastighed i både x- og y-retning

Vi havde et problem med at det på skærmen lignede at boldens hastighed variede meget afhængigt af om dens vektor var relativt vandret (lille absolut y-komposant) eller relativt lodret (lille x-komposant). Dette skyldes at monospace-karakteren er meget tæt på at være dobbelt så høj som den er bred. Vores løsningen på dette var at gøre boldens vektors x-komposant dobbelt så stor, ved at bitshifte den 1 til venstre. På den måde er bolden ikke længere en enhedsvektor med mindre dens x-komposant er nul.

Når bolden så rammer ind i noget bitshiftes `ball.vector.x` 1 til højre (divideres med to), for at gøre boldens vektor til enhedsvektor igen, sådan så der ikke opstår rod når boldens vektor roteres.

Skift til Putty Terminal

Som en del af de avancerede mål, skiftede vi fra Hyper Terminal over til Putty Terminal. Den eneste grund til dette er man i Putty Terminal kan lave meget større oplosning, sådan så spillet ser flottere ud og kan spilles som et fullscreen-spil. De eneste komplikationer der var forbundet med dette var at nogle få af ASCII-karaktererne blev tegnet på en anden måde i Putty end tiltænkt, selvom både Hyper og Putty terminalerne bruger charset 850. Løsningen på dette var bare at bruge nogle andre ASCII-karakterer.

Implementering af en lille smule vilkårlighed ved hver afbøjning

Som i professionelle computerspil, har vi sørget for at implementere en mindre vilkårlighed i hvor meget bolden afbøjes hver gang den roteres, dvs. både når den rammer striker, brikker, loft eller vægge. Bolden får da sin almindelige beregnede afbøjningsvinkel med en adderet pseudovilkårlig rotation på mellem minus 3 og plus 4 ud af 512. Det vil sige en vinkel på mellem ca. minus 2,1 og plus 2,8 grader. Dette er en vinkel lille nok til man med øjet aldrig ved ligge mærke til afbøjningsvinklen ikke er helt lig med indgangsvinklen på brikker, loft og vægge, men samtidig stor nok til at bolden ikke bliver reflekteret rundt sådan at den er fanget i et fast mønster.

Justering af hvor ofte bolden tegnes

Som nævnt i afsnit **3.2** havde vi lidt problemer med at uarten ikke altid kunne tegne bolden hurtigt nok, efter vi lavede bolden til at fylde 4x2 monospace-karakterer. Vi oplevede at hvis delayet mellem aftegningerne af bolden blev for lille, så kunne man risikere ofte ikke at kunne se bolden, da vores kode jo er lavet på den måde at bolden først slettes, derefter bevæger sig

og så tegnes igen. Altså uarten kunne ikke nå at gennemføre hele processen med det resultat at boldene kun slettedes og ikke blev tegnet ordentligt op igen. Desuden opleve vi flere gange at terminalen frøs, hvis vi prøvede at tegne bolden for hurtigt.

Som en løsning på dette justerede vi hastigheden sådan så når der er under 20ms delayet mellem aftegningerne af bolden, så tegnes den kun hver anden gang. Dette letter arbejdet for uarten nok til at det bolden ser ordentlig ud hele tiden. Vi eksperimenterede også med kun at tegne bolden hver 3 gang, men det ser for forstyrrende ud for øjet, så vi fjernede det igen.

4.7 Blokdiagram

De tre lag vores program er opdelt i, er vist på Figur 14. Application Layer er de funktioner der er specifikke for vores program. Application Interface Layer indeholder generelle funktioner og Hardware Abstraction Layer indeholder de funktioner der er specifikke for vores hardware. Skulle man derfor implementere et andet program på samme mikroprocessor kunne man genbruge funktionerne fra Hardware Abstraction Layer og fra Application Layer. Skulle man derimod lave samme spil på en anden processorarkitektur kunne man genbruge funktionerne i Application Layer og Application Interface Layer. Skulle man lave et nyt program på en ny arkitektur kunne man kun genbruge funktionerne fra Application Interface Layer.

Vi har delt funktionerne op så Hardware Abstraction Layer indeholder de funktioner der skriver til og læser fra hardwaren. Det er funktionerne der har med LED-displayet, timerne, knapperne og gameporten at gøre. I Application Interface Layer har vi funktionerne der har med matematikken (vektorrotation) at gøre samt funktionerne der skriver til terminalen. I Application Layer har vi funktionerne der styrer selve spillet, samt vores levels, menuer og ASCII-art.

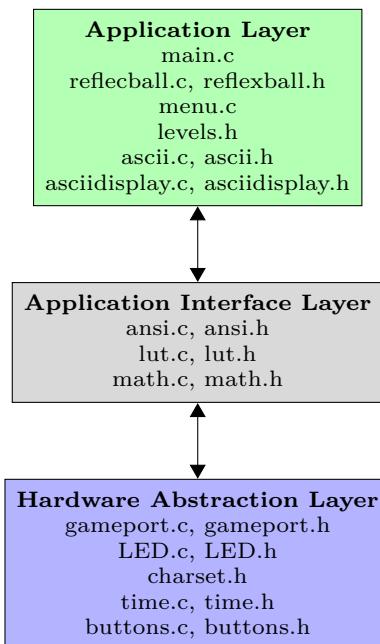


Figure 14: Blokdiagram over de tre lag i koden

For at få en fornemmelse for strukturen af koden, kan man også se på hvilke filer de forskellige filer inkluderer. På Figur 15 er det vist hvilke filer der er inkluderet af main.c. Filerne ez8.h og sio.h er to filer der f.eks. indeholder mikroprocessorens funktioner til at skrive tekst til terminalen. De er specifikke for Zilog 6403-mikroprocessoren. Ud over disse filer inkluderer main.c filer fra alle tre lag i vores softwarearkitektur. For at få et bedre overblik af hvordan filerne inkluderer hinanden har vi på Figur 16 vist hvilke filer der er inkluderet af reflexball.c. Et fuldt overblik over hvilke filer der linker til hinanden m.m. kan findes på <http://lauszus.github.io/ReflexBall/files.html>.

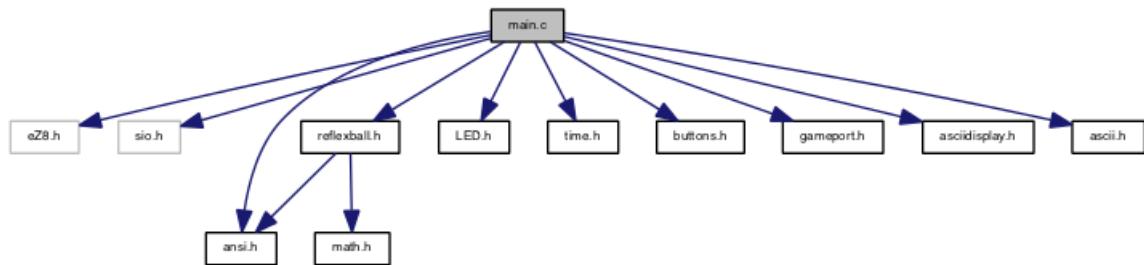


Figure 15: Her er det vist hvilke funktioner main.c inkluderer

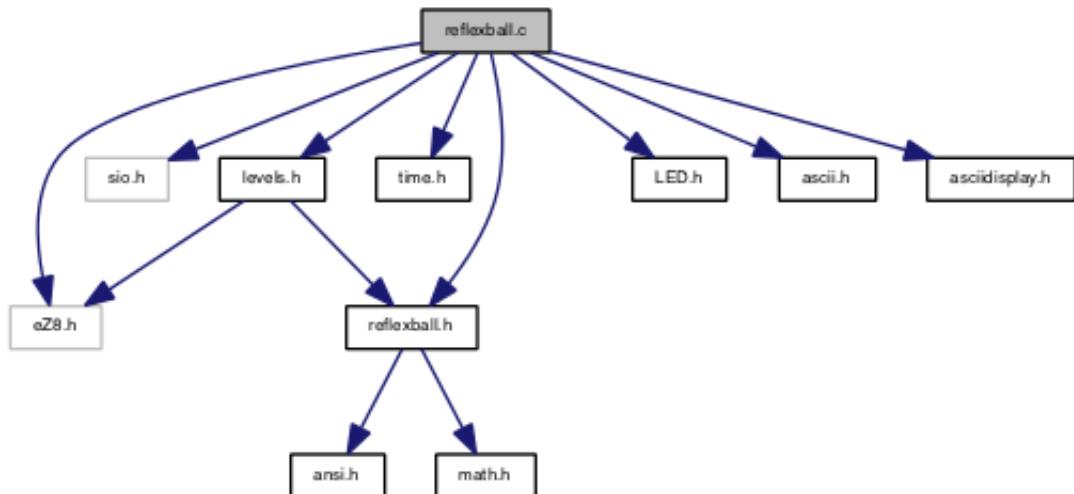


Figure 16: Her er det vist hvilke funktioner reflexball.c inkluderer

4.8 Generelt flowchart

På figur 17 ses et overordnet flowchart for vores kode. Det beskriver den overordnede virkemåde af koden uden at gå helt i dybten, da det ellers ville blive for stort og kompliceret. Et flowchart over briklogikken kan ses i figur 9.

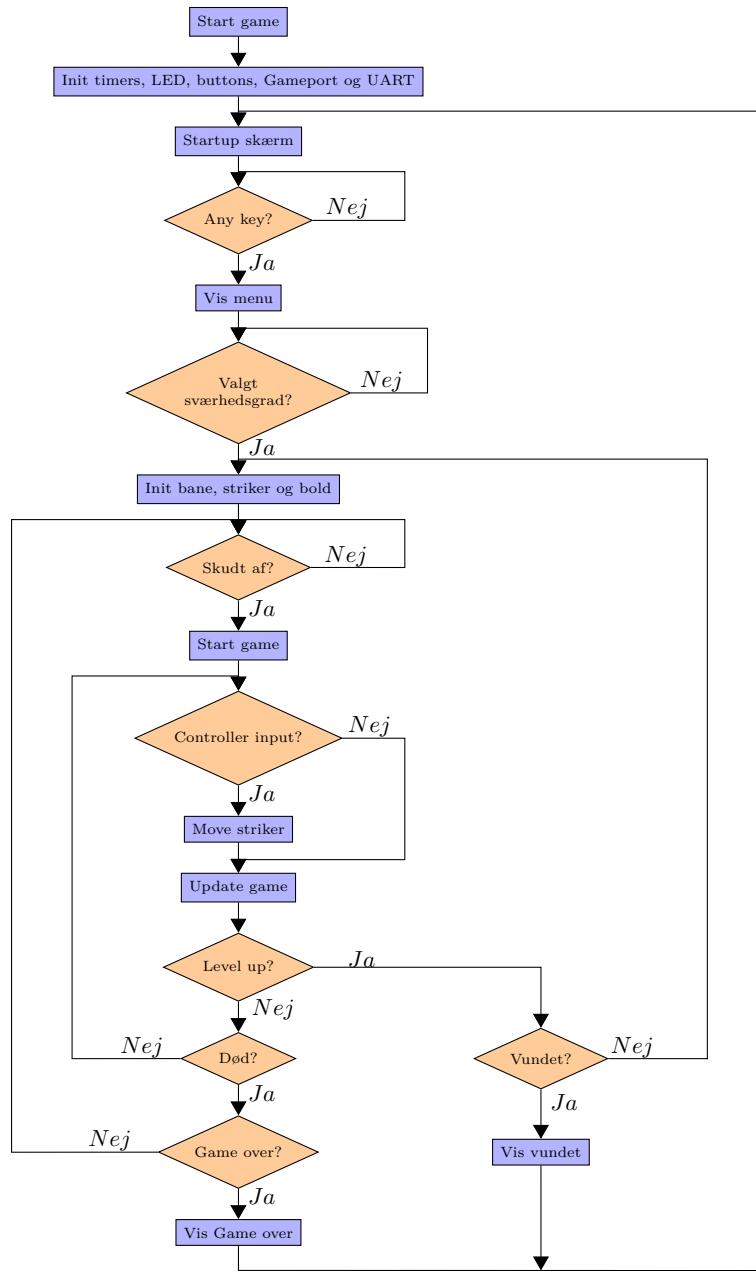


Figure 17: Flowchart for den overordnede virkemåde af koden

5 Verifikation

I dette afsnit vil vi kort gennemgå hvordan vi testede om de forskellige delmål var opfyldt.

Delmål Basics

- **Detektere tastatur-input**
Testet ved at udskrive inputtet på skærmen.
- **Tegne banen, Bevæge strikeren, Bevægelig bold**
Testet ved at se på skærmen om der skete det vi ønskede.
- **Bolden reflekteres af striker, loft og vægge**
Testet ved at prøve spillet.
- **Det skal detekteres når bolden er under strikeren (man er død)**
Testet ved at skrive tekst til skærmen og stoppe spillet når dette skete.

Delmål advanced

- **Liv-system, Pointsystem**
Testet ved at skrive variablen indeholdende antallet af liv på skærmen.
- **Ændring i boldhastighed i takt med ens score stiger**
Testet ved at skrive til skærmen samt at få hastigheden til at stige hurtigt så det var let at se hvornår det skete.
- **Internt 18.14 koordinatsystem**
Testet ved at omregne boldens koordinater fra 18.14 til decimaltal og skrive dem til skærmen.
- **Bolden skal starte i en tilfældig vinkel**
Testet ved at skrive vinklen på skærmen.
- **Striker-zoner**
Testet ved at skrive de forskellige zoner til skærmen når strikeren initialiseres. Maks-vinklen er skrevet ved at skrive før- og efter-vinkel til skærmen når maks-vinklen nås.
- **Stor bold**
Kan ses på skærmen.
- **Forskellige sværhedsgrader**
Testet ved at se om striker-størrelse og hastighed svarer til sværhedsgraden.

Brikker

- **Banen skal indeholde brikker**
Kan ses på skærmen.
- **Forskellige typer brikker**
Testet ved at ramme brikkerne så mange gange som de har liv og se at de forsvinder.

- **Der skal være forskellige levels med brikker i forskellige mønstre**
Testet ved at starte spillet i de forskellige baner. Overgangen mellem banerne er testet ved at gennemføre en bane.
- **Når bolden rammer brikker og kanter skal den reflekteres på en logisk måde så fysikken i spillet ser realistisk ud**
Testet ved at spille spillet en masse hvor bolden både kørte med høj og lav hastighed.

Helhedsindtryk

- **LED display med score, liv og levels**
Testet ved at se om det rigtige blev skrevet ud.
- **Menu, beskeder ved Game Over, besked hvis spillet vindes**
Game over er testet ved at blive tabe spillet. Besked når spillet vindes er testet ved kun at lave en bane i spillet og så klare den.

Hardware

- **Styring med intern hardware (knapper på microcontrolleren)**
Testet ved at skrive input fra knappen til skærmen.
- **Tilslutning af ekstern hardware (DEXXA Steering Wheel)**
Testet ved skrive input til skærmen. Inputtet bliver omregnet til en værdi der bestemmer hvor langt vi rykker strikeren. Denne omregning er også testet ved at skrive den til skærmen. Rattet er desuden testet ved at spille spillet med det som controller.

6 Brugermanual

I dette afsnit vil vi beskrive en simpel brugermanual til spillet. Vi vil fokusere på den primære styringsmetode vha. rattet og gearstangen. Spillet kan dog også spillet vha. computerens keyboard eller knapperne på microcontrolleren.

Det første der møder en når man starter spillet er skærmbilledet der ses på figur 18. Den blinkende tekst under logoet angiver at brugeren skal trykke på en valgfri tast for at starte spillet. ASCII arten af rattet skulle gerne give en indikation af hvordan spillet styres. Et billede af rattet til sammenligning kan ses i figur 19.



Figure 18: Screenshot af startup skærmen

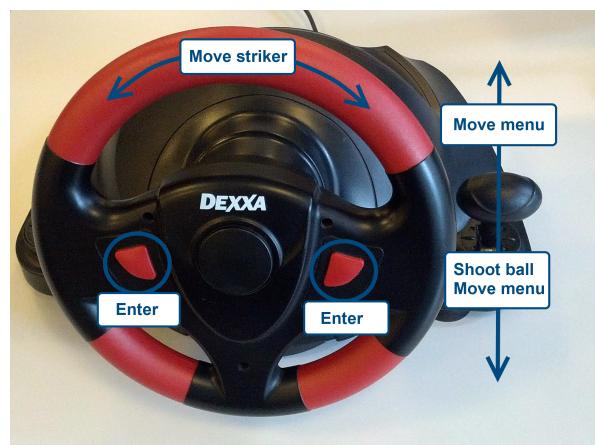


Figure 19: Oversigt over styring af rattet

Derefter vises menuen svarende til den i figur 20. Her kan brugeren vælge sværhedsgraden. Brugeren kan således flytte bolden op og ned vha. gearet. Når den ønskede sværhedsgrad er valgt kan spillet startes ved at trykke på en af de to knapper der sidder på rattet. Den valgte sværhedsgrad bestemmer derefter bredden af strikeren, samt hvor hurtigt bolden skal begynde at forøge sin hastighed jo sværre mode, jo hurtigere forøges boldhastigheden. Ved valg af Chuck Norris mode er boldens hastighed dog den maksimale fra starten. For mere information henvises til funktionen asciidisplay i appendiks C.

Efter dette begynder selve spillet. Når spillet starter har man 3 liv og for hver gang det ikke lykkes en at fange bolden med strikeren, inden den ryger forbi, mister man et liv. Man skyder bolden afsted ved enten af at hive gearstangen tilbage eller trykke på en af de to knapper på rattet. Bolden vil da skydes afsted med en vilkårlig vinkel på 45-135 grader. Herefter gælder det blot om at skyde alle brikkerne ned ved at bevæge strikeren ved at dreje på rattet. Der er forskellig afbøjningsvinkel afhængigt af hvor bolden rammer strikeren. En god spiller kan bruge dette til at få bolden afbøjet i den retning han/hun gerne vil.

Hver gang brugeren rammer en brik får 1 point. Bemærk at det ikke giver point når strikeren rammer bolden, så det belønner sig ikke at forlænge banen, ved at sørge for ikke at ramme brikker. Scoren bliver løbende vist på LED displayet på microcontrolleren så længe man er i live. Hvis man dør scrolles pointene ud på LED displayet og derefter scrolles antal liv man



Figure 20: Screenshot af menuen

har tilbage igennem og til sidst scrolles pointene frem igen og stopper når de udfylder hele displayet, se figur 13. Hvis man er utålmodig og ikke kan vente til der er scrollt færdig kan man sagtens starte spillet imens.

Hvis brugeren skyder alle brikkerne i stykker avancerer brugeren til næste level og får som belønning 2 ekstra liv. Her vil LED displayet begynde at scrollle. Først kommer beskeden "Level up!", dernæst scrolles hvilket level man er nået til igennem, så hvor mange liv man nu har og til sidst scrolles pointene tilbage ind igen. Når en ny bane påbegyndes nulstilles boldens hastighed til starthastigheden på den givne sværhedsgrad. Men pas på, banerne bliver sværere og sværere!

Et overblik over alle de 6 levels i spillet kan ses på figurerne nedenfor. Bemærk at der i level 6 er en hel udfyldt række af usynlige brikker, der først bliver synlige når man har ramt dem én gang. Disse brikker har 5 liv og er derfor væsentligt sværere at slå i stykker. Se levels i appendiks C.

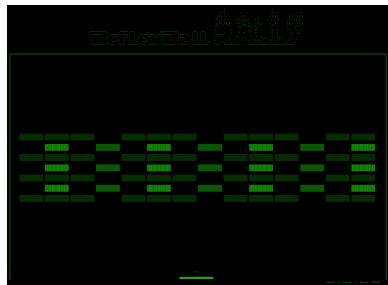


Figure 21: Level 1

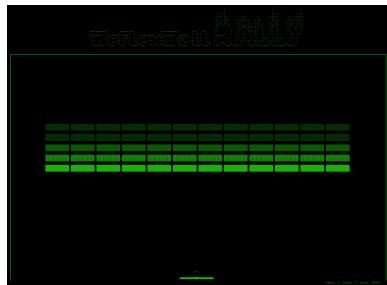


Figure 22: Level 2

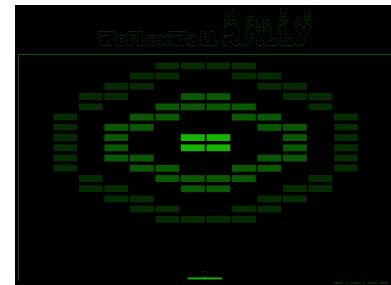


Figure 23: Level 3

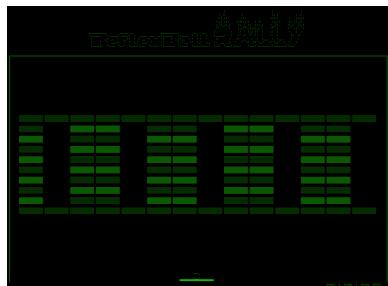


Figure 24: Level 4

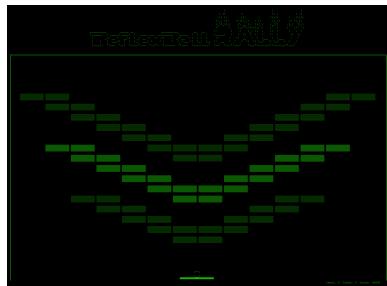


Figure 25: Level 5

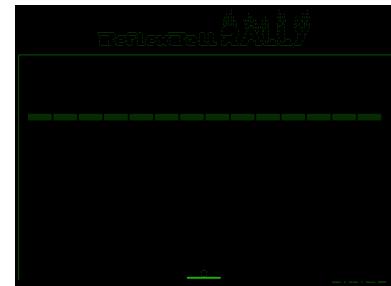


Figure 26: Level 6

Hvis man taber spillet vises "Game Over!" skrevet med ASCII art, samt 1 tilfældig ud af 8 forskellige undertitler (som ikke nødvendigvis altid er hverken lige politisk korrekte eller venlige). Figur 27 viser et screenshot af dette. Bliver man Game Over vil LED displayet blive ved med at scrollle "Game Over" efterfulgt af ens score, indtil man klikker på en vilkårlig tast. Når dette gøres, kommer man tilbage til startmenuen.



Figure 27: Eksempel på screenshot game over tekst

Hvis man vinder spillet på Easy, Medium eller Hard vises et screenshot magen til figur 28 som belønning for det hårde slid. Hvis man derimod skulle vinde spillet på sværhedsgraden Chuck Norris (hvis dette overhovedet kan lade sig gøre), fås en sølle belønning, da et ASCII-Art billede af Chuck Norris med underteksten "Only Chuck Norris wins Chuck Norris mode!" vises kort (se figur 29), efterfulgt af at spillet vil starte forfra i level 1 på Chuck Norris mode.

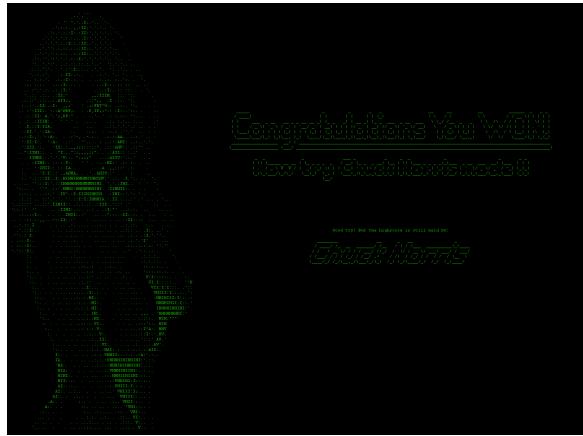


Figure 28: Screenshot ved gennemførsel af spillet

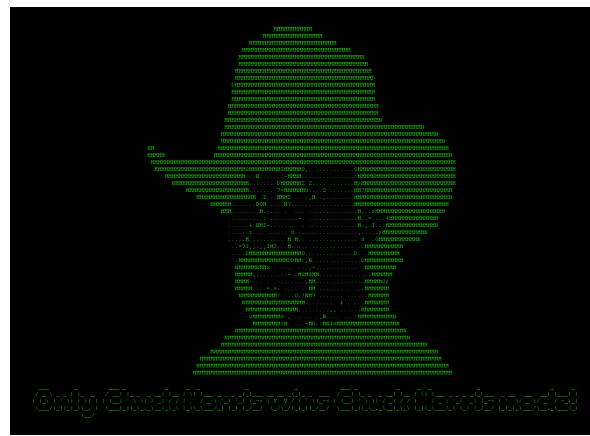


Figure 29: Screenshot ved gennemførsel af spillet i Chuck Norris mode

7 Diskussion

Dette projekt valgte vi at dele op i en række delmål. Delmålene var delt op i grupperne Basics, Advanced, Brikker, Helhedsindtryk og Hardware. Basics var det der skulle til for at få et fungerende spil og de efterfølgende mål har skullet opfyldt vores ønsker om at lave et realistisk spil med godt gameplay. Ud fra disse delmål, samt det ekstra mål at man skulle kunne få powerups i spillet, lavede vi tidsplanen vist på Figur 30, øverst. Som det kan ses nederst på samme figur, tog implementeringen af brikkerne dog væsentlig længere tid end forventet og det gjorde at vi ikke nåede vores mål om at lave powerups. En anden forskel på vores oprindelige tidsplan og hvad vi rent faktisk gjorde, var at vi startede tidligere på at dokumentere projektet og dette er dermed blevet gjort løbende. Undervejs i forløbet valgte vi at lave ASCII-art til spillet. Vi valgte at gøre dette da vi havde et spil hvor fysikken fungerede tilfredsstillende. Med ASCII-arten vil vi gerne give spillet en stemning af et old-school, gennemført arkadespil.

Oprindelige mål og plan

Mål	Fredag d. 14.	Mandag d. 17.	Tirsdag d. 18.	Onsdag d. 19.	Torsdag d. 20.	Fredag d. 21.	Mandag d. 24.	Tirsdag d. 25.	Onsdag d. 26.	Torsdag d. 27.	Fredag d. 28.
Delmål Basics											
Advanced (liv, hastighed, startvinkel)											
Brikker											
Display											
DEXXA Steering Wheel											
Start Menu, Game Over, Game Won											
Achievements (Power Up's og Down's)											
Betæsler, test og fintuning af koden											
Rapport											

Nåede mål og plan

Mål	Fredag d. 14.	Mandag d. 17.	Tirsdag d. 18.	Onsdag d. 19.	Torsdag d. 20.	Fredag d. 21.	Mandag d. 24.	Tirsdag d. 25.	Onsdag d. 26.	Torsdag d. 27.	Fredag d. 28.
Delmål Basics											
Advanced (liv, hastighed, startvinkel)											
Brikker											
Display											
DEXXA Steering Wheel											
Start Menu, Game Over, Game Won											
ASCII Art											
Achievements (Power Up's og Down's)											
Betæsler, test og fintuning af koden											
Rapport											

Figure 30: Øverst: Tidsplan over de forskellige delmål og hvornår de skulle være færdige. Nederst: Hvad vi rent faktisk lavede og hvornår

Forsinkelserne i projektet som kan ses på Figur 30 skyldtes primært at vi, sammen med de avancerede delmål, valgte at gøre bolden større. Da vi skiftede til terminalen Putty i stedet for HyperTerminal og fik mulighed for at køre terminalen i fuldskærm valgte vi at lave bolden 4×2 karakterer. Dette gjorde, at bolden kunne ramme flere brikker af gangen, og briklogikken blev af den grund meget kompleks, se flowchartet på Figur 9 og Appendix reflexball. En yderligere udfordring var, at bolden bevægede sig to karakterer af gangen i x-retningen på skærmen. Der kunne derfor komme en stor del af bolden ind i brikkerne, hvilket også vanskeliggjorde, at fastslå, hvordan brikken var blevet ramt af bolden.

En stor del af arbejdet med brikkerne bestod af debugging og rettelser i koden. For at teste briklogikken skrev vi tekst ud på skærmen der fortalte os hvordan vi havde ramt brikkerne, om flagene `dontReflectX` og `dontReflectY` var blevet sat og om det var detekteret at der var brikker ved siden af og over/under den ramte brik. På den måde kunne vi finde præcis de steder i koden hvor der var problemer med vores logik.

Hvis vi skulle have implementeret powerups i spillet kunne det være gjort ved at tjekke om en brik døde hver gang man ramte en brik. Hvis den gjorde det, kunne vi f.eks. lade de sidste tre bit af millis() bestemme om man fik en powerup. Hvis de sidste tre bit gav f.eks. 0 kunne man få en powerup. På den måde ville det være tilfældigt om man fik en powerup, man ville have $1/8$ sandsynlighed for at det skete.

Den store udfordring med powerups er dog at implementere en masse forskellige sjove features de skulle have. En powerup kunne f.eks. give langsom hastighed, flere bolde, bredere striker, større/mindre bold, blinkende eller usynlig striker, blinkende level, flere brikker, mulighed for at skyde eller mange andre ting. For at det skulle være en gennemført implementering af powerups skulle vi dog lave flere forskellige funktioner, og dette ville derfor blive et forholdsvis stort arbejde. Vi valgte derfor at fokusere på at lave et gennemført spil hvor detaljerne var i højsædet.

Vi forsøgte så vidt muligt at mindske antallet af globale variabler, da det hurtigt ville gøre koden meget uoverskuelig.

I de enkelte .c filer har vi forsøgt at bruge så få globale variabler defineret i andre filer som muligt, da det specielt ville gøre det hele meget uoverskueligt.

Det var dog svært helt at undgå globale variabler i takt med at koden blev mere og mere kompliceret. Specielt reflexball.c bruger en række globale variable. Vi mener dog ikke at de er brugt i unødvendigt, da de er nødvendige for at gemme information såsom brikken og strikerens position, antallet af liv brugerne har tilbage, tiden siden sidste iteration etc. Vi har dog bestræbt os på at kalde de forskellige variabler så sigende navne som muligt, så der forhåbentligt opstår så lidt forvirring som muligt.

8 Konklusion

Efter spillet et blevet designet, skrevet og uploadet til Zilog 6403 microcontrolleren, er alle spillets facetter blevet gennemtestet og fundet i overensstemmelse med det forventede resultat. Vi kan derfor konkludere at den skrevne kode implementerer spillet Reflex Ball på microcontrolleren og i hyperterminalen korrekt i forhold til vores specifikationskrav, bortset fra at vi aldrig nåede at implementere achievements-delen. Vi kan derudover også konkludere at alle spillets udvidelser virker som forventet uden fejl og mangler.

A Introduktionsopgaver

Som kort beskrevet i Introduktionen begyndte vi dette projekt med at lave en række opgaver, både håndregningsopgaver og en del programmeringsopgaver.

A.1 Håndregningsopgaver

Vores besvarelser af håndregningsopgaverne omhandlende bit- og hexadecimalmanipulation kan ses på figur 31, figur 32, figur 33 og figur 34.

Exercise 1.1					
Opgave 1					
	Unsigned	Signed-magnitude	Ones-complement	Twos-complement	Biased
56	00111000 (8)	00110000 (8)	00111000 (8)	00111000 (8)	10110111 (8)
178	10110010 (8)	0000 0000 1011 0010 (16)	0000 0000 1011 0010 (16)	0000 0000 1011 0010 (16)	1000 0000 1011 0001 (16)
1002	0000 0011 1110 1010 (16)	0000 0011 1110 1010 (16)	0000 0011 1110 1010 (16)	0000 0011 1110 1010 (16)	1000 0011 1110 1001 (16)
7586	0001 1101 1010 0010 (16)	0001 1101 1010 0010 (16)	0001 1101 1010 0010 (16)	0001 1101 1010 0010 (16)	1001 1101 1010 0001 (16)

Opgave 2					
	Unsigned	Signed-magnitude	Ones-complement	Twos-complement	Biased
-56	-	1011 1000 (8)	11000111 (8)	11001000 (8)	01000111 (8)
-177	-	1000 0000 1011 0001 (16)	1111 1111 0100 1110 (16)	1111 1111 0100 1111 (16)	0111 1111 0100 1110 (16)
-1003	-	1000 0011 1110 1011 (16)	1111 1100 0001 0100 (16)	1111 1100 0001 0101 (16)	0111 1100 0001 0100 (16)
-7586	-	1001 1101 1010 0010 (16)	1110 0010 0101 1101 (16)	1110 0010 0101 1110 (16)	0110 0010 0101 1101 (16)

Figure 31: Vores besvarelse af Exercise 1.1

Exercise 2.1					
178	0000 0000 1011 0010 (16)				
-177	1111 1111 0100 1111 (16)				
	= 1 0000 0000 0000 0001				
1001	0000 0011 1110 1001 (16)				
-1002	1111 1100 0001 0110 (16)				
	= 1111 1111 1111 1111				
7576	0001 1101 1001 1000 (16)				
-7586	1110 0010 0101 1110 (16)				
	= 1111 1111 1111 0110				

Figure 32: Vores besvarelse af Exercise 2.1

Exercise 3.1		
	Question	Answer
0	What is 0x14 in binary?	0b 0001 0100
1	What is 0xE6 in binary?	0b 1110 0110
2	What is 0x58 in binary?	0b 0101 1000
3	What is 10110111b in hexadecimal?	0xB7
4	What is 10011111b in hexadecimal?	0x9F
5	What is 00111101b in hexadecimal?	0x3D
6	How do you turn on bit 4 and 7 in a byte?	$ = 1001\ 0000$ eller $ = (1<<4) \mid (1<<7)$
7	How do you turn off bit 2 and 5 in a byte?	$&= ~(0010\ 0100)$ eller $&= ~(1<<2) \mid (1<<5)$
8	How do you write -1 in binary, assuming a byte?	0b 1111 1111
9	How do you write -2 in binary, assuming a byte?	0b 1111 1110
10	What is 0000 1000b in decimal, assuming unsigned notation?	8
11	What is 0000 1000b in decimal, assuming signed notation?	8
12	What is 1111 1110b in decimal, assuming unsigned notation?	254
13	What is 1111 1110b in decimal, assuming signed notation?	-2
14	What is the highest value you can store in a signed byte (write in decimal and binary notation)?	127, 0b0111 1111
15	What is the highest value you can store in an unsigned byte (write in decimal and binary notation)?	255, 0b1111 1111
16	What is the lowest value you can store in a signed byte (write in decimal and binary notation)?	-128, 0b1000 0000
17	What is the lowest value you can store in an unsigned byte (write in decimal and binary notation)?	0, 0b0000 0000
18	How do you tell if a signed byte is positive or negative?	If MSB=1, negative. If MSB=0, positive
19	How do you change the sign of a binary number?	number = ~number +1
20	How do you multiply by two?	Bitshift 1 left. ($<<1$)
21	How do you divide by two?	Bitshift 1 right. ($>>1$)
22	When dividing by two, how is the rounding performed?	Always rounding down (integer division)

Figure 33: Vores besvarelse af Exercise 3.1

Exercise 4.4		
	Question	Answer
23	How do you convert a char into 8.8 notation?	(int) char << 8
24	How do you convert an 8.8 value to a char?	(char) (int >> 8)
25	What is the smallest, positive, non-zero value you can store?	$2^{(-8)} = 1/256$
26	What happens if you add two 8.8 values?	You will get a correct result
27	What happens if you multiply two 8.8 values?	It works only if you right-bitshift ($>>$) number of bits after the point after multiplying
28	What happens if you multiply an 8.8 value with a char?	It works, you will get an 8.8 value
29	What about rounding when you convert an 8.8 value to a char?	Everything after bit will be rounded down to 0.
30	How do you make it round correctly?	Add MSB after point to LSB before point before converting

Figure 34: Vores besvarelse af Exercise 4.4

A.2 Programmeringsopgaver

En del af de programmer vi skrev under introforløbet endte vi med at bruge i ReflexBall-spillet i modificerede versioner. Disse inkluderer ansi, charset, buttons, LED, lut, math og time kan ses i Appendixet **C kode** i deres modificerede versioner. I Appendixet **C kode** er der fjernet noget af det oprindelige kode i ansi og i time, hvilket skyldes at vi ikke brugte de givne funktioner i implementeringen af ReflexBall. Disse fjernede dele af **ansi.c** og **time.c** kan ses nedenfor.

A.3 ansi

ansi.c

```

1 void drawBanner(unsigned char x1, unsigned char y1, unsigned char left, unsigned char right, char* title) {
2     gotoxy(x1+1,y1);
3     printf("%c",left);
4     reverse(1);
5     printf("%s",title);
6     reverse(0);
7     printf("%c",right);
8 }
9
10 void window(unsigned char x1, unsigned char y1, unsigned char x2, unsigned char y2, char* title, char style) {
11     char* bannerTitle = title;
12     unsigned char leftTop, rightTop, leftBot, rightBot, verSide, horSide, leftCross, rightCross;
13
14     if (style) {
15         leftTop = 201;
16         rightTop = 187;
17         leftBot = 200;
18         rightBot = 188;
19         verSide = 186;
20         horSide = 205;
21         leftCross = 185;
22         rightCross = 204;
23     } else {
24         leftTop = 218;
25         rightTop = 191;
26         leftBot = 192;
27         rightBot = 217;
28         verSide = 179;
29         horSide = 196;
30         leftCross = 180;
31         rightCross = 195;
32     }
33
34     drawTopBot(x1,y1,x2-x1-1,leftTop,rightTop,horSide);
35     drawSides(x1,y1,x2,y2,verSide);
36     drawTopBot(x1,y2,x2-x1-1,leftBot,rightBot,horSide);
37
38     if (strlen(title) > x2-x1-7)
39         bannerTitle = "Err";
40     drawBanner(x1,y1,leftCross,rightCross,bannerTitle);
41
42     gotoxy(x1+2,y1+2);
43
44     saveCursor();
45 }
46
47 void printWindow(char* string) {
48     int position = 0;
49     getSavedCursor();
50     while(*string != '\0') {
51         if (*string == '\n') {
52             moveCursor(DOWN,1);
53             moveCursor(BACK,position);
54             position = 0;
55         } else {
56             printf("%c",*string);
57             position++;
58         }
59         string++;
60     }
61     saveCursor();

```

62 }

A.4 time

time.c

```

1 volatile unsigned char newTime = 0;
2 volatile time_t time;
3
4 void resetTime() {
5     time.hour = 0;
6     time.min = 0;
7     time.sec = 0;
8     time.cs = 0;
9
10    TOH = 0; // Reset timer value
11    TOL = 1;
12 }
13
14 void printTime(unsigned splittime) {
15     if (splittime != 3)
16         gotoxy(32,7+splittime);
17     else
18         gotoxy(32,7);
19     printf("%01d:%02d:%02d.",time.hour,time.min,time.sec);
20     if (splittime)
21         printf("%02d",time.cs);
22     else
23         printf(" -- ");
24 }
25
26 #pragma interrupt
27 void timer0int() {
28     time.cs++;
29
30     if (time.cs == 100) {
31         time.cs = 0;
32         time.sec++;
33         newTime = 1;
34         if (time.sec == 60) {
35             time.sec = 0;
36             time.min++;
37             if (time.min == 60) {
38                 time.min = 0;
39                 time.hour++; // To the infinity!!
40             }
41         }
42     }
43 }
44
45 void initWindow() {
46     color(2,4);
47     clrscr();
48     window(10,5,45,11,"Stop watch",1);
49     printWindow("Time since start: 0:00:00.--\n");
50     printWindow("Split time 1: --:--:--.\n");
51     printWindow("Split time 2: --:--:--.\n");
52 }
53
54 void initTimers() {
55     DI(); // Disable interrupt
56
57     TOCTL = 0; // TEN – disable timer

```

```
58 TOCTL |= PRE4; // PRES – Prescaler
59 TOCTL |= (1 << 0); // TMODE – continuous mode
60
61 TOH = 0;
62 TOL = 1;
63
64 TORH = 46080 >> 8; // Interrupt every 10ms
65 TORL = 46080 & 0xFF;
66
67 SET_VECTOR(TIMERO, timer0int); // Enter the timer0int function at each interrupt
68
69 // Set timer0 priority to high
70 IRQOENH |= PRIORITY_TIMERO;
71 IRQOENL |= PRIORITY_TIMERO;
72
73 resetTime();
74
75 //T0CTL |= (1 << 7); // TEN – enable timer
76
77 EI(); // Enable interrupt
78 }
```

B Java kode

Nedenfor ses den Java kode vi brugte til at konvertere en given ASCII art gemt i tekst-fil til et array, som man kunne indsætte direkte i vores C kode. Den sørger således for først at finde bredden og højden af tegnene, da den skal bruge disse informationer til at definerer array'et. Derefter indsætter den en ekstra backslashes for hver gang den enten finder en backslash, anførselstegn eller accent. Til sidst indsætter den anførselstegn rundt om strengene og fylder de der ikke er lange nok ud med mellemrum, så alle strengene er lige så lange som den bredeste streng.

Til sidst printer den resultatet ud på konsollen, som herefter kan kopieres direkte ind i C koden.

B.1 BackslashEscapes

BackslashEscapes.java

```

1 import java.io.File;
2 import java.io.IOException;
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.Scanner;
6 import javax.swing.JFileChooser;
7
8
9 /**
10 * This class is used to generate an array from an ASCII art that can be put directly into the .c file.
11 */
12 public class BackslashEscapes {
13
14     public static void main(String[] args) throws IOException {
15         JFileChooser chooser = new JFileChooser(new File(System.getProperty("user.home"))); // Create the
16             File Chooser instance
17         int returnVal = chooser.showOpenDialog(null); // Show the file dialog
18         if(returnVal != JFileChooser.APPROVE_OPTION) // Check if user didn't press OK
19             return; // If not return
20         openFile(chooser.getSelectedFile().getPath());
21     }
22
23     private static void openFile(String filepath) throws IOException {
24         List<String> list = new ArrayList<String>();
25         Scanner fileScanner = new Scanner(new File(filepath));
26
27         while(fileScanner.hasNextLine()) // Read the entire file into the list
28             list.add(fileScanner.nextLine());
29
30         int height = 0;
31         int width = 0;
32
33         for (String line : list) { // This is used to calculate the necessary height and width of the array
34             System.out.println(line);
35             height++;
36             line = line.replaceAll("\s+", ""); // Remove spaces to the right
37             if (width < line.length())
38                 width = line.length();
39         }
40
41         System.out.println("\nrom const char name[" + height + "][" + (width+1) + "] = {"); // Make room for null-
42             character
43         for (String line : list) {

```

```
42 int spaces = 0;
43 line = line.replaceAll("\s+", ""); // Remove spaces to the right
44 for (int i=line.length();i<width;i++)
45     spaces++;
46 line = line.replaceAll("\\\\", "\\\\"); // Replace \ with \\
47 line = line.replaceAll("\\\"", "\\\""); // Replace " with \
48 line = line.replaceAll("'", "\\'"); // Replace ' with \
49 line = "\\t\\" + line; // Put quota in front of string
50
51 for (int i=0;i<spaces;i++)
52     line += " "; // Fill up lines with spaces, so they are all the same length
53 line += "\\n"; // Put a quota and a comma at the end
54
55 System.out.println(line);
56 }
57 System.out.println("};"); // And finally close the array
58 }
59 }
```

C C kode

Nedenfor ses hele vores færdige kode i alfabetisk rækkefølge.

Koden er skrevet i compileren Z8Encore! og implementeret på en Zilog 6403 mircococontroller. Da den er skrevet i C ville det dog ikke være så svært at porte den til anden hardware arkitektur. Man skulle således kunne ændre "Hardware Interface Layer" vist i figur 14.

C.1 ansi

ansi.h

```

1 #ifndef _ansi_h_
2 #define _ansi_h_
3
4 #define ESC 0x1B
5
6 #define UP 'A'
7 #define DOWN 'B'
8 #define FORWARD 'C'
9 #define BACK 'D'
10
11 // Public
12 void fgcolor(unsigned char foreground);
13 void bgcolor(unsigned char background);
14 void color(unsigned char foreground, unsigned char background);
15 void resetbgcolor();
16 void clrscr();
17 void clreol();
18 void gotoxy(unsigned char x, unsigned char y);
19 void underline(char on);
20 void blink(char on);
21 void reverse(char on);
22 unsigned char strlen(char* string);
23 void drawTopBot(unsigned char x, unsigned char y, unsigned char width, unsigned char left, unsigned char right,
     unsigned char side);
24 void drawSides(unsigned char x1, unsigned char y1, unsigned char x2, unsigned char y2, unsigned char side);
25 void saveCursor();
26 void getSavedCursor();
27 void moveCursor(char dir, unsigned char n);
28
29 // Private
30 void graphicCommand(char command);
31
32 #endif

```

ansi.c

```

1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "ansi.h"
4
5 void fgcolor(unsigned char foreground) {
6 /* Value foreground Value foreground
7 -----
8 0 Black 8 Dark Gray
9 1 Red 9 Light Red
10 2 Green 10 Light Green
11 3 Brown 11 Yellow
12 4 Blue 12 Light Blue
13 5 Purple 13 Light Purple
14 6 Cyan 14 Light Cyan

```

```

15      7 Light Gray 15 White
16  */
17  int type = 22; // normal text
18  if (foreground > 7) {
19      type = 1; // bold text
20      foreground -= 8;
21  }
22  printf("%c[%d;%dm", ESC, type, foreground+30);
23 }
24
25 void bgcolor(unsigned char background) {
26 /* IMPORTANT: When you first use this function you cannot get back to true white background in HyperTerminal.
27 Why is that? Because ANSI does not support true white background (ANSI white is gray to most human eyes).
28 The designers of HyperTerminal, however, preferred black text on white background, which is why
29 the colors are initially like that, but when the background color is first changed there is no
30 way coming back.
31 Hint: Use resetbgcolor(); clrscr(); to force HyperTerminal into gray text on black background.
32
33 Value Color
34 -----
35 0 Black
36 1 Red
37 2 Green
38 3 Brown
39 4 Blue
40 5 Purple
41 6 Cyan
42 7 Gray
43 */
44  printf("%c[%dm", ESC, background+40);
45 }
46
47 void color(unsigned char foreground, unsigned char background) { // combination of fgcolor() and bgcolor() – uses
48 less bandwidth
49  int type = 22; // normal text
50  if (foreground > 7) {
51      type = 1; // bold text
52      foreground -= 8;
53  }
54  printf("%c[%d;%d;%dm", ESC, type, foreground+30, background+40);
55 }
56
57 void resetbgcolor() {
58  printf("%c[m", ESC); // Gray on black text, no underline, no blink, no reverse
59 }
60
61 void clrscr() {
62  printf("%c[2J", ESC);
63 }
64
65 void clreol() {
66  printf("%c[K", ESC);
67 }
68
69 void gotoxy(unsigned char x, unsigned char y) {
70  printf("%c[%d;%dH", ESC, y, x);
71 }
72
73 void graphicCommand(char command) {
74  printf("%c[%dm", ESC, command);
75 }
76
77 void underline(char on) {
78  char command = 4;
79  if (!on)
        command += 20;

```

```
80     graphicCommand(command);
81 }
82
83 void blink(char on) {
84     char command = 5;
85     if (!on)
86         command += 20;
87     graphicCommand(command);
88 }
89
90 void reverse(char on) {
91     char command = 7;
92     if (!on)
93         command += 20;
94     graphicCommand(command);
95 }
96
97 unsigned char strlen(char* string) {
98     unsigned char length = 0;
99     while(*string++ != '\0')
100         length++;
101     return length;
102 }
103
104 void drawTopBot(unsigned char x, unsigned char y, unsigned char width, unsigned char left, unsigned char right,
105                 unsigned char side) {
106     int i;
107     gotoxy(x,y);
108     printf("%c",left);
109     for (i=0;i<width;i++)
110         printf("%c",side);
111     printf("%c",right);
112 }
113
114 void drawSides(unsigned char x1, unsigned char y1, unsigned char x2, unsigned char y2, unsigned char side) {
115     int i, j;
116     for (i=y1+1;i<y2;i++) {
117         gotoxy(x1,i);
118         printf("%c",side);
119         gotoxy(x2,i);
120         printf("%c",side);
121     }
122 }
123
124 void saveCursor() {
125     printf("%c[s", ESC);
126 }
127
128 void getSavedCursor() {
129     printf("%c[u", ESC);
130 }
131
132 void moveCursor(char dir, unsigned char n) {
133     printf("%c[%d%c", ESC, n, dir);
```

C.2 ascii

ascii.h

```
1 #ifndef __ascii_h__
2 #define __ascii_h__
3
4 extern rom const char titleAscii1[11][127];
5 extern rom const char titleAscii2[11][127];
6 extern rom const char menuAscii[10][78];
7 extern rom const char easyAscii[8][24];
8 extern rom const char mediumAscii[6][38];
9 extern rom const char hardAscii[6][25];
10 extern rom const char chuckAscii1[9][66];
11 extern rom const char chuckAscii2[9][66];
12 extern rom const char wheelAscii[30][85];
13
14 extern rom const char gameOverAscii[8][75];
15 extern rom const char amigoAscii[5][61];
16 extern rom const char driveAscii[5][79];
17 extern rom const char havNoBallsAscii[5][135];
18 extern rom const char openEyesAscii[5][100];
19 extern rom const char patienceAscii[5][115];
20 extern rom const char notPassAscii[5][89];
21 extern rom const char thereIsNoBallAscii[5][127];
22 extern rom const char deadAscii[5][94];
23
24 extern rom const char ladyAscii[81][73];
25 extern rom const char chuckNorrisAscii[50][89];
26 extern rom const char congratulationsAscii[8][126];
27 extern rom const char nowTryAscii[5][108];
28 extern rom const char onlyChuckAscii[5][158];
29 extern rom const char chuckNorrisTextAscii[5][63];
30
31 #endif
```



```

204 rom const char ladyAscii[81][73] = {
205     " . . . . ,  

206     " .\|\\|\| . \| . ,  

207     " .\|\\|\| .\|..I:.\|.. \| . ,  

208     " .\|..,,,:II:\|\\|\| .. \| . ,  

209     " .\|..,\|..I:II:\|\\|\| .. \| . ,  

210     " .\|\\|\| .\|..I:..I:.\|.. \| . ,  

211     " .\|\\|\| .\|..I:..I:.\|.. \| . ,  

212     " .\|\\|\| .\|..I:..I:.\|.. \| . ,  

213     " .\|\\|\| .\|..I:..I:.\|.. \| . ,  

214     " .\|..I:..I:.\|..I:.\|.. \| . ,  

215     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

216     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

217     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

218     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

219     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

220     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

221     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

222     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

223     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,  

224     " .\|..I:..I:.\|..I:.\|..I:.\|.. \| . ,FBTV"X:.. . . \| . . . ,  

225     " .\|..I:..I:.\|..A\"PBF;..P;IP;\"I:..I:.\|.. . . ,  

226     " .\|..I:..I:.\|..A\"V\|..PP;\\" . . . \|.. . . . \|.. . . . ,  

227     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

228     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

229     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

230     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

231     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

232     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

233     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

234     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

235     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

236     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

237     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

238     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

239     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

240     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

241     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

242     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

243     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

244     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

245     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

246     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

247     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

248     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

249     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

250     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

251     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

252     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

253     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

254     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

255     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

256     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

257     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

258     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

259     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

260     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

261     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

262     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

263     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

264     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

265     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

266     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

267     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

268     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,  

269     " .\|..I:..I:.\|..A\"V\|..;IIIH..m.\|.. . . ,

```


C.3 asciidisplay

asciidisplay.h

```
1 #ifndef _asciidisplay_h_
2 #define _asciidisplay_h_
3
4 // Public
5 unsigned char strlen_rom(rom const char *string);
6 void printAscii(rom const char *str, unsigned char size);
7 void printAsciiXY(rom const char *str, unsigned char size, unsigned char x, unsigned char y);
8 void initStartMenu(unsigned char x1, unsigned char y1, unsigned char x2, unsigned char y2);
9 unsigned char startMenu();
10 void printMenu();
11 unsigned char updateMenu();
12 void showWon();
13 void showGameOver();
14
15 // Private
16 void clearMenuBall(unsigned char x, unsigned char y);
17 void drawMenuBall(unsigned char x, unsigned char y);
18 void moveBall(char dir);
19 void calculateDifficulty();
20
21 #endif
```

asciidisplay.c

```
1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "asciidisplay.h"
4 #include "ansi.h"
5 #include "LED.h"
6 #include "time.h"
7 #include "buttons.h"
8 #include "gameport.h"
9 #include "ascii.h"
10 #include "reflexball.h"
11
12 const unsigned char ballMenuYPos[4] = { 40, 48, 55, 64 };
13 const unsigned char ballXPos = 85;
14 unsigned char ballY, lastBallY;
15
16 unsigned char menuState;
17 unsigned long timer;
18 unsigned char oldButtonsWheel, oldButtonsBoard;
19
```

```

20 unsigned char xMin, yMin, xMax, yMax;
21
22 const char *startString = "Press any key to continue";
23
24 unsigned char strlen_rom(rom const char *string) {
25     unsigned char length = 0;
26     while ((char)*string++ != '\0')
27         length++;
28     return length;
29 }
30
31 void printAscii(rom const char *str, unsigned char size) {
32     unsigned char i;
33
34     for (i=1;i<=size;i++) {
35         getSavedCursor();
36         moveCursor(DOWN,i);
37         while ((char)*str != '\0')
38             printf("%c",*str++);
39         str++;
40     }
41     getSavedCursor();
42     moveCursor(DOWN,size);
43     saveCursor();
44 }
45
46 void printAsciiXY(rom const char *str, unsigned char size, unsigned char x, unsigned char y) {
47     unsigned char i, length = strlen_rom(str);
48     for (i=0;i<size;i++) {
49         gotoxy(x,y+i);
50         while ((char)*str != '\0')
51             printf("%c",*str++);
52         str++;
53     }
54     moveCursor(BACK,length);
55 }
56
57 void clearMenuBall(unsigned char x, unsigned char y) {
58     gotoxy(x,y);
59     printf(" ");
60     gotoxy(x,y+1);
61     printf(" ");
62 }
63
64 void drawMenuBall(unsigned char x, unsigned char y) {
65     const unsigned char top = 238, bottom = 95, slash = '/', backSlash = '\\';
66
67     clearMenuBall(ballXPos,lastBallY);
68
69     lastBallY = y;
70
71     gotoxy(x,y);
72     printf("%c%c%c%c",slash,top,top,backSlash);
73     gotoxy(x,y+1);
74     printf("%c%c%c%c",backSlash,bottom,bottom,slash);
75 }
76
77 void moveBall(char dir) {
78     if (ballY + dir >= sizeof(ballMenuYPos) || ballY + dir < 0)
79         return;
80
81     ballY += dir;
82     drawMenuBall(ballXPos,ballMenuYPos[ballY]);
83 }
84
85 void showWon() {

```

```

86 const char* highscoreString = "Good try! But the highscore is still held by:";
87 clrscr();
88 if (divider == 1) { // Chuck Norris mode
89     printAsciiXY(chuckNorrisAscii[0],sizeof(chuckNorrisAscii)/sizeof(chuckNorrisAscii[0]),(xMin+xMax)/2-
90         strlen_rom(chuckNorrisAscii[0])/2,(yMin+yMax)/2-(sizeof(chuckNorrisAscii)/sizeof(chuckNorrisAscii[0]))/
91         2-10);
90     moveCursor(DOWN,1);
91     moveCursor(BACK,strlen_rom(onlyChuckAscii[0])/2-strlen_rom(chuckNorrisAscii[0])/2);
92     saveCursor();
93     printAscii(onlyChuckAscii[0],sizeof(onlyChuckAscii)/sizeof(onlyChuckAscii[0]));
94 } else {
95     gotoxy(0,0);
96     saveCursor();
97     printAscii(ladyAscii[0],sizeof(ladyAscii)/sizeof(ladyAscii[0]));
98
99     gotoxy((xMin+xMax)/2-strlen_rom(congratulationsAscii[0])/2+strlen_rom(ladyAscii[0])/2,20);
100    saveCursor();
101    printAscii(congratulationsAscii[0],sizeof(congratulationsAscii)/sizeof(congratulationsAscii[0]));
102    moveCursor(DOWN,1);
103    moveCursor(FORWARD,strlen_rom(congratulationsAscii[0])/2-strlen_rom(nowTryAscii[0])/2);
104    saveCursor();
105    printAscii(nowTryAscii[0],sizeof(nowTryAscii)/sizeof(nowTryAscii[0]));
106
107    moveCursor(DOWN,10);
108    moveCursor(FORWARD,strlen_rom(nowTryAscii[0])/2-strlen(highscoreString)/2);
109    blink(1);
110    printf("%s",highscoreString);
111    blink(0);
112    moveCursor(DOWN,1);
113    moveCursor(BACK,strlen(highscoreString)+strlen_rom(chuckNorrisTextAscii[0])/2-strlen(highscoreString)/2);
114    saveCursor();
115    printAscii(chuckNorrisTextAscii[0],sizeof(chuckNorrisTextAscii)/sizeof(chuckNorrisTextAscii[0]));
116 }
117
118 while (!getGameportButtons() && !readButtons() && !kbhit()); // Wait for button press
119 }
120
121 void showGameOver() {
122     clrscr();
123     printAsciiXY(gameOverAscii[0],sizeof(gameOverAscii)/sizeof(gameOverAscii[0]),(xMin+xMax)/2-strlen_rom(
124         gameOverAscii[0])/2,(yMin+yMax)/2-(sizeof(gameOverAscii)/sizeof(gameOverAscii[0]))/2-5);
124     moveCursor(DOWN,1);
125
126     switch (millis() & 0x7) { // Pseudo random number from 0-7
127         case 0:
128             if (strlen_rom(gameOverAscii[0]) > strlen_rom(amigoAscii[0]))
129                 moveCursor(FORWARD,strlen_rom(gameOverAscii[0])/2-strlen_rom(amigoAscii[0])/2);
130             else
131                 moveCursor(BACK,strlen_rom(amigoAscii[0])/2-strlen_rom(gameOverAscii[0])/2);
132             saveCursor();
133             printAscii(amigoAscii[0],sizeof(amigoAscii)/sizeof(amigoAscii[0]));
134             break;
135         case 1:
136             if (strlen_rom(gameOverAscii[0]) > strlen_rom(driveAscii[0]))
137                 moveCursor(FORWARD,strlen_rom(gameOverAscii[0])/2-strlen_rom(driveAscii[0])/2);
138             else
139                 moveCursor(BACK,strlen_rom(driveAscii[0])/2-strlen_rom(gameOverAscii[0])/2);
140             saveCursor();
141             printAscii(driveAscii[0],sizeof(driveAscii)/sizeof(driveAscii[0])); // Center text below "Game Over!"
142             break;
143         case 2:
144             if (strlen_rom(gameOverAscii[0]) > strlen_rom(havNoBallsAscii[0]))
145                 moveCursor(FORWARD,strlen_rom(gameOverAscii[0])/2-strlen_rom(havNoBallsAscii[0])/2);
146             else
147                 moveCursor(BACK,strlen_rom(havNoBallsAscii[0])/2-strlen_rom(gameOverAscii[0])/2);

```

```

148     saveCursor();
149     printAscii(havNoBallsAscii[0],sizeof(havNoBallsAscii)/sizeof(havNoBallsAscii[0])); // Center text below "Game Over!"
150     break;
151 case 3:
152     if (strlen_rom(gameOverAscii[0]) > strlen_rom(openEyesAscii[0]))
153         moveCursor(FORWARD,strlen_rom(gameOverAscii[0])/2-strlen_rom(openEyesAscii[0])/2);
154     else
155         moveCursor(BACK,strlen_rom(openEyesAscii[0])/2-strlen_rom(gameOverAscii[0])/2);
156     saveCursor();
157     printAscii(openEyesAscii[0],sizeof(openEyesAscii)/sizeof(openEyesAscii[0])); // Center text below "Game Over !
158     break;
159 case 4:
160     if (strlen_rom(gameOverAscii[0]) > strlen_rom(patienceAscii[0]))
161         moveCursor(FORWARD,strlen_rom(gameOverAscii[0])/2-strlen_rom(patienceAscii[0])/2);
162     else
163         moveCursor(BACK,strlen_rom(patienceAscii[0])/2-strlen_rom(gameOverAscii[0])/2);
164     saveCursor();
165     printAscii(patienceAscii[0],sizeof(patienceAscii)/sizeof(patienceAscii[0])); // Center text below "Game Over !
166     break;
167 case 5:
168     if (strlen_rom(gameOverAscii[0]) > strlen_rom(notPassAscii[0]))
169         moveCursor(FORWARD,strlen_rom(gameOverAscii[0])/2-strlen_rom(notPassAscii[0])/2);
170     else
171         moveCursor(BACK,strlen_rom(notPassAscii[0])/2-strlen_rom(gameOverAscii[0])/2);
172     saveCursor();
173     printAscii(notPassAscii[0],sizeof(notPassAscii)/sizeof(notPassAscii[0])); // Center text below "Game Over!
174     break;
175 case 6:
176     if (strlen_rom(gameOverAscii[0]) > strlen_rom(thereIsNoBallAscii[0]))
177         moveCursor(FORWARD,strlen_rom(gameOverAscii[0])/2-strlen_rom(thereIsNoBallAscii[0])/2);
178     else
179         moveCursor(BACK,strlen_rom(thereIsNoBallAscii[0])/2-strlen_rom(gameOverAscii[0])/2);
180     saveCursor();
181     printAscii(thereIsNoBallAscii[0],sizeof(thereIsNoBallAscii)/sizeof(thereIsNoBallAscii[0])); // Center text
182         below "Game Over!"  

183     break;
184 case 7:
185     if (strlen_rom(gameOverAscii[0]) > strlen_rom(deadAscii[0]))
186         moveCursor(FORWARD,strlen_rom(gameOverAscii[0])/2-strlen_rom(deadAscii[0])/2);
187     else
188         moveCursor(BACK,strlen_rom(deadAscii[0])/2-strlen_rom(gameOverAscii[0])/2);
189     saveCursor();
190     printAscii(deadAscii[0],sizeof(deadAscii)/sizeof(deadAscii[0])); // Center text below "Game Over!
191     break;
192 }  

193 while (!getGameportButtons() && !readButtons() && !kbhit()); // Wait for button press
194 }
195 void initStartMenu(unsigned char newX1, unsigned char newY1, unsigned char newX2, unsigned char newY2) {
196     xMin = newX1;
197     yMin = newY1;
198     xMax = newX2;
199     yMax = newY2;
200
201     blink(1);
202     gotoxy((xMin+xMax)/2-strlen(startString)/2,(yMin+yMax)/2-10);
203     printf("%s",startString);
204     blink(0);
205
206     printAsciiXY(wheelAscii[0],sizeof(wheelAscii)/sizeof(wheelAscii[0]),(xMin+xMax)/2-strlen_rom(wheelAscii[0])/2,
207                 yMax-sizeof(wheelAscii)/sizeof(wheelAscii[0])-15);
208     gotoxy((xMin+xMax)/2-strlen_rom(titleAscii1[0])/2,(yMin+yMax)/2-25);

```

```

209     saveCursor();
210
211     timer = 0;
212 }
213
214 unsigned char startMenu() {
215     if (millis() - timer > 200) {
216         timer = millis();
217
218         switch (menuState) {
219             case 0:
220                 printAscii(titleAscii1[0],sizeof(titleAscii1)/sizeof(titleAscii1[0]));
221                 moveCursor(UP,sizeof(titleAscii1)/sizeof(titleAscii1[0]));
222                 saveCursor();
223                 menuState = 1;
224                 break;
225             case 1:
226                 printAscii(titleAscii2[0],sizeof(titleAscii2)/sizeof(titleAscii2[0]));
227                 moveCursor(UP,sizeof(titleAscii2)/sizeof(titleAscii2[0]));
228                 saveCursor();
229                 menuState = 0;
230                 break;
231         }
232     }
233
234     if (getGameportButtons()) {
235         oldButtonsWheel = getGameportButtons(); // Update oldButtons value, so it doesn't skip the next menu
236         return 1;
237     } else if (readButtons()) {
238         oldButtonsWheel = readButtons(); // Update oldButtons value, so it doesn't skip the next menu
239         return 1;
240     } else if (kbhit())
241         return 1;
242
243     return 0;
244 }
245
246 void printMenu() {
247     printAsciiXY(menuAscii[0],sizeof(menuAscii)/sizeof(menuAscii[0]),(xMin+xMax)/2-strlen_rom(menuAscii[0])/2,(yMin
248         +yMax)/2-15);
249     moveCursor(FORWARD,strlen_rom(menuAscii[0])/2-strlen_rom(chuckAscii[0])/2+10);
250     moveCursor(DOWN,1);
251     saveCursor();
252
253     printAscii(easyAscii[0],sizeof(easyAscii)/sizeof(easyAscii[0]));
254     printAscii(mediumAscii[0],sizeof(mediumAscii)/sizeof(mediumAscii[0]));
255     moveCursor(DOWN,1);
256     saveCursor();
257     printAscii(hardAscii[0],sizeof(hardAscii)/sizeof(hardAscii[0]));
258     moveCursor(DOWN,1);
259     saveCursor();
260
261     menuState = 0;
262     timer = 0;
263
264     ballY = lastBallY = 0;
265     drawMenuBall(ballXPos,ballMenuYPos[ballY]);
266 }
267
268 unsigned char updateMenu() {
269     int input;
270     unsigned char buttons, buttonsClick;
271
272     switch (menuState) {
273         case 0:

```

```

274     if (millis() - timer > 200) {
275         timer = millis();
276         printAscii(chuckAscii1[0], sizeof(chuckAscii1)/sizeof(chuckAscii1[0]));
277         moveCursor(UP,sizeof(chuckAscii1)/sizeof(chuckAscii1[0]));
278         saveCursor();
279         menuState = 1;
280     }
281     break;
282 case 1:
283     if (millis() - timer > 200) {
284         timer = millis();
285         printAscii(chuckAscii2[0], sizeof(chuckAscii2)/sizeof(chuckAscii2[0]));
286         moveCursor(UP,sizeof(chuckAscii2)/sizeof(chuckAscii2[0]));
287         saveCursor();
288         menuState = 0;
289     }
290     break;
291 }
292 buttons = getGameportButtons();
293 if (buttons != oldButtonsWheel) {
294     buttonsClick = buttons & ~oldButtonsWheel; // Only look at the buttons that have changed
295     oldButtonsWheel = buttons;
296
297     if (buttonsClick & 0x1) // Gear forward
298         moveBall(-1);
299     else if (buttonsClick & 0x4) // Gear backward
300         moveBall(1);
301     else if (buttonsClick & 0xA) { // Either of the wheel buttons
302         calculateDifficulty();
303         return 1;
304     }
305 } else if (kbhit()) {
306     input = getch();
307     if (input == ' ') { // Space
308         calculateDifficulty();
309         return 1;
310     } else if (input == 65) // Up
311         moveBall(-1);
312     else if (input == 66) // Down
313         moveBall(1);
314 } else {
315     buttons = readButtons();
316     if (buttons != oldButtonsBoard) {
317         buttonsClick = buttons & ~oldButtonsBoard; // Only look at the buttons that have changed
318         oldButtonsBoard = buttons;
319
320         if (buttonsClick & 0x2) { // Center
321             calculateDifficulty();
322             return 1;
323         } else if (buttonsClick & 0x4) // Left
324             moveBall(-1);
325         else if (buttonsClick & 0x1) // Right
326             moveBall(1);
327     }
328 }
329 return 0;
330 }
331
332 void calculateDifficulty() {
333     if (ballY == 0) { // Easy
334         divider = 10;
335         strikerWidth = 30;
336     } else if (ballY == 1) { // Medium
337         divider = 5;
338         strikerWidth = 20;
339     } else if (ballY == 2) { // Hard

```

```

340     divider = 2;
341     strikerWidth = 10;
342 } else { // Chuck Norris
343     divider = 1;
344     strikerWidth = 4;
345 }
346 }
```

C.4 buttons

buttons.h

```

1 #ifndef _buttons_h_
2 #define _buttons_h_
3
4 // Public
5 void initButtons();
6 unsigned char readButtons();
7 unsigned char readkey();
8
9 #endif
```

buttons.c

```

1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "buttons.h"
4 #include "time.h"
5
6 void initButtons() {
7     PDDD = (1 << 3);
8     PFDD = (1 << 6) | (1 << 7);
9 }
10
11 unsigned char readButtons() {
12     unsigned char inD, inF;
13     inD = (~PDIN >> 3) & 0x1;
14     inF = (~PFIN >> 6) & 0x3;
15     return ((inF >> 1) | ((inF & 0x1) << 1) | (inD << 2));
16 }
17
18 unsigned char readkey() { // Read button with debounce
19     unsigned char output = readButtons();
20     delay_ms(50); // Wait 50ms
21     output &= readButtons(); // Check if it is still high
22     return output;
23 }
```

C.5 charset

charset.h

```

1 ****
2 *
3 * File Name: charset.h
4 *
5 * Last modified:
6 * Michael Thomsen 21/03/2005
```

```

7  /*
8   * Change log: none
9   ****
10
11
12 /** Supported ASCII characters ***
13
14 ASCII CHAR ASCII CHAR ASCII CHAR ASCII CHAR ASCII CHAR ASCII CHAR
15 0x20 0x30 0 0x40 @ 0x50 P 0x60 ` 0x70 p
16 0x21 ! 0x31 1 0x41 A 0x51 Q 0x61 a 0x71 q
17 0x22 " 0x32 2 0x42 B 0x52 R 0x62 b 0x72 r
18 0x23 # 0x33 3 0x43 C 0x53 S 0x63 c 0x73 s
19 0x24 $ 0x34 4 0x44 D 0x54 T 0x64 d 0x74 t
20 0x25 % 0x35 5 0x45 E 0x55 U 0x65 e 0x75 u
21 0x26 & 0x36 6 0x46 F 0x56 V 0x66 f 0x76 v
22 0x27 ' 0x37 7 0x47 G 0x57 W 0x67 g 0x77 w
23 0x28 ( 0x38 8 0x48 H 0x58 X 0x68 h 0x78 x
24 0x29 ) 0x39 9 0x49 I 0x59 Y 0x69 i 0x79 y
25 0x2A * 0x3A : 0x4A J 0x5A Z 0x6A j 0x7A z
26 0x2B + 0x3B ; 0x4B K 0x5B [ 0x6B k 0x7B {
27 0x2C , 0x3C < 0x4C L 0x5C \ 0x6C l 0x7C |
28 0x2D - 0x3D = 0x4D M 0x5D ] 0x6D m 0x7D }
29 0x2E . 0x3E > 0x4E N 0x5E ^ 0x6E n 0x7E ~
30 0x2F / 0x3F ? 0x4F O 0x5F _ 0x6F o
31
32 0x91 æ
33 0x9B ø
34 0x86 å
35 0x92 Å
36 0x9D Ø
37 0x8F Å
38
39 Note:
40 1. The ASCII character 0x20 is "SPACE" or a blank.
41 2. The ASCII character 0x7F(DEL) is not supported by the matrix below.
42
43 */
44
45 #ifndef __CHARSET_H__
46 #define __CHARSET_H__
47
48 const unsigned char rom character_data[95+9][5] = {
49 {0x00, 0x00, 0x00, 0x00, 0x00},
50 {0x00, 0x5F, 0x5F, 0x00, 0x00},
51 {0x00, 0x07, 0x00, 0x07, 0x00},
52 {0x14, 0x7F, 0x14, 0x7F, 0x14},
53 {0x24, 0x2A, 0x7F, 0x2A, 0x12},
54 {0x23, 0x13, 0x08, 0x64, 0x62},
55 {0x36, 0x49, 0x55, 0x22, 0x50},
56 {0x00, 0x05, 0x03, 0x00, 0x00},
57 {0x00, 0x1C, 0x22, 0x41, 0x00},
58 {0x00, 0x41, 0x22, 0x1C, 0x00},
59 {0x14, 0x08, 0x3E, 0x08, 0x14},
60 {0x08, 0x08, 0x3E, 0x08, 0x08},
61 {0x00, 0x50, 0x30, 0x00, 0x00},
62 {0x08, 0x08, 0x08, 0x08, 0x08},
63 {0x00, 0x60, 0x60, 0x00, 0x00},
64 {0x20, 0x10, 0x08, 0x04, 0x02},
65 {0x3E, 0x51, 0x49, 0x45, 0x3E},
66 {0x00, 0x42, 0x7F, 0x40, 0x00},
67 {0x42, 0x61, 0x51, 0x49, 0x46},
68 {0x22, 0x49, 0x49, 0x49, 0x36},
69 {0x18, 0x14, 0x12, 0x7F, 0x10},
70 {0x2F, 0x49, 0x49, 0x49, 0x31},
71 {0x3E, 0x49, 0x49, 0x49, 0x32},
72 {0x03, 0x01, 0x71, 0x09, 0x07},

```

```
73 {0x36, 0x49, 0x49, 0x49, 0x36},  
74 {0x26, 0x49, 0x49, 0x49, 0x3E},  
75 {0x00, 0x36, 0x36, 0x00, 0x00},  
76 {0x00, 0x56, 0x36, 0x00, 0x00},  
77 {0x08, 0x14, 0x22, 0x41, 0x00},  
78 {0x14, 0x14, 0x14, 0x14, 0x14},  
79 {0x00, 0x41, 0x22, 0x14, 0x08},  
80 {0x02, 0x01, 0x51, 0x09, 0x06},  
81 {0x32, 0x49, 0x79, 0x41, 0x3E},  
82 {0x7C, 0x0A, 0x09, 0x0A, 0x7C},  
83 {0x7F, 0x49, 0x49, 0x49, 0x36},  
84 {0x3E, 0x41, 0x41, 0x41, 0x22},  
85 {0x7F, 0x41, 0x41, 0x41, 0x3E},  
86 {0x7F, 0x49, 0x49, 0x49, 0x41},  
87 {0x7F, 0x09, 0x09, 0x09, 0x01},  
88 {0x3E, 0x41, 0x49, 0x49, 0x7A},  
89 {0x7F, 0x08, 0x08, 0x08, 0x7F},  
90 {0x00, 0x41, 0x7F, 0x41, 0x00},  
91 {0x30, 0x40, 0x40, 0x40, 0x3F},  
92 {0x7F, 0x08, 0x14, 0x22, 0x41},  
93 {0x7F, 0x40, 0x40, 0x40, 0x40},  
94 {0x7F, 0x02, 0x0C, 0x02, 0x7F},  
95 {0x7F, 0x02, 0x04, 0x08, 0x7F},  
96 {0x3E, 0x41, 0x41, 0x41, 0x3E},  
97 {0x7F, 0x09, 0x09, 0x09, 0x06},  
98 {0x3E, 0x41, 0x51, 0x21, 0x5E},  
99 {0x7F, 0x09, 0x09, 0x09, 0x76},  
100 {0x26, 0x49, 0x49, 0x49, 0x32},  
101 {0x01, 0x01, 0x7F, 0x01, 0x01},  
102 {0x3F, 0x40, 0x40, 0x40, 0x3F},  
103 {0x1F, 0x20, 0x40, 0x20, 0x1F},  
104 {0x3F, 0x40, 0x38, 0x40, 0x3F},  
105 {0x63, 0x14, 0x08, 0x14, 0x63},  
106 {0x03, 0x04, 0x78, 0x04, 0x03},  
107 {0x61, 0x51, 0x49, 0x45, 0x43},  
108 {0x7F, 0x41, 0x41, 0x00, 0x00},  
109 {0x02, 0x04, 0x08, 0x10, 0x20},  
110 {0x00, 0x41, 0x41, 0x7F, 0x00},  
111 {0x04, 0x02, 0x01, 0x02, 0x04},  
112 {0x40, 0x40, 0x40, 0x40, 0x40},  
113 {0x00, 0x01, 0x02, 0x04, 0x00},  
114 {0x20, 0x54, 0x54, 0x54, 0x78},  
115 {0x7F, 0x48, 0x44, 0x44, 0x38},  
116 {0x38, 0x44, 0x44, 0x44, 0x20},  
117 {0x38, 0x44, 0x44, 0x48, 0x7F},  
118 {0x38, 0x54, 0x54, 0x54, 0x18},  
119 {0x08, 0x7E, 0x09, 0x01, 0x02},  
120 {0x0C, 0x52, 0x52, 0x52, 0x3E},  
121 {0x7F, 0x08, 0x04, 0x04, 0x78},  
122 {0x00, 0x44, 0x7D, 0x40, 0x00},  
123 {0x20, 0x40, 0x44, 0x3D, 0x00},  
124 {0x7F, 0x10, 0x28, 0x44, 0x00},  
125 {0x00, 0x41, 0x7F, 0x40, 0x00},  
126 {0x7C, 0x04, 0x18, 0x04, 0x78},  
127 {0x7C, 0x08, 0x04, 0x04, 0x78},  
128 {0x38, 0x44, 0x44, 0x44, 0x38},  
129 {0x7C, 0x14, 0x14, 0x14, 0x08},  
130 {0x08, 0x14, 0x14, 0x18, 0x7C},  
131 {0x7C, 0x08, 0x04, 0x04, 0x08},  
132 {0x48, 0x54, 0x54, 0x54, 0x20},  
133 {0x04, 0x3F, 0x44, 0x40, 0x20},  
134 {0x3C, 0x40, 0x40, 0x20, 0x7C},  
135 {0x1C, 0x20, 0x40, 0x20, 0x1C},  
136 {0x3C, 0x40, 0x38, 0x40, 0x3C},  
137 {0x44, 0x28, 0x10, 0x28, 0x44},  
138 {0x0C, 0x50, 0x50, 0x50, 0x3C},
```

```

139 {0x44, 0x64, 0x54, 0x4C, 0x44},
140 {0x00, 0x08, 0x36, 0x41, 0x00},
141 {0x00, 0x00, 0x7F, 0x00, 0x00},
142 {0x00, 0x41, 0x36, 0x08, 0x00},
143 {0x08, 0x04, 0x08, 0x10, 0x08},
144
145 // Some special characters we created
146 {0x24, 0x54, 0x78, 0x54, 0x58}, // æ
147 {0x38, 0x64, 0x54, 0x4C, 0x38}, // ø
148 {0x24, 0x54, 0x55, 0x54, 0x78}, // å
149 {0x7E, 0x09, 0x7F, 0x49, 0x49}, // Æ
150 {0x3C, 0x62, 0x5A, 0x46, 0x3C}, // Ø
151 {0x78, 0x14, 0x15, 0x14, 0x78}, // Å
152 {0X7F, 0x10, 0x20, 0x20, 0x1F}, //
153 {0x20, 0x46, 0x40, 0x46, 0x20}, // :)
154 {0x1C, 0x3E, 0x7C, 0x3E, 0x1C} // <3
155 };
156
157 #endif /*! _ACHARSET_H_ */

```

C.6 gameport

gameport.h

```

1 #ifndef _gameport_h_
2 #define _gameport_h_
3
4 // Public
5 void initGameport();
6 unsigned char getGameportButtons();
7 char readSteeringWheel();
8
9 // Private
10 unsigned int readADC(unsigned char channel);
11
12 #endif

```

gameport.c

```

1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "gameport.h"
4 #include "ansi.h"
5
6 unsigned int readADC(unsigned char channel) {
7     unsigned char inHigh, inLow;
8     unsigned int ADC_data;
9
10    ADCCTL = 0x80 | 0x20 | (channel & 0x0F); // Enable ADC on the selected channel and use external voltage (3.3V) as
11    VREF
12    while (ADCCTL & 0x80); // Wait for conversion to be completed
13    inHigh = ADCD_H; // ADC high byte
14    inLow = ADCD_L; // ADC low low byte
15    ADC_data = ((unsigned int)inHigh << 2) | inLow >> 6; // ADC output word
16
17    return ADC_data;
18 }
19 void initGameport() {
20     PBDD = (1 << 5) | (1 << 4) | (1 << 3) | (1 << 2); // Use PB2, PB3, PB4 and PB5 as digital inputs
21     PBAF = (1 << 1) | (1 << 0); // Use PB0 and PB1 for ADC conversion

```

```

22 }
23
24 unsigned char getGameportButtons() {
25     return (((~PBIN) & 0x3C) >> 2);
26 }
27
28 char readSteeringWheel() {
29     int val = readADC(1);
30
31     //gotoxy(50,10);
32     //printf("Val: %04d",val);
33
34     // The driving wheel is not linear therefor this table is needed
35     if (val > 1000)
36         return -6; // We will move it more aggressively to the side
37     else if (val > 796)
38         return -4;
39     else if (val > 712)
40         return -2;
41     else if (val > 610)
42         return -1;
43     else if (val > 570)
44         return 0;
45     else if (val > 500)
46         return 1;
47     else if (val > 466)
48         return 2;
49     else if (val > 440)
50         return 4;
51     else
52         return 6;
53 }
```

C.7 LED

LED.h

```

1 #ifndef _led_h_
2 #define _led_h_
3
4 #define PRE1 (0 << 3);
5 #define PRE2 (1 << 3);
6 #define PRE4 (2 << 3);
7 #define PRE8 (3 << 3);
8 #define PRE16 (4 << 3);
9 #define PRE32 (5 << 3);
10 #define PRE64 (6 << 3);
11 #define PRE128 (7 << 3);
12
13 #define SCROLL_SPEED 5
14
15 // Public
16 void initLED(); // Initialize the LED display
17 void LEDsetString(char *string); // Used to scroll a string
18 void LEDRunOnce(char *firstString, char* secondString); // Used to scroll the first string once and then show the
   second string afterwards
19
20 // Private
21 void clockLed(unsigned char digit);
22 unsigned char convertChar(char input);
23 void moveVideoBuffer();
24 void LEDupdate();
25 void timer2int();
```

26
27 #endif

LED.c

```

1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "charset.h"
4 #include "LED.h"
5 #include "ansi.h"
6
7 unsigned char digit = 0, column = 0, delayCounter = 0, index = 0, stringLength = 0;
8 char videoBuffer[5][6];
9
10 volatile char runOnce, *pSecondString; // runOnce is used scroll a text only once, the pointer will point to the second
   string that is shown after the scroll is complete
11 char *pString; // Pointer to string in ram
12
13
14 void clockLed(unsigned char digit) {
15     if (digit == 0) {
16         PEOUT &= ~(1 << 7);
17         PEOUT |= (1 << 7);
18     }
19     else if (digit == 1) {
20         PGOUT |= (1 << 7);
21         PGOUT &= ~(1 << 7);
22     }
23     else if (digit == 2) {
24         PEOUT &= ~(1 << 5);
25         PEOUT |= (1 << 5);
26     }
27     else if (digit == 3) {
28         PEOUT &= ~(1 << 6);
29         PEOUT |= (1 << 6);
30     }
31 }
32
33 unsigned char convertChar(char input) { // Convert to some of our own characters
34     unsigned char c;
35     if (input == 'ı')
36         c = 'ı' + 1;
37     else if (input == '\oe')
38         c = 'œ' + 2;
39     else if (input == 'å')
40         c = 'å' + 3;
41     else if (input == 'Æ')
42         c = 'Æ' + 4;
43     else if (input == 'Ø')
44         c = 'Ø' + 5;
45     else if (input == 'Å')
46         c = 'Å' + 6;
47     else if (input == ' ')
48         c = ' ' + 7;
49     else if (input == ')') // Smiley
50         c = ')' + 8;
51     else if (input == '<3')
52         c = '<3' + 9;
53     else
54         c = input;
55
56     return c;
57 }
58
59 void LEDsetString(char *string) {

```

```

60     unsigned char i, j;
61
62     DI(); // Disable all interrupts
63
64     runOnce = 0; // Reset flag
65     stringLength = strlen(string); // Calculate length of string
66     pString = string; // Set pointer to the start of the string
67
68     for (i=0; i < 5; i++) {
69         for (j=0;j<5;j++)
70             videoBuffer[i][j] = character_data[convertChar(*pString)-0x20][j];
71         videoBuffer[i][5] = 0;
72
73         pString++;
74         if (*pString == '\0') {
75             pString -= stringLength;
76             break; // Break if we have reached the end of the string – this is due to the string being less than five characters wide
77         }
78     }
79     for (i = stringLength; i < 4; i++) {
80         for (j = 0; j < 5; j++)
81             videoBuffer[i][j] = character_data[' ' -0x20][j]; // Fill out the rest of the string with spaces if the string is less than five characters
82     }
83
84     digit = column = delayCounter = index = 0; // Reset all values used for multiplexing
85
86     EI(); // Enable all interrupts
87 }
88
89 void LEDRunOnce(char *firstString, char* secondString) { // Used to scroll the first string once and then show the second string afterwards
90     LEDsetString(firstString);
91     runOnce = 1;
92     pSecondString = secondString; // We will save the location of the second string
93 }
94
95 void moveVideoBuffer() {
96     unsigned char i, j;
97
98     for (i=0; i < 5; i++) {
99         for (j=0;j<5;j++) {
100             if (i < 4)
101                 videoBuffer[i][j] = videoBuffer[i+1][j]; // Shift all one to the left
102             else
103                 videoBuffer[4][j] = character_data[convertChar(*pString)-0x20][j]; // Read the next character in the string
104         }
105     }
106     pString++;
107     if (*pString == '\0') { // Check if we have reached the end of the string
108         if (runOnce) { // This wil actually abort when it loads the last character in the 5th digit, so you have to put a space in end of the sentence
109             runOnce = 0;
110             LEDsetString(pSecondString);
111         } else
112             pString -= stringLength;
113     }
114 }
115
116 void LEDupdate() { // This function is called inside the interrupt
117     PGOUT = (PGOUT & (1 << 7)) | *(&videoBuffer[0][0] + digit*6 + column + index);
118     PEOUT |= 0x1F; // Set all cathodes high
119     PEOUT &= ~(1 << (4-column)); // Set one cathodes low decided by column
120
121     clockLed(digit);

```

```

122 if (++digit == 4) {
123     digit = 0;
124     if (++column == 5) {
125         column = 0;
126         if (++delayCounter == SCROLL_SPEED && stringLength > 4) { // We don't have to scroll the text if there is
127             less than five characters
128             delayCounter = 0;
129             if (++index > 5) {
130                 index = 0;
131                 moveVideoBuffer();
132             }
133         }
134     }
135 }
136
137 #pragma interrupt
138 void timer2int() {
139     LEDupdate();
140 }
141
142 void initLED() {
143     unsigned char i;
144     PEDD = 0; // All output
145     PGDD = 0; // All output
146     PEOUT = 0x1F; // Set clocks to low and cathodes to high
147     PGOUT = 0; // Set all low
148
149     for (i=0;i<4;i++) // Turn all off by default
150         clockLed(i);
151
152     DI(); // Disable interrupt
153
154     T2CTL = 0; // TEN – disable timer
155     T2CTL |= PRE1; // PRES – Prescaler
156     T2CTL |= (1 << 0); // TMODE – continuous mode
157
158     T2H = 0;
159     T2L = 1;
160
161     T2RH = 9216 >> 8; // Interrupt every 500us
162     T2RL = 9216 & 0xFF;
163
164     SET_VECTOR(TIMER2, timer2int); // Enter the timer0int function at each interrupt
165
166     // Set timer2 priority to low
167     IRQOENH &= ~(1 << 7);
168     IRQOENL |= (1 << 7);
169
170     T2CTL |= (1 << 7); // TEN – enable timer
171
172     EI(); // Enable interrupt
173 }
```

C.8 levels

levels.h

```

1 #ifndef _levels_h_
2 #define _levels_h_
3
4 #include <eZ8.h> // special encore constants, macros and flash routines
5 #include "reflexball.h"
```

```

6
7 // This array contains all the levels for the game
8 unsigned char rom levels[6][BRICK_TABLE_HEIGHT][BRICK_TABLE_WIDTH] = {
9 { // Bier
10 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
11 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
12 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
13 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
14 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
15 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
16 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
17 { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0 },
18 { 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 3 },
19 { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1 },
20 { 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 3 },
21 { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1 },
22 { 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 3 },
23 { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1 },
24 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
25 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
26 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
27 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
28 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
29 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
30 },
31 { // Basic
32 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
33 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
34 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
35 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
36 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
37 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
38 { 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 },
39 { 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 },
40 { 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0 },
41 { 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0 },
42 { 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0 },
43 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
44 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
45 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
46 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
47 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
48 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
49 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
50 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
51 },
52 { // Cirkel
53 { 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },
54 { 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0 },
55 { 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0 },
56 { 0, 0, 1, 1, 0, 0, 2, 2, 0, 0, 1, 1, 0, 0 },
57 { 0, 0, 1, 0, 0, 2, 2, 2, 0, 0, 1, 0, 0 },
58 { 0, 1, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0, 1, 0 },
59 { 0, 1, 0, 2, 2, 0, 0, 0, 0, 2, 2, 0, 1, 0 },
60 { 0, 1, 0, 2, 0, 0, 4, 4, 0, 0, 2, 0, 1, 0 },
61 { 0, 1, 0, 2, 0, 0, 4, 4, 0, 0, 2, 0, 1, 0 },
62 { 0, 1, 0, 2, 2, 0, 0, 0, 2, 2, 0, 1, 0 },
63 { 0, 1, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0, 1, 0 },
64 { 0, 0, 1, 0, 2, 2, 2, 2, 0, 0, 1, 0, 0 },
65 { 0, 0, 1, 1, 0, 0, 2, 2, 0, 0, 1, 1, 0, 0 },
66 { 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0 },
67 { 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0 },
68 { 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0 },
69 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
70 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
71 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

```

```

72 },
73 { // Blokke
74 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
75 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
76 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
77 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
78 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
79 { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
80 { 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0 },
81 { 2, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 0 },
82 { 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0 },
83 { 2, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 0 },
84 { 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0 },
85 { 2, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 0 },
86 { 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0 },
87 { 2, 0, 1, 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 0 },
88 { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
89 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
90 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
91 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
92 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
93 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
94 },
95 { // Batman
96 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
97 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
98 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
99 { 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1 },
100 { 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0 },
101 { 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0 },
102 { 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0 },
103 { 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0 },
104 { 0, 2, 2, 0, 0, 1, 1, 1, 0, 0, 2, 2, 0 },
105 { 0, 0, 2, 2, 0, 0, 1, 1, 0, 0, 2, 2, 0, 0 },
106 { 0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 2, 0, 0, 0 },
107 { 0, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0, 0 },
108 { 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0 },
109 { 0, 0, 1, 1, 0, 0, 2, 2, 0, 0, 1, 1, 0 },
110 { 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0 },
111 { 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0 },
112 { 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0 },
113 { 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0 },
114 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
115 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
116 },
117 { // Usynlig
118 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
119 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
120 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
121 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
122 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
123 { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
124 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
125 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
126 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
127 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
128 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
129 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
130 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
131 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
132 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
133 { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 },
134 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
135 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
136 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
137 }

```

```
138 };
139
140 #endif
```

C.9 lut

lut.h

```
1 // =====
2 //
3 // Exported by Cearn's excellut v1.0
4 // (comments, kudos, flames to daytshen@hotmail.com)
5 //
6 //
7 // =====
8 #ifndef LUT_H
9 #define LUT_H
10 // === LUT SIZES ===
11 #define SIN_SIZE 512
13
14 // === LUT DECLARATIONS ===
15 extern const signed short rom SIN[512];
16
17 #endif // LUT_H
```

lut.c

```
1 // =====
2 // Look-Up Tables
3 // SIN: sin(x*pi/256)
4 //
5 // Exported by Cearn's excellut v1.0
6 // (comments, kudos, flames to daytshen@hotmail.com)
7 //
8 //
9
10 #include "lut.h"
11 //

12 // SIN: a 512 long LUT of 16bit values in 2.14 format
13 // sin(x*pi/256)
14 const signed short rom SIN[512] =
15 {
16     0x0000, 0x00C9, 0x0192, 0x025B, 0x0324, 0x03ED, 0x04B5, 0x057E,
17     0x0646, 0x070E, 0x07D6, 0x089D, 0x0964, 0x0A2B, 0x0AF1, 0x0BB7,
18     0x0C7C, 0x0D41, 0x0E06, 0x0ECA, 0x0F8D, 0x1050, 0x1112, 0x11D3,
19     0x1294, 0x1354, 0x1413, 0x14D2, 0x1590, 0x164C, 0x1709, 0x17C4,
20     0x187E, 0x1937, 0x19EF, 0x1AA7, 0x1B5D, 0x1C12, 0x1CC6, 0x1D79,
21     0x1E2B, 0x1EDC, 0x1F8C, 0x203A, 0x20E7, 0x2193, 0x223D, 0x22E7,
22     0x238E, 0x2435, 0x24DA, 0x257E, 0x2620, 0x26C1, 0x2760, 0x27FE,
23     0x289A, 0x2935, 0x29CE, 0x2A65, 0x2AFB, 0x2B8F, 0x2C21, 0x2CB2,
24 }
```

```

25 0x2D41,0x2DCF,0x2E5A,0x2EE4,0x2F6C,0x2FF2,0x3076,0x30F9,
26 0x3179,0x31F8,0x3274,0x32EF,0x3368,0x33DF,0x3453,0x34C6,
27 0x3537,0x35A5,0x3612,0x367D,0x36E5,0x374B,0x37B0,0x3812,
28 0x3871,0x38CF,0x392B,0x3984,0x39DB,0x3A30,0x3A82,0x3AD3,
29 0x3B21,0x3B6D,0x3BB6,0x3BFD,0x3C42,0x3C85,0x3CC5,0x3D03,
30 0x3D3F,0x3D78,0x3DAF,0x3DE3,0x3E15,0x3E45,0x3E72,0x3E9D,
31 0x3EC5,0x3EEB,0x3FOF,0x3F30,0x3F4F,0x3F6B,0x3F85,0x3F9C,
32 0x3FB1,0x3FC4,0x3FD4,0x3FE1,0x3FEC,0x3FF5,0x3FFF,0x3FFF,
33
34 0x4000,0x3FFF,0x3FFF,0x3FF5,0x3FEC,0x3FE1,0x3FD4,0x3FC4,
35 0x3FB1,0x3F9C,0x3F85,0x3F6B,0x3F4F,0x3F30,0x3F0F,0x3EEB,
36 0x3EC5,0x3E9D,0x3E72,0x3E45,0x3E15,0x3DE3,0x3DAF,0x3D78,
37 0x3D3F,0x3D03,0x3CC5,0x3C85,0x3C42,0x3BB6,0x3B6D,
38 0x3B21,0x3AD3,0x3A82,0x3A30,0x39DB,0x3984,0x392B,0x38CF,
39 0x3871,0x3812,0x37B0,0x374B,0x36E5,0x367D,0x3612,0x35A5,
40 0x3537,0x34C6,0x3453,0x33DF,0x3368,0x32EF,0x3274,0x31F8,
41 0x3179,0x30F9,0x3076,0x2FF2,0x2F6C,0x2EE4,0x2E5A,0x2DCF,
42
43 0x2D41,0x2CB2,0x2C21,0x2B8F,0x2AFB,0x2A65,0x29CE,0x2935,
44 0x289A,0x27FE,0x2760,0x26C1,0x2620,0x257E,0x24DA,0x2435,
45 0x238E,0x22E7,0x223D,0x2193,0x20E7,0x203A,0x1F8C,0x1EDC,
46 0x1E2B,0x1D79,0x1CC6,0x1C12,0x1B5D,0x1AA7,0x19EF,0x1937,
47 0x187E,0x17C4,0x1709,0x164C,0x1590,0x14D2,0x1413,0x1354,
48 0x1294,0x11D3,0x1112,0x1050,0xF8D,0x0ECA,0x0E06,0x0D41,
49 0x0C7C,0x0BB7,0x0AF1,0x0A2B,0x0964,0x089D,0x07D6,0x070E,
50 0x0646,0x057E,0x04B5,0x03ED,0x0324,0x025B,0x0192,0x00C9,
51
52 0x0000,0xFF37,0xFE6E,0xFDA5,0xFCDC,0xFC13,0xFB4B,0xFA82,
53 0xF9BA,0xF8F2,0xF82A,0xF763,0xF69C,0xF5D5,0xF50F,0xF449,
54 0xF384,0xF2BF,0xF1FA,0xF136,0xF073,0xEF0,0xEEEE,0xEE2D,
55 0xED6C,0xECAC,0xEBED,0xEB2E,0xEA70,0xE9B4,0xE8F7,0xE83C,
56 0xE782,0xE6C9,0xE611,0xE559,0xE4A3,0xE3EE,0xE33A,0xE287,
57 0xE1D5,0xE124,0xE074,0xDFC6,0xDF19,0xDE6D,0xDDC3,0xDD19,
58 0xDC72,0xDBCB,0xDB26,0xDA82,0xD9E0,0xD93F,0xD8A0,0xD802,
59 0xD766,0xD6CB,0xD632,0xD59B,0xD505,0xD471,0xD3DF,0xD34E,
60
61 0xD2BF,0xD231,0xD1A6,0xD11C,0xD094,0xD00E,0xCF8A,0xCF07,
62 0xCE87,0xCE08,0xCD8C,0xCD11,0xCC98,0xCC21,0xCBAD,0xCB3A,
63 0xCA9,0xCA5B,0xC9EE,0xC983,0xC91B,0xC8B5,0xC850,0xC7EE,
64 0xC78F,0xC731,0xC6D5,0xC67C,0xC625,0xC5D0,0xC57E,0xC52D,
65 0xC4DF,0xC493,0xC44A,0xC403,0xC3BE,0xC37B,0xC33B,0xC2FD,
66 0xC2C1,0xC288,0xC251,0xC21D,0xC1EB,0xC1BB,0xC18E,0xC163,
67 0xC13B,0xC115,0xC0F1,0xC0D0,0xC0B1,0xC095,0xC07B,0xC064,
68 0xC04F,0xC03C,0xC02C,0xC01F,0xC014,0xC00B,0xC005,0xC001,
69
70 0xC000,0xC001,0xC005,0xC00B,0xC014,0xC01F,0xC02C,0xC03C,
71 0xC04F,0xC064,0xC07B,0xC095,0xC0B1,0xC0D0,0xC0F1,0xC115,
72 0xC13B,0xC163,0xC18E,0xC1BB,0xC1EB,0xC21D,0xC251,0xC288,
73 0xC2C1,0xC2FD,0xC33B,0xC37B,0xC3BE,0xC403,0xC44A,0xC493,
74 0xC4DF,0xC52D,0xC57E,0xC5D0,0xC625,0xC67C,0xC6D5,0xC731,
75 0xC78F,0xC7EE,0xC850,0xC8B5,0xC91B,0xC983,0xC9EE,0xCA5B,
76 0xCA9,0xCB3A,0xCBAD,0xCC21,0xCC98,0xCD11,0xCD8C,0xCE08,
77 0xCE87,0xCF07,0xCF8A,0xD00E,0xD094,0xD11C,0xD1A6,0xD231,
78
79 0xD2BF,0xD34E,0xD3DF,0xD471,0xD505,0xD59B,0xD632,0xD6CB,
80 0xD766,0xD802,0xD8A0,0xD93F,0xD9E0,0xDA82,0xDB26,0xDBCB,
81 0xDC72,0xDD19,0xDDC3,0xDE6D,0xDF19,0xDFC6,0xE074,0xE124,
82 0xE1D5,0xE287,0xE33A,0xE3EE,0xE4A3,0xE559,0xE611,0xE6C9,
83 0xE782,0xE83C,0xE8F7,0xE9B4,0xEA70,0xEB2E,0xEBED,0xECAC,
84 0xED6C,0xEE2D,0xEEEE,0xEF0,0xF073,0xF136,0xF1FA,0xF2BF,
85 0xF384,0xF449,0xF50F,0xF5D5,0xF69C,0xF763,0xF82A,0xF8F2,
86 0xF9BA,0xFA82,0xFB4B,0xFC13,0xFCDC,0xFDA5,0xFE6E,0xFF37,
87 };

```

C.10 main

main.c

```

1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "reflexball.h"
4 #include "ansi.h"
5 #include "LED.h"
6 #include "time.h"
7 #include "buttons.h"
8 #include "gameport.h"
9 #include "asciidisplay.h"
10 #include "ascii.h"
11
12 unsigned long wheelTimer;
13
14 void main() {
15     int input;
16     unsigned char buttons;
17     char wheel;
18
19     initTimers();
20     initLED();
21     initButtons();
22     initGameport();
23     init_uart(_UART0,_DEFREQ,BAUD_115200);
24
25     color(2,0); // Green foreground, black background
26
27 Start:
28     clrscr();
29
30     LEDsetString(" ReflexBall RALLY!");
31
32     initStartMenu(3,1,224,82); // x1, y1, x2, y2
33     while(!startMenu()); // Wait for any key to be pressed
34     clrscr();
35     printMenu();
36     while(!updateMenu()); // Wait until difficulty is chosen
37
38     LEDsetString(" "); // Clear display
39     initReflexBall(3,15,224,82,1); // x1, y1, x2, y2, style
40
41     for(;;) {
42         buttons = getGameportButtons();
43
44         if (buttons & 0xE) // Gear backward or button press
45             startGame();
46         else if (millis() - wheelTimer > 20) { // We have to limit the update rate or it will move to fast
47             wheelTimer = millis();
48             wheel = readSteeringWheel();
49             if (wheel != 0)
50                 moveStriker(wheel);
51         }
52         else { // The driving wheel overrules the other controls
53             buttons = readButtons();
54             if (buttons) {
55                 if (buttons & 0x2) // Center
56                     startGame();
57                 else if (buttons & 0x4) // Left
58                     moveStriker(-1);
59                 else if (buttons & 0x1) // Right
60                     moveStriker(1);
61             }
62         }
63     }
64 }
```

```

62     else if (kbhit()) {
63         input = getch();
64         if (input == ' ') // Space
65             startGame();
66         else if (input == 68) // Left
67             moveStriker(-2);
68         else if (input == 67) // Right
69             moveStriker(2);
70     }
71 }
72 updateGame();
73 if (restartGame)
74     goto Start; // Goto back to the start of the game
75 }
76 }
```

C.11 math

math.h

```

1 #ifndef _math_h_
2 #define _math_h_
3
4 #define FIX14_SHIFT 14
5
6 #define FIX14_MULT(a, b) ((a*b) >> FIX14_SHIFT)
7 #define FIX14_DIV(a, b) (a << FIX14_SHIFT / b)
8
9 typedef struct {
10     long x,y;
11 } TVector;
12
13 // Public
14 long expand(long input);
15 void printNumber(long input, unsigned char decimalBits, unsigned char decimal);
16 void printFix(long input, unsigned char decimal);
17 long sin(int val);
18 long cos(int val);
19 void initVector(TVector* v, long x, long y);
20 void rotate(TVector* v, int val);
21 void printVector(TVector* v);
22
23 #endif
```

math.c

```

1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "math.h"
4 #include "lut.h"
5
6 long expand(long input) {
7     return input << 2; // Convert an 18.14 to 16.16
8 }
9
10 void printNumber(long input, unsigned char decimalBits, unsigned char decimal) {
11     int i;
12     unsigned long pow = 10, mask = 0xFFFFFFFF, result, output;
13
14     if ((input & 0x80000000) != 0) {
15         printf("-");
16         input = -input;
```

```

17 }
18
19 if (decimal > 5)
20     decimal = 5;
21
22 for (i=1;i<decimal;i++)
23     pow *= 10;
24
25 mask >>= (32 - decimalBits);
26 result = pow * (unsigned long)(input & mask);
27 output = result >> decimalBits;
28 output += (result >> (decimalBits-1)) & 0x1; // Round correctly
29
30 printf("%ld.%0*ld", input >> decimalBits, decimal, output);
31 }
32
33 void printFix(long input, unsigned char decimal) {
34     printNumber(input, FIX14_SHIFT, decimal);
35 }
36
37 long sin(int val) {
38     return SIN[val & 0x1FF];
39 }
40
41 long cos(int val) {
42     return sin(val + 128);
43 }
44
45 void initVector(TVector* v, long x, long y) {
46     v->x = x << FIX14_SHIFT;
47     v->y = y << FIX14_SHIFT;
48 }
49
50 void rotate(TVector* v, int val) {
51     long sinVal, cosVal, tempX;
52     sinVal = sin(val);
53     cosVal = cos(val);
54     tempX = v->x;
55
56     v->x = FIX14_MULT(v->x,cosVal) - FIX14_MULT(v->y,sinVal);
57     v->y = FIX14_MULT(tempX,sinVal) + FIX14_MULT(v->y,cosVal);
58 }
59
60 void printVector(TVector* v) {
61     printf("(\n");
62     printFix(v->x,4);
63     printf(",\n");
64     printFix(v->y,4);
65     printf(")\n");
66 }
```

C.12 reflexball

reflexball.h

```

1 #ifndef _reflexball_h_
2 #define _reflexball_h_
3
4 #include "ansi.h"
5 #include "math.h"
6
7 #define STRIKER_MAX_WIDTH 30 // This has to be even
8 #define STRIKER_MAX_ANGLE 64 // 360 deg = 512
```

```

9 #define NLIVES 3
10 #define DEFAULT_DIFFICULTY 40
11 #define UART_MAX_SPEED 20 // If it gets below this value, we will only draw it every second time or if it hits an
   object, this is because the UART can not send the characters fast enough
12 #define MAX_DIFFICULTY 10 // If speed gets under this value the UART can no longer keep up
13 #define BALL_WIDTH 4 // The modulus of this number should be even
14 #define BALL_HEIGHT 2
15
16 #define BRICK_TABLE_WIDTH 14
17 #define BRICK_TABLE_HEIGHT 20
18
19
20 typedef struct {
21     long x, y; // (x,y) is in the top left corner — these use the 18.14 format
22     unsigned char width, height;
23     TVector vector;
24 } Ball;
25
26 typedef struct {
27     unsigned char x, y, width;
28 } Striker;
29
30 typedef struct {
31     unsigned char x, y; // (x,y) is in the top left corner
32     unsigned char lives;
33     unsigned char width, height;
34 } Brick;
35
36 extern unsigned char divider; // This is the difficulty set in the beginning
37 extern unsigned char strikerWidth; // This is the striker width determent from the selected difficulty
38 extern unsigned char restartGame; // True if the user have won the game
39
40 // Public
41 void initReflexBall(unsigned char newX1, unsigned char newY1, unsigned char newX2, unsigned char newY2, char style)
42 ;
42 void startGame();
43 void stopGame();
44 void updateGame();
45 void moveStriker(char dir);
46
47 // Private
48 void printLevel();
49 void printLives();
50 void printScore();
51 void showScoreLED();
52 void scrollLiveInGameLED();
53 void scrollLevelUp();
54 void scrollAll();
55 void dead();
56 unsigned char getTerminalCoordinate(long input);
57 void gotoxyBall(long x, long y);
58 void clearBigBall(long x, long y);
59 void drawBigBall();
60 void drawBrick(Brick *brick);
61 void checkIteration(unsigned char x, unsigned char y);
62 void setBallPos(unsigned char x, unsigned char y);
63 void iterate();
64 void drawStriker();
65 void ballPosStriker();
66 void initStriker(unsigned char x, unsigned y, unsigned char width);
67 void initBricks(char clear);
68 void initBall();
69 void drawLevel();
70 void levelUp();
71
72 #endif

```

reflexball.c

```

1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "reflexball.h"
4 #include "time.h"
5 #include "levels.h"
6 #include "LED.h"
7 #include "ascii.h"
8 #include "asciidisplay.h"
9
10 Ball ball;
11 Striker striker;
12 Brick bricks[BRICK_TABLE_HEIGHT][BRICK_TABLE_WIDTH];
13 unsigned char strikerAngle[STRIKER_MAX_WIDTH/2-1+BALL_WIDTH/2-1];
14 unsigned int bricksLives; // Stores the total number of lives of the bricks
15
16 unsigned long gameTimer;
17 unsigned char gameStarted = 0, alive;
18
19 unsigned char x1, y1, x2, y2;
20
21 unsigned int score, levelScore; // The total score and the score obtained in this level
22 unsigned char divider; // This is the difficulty set in the beginning
23 unsigned char strikerWidth; // This is the striker width determent from the selected difficulty
24
25 unsigned char lives, level;
26 volatile char runOnceBuf[50], ledBuf[50]; // We need this global buffer as the LED code creates a pointer to the
   memory location
27
28 char drawCounter = 0;
29 char drawCounterMax = 1;
30 char drawBallNow;
31
32 long lastX, lastY;
33
34 unsigned char restartGame; // True if the user have won the game
35
36 void printLevel() {
37   gotoxy(x2-35,y2);
38   printf("Level: %d", level+1);
39 }
40 void printLives() {
41   gotoxy(x2-25,y2);
42   printf("Lives: %d", lives);
43 }
44 void printScore() {
45   gotoxy(x2-15,y2);
46   printf("Score: %04d$", score);
47 }
48
49 void showScoreLED() {
50   sprintf(ledBuf,"%04d",score);
51   LEDsetString(ledBuf);
52 }
53 void scrollLiveInGameLED() {
54   sprintf(ledBuf,"%04d",score);
55   sprintf(runOnceBuf,"%s$ Lives:%d %s ", ledBuf, lives, ledBuf); // We have to put a space behind as the LED
   routine aborts when it loaded the last character in the 5th digit
56
57 LEDRunOnce(runOnceBuf,ledBuf);
58 }
59 void scrollLevelUp() {
60   sprintf(ledBuf,"%04d", score);

```

```

61 sprintf(runOnceBuf, "%s$ Level up! Level:%d Lives:%d %s ", ledBuf, level+1, lives, ledBuf); // We have to put a
62   space behind as the LED routine aborts when it loaded the last character in the 5th digit
63 LEDRunOnce(runOnceBuf,ledBuf);
64 }
65 void scrollAll() {
66   sprintf(ledBuf,"%04d",score);
67   sprintf(runOnceBuf,"%s$ Level:%d Lives:%d %s ", ledBuf, level+1, lives, ledBuf); // We have to put a space
68   behind as the LED routine aborts when it loaded the last character in the 5th digit
69   LEDRunOnce(runOnceBuf,ledBuf);
70 }
71
72 void dead() {
73   if(--lives == 0) {
74     delay_ms(1000); // Wait a bit before showing Game Over ASCII string
75     sprintf(ledBuf,"%04d$ Game Over! Score:", score);
76     LEDsetString(ledBuf);
77     showGameOver();
78     restartGame = 1;
79     return;
80   } else {
81     scrollLiveInGameLED();
82     printLives();
83   }
84
85 alive = 0;
86 stopGame();
87 }
88
89 unsigned char getTerminalCoordinate(long input) {
90   unsigned char output = input >> FIX14_SHIFT;
91   output += (input >> (FIX14_SHIFT-1)) & 0x1; // Round correctly
92   return output;
93 }
94
95 void gotoxyBall(long x, long y) {
96   gotoxy(getTerminalCoordinate(x),getTerminalCoordinate(y));
97 }
98
99 void clearBigBall(long x, long y) {
100   gotoxyBall(x,y);
101   printf(" ");
102   gotoxyBall(x,y + ((long)1 << FIX14_SHIFT));
103   printf(" ");
104 }
105
106 void drawBigBall() {
107   const unsigned char top = 238, bottom = 95, slash = '/', backSlash = '\\';
108
109   if ((++drawCounter != drawCounterMax && !drawBallNow) || !alive) // Check if we should skip the drawing in this
110     iteration or if the game is not running
111   return;
112   drawCounter = 0;
113   drawBallNow = 0;
114
115   clearBigBall(lastX,lastY);
116
117   lastX = ball.x;
118   lastY = ball.y;
119
120   gotoxyBall(ball.x,ball.y);
121   printf("%c%c%c%c",slash,top,top,backSlash);
122   gotoxyBall(ball.x,ball.y + ((long)1 << FIX14_SHIFT));
123   printf("%c%c%c%c",backSlash,bottom,bottom,slash);
124 }
```

```

124
125 void drawBrick(Brick *brick) {
126     unsigned char i, j, brickStyle;
127     for (i=0;i<brick->height;i++) {
128         gotoxy(brick->x,brick->y+i);
129         for (j=0;j<brick->width;j++) {
130             if (!brick->lives)
131                 brickStyle = 15;
132             else if (brick->lives == 1)
133                 brickStyle = 176;
134             else if (brick->lives == 2)
135                 brickStyle = 177;
136             else if (brick->lives == 3)
137                 brickStyle = 178;
138             else if (brick->lives == 4)
139                 brickStyle = 219;
140             else // Invisible
141                 brickStyle = 15;
142             printf("%c",brickStyle);
143         }
144     }
145 }
146
147 void checkIteration(unsigned char x, unsigned char y) {
148     unsigned char i, j, dontDeflectX = 0, dontDeflectY = 0, deflectedX = 0, deflectedY = 0;
149     char distance; // Distance from center to ball position on striker
150     int angle;
151
152     if ((x > x1 && x+ball.width < x2 && y > y1 && y+ball.height-1 < y2-1) && gameStarted) {
153         for (i=0;i<BRICK_TABLE_HEIGHT;i++) {
154             for (j=0;j<BRICK_TABLE_WIDTH;j++) {
155                 if (!bricks[i][j].lives) // Skip if lives == 0
156                     continue;
157
158                 if (y-1 == bricks[i][j].y+bricks[i][j].height-1 || y+ball.height-1 == bricks[i][j].y-1 || x+ball.width-1
159                     == bricks[i][j].x-1 || x-1 == bricks[i][j].x+bricks[i][j].width-1)
160                     drawBallNow = 1; // Make sure it draws the ball in this iteration
161
162                 // Check if the ball hit a brick
163                 if (y <= bricks[i][j].y+bricks[i][j].height-1 && y+ball.height-1 >= bricks[i][j].y && x+ball.width-1
164                     >= bricks[i][j].x && x <= bricks[i][j].x+bricks[i][j].width-1) {
165                     score++;
166                     levelScore++;
167                     printScore();
168                     showScoreLED();
169
170                     // Check if the ball hit the top or the bottom of the brick
171                     if (y <= bricks[i][j].y+bricks[i][j].height-1 && y+ball.height-1 >= bricks[i][j].y) {
172                         if (!deflectedY) {
173                             if (x+ball.width-1 == bricks[i][j].x || x+ball.width-2 == bricks[i][j].x || x == bricks[i][j].x+
174                                 bricks[i][j].width-1 || x == bricks[i][j].x+bricks[i][j].width-2) { // Left or right side
175                             if (y+ball.height-1 >= bricks[i][j].y && y <= bricks[i][j].y) { // Top left or right corner on brick
176                                 if (i > 0 && bricks[i-1][j].lives) { // Make sure that there can actually be a brick above it
177                                     dontDeflectY = 1; // Don't deflect it
178                                     //gotoxy(x2-20,2);
179                                     //printf("Alive above! %04d",dontCounter++);
180                                 }
181                             }
182                         }
183                         if (y <= bricks[i][j].y+bricks[i][j].height-1 && y+ball.height-1 >= bricks[i][j].y+bricks[i][j].
184                             height-1) { // Bottom left or right corner on brick
185                             if (i+1 < BRICK_TABLE_HEIGHT && bricks[i+1][j].lives) { // Make sure that there can actually be
186                                 a brick underneath it
187                                 dontDeflectY = 1; // Don't deflect it
188                                 //gotoxy(x2-20,2);
189                                 //printf("Alive below! %04d",dontCounter++);
190                             }
191                         }
192                     }
193                 }
194             }
195         }
196     }
197 }
```

```

185         }
186     } else if (y >= bricks[i][j].y && y+ball.height-1 <= bricks[i][j].y+bricks[i][j].height-1) // Check
187         if we hit the brick directly from the side
188     dontDeflectY = 1;
189
190     if (y+ball.height-1 >= bricks[i][j].y && y <= bricks[i][j].y) { // Top of brick
191         if (ball.vector.y & 0x80000000) { // Negative
192             dontDeflectY = 1; // Don't deflect it
193             //gotoxy(x2-20,6);
194             //printf("Top negative! %04d",dontCounter++);
195         }
196         if (y <= bricks[i][j].y+bricks[i][j].height-1 && y+ball.height-1 >= bricks[i][j].y+bricks[i][j].
197             height-1) { // Bottom of brick
198             //gotoxy(x2-20,6);
199             //printf("Bottom ");
200             if (!(ball.vector.y & 0x80000000)) { // Positive
201                 dontDeflectY = 1; // Don't deflect it
202                 //printf("positive! %04d",dontCounter++);
203             } /*else
204                 printf("negative! %04d",dontCounter++);*/
205         }
206
207         if (!dontDeflectY) {
208             deflectedY = 1;
209             ball.vector.y = -ball.vector.y;
210             ball.y += 2*ball.vector.y;
211         } /*else if (dontDeflectY) {
212             gotoxy(10,4);
213             printf("dontDeflectY: %d",dontCounter++);
214         }*/
215     }
216     // Check if the ball hit one of the sides of the brick
217     if (x+ball.width-1 == bricks[i][j].x || x+ball.width-2 == bricks[i][j].x || x == bricks[i][j].x+bricks[i].
218         [j].width-1 || x == bricks[i][j].x+bricks[i][j].width-2) {
219         if (!deflectedX) {
220             if ((y+ball.height-1 >= bricks[i][j].y && y <= bricks[i][j].y) || (y <= bricks[i][j].y+bricks[i][j].
221                 height-1 && y+ball.height-1 >= bricks[i][j].y+bricks[i][j].height-1)) { // Top or bottom of
222                 brick
223                 if (x+ball.width-1 == bricks[i][j].x || x+ball.width-2 == bricks[i][j].x) { // Top or bottom left
224                     side
225                     if (j > 0 && bricks[i][j-1].lives) { // Check if there is one alive to the left of the brick
226                         dontDeflectX = 1; // Don't deflect it
227                         //gotoxy(x2-20,4);
228                         //printf("Alive left! %04d",dontCounter++);
229                     }
230                     if (x == bricks[i][j].x+bricks[i][j].width-1 || x == bricks[i][j].x+bricks[i][j].width-2) { // Top or
231                         bottom right side
232                         if (j+1 < BRICK_TABLE_WIDTH && bricks[i][j+1].lives) { // Check if there is one alive to the right
233                             of the brick
234                             dontDeflectX = 1; // Don't deflect it
235                             //gotoxy(x2-20,4);
236                             //printf("Alive right! %04d",dontCounter++);
237                         }
238                     }
239                 }
240             }
241         }
242     }

```

```

243         if (x == bricks[i][j].x+bricks[i][j].width-1 || x == bricks[i][j].x+bricks[i][j].width-2) { // Right
244             side
245             if (!(ball.vector.x & 0x80000000)) { // Positive
246                 dontDeflectX = 1; // Don't deflect it
247                 //gotoxy(x2-20,8);
248                 //printf("Right positive! %04d",dontCounter++);
249             }
250         }
251         if (y != bricks[i][j].y) { // It hasn't hit the brick directly from the side
252             if (y < bricks[i][j].y) { // Top
253                 if (!(ball.vector.y & 0x80000000) && !(i > 0 && bricks[i-1][j].lives)) { // Check if it is positive
254                     and make sure there is no brick above
255                     dontDeflectX = 1; // Don't deflect it
256                     //gotoxy(x2-20,14);
257                     //printf("Top none above! %04d",dontCounter++);
258                 }
259             } else { // Bottom
260                 if (ball.vector.y & 0x80000000 && !(i+1 < BRICK_TABLE_HEIGHT && bricks[i+1][j].lives)) { // Check if it is negative and make sure there is no brick below
261                     dontDeflectX = 1; // Don't deflect it
262                     //gotoxy(x2-20,14);
263                     //printf("Bottom none below! %04d",dontCounter++);
264                 }
265             }
266         }
267         if (!dontDeflectX) {
268             deflectedX = 1;
269             ball.vector.x = -ball.vector.x;
270             ball.x += 2*ball.vector.x;
271         } /*else if (dontDeflectX) {
272             gotoxy(10,2);
273             printf("dontDeflectX: %d",dontCounter++);
274         }*/
275     }
276 }
277 bricks[i][j].lives--;
278 bricksLives--;
279 drawBrick(&bricks[i][j]);
280 if (!bricksLives) {
281     levelUp();
282     return;
283 }
284 ball.vector.x >= 1; // We need to set this back again before rotating
285 angle = (int)(millis() & 0x7) - 3; // Pseudo random number from -3 to 4
286 //gotoxy(x1+10,y2-10);
287 //printf("Angle: %02d", angle);
288 rotate(&ball.vector, angle); // Slightly rotate the ball from -3 to 4, to make the game more exciting
289 ball.vector.x <= 1; // The x vector needs to be twice as large due to the y-axis being twice as high on
290     the screen
291 }
292 }
293
294 if (dontDeflectX && dontDeflectY) {
295     //gotoxy(x2-20,12);
296     //printf("CornerHit: %d",dontCounter++);
297     ball.vector.y = -ball.vector.y;
298     ball.y += 2*ball.vector.y;
299     ball.vector.x = -ball.vector.x;
300     ball.x += 2*ball.vector.x;
301 }
302 }
303 else if (((y+ball.height-1 > striker.y) || (y+ball.height-1 == striker.y && (x+ball.width <= striker.x || x
304     >= striker.x+striker.width))) && gameStarted) { // If you are below the striker then player must be dead

```

```

304     drawBallNow = 1;
305     drawBigBall();
306     dead();
307     return;
308 }
309 else if (gameStarted) {
310     if (x <= x1 || x+ball.width >= x2) {
311         ball.vector.x = -ball.vector.x;
312         ball.x += 2*ball.vector.x;
313     }
314     if ((y <= y1) || (y+ball.height-1 == striker.y && x+ball.width > striker.x && x < striker.x+striker.
315         width)) {
316         if (y+ball.height-1 == striker.y) { // Check if we hit the striker
317             printScore();
318             showScoreLED();
319
320             distance = x+ball.width-1 - striker.x;
321
322             if (distance < striker.width/2-1+BALL_WIDTH/2-1) // The ball hit the left side
323                 angle = strikerAngle[((striker.width/2-1)+(BALL_WIDTH/2-1))-1]-distance];
324             else if (distance > striker.width/2+1+BALL_WIDTH/2-1) // The ball hit the right side
325                 angle = -(int)strikerAngle[distance-((striker.width/2+1)+(BALL_WIDTH/2-1))+1]; // IMPORTANT:
326                 remember to cast to int before the minus sign
327             else
328                 angle = 0;
329
330             if (angle != 0) {
331                 /*gotoxy(10,20);
332                 printf(" ");
333                 gotoxy(10,20);
334                 printf("Rotating: %d", angle);*/
335
336             ball.vector.x >= 1; // We need to set this back again before rotating
337             rotate(&ball.vector,angle);
338
339             if (((ball.vector.y >> 12) & 0x7) == 0) { // Check if below 1/4 or approximately 15 degrees
340                 /*gotoxy(10,16);
341                 printf("BeforeY: ");
342                 printFix(ball.vector.y,4);*/
343                 ball.vector.y = 1 << 12; // 1/4
344                 /*printf(" AfterY: ");
345                 printFix(ball.vector.y,4);*/
346
347                 /*printf(" BeforeX: ");
348                 printFix(ball.vector.x,4);*/
349                 if (ball.vector.x & 0x80000000) // Negative
350                     ball.vector.x = -(long)15864; // -(cos(arcsin(1/4)) << 14)
351                 else
352                     ball.vector.x = 15864; // cos(arcsin(1/4)) << 14
353                 /*printf(" AfterX: ");
354                 printFix(ball.vector.x,4);*/
355             }
356             ball.vector.x <= 1; // The x vector needs to be twice as large due to the y-axis being twice as high on
357             the screen
358         }
359         /*gotoxy(10,10);
360         printf("%02d %02d",distance,angle);*/
361         ball.vector.y = -ball.vector.y;
362         ball.y += 2*ball.vector.y;
363     }
364 }

```

```

366 if (x == x1+1 || x+ball.width == x2-1 || y == y1+1 || (y+ball.height-1 == striker.y-1 && x+ball.width >
367     striker.x && x < striker.x+striker.width)) // Check if we are next to the sides, top or striker
368     drawBallNow = 1; // If so then draw the ball in this iteration
369
370 drawBigBall();
371 }
372
373 void setBallPos(unsigned char x, unsigned char y) {
374     ball.x = (long)x << FIX14_SHIFT;
375     ball.y = (long)y << FIX14_SHIFT;
376 }
377
378 void iterate() {
379     if (alive && gameStarted) {
380         ball.x += ball.vector.x;
381         ball.y += ball.vector.y;
382     }
383     checkIteration(getTerminalCoordinate(ball.x),getTerminalCoordinate(ball.y));
384 }
385
386 void drawStriker() {
387     const unsigned char strikerStyle = 223;
388     unsigned char i;
389     gotoxy(striker.x,striker.y);
390     if (divider == 1) // Blink striker if Chuck Norris mode is on
391         blink(1);
392     for (i=0;i<striker.width;i++)
393         printf("%c",strikerStyle);
394     if (divider == 1)
395         blink(0);
396 }
397
398 void ballPosStriker() {
399     setBallPos(striker.x+striker.width/2-ball.width/2,striker.y-ball.height);
400     iterate();
401     drawStriker(); // Redraw striker in case the ball clears part of the stikers
402 }
403
404 void moveStriker(char dir) { // Take care of moving the striker left or right
405     char absDir = dir;
406     if (!lives) // If no lives left then return
407         return;
408     if (dir < 0)
409         absDir = -dir;
410     if ((int)striker.x + dir <= x1)
411         striker.x = x1+1;
412     else if (striker.x+striker.width-1 + dir >= x2)
413         striker.x = x2-striker.width;
414     else
415         striker.x += dir;
416
417     if (dir < 0) // Left
418         gotoxy(striker.x+striker.width,striker.y);
419     else // Right
420         gotoxy(striker.x-dir,striker.y);
421
422     for (; absDir > 0; absDir--)
423         printf(" "); // Clear old char
424
425     drawStriker();
426
427     if (!gameStarted) {
428         ballPosStriker();
429         if (!alive)
430             startGame();

```

```

431     }
432 }
433
434 void initStriker(unsigned char x, unsigned y, unsigned char width) {
435     unsigned char i, dAngle, strikerZones;
436
437     if (width > STRIKER_MAX_WIDTH)
438         width = STRIKER_MAX_WIDTH;
439
440     strikerZones = width/2-1+BALL_WIDTH/2-1; // The striker is split up into zones to the left and right of the striker
441     center
442     striker.x = x - width/2;
443     striker.y = y;
444     striker.width = width;
445     drawStriker();
446
447     dAngle = STRIKER_MAX_ANGLE/strikerZones; // Delta angle between striker characters
448
449 //gotoxy(10,8);
450 for (i = 1; i <= strikerZones; i++) {
451     strikerAngle[i-1] = dAngle*i; // Fill in the array – the longer distance from the center, the larger angle it will
452     be reflected with
453     //printf("%d ", strikerAngle[i-1]);
454 }
455
456 void stopGame() {
457     gameStarted = 0;
458 }
459
460 void initBricks(char clear) {
461     unsigned char i, j;
462
463     bricksLives = 0; // Reset
464
465     for (i=0;i<BRICK_TABLE_HEIGHT;i++) {
466         for (j=0;j<BRICK_TABLE_WIDTH;j++) {
467             bricks[i][j].width = 14;
468             bricks[i][j].height = 2;
469
470             bricks[i][j].lives = levels[level][i][j];
471             bricksLives += bricks[i][j].lives;
472             bricks[i][j].x = x1+6 + (bricks[i][j].width+1)*j;
473             bricks[i][j].y = y1+3 + (bricks[i][j].height+1)*i;
474             if (bricks[i][j].lives || clear)
475                 drawBrick(&bricks[i][j]);
476         }
477     }
478 }
479
480 void startGame() {
481     int startAngle;
482     if (!alive) {
483         alive = 1;
484         if (lives == 0) {
485             score = 0;
486             levelScore = 0;
487             showScoreLED();
488             initReflexBall(x1,y1,x2,y2,1);
489
490             gameTimer = 0;
491             printLives();
492             printScore();
493             printLevel();
494         }

```

```

495     ballPosStriker();
496 } else if (!gameStarted) {
497     initVector(&ball.vector,1,0);
498     startAngle = (millis() & 0x7F) - 192; // Calculate a "random" angle from 0–127 (0–89.3 deg) and then
499     subtract 192 (135 deg)
500
501     //gotoxy(10,6);
502     //printf("Start angle: %02d",(startAngle*360)/512);
503
504     rotate(&ball.vector, startAngle);
505
506     /*printf(" ");
507     printFix(ball.vector.x,4);
508     printf(" ");
509     printFix(ball.vector.y,4);*/
510
511     ball.vector.x <<= 1; // The x vector needs to be twice as large due to the y-axis being twice as high on the
512     screen
513     gameTimer = 0;
514     gameStarted = 1;
515     updateGame();
516 }
517
518 void updateGame() {
519     int speed = DEFAULT_DIFFICULTY - levelScore/divider; // Calculate teh speed using the score obtained in the
520     current level and the divider set in the start up menu
521     if (speed < MAX_DIFFICULTY || divider == 1) // Limit maximum speed and set maximum speed as default if Chuck
522         Norris mode is enabled
523     speed = MAX_DIFFICULTY;
524
525     if (speed < UART_MAX_SPEED) // If it gets below this value, we will only draw it every second time or if it hits an
526         object, this is because the UART can not send the characters fast enough
527     drawCounterMax = 2;
528     else
529         drawCounterMax = 1;
530
531     if (millis() - gameTimer > speed && gameStarted) {
532         gameTimer = millis();
533         iterate();
534     }
535
536 void initBall() {
537     setBallPos(striker.x+striker.width/2-2,striker.y-2); // Initialize the ball position to the striker position, as
538     ballPosStriker will clear this
539     ball.width = BALL_WIDTH; // This has to be even
540     ball.height = BALL_HEIGHT;
541     initVector(&ball.vector,0,0);
542 }
543
544 void drawLevel(char clear) {
545     unsigned char i;
546
547     gotoxy(striker.x,striker.y);
548     for (i=0;i<striker.width;i++)
549         printf(" "); // Clear old striker
550     initStriker((x2-x1)/2+x1,y2-1,strikerWidth); // The width of the striker should always be even
551
552     initBall(); // Initialize to striker position
553
554     alive = 1;
555     gameStarted = 0;
556     ballPosStriker();
557
558     printLives();
559     printScore();

```

```

555     printLevel();
556
557     gameTimer = 0;
558
559     initBricks(clear);
560 }
561
562 void levelUp() {
563     level++;
564     if (level >= sizeof(levels)/sizeof(levels[0])) { // Check if the user won the game
565         showWon();
566         if (divider != 1) { // If not in Chuck Norris mode, then the user wins the game
567             restartGame = 1;
568             return;
569         }
570         initReflexBall(x1,y1,x2,y2,1);
571         return;
572     }
573     lives += 2;
574
575     levelScore = 0;
576     scrollLevelUp();
577     drawLevel(1);
578 }
579
580 void initReflexBall(unsigned char newX1, unsigned char newY1, unsigned char newX2, unsigned char newY2, char style)
581 {
582     unsigned char leftTop, rightTop, leftBot, rightBot, verSide, horSide, leftCross, rightCross;
583
584     x1 = newX1;
585     y1 = newY1;
586     x2 = newX2;
587     y2 = newY2;
588
589     if (style) { // Bold
590         leftTop = 201;
591         rightTop = 187;
592         leftBot = 200;
593         rightBot = 188;
594         verSide = 186;
595         horSide = 205;
596         leftCross = 185;
597         rightCross = 204;
598     } else { // Normal
599         leftTop = 218;
600         rightTop = 191;
601         leftBot = 192;
602         rightBot = 217;
603         verSide = 179;
604         horSide = 196;
605         leftCross = 180;
606         rightCross = 195;
607     }
608
609     lastX = (long)((x1+x2)/2) << FIX14_SHIFT; // Sets the last ball coordinates just above striker, so it clears the ball
610     // correctly the first time
611     lastY = (long)(y2-1-BALL_HEIGHT) << FIX14_SHIFT;
612
613     clrscr(); // Clear the screen
614
615     // Put the title in the top of the screen — this is done when animateTitle() is called inside updateGame()
616     printAsciiXY(titleAscii1[0],sizeof(titleAscii1)/sizeof(titleAscii1[0]),(x1+x2)/2-strlen_rom(titleAscii1[0])/2,
617                 y1/2-(sizeof(titleAscii1)/sizeof(titleAscii1[0]))/2);
618
619     drawSides(x1,y1,x2,y2,verSide);
620     drawTopBot(x1,y1,x2-x1-1,leftTop,rightTop,horSide);

```

```

618     restartGame = 0;
619     level = 0;
620     score = 0;
621     levelScore = 0;
622     lives = NLIVES;
623     drawLevel(0);
624     scrollAll();
625 }

```

C.13 time

time.h

```

1 #ifndef _time_h_
2 #define _time_h_
3
4 #define PRE1 (0 << 3);
5 #define PRE2 (1 << 3);
6 #define PRE4 (2 << 3);
7 #define PRE8 (3 << 3);
8 #define PRE16 (4 << 3);
9 #define PRE32 (5 << 3);
10 #define PRE64 (6 << 3);
11 #define PRE128 (7 << 3);
12
13 #define PRIORITY_TIMER0 (1 << 5)
14 #define PRIORITY_TIMER1 (1 << 6)
15 #define PRIORITY_TIMER2 (1 << 7)
16
17 // Public
18 void initTimers();
19 unsigned long millis();
20 void delay_ms(unsigned long time);
21
22 // Private
23 void timer1int();
24
25 #endif

```

time.c

```

1 #include <eZ8.h> // special encore constants, macros and flash routines
2 #include <sio.h> // special encore serial i/o routines
3 #include "time.h"
4 #include "ansi.h"
5
6 volatile unsigned long delayTimer, mscounter;
7
8 void initTimers() {
9     DI(); // Disable interrupt
10
11     T1CTL = 0; // TEN – disable timer
12     T1CTL |= PRE1; // PRES – Prescaler
13     T1CTL |= (1 << 0); // TMODE – continuous mode
14
15     T1H = 0;
16     T1L = 1;
17
18     T1RH = 18432 >> 8; // Interrupt every 1ms
19     T1RL = 18432 & 0xFF;
20

```

```
21 SET_VECTOR(TIMER1, timer1int); // Enter the timer1int function at each interrupt
22
23 // Set timer1 priority to high
24 IRQOENH |= PRIORITY_TIMER1;
25 IRQOENL |= PRIORITY_TIMER1;
26
27 delayTimer = 0;
28 mscounter = 0;
29
30 T1CTL |= (1 << 7); // TEN – enable timer
31
32 EI(); // Enable interrupt
33 }
34
35 unsigned long millis() {
36     return mscounter;
37 }
38
39 void delay_ms(unsigned long time) { // This is not that accurate, but good enough for our needs
40     delayTimer = time;
41     while(delayTimer);
42 }
43
44 #pragma interrupt
45 void timer1int() {
46     delayTimer--;
47     mscounter++;
48 }
```