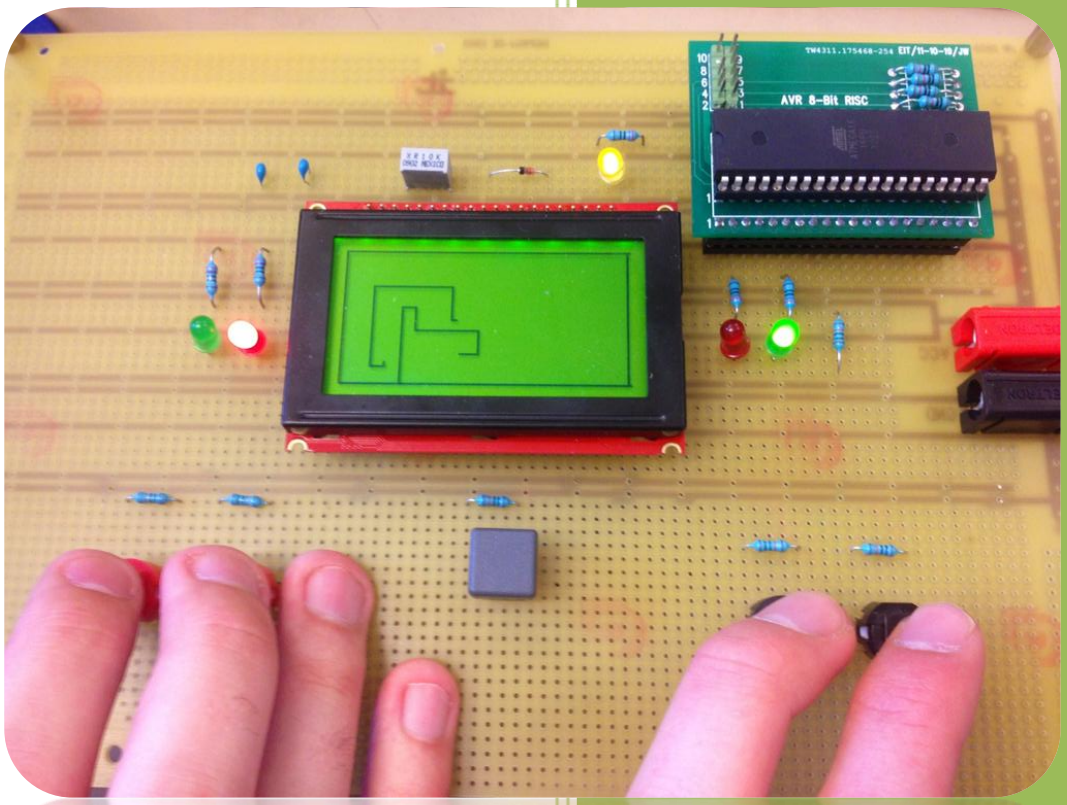
EITF11

WormFight



Axel Eriksson, Felix Geuken

Handledare: Bertil Lindvall

EITF11

Innehåll

WormFight

Inledning
För att utveckla våra färdigheter inom hårdvar- och mjukvaruutveckling har vi valt att utveckla en spelkonsol.

Tanken är att göra ett spel som kallas ”WormFight”. Det är ett spel som har inspirerats av det klassiska spelet ”Snake” från slutet av 1970-talet. Tanken är man ska vara två spelare som spelar mot varandra med var sin mask och målet kommer att vara att överleva så länge som möjligt. Spelet avslutas om någon av spelarna kör in i väggarna eller kör in i den andra masken.

Kravspecifikation
Spelet ska fungera enligt ursprungsidén med tillägget att man även ska kunna krocka in i den andra spelaren. För att kunna navigera krävs fyra stycken knappar, två till varje spelare, som möjliggör riktningsändringar till både höger och vänster.
Maskarnas framfart visas på en display och utgör också själva spelplanen där kampen utförs. Spelet ska avbrytas av att en mask kör in i någon av de fyra väggarna, motståndaren eller sig själv. En grön diod tänds då för vinnaren och en röd för förloraren. Krockar båda spelarna samtidigt tänds röda dioder för båda spelarna. För att starta spelet krävs också en knapp som man trycker på, vilket även ska fungera om man vill starta om spelet efter en avslutad spelomgång.

Teori - Hårdvara

Processor - AVR ATMega16
Processorn som vi har använt i det här projektet är en ATMega16 vilken har 1kb arbetsminne samt 40 ben där majoriteten är fördelade till fyra portar (A-D).

- Port A använder vi för att skicka och ta emot data från displayen.
  Port B används till övriga funktioner för displayen såsom Read/Write och ChipSelect. Porten används också till att reglera några av lysdioderna.
- Port C är till stora delar reserverad för debugging med JTAG men också några lysdioder.
- Port D används huvudsakligen till att lyssna på knapparna.

Display - GDM12864C
Displayen vi använder till det här projektet är en LED-display med 128x64 pixlar. Pixlarna är fördelade på två delar med vardera 64x64 pixlar. Pixlarna är ytterligare fördelade i 8 ”sidor” (pages) i x-led där varje sida innehåller 8 pixlar (det vill säga en byte).

Knappar
Knapparna som vi har valt att använda inom det här projektet är av en mycket enkel modell. När knappen är nedtryckt blir strömmen till aktuellt ben på processorn låg,

annars hålls den hög. Processorn lyssnar i början av varje spelrunda på varje knapp genom att avgöra spänningen till ett specifikt ben och avgör sedan om en riktningsändring ska ske eller om masken ska fortsätta i befintlig riktning.

Lysdioder

Dioderna vi använder är enbart för ytterligare effekt samt avgöra fall då det ser ut som att båda spelarna har krockat. Dessa är kopplade till var sin pinne på processor som sänder ut en ström och tänder dioderna.

Resistorer

Resistorerna använder vi för att minska spänningen till processorn från knapparna och lysdioderna. Till dioderna är de nödvändiga då dioden inte tål för hög spänning. Vi har även använt oss av en reglerbar resistor för att kunna ställa in en behaglig vinkel på skärmen.

Kondensator

Kondensatorerna använde vi i vår konstruktion för att undvika att få spänningssvängningar, det vill säga få en jämnare spänning till framförallt skärm och processor.

JTAG

Modul som kopplas till processorn för att kunna köra processorn i AVR Studio 4. Nödvändig för att kunna felsöka och förbättra spelet.

Metod

Konstruktion

Efter färdigställandet av kopplingsschemat och efter mottagande av nödvändig hårdvara började vi att konstruera själva WormFight-konsolen. Tanken var att skapa en konsol så att spelarna kan spela bekvämt utan att trängas samtidigt som att man ser displayen tydligt. Med hjälp av en basplatta av större modell genomfördes detta. Då vår prototyp inte har särkilt mycket hårdvara var den fysiska konstruktionen relativt snabbt monterad och klar för mjukvara.

Efter några dagars effektivt lödande och kopplande verkade konsolen fungera som den skulle och benen mottog rätt signaler från exempelvis knapparna. Detta testades kontinuerligt med hjälp av logikpenna för att avgöra den specifika signalen.

Efter en tids lärande genom att sköta varje pinne för sig i AVR Studio 4 verkade således hårdvaran fungera och utföra det den sades åt att utföra. Displayen ritade ut angiven data och lysdioderna tändes på begäran. C-funktionerna började sedan ta form som löste det vi hade knappat pinne-för-pinne mycket fortare och noggrannare.

Vid senare tillfällen lades kondensatorerna till i konstruktionen då problem uppstod med spel utan JTAG inkopplat.

Spelet

För att kontrollera de båda maskarna i spelet använde vi oss av ett flertal variabler och vektorer där aktuell data sparas. Försök gjordes även med matriser som bas, men på grund av begränsningar i arbetsminne fick dessa idéer skrotas.

Masken i sig är en serie pixlar på rad. Dels för enkelhet, dels för känslan av större spelplan utgörs masken endast av en pixel i bredd. Längden avgörs sedan till viss del av hur länge masken överlever men också av en begränsning till 100 pixlar på grund av minnesbrist. Med hjälp av vektorer kan vi på så sätt hålla koll på hela masken och med hjälp av detta också få hela masken att flytta på sig.

För att kontrollera maskens riktning introducerades en variabel som enbart håller koll på riktning. Knapparna reglerar sedan denna variabel som sedan genom en metod som sedan avgör om masken ska röra sig uppåt, nedåt, höger eller vänster.

För att identifiera en krasch används metoder som kontrollerar om nästa pixel som masken ska röra sig till finns i någon av maskarnas vektorer eller om masken kommer att röra sig in i väggen. I sådana fall avbryts spelet.

Har ingen krasch identifierats fortsätter spelet med en till runda med förflyttningar av de båda maskarna. Beroende på vilken mask det är som krockar utformade vi metoder som ger ytterligare feedback till spelarna genom att tända lysdioder med olika färg ovanför spelarnas knappar.

Resultat

Resultatet är en fullt fungerande spelkonsol enligt vår kravspecifikation. Det går att spela två spelare samtidigt mot varandra och så länge ingen av maskarna kör in i väggen, den egna masken eller motståndarmasken så fortsätter spelet. De fem knapparna fungerar som önskat och de styr maskarna åt rätt håll. Enda nackdelen med knapparna är att de är väldigt känsliga för knapptryckningar och spelarna kan lätt råka göra två svängningar när man bara vill göra en. Det finns möjlighet att göra knapparna mindre känsliga men då skulle spelet bli mycket långsammare vilket vi anser är sämre för spelupplevelsen. Vi fick även dioderna att lysa som önskat.

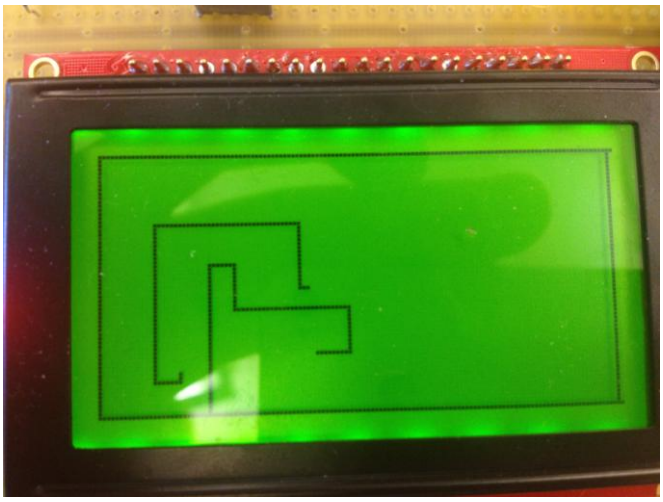
Diskussion

Sammanfattningsvis kan man säga att allting som kunde gå fel gick fel. Många gånger kändes det som att det var slumpen som avgjorde om det gick bra eller inte. Men på något sätt vi fick ihop konstruktionen och kunde konstatera att det varken var fel på processorn, skärmen eller AVR-studios vilket vi stundtals hade antagit.

Av de problem som uppstod var många av den sort som är nästintill omöjliga att förutse. Ett handlade om en optimeringsfunktion som finns inbyggd i AVR-studio som vi inte visste förrän i slutet att den existerade. Efter att denna hade stängts slutade AVR att anropa funktioner utan att vi hade kodat anropen, vilket gjorde att allting fungerade mycket bättre..Ett annat handlade om att spänningen till processorn och skärmen var ojämn vilket krävde att två kondensatorer fick läggas till i konstruktionen. Efter dessa

fungerade allt fysiskt som det skulle. På grund av våra begränsade tidigare erfarenheter tog dessa problem mycket längre tid att förstå än vad de borde ha tagit.

I övrigt handlade de största problemen som vi mötte under projektets gång om kommunikationen mellan processorn och displayen. Det tog ett tag att förstå och kunna utnyttja skärmens uppbyggnad och framförallt hur ”sidorna” i x-led fungerade.



Ett annat problem som tog en del tid var uppbyggnaden av själva spelplanen. Ursprungsidén var en matris med samma dimensioner som skärmen men denna idé fick skrotas då arbetsminnet blev alldeles för högt belastat. Problemet med minnesbrist har vi inte stött på tidigare då vi har arbetat på kraftigare processorer som inte lider av samma problem. Att spara varje besökt pixel löste minnesproblemet men istället var skärmen nu tvungen att läsas av för att inte radera andra markerade pixlar på samma sida.

Att markera varje besökt pixel ledde också till minnesproblem, men inte i samma utsträckning som tidigare. Den begränsning som uppkom är att vi max kunde spara undan omkring 200 pixlar. Grundtanken var att masken skulle bli längre och längre med spelets gång utan men detta gör att maskarna bara kunde bli 100 pixlar vardera.

Den slutliga lösningen blev att maskarna också kunde röra sig framåt det vill säga att den sista pixeln tas bort efterhand. Det leder inte bara att spelet varar längre utan också gör att spelarna verkligen måste jobba ordentligt för att kunna stänga in sin motståndare. Sammantaget en bättre spelupplevelse.

Problemet vi fortfarande har med att knapparna är väldigt känsliga kan eventuellt förbättras med hjälp av en ”OR”-brygga. Detta kan dock leda till att om en spelare väljer att fortsätta hålla sin knapp inne blir den andra spelaren helt begränsad i sina rörelser. Vår lösning med att lyssna på varje spelare blir i vår mening rättvisare och bättre för vår användning.

I allmänt har vi fått ökade kunskaper inom både hård- och mjukvaruutveckling. Svårighetsgraden kan dock ses som överkant trots en relativt enkel konstruktion. Att varken ha programmerat i C eller arbetat med hårdvara tidigare gjorde att även enkla problem kunde dra ut på tiden. De programmeringskunskaper vi hade sedan tidigare kunde i viss mån användas med till exempel grundförståelsen för språket men att göra metoder ben-specifika samt att C inte är objektorienterat som JAVA gjorde som sagt att tiden kunde stundtals springa iväg.

Bristen på kunskap har dock gett ytterligare erfarenheter i att söka och hitta relevant information vilket kan ses som nyttigt.

Referenser

Kernghan, Brian W. och Ritchie, Dennis M. 1988. *The C Programming Language*. 2. uppl. Englewood Cliffs: Prentice Hall.

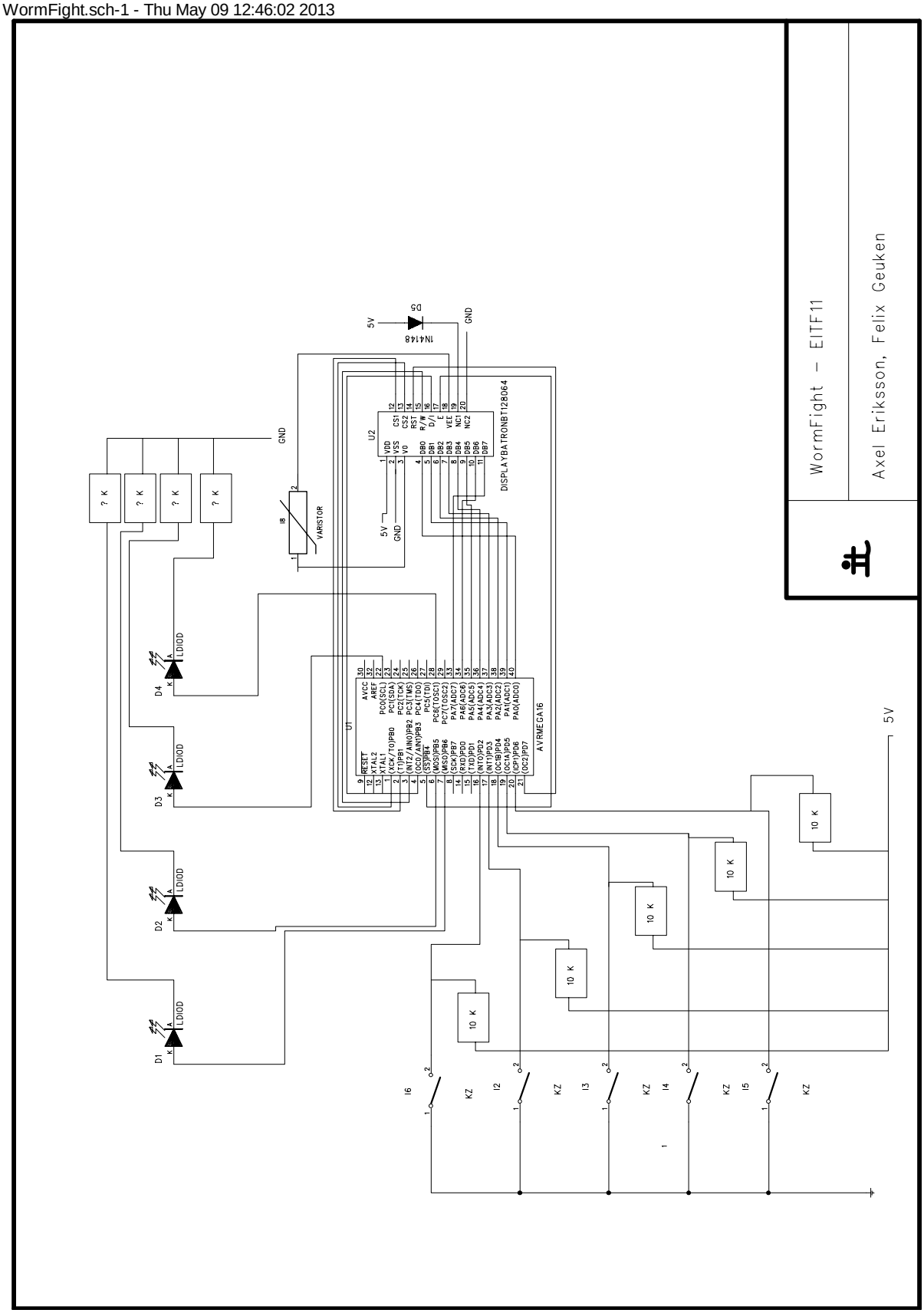
Datablad för display (GDM12864HLCM.http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Display/GDM12864H.pdf) (Hämtad 2013-05-09)

Datablad för processor AVR ATMega16 (http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf) (Hämtad 2013-05-09)

Bilagor

Kopplingsschema

WormFight.sch-1 - Thu May 09 12:46:02 2013



Källkod

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

unsigned short int snakeOneX;
unsigned short int snakeOneY;
short int snakeOneAngle;
unsigned short int snakeTwoX;
unsigned short int snakeTwoY;
short int snakeTwoAngle;
short int cause;
unsigned short int buttons;
unsigned short int snakeTwo[100][2];
unsigned short int snakeOne[100][2];
unsigned short int count;
unsigned short int gameNbr = 1;

void actionEventE()
{
		PORTB = PORTB | 0x10;
		PORTB = PORTB & 0xEF;
		PORTB = PORTB | 0x10;
}

// Tänder den vänstra gröna lampan
void lightLeftGreen()
{
		PORTB = PORTB | 0x20;
}

// Släcker den vänstra gröna lampan
void unlightLeftGreen()
{
		PORTB = PORTB & 0xDF;
}

// Tänder den vänstra röda lampan
void lightLeftRed()
{
		PORTB = PORTB | 0x40;
}

// Släcker den vänstra röda lampan
void unlightLeftRed()
{
		PORTB = PORTB & 0xBF;
}

// Tänder den högra gröna lampan
```

```
void lightRightGreen()
{
		PORTC = PORTC | 0x40;
}

// Släcker den högra gröna lampan
void unlightRightGreen()
{
		PORTC = PORTC & 0xBF;
}

// Tänder den högra vänstra lampan
void lightRightRed()
{
		PORTC = PORTC | 0x01;
}

// Släcker den högra vänstra lampan
void unlightRightRed()
{
		PORTC = PORTC & 0xFE;
}

unlightAll()
{
		unlightLeftGreen();
		unlightLeftRed();
		unlightRightGreen();
		unlightRightRed();
}

// Kontrollerar att skärmen är redo att mottaga ny information.
void checkIfReady()
{
		DDRA = 0x00;
		short int tmp = PORTB;
		PORTB = PORTB | 0x15;
		PORTB = PORTB & 0xF5;
		actionEventE();
		while(PINA != 0x00){
			actionEventE();
		}

		PORTB = PORTB | 0x16;
		PORTB = PORTB & 0xF6;
		actionEventE();
		while(PINA != 0x00){
			actionEventE();
		}
```

```
		DDRA = 0xFF;
		PORTB = tmp;
}

int transformToPinout(int z)
{
		if(z == 0)
		{
			return 0x01;
		}
		else if (z == 1)
		{
			return 0x02;
		}
		else if (z == 2)
		{
			return 0x04;
		}
		else if (z == 3)
		{
			return 0x08;
		}
		else if (z == 4)
		{
			return 0x10;
		}
		else if (z == 5)
		{
			return 0x20;
		}
		else if (z == 6)
		{
			return 0x40;
		}
		else
		{
			return 0x80;
		}
}

// Lyssnar på den vänstra svarta knappen
void readLeftBlackButton()
{
		short int directions = PIND | 0xDF;
		if(directions == 0xDF)
		{
			snakeTwoAngle -= 1;
		}
```

```
}

// Lyssnar på den högra svarta knappen
void readRightBlackButton()
{
		short int directions = PIND | 0xBF;
		if(directions == 0xBF)
		{
			snakeTwoAngle += 1;
		}
}

// Lyssnar på den vänstra röda knappen
void readLeftRedButton()
{
		short int directions = PIND | 0xFB;
		if(directions == 0xFB)
		{
			snakeOneAngle -= 1;
		}
}

// Lyssnar på den högra röda knappen
void readRightRedButton()
{
		short int directions = PIND | 0xF7;
		if(directions == 0xF7)
		{
			snakeOneAngle += 1;
		}
}

void delay(short int t){
		while(t > 0)
		{
			for(int i = 800; i >= 0; i--);
			t--;
		}
}

// Kontrollerar knapparna
void readButtons()
{
		readLeftBlackButton();
		delay(20);
		readRightBlackButton();
		delay(20);
		readLeftRedButton();
		delay(20);
```

```
		readRightRedButton();
}

// Ger nästa x-koordinat
int getNextX(int snakeAngle, int snakeX)
{
		if(snakeAngle % 4 == 0)
		{
			return snakeX - 1;
		} else
		{
			return snakeX + 1;
		}
}

// Ger nästa y-koordinat
int getNextY(int snakeAngle, int snakeY)
{
		if(snakeAngle % 4 == 1)
		{
			return snakeY + 1 ;
		} else
		{
			return snakeY - 1;
		}
}

// Kontrollerar om pixeln är markerad eller ej.
int isOccupied(short int x, short int y)
{
		for(short int i = 0; i < 100; i++){
			if((x == snakeOne[i][0] && y == snakeOne[i][1]) || (x ==
snakeTwo[i][0] && y == snakeTwo[i][1]))
			{
			return 1;
			}
		}
		return 0;
}

// Kontrollerar om masken är utanför banan.
int outOfBounds(short int x, short int y)
{
		if(x > 62 || x < 1 || y < 1 || y > 126)
		{
			return 1;
		}
		return 0;
}
```

```
// Kontrollerar om angiven mask har kraschat.
int hasCrashed(int snakeAngle,  int snakeX, int snakeY)
{
		short int direction = snakeAngle % 2;
		short int x;
		short int y;
		if (direction == 0)
		{
			x = getNextX(snakeAngle, snakeX);
			y = snakeY;
		} else
		{
			y = getNextY(snakeAngle, snakeY);
			x = snakeX;
		}
		int out = outOfBounds(x, y);
		int is = isOccupied(x, y);
		if (out == 1 || is == 1)
		{
			return 1;
		}
		return 0;
}

// Kontrollerar om någon mask har kraschat.
void checkIfCrashed()
{
		short int statusOne = hasCrashed(snakeOneAngle, snakeOneX, snakeOneY);
		short int statusTwo = hasCrashed(snakeTwoAngle, snakeTwoX,
snakeTwoY);

		if(statusOne == 1 && statusTwo == 0)
		{
			// spelare ett har krockat
			cause = 1;
		} else if(statusOne == 0 && statusTwo == 1)
		{
			// spelare två har krockat
			cause = 2;
		} else if(statusOne == 1 && statusTwo == 1)
		{
			// båda spelarna har krockat
			cause = 3;
		}
		else if(getNextX(snakeOneAngle,snakeOneX) ==
getNextX(snakeTwoAngle,snakeTwoX) && getNextY(snakeOneAngle,snakeOneY) ==
getNextY(snakeTwoAngle,snakeTwoY)){
			// båda spelarna har krockat
```

```
                        cause = 3;
                }
}

int readCurrentPage(short int xPage, short int y)
{
                short int tmp = 0;
                for(int i = 0; i < 100; i++)
                {
                        if(snakeOne[i][1] == y && snakeOne[i][0] / 8 == xPage){
                                tmp += transformToPinout(snakeOne[i][0] % 8);
                        }
                        if(snakeTwo[i][1] == y && snakeTwo[i][0] / 8 == xPage){
                                tmp += transformToPinout(snakeTwo[i][0] % 8);
                        }
                }
                if(xPage == 0)
                {
                        tmp += transformToPinout(0);
                }
                if(xPage == 7)
                {
                        tmp += transformToPinout(7);
                }
                return tmp;
}

void unmarkPixel(short int x, short int y)
{
                DDRA = 0xFF;
                PORTB = PORTB & 0xF0;

                int yAddress;
                checkIfReady();
                if(y < 64){
                        yAddress = y;                   // Markerar ena skärmhalvan
                        PORTB = PORTB | 0x01;
                } else
                {
                        yAddress = y - 64; // Markerar andra skärmhalvan
                        PORTB = PORTB | 0x02;
                }

                int xPage = x / 8;
                short int xDot = x % 8;

                PORTA = 0x40 + yAddress;
                checkIfReady();
                actionEventE();
```

```
		PORTA = 0xB8 + xPage;
		checkIfReady();
		actionEventE();


		PORTA = 0xC0;
		checkIfReady();
		actionEventE();
			// Sätter z = 0

		PORTA = readCurrentPage(xPage, y) & (0xFF - transformToPinout(xDot));
		PORTB = PORTB | 0x08;
		checkIfReady();
		actionEventE();

		PORTB = PORTB & 0xF7;
		PORTA = 0x00;
}

// Markerar angiven pixel.
void markPixel(short int x, short int y)
{
		DDRA = 0xFF;
		PORTB = PORTB & 0xF0;

		int yAddress;
		checkIfReady();
		if(y < 64){
			yAddress = y;			// Markerar ena skärmhalvan
			PORTB = PORTB | 0x01;
		} else
		{
			yAddress = y - 64; // Markerar andra skärmhalvan
			PORTB = PORTB | 0x02;
		}

		int xPage = x / 8;
		short int xDot = x % 8;

		PORTA = 0x40 + yAddress;
		checkIfReady();
		actionEventE();

		PORTA = 0xB8 + xPage;
		checkIfReady();
		actionEventE();
```

```
		PORTA = 0xC0;
		checkIfReady();
		actionEventE();
			// Sätter z = 0

		PORTA = readCurrentPage(xPage, y) | transformToPinout(xDot);
		PORTB = PORTB | 0x08;
		checkIfReady();
		actionEventE();

		PORTB = PORTB & 0xF7;
		PORTA = 0x00;
}

// Flyttar mask 1.
void moveOne()
{
		if (snakeOneAngle % 2 == 0)
		{
			snakeOneX = getNextX(snakeOneAngle, snakeOneX);
		} else
		{
			snakeOneY = getNextY(snakeOneAngle, snakeOneY);
		}
		markPixel(snakeOneX, snakeOneY);
		if(count > 99)
		{
			unmarkPixel(snakeOne[count % 100][0], snakeOne[count %
100][1]);
		}
		snakeOne[count % 100][0] = snakeOneX;
		snakeOne[count % 100][1] = snakeOneY;
}

// Flyttar mask 2.
void moveTwo()
{
		if (snakeTwoAngle % 2 == 0)
		{
			snakeTwoX = getNextX(snakeTwoAngle, snakeTwoX);
		} else
		{
			snakeTwoY = getNextY(snakeTwoAngle, snakeTwoY);
		}
		markPixel(snakeTwoX, snakeTwoY);
		if(count > 99)
		{
			unmarkPixel(snakeTwo[count % 100][0], snakeTwo[count %
100][1]);
```

```
		}
		snakeTwo[count % 100][0] = snakeTwoX;
		snakeTwo[count % 100][1] = snakeTwoY;
}

// Metod för effekt vid slut.
void gameOver()
{
		if (cause == 1)
		{
				lightRightGreen();
				lightLeftRed();
		}
		else if (cause == 2)
		{
				lightLeftGreen();
				lightRightRed();
		}
		else if(cause == 3)
		{
				lightLeftRed();
				lightRightRed();
		}
}

// Huvudloop för spelet. Den yttre while-loopen möjliggör för flera omgångar av spelet
// efter varandra.
void game()
{
		while(1)
		{
				buttons = PIND | 0xEF;
				if (buttons == 0xEF)
				{
						if (gameNbr > 1)
						{
								unlightAll();
								unmarkRows();
								gameSetup();
								delay(500);
						}
						while(1)
						{
								readButtons();
								delay(1);
								checkIfCrashed();
								if(cause == 1 || cause == 2 || cause ==
3)
								{
```

```
                                                            break;
                                                    }
                                                    moveOne();
                                                    moveTwo();
                                                    count++;
                                            }
                                            gameOver();
                                            gameNbr++;
                                    }
                            }
}

void markWholePage(short int xPage, short int y)
{
                    DDRA = 0xFF;
                    PORTB = PORTB & 0xF0;

                    int yAddress;
                    checkIfReady();
                    if(y < 64){
                                    yAddress = y;                   // Markerar ena skärmhalvan
                                    PORTB = PORTB | 0x01;
                    } else
                    {
                                    yAddress = y - 64; // Markerar andra skärmhalvan
                                    PORTB = PORTB | 0x02;
                    }

                    PORTA = 0x40 + yAddress;
                    checkIfReady();
                    actionEventE();

                    PORTA = 0xB8 + xPage;
                    checkIfReady();
                    actionEventE();


                    PORTA = 0xC0;
                    checkIfReady();
                    actionEventE();
                                    // Sätter z = 0


                    PORTA = 0xFF;
                    PORTB = PORTB | 0x08;
                    checkIfReady();
                    actionEventE();

                    PORTB = PORTB & 0xF7;
```

```
		PORTA = 0x00;
}

void makeFrame()
{
		for (short int k = 0; k < 128; k++)
		{
			markPixel(0,k);
			markPixel(63,k);
		}
		for (short int i = 0; i < 8; i++)
		{
			markWholePage(i,0);
			markWholePage(i,126);
		}
}

void unmarkRow(int x, int y)
{
		DDRA = 0xFF;
		PORTB = PORTB & 0xF0;
		int yAddress;
		if(y < 64){
			yAddress = y;
			PORTB = PORTB | 0x02;
		} else
		{
			yAddress = y - 64;
			PORTB = PORTB | 0x01;
		}

		PORTA = 0x40 + yAddress;
		checkIfReady();
		actionEventE();

		PORTA = x;
		PORTA = 0xB8 + x;
		checkIfReady();
		actionEventE();

		PORTA = 0x00;
		PORTB = PORTB | 0x08;
		checkIfReady();
		actionEventE();
		PORTB = PORTB & 0xF7;
		PORTA = 0x00;
}

void unmarkRows()
```

```
{
		for(int y = 0; y < 128; y++)
		{
			for(int x = 0; x < 8; x++)
			{
				unmarkRow(x,y);
			}
		}
}

void ddrSet()
{
		DDRA = 0xFF;
		DDRB = 0x7F;
		DDRC = 0x41;
		DDRD = 0x80;
}

void checkIfReadyCS1()
{
		DDRA = 0x00;
		short int tmp = PORTB;
		PORTB = PORTB | 0x15;
		PORTB = PORTB & 0xF5;
		actionEventE();
		while(PINA != 0x20){
			actionEventE();
		}

		DDRA = 0xFF;
		PORTB = tmp;
}

void checkIfReadyCS2()
{
		DDRA = 0x00;
		short int tmp = PORTB;
		PORTB = PORTB | 0x16;
		PORTB = PORTB & 0xF6;
		actionEventE();
		while(PINA!= 0x20){
			actionEventE();
		}

		DDRA = 0xFF;
		PORTB = tmp;
}

void startUpScreen()
```

```
{
		PORTA = 0x3F;
		PORTB = 0x01;
		checkIfReadyCS1();
		actionEventE();


		PORTB = 0x02;
		checkIfReadyCS2();
		actionEventE();
}

void clearMemorySnakeOne()
{
		for(int i = 0; i < 100; i++)
		{
			for(int j = 0; j < 2; j++)
			{
				snakeOne[i][j] = 0;
			}
		}
}

void clearMemorySnakeTwo()
{
		for(int i = 0; i < 100; i++)
		{
			for(int j = 0; j < 2; j++)
			{
				snakeTwo[i][j] = 0;
			}
		}
}

void gameSetup()
{
		clearMemorySnakeOne();
		clearMemorySnakeTwo();
		makeFrame();
		snakeOneX = 30;
		snakeOneY = 90;
		snakeOneAngle = 102;
		markPixel(30,30);
		snakeTwoX = 30;
		snakeTwoY = 30;
		snakeTwoAngle = 102;
		markPixel(30,90);
		cause = 0;
		count = 0;
```

```
}

void reset()
{
		PORTD = 0x00;
		actionEventE();
		PORTD = 0x80;
		actionEventE();
		PORTD = 0x00;
		actionEventE();
		PORTD = 0x80;
		actionEventE();
}

void setUp()
{
		ddrSet();
		reset();
		startUpScreen();
		checkIfReady();
		unmarkRows();
		gameSetup();
}

void main(void)
{
		setUp();
		game();
}
```