Luffaren

Ett luffarschack av och för luffarschackgamers

Projekt Digitala System, EITA15
20/5 - 2022

Handledare:
Bertil Lindvall, Lars- Göran Larsson

Gruppmedlemmar grupp 2:
Emil Adolfsson, Molly Beijar Bittar, Alma Flygare,
Emil Jakobsson, Christoffer Johansson Lundqvist

Sammanfattning

Denna rapport beskriver projektarbetet i kursen Digitala System, EITA15, där gruppen med hjälp av tidigare inlärda kunskaper från kursen ska konstruera valfri produkt. För denna rapport valde gruppen att konstruera ett luffarschack där neopixlar skulle visa en spelplan och en knappsats som användes för att avgöra vart spelarna ville placera markörerna. Knappsatsen skulle även ge möjligheten att byta mellan att spela mot en annan spelare och att spela mot en dator i olika svårighetsgrader. För att nollställa spelplanen skulle det även finnas en resetknapp på knappsatsen.

Projektarbetet började med inhämtning av information för samtliga hårdvarudelar från respektive datablad. Utefter informationen som inhämtats ritades ett första utkast av kopplingsschema. Utifrån kopplingsschemat virades och löddes samtliga komponenter på ett kopplingsbräde. Samtidigt påbörjades programmeringen i programmet Atmel Studio 7 där första fokus var att enskilt tända neopixlarna i valda färger. Därefter övergick programmeringen i att styra spel mellan två spelare för att sedan övergå i spel mot dator.

Projektet resulterade i ett luffarschack med en 3x3 spelplan styrd av en knappsats. På knappsatsen väljs markör som tänder den representerade neopixeln i färgen för spelaren. Bredvid spelplanen finns en tionde neopixel som visar färgen för den spelaren som ska lägga nästa drag. På knappsatsen väljer användaren om spel mot dator önskas genom att klicka på önskad svårighetsgrad där den svåraste ej går att vinna mot. För att återställa spelplanen klickar användaren på knappsatsens resetknapp.

Nyckelord

ATmega1284, Luffarschack, C, neopixel, AI

Abstract

This report describes the project made in the course Digitala System, EITA15, where the group creates a product by using their previous knowledge from the course. For this report the group chose to create a Tic Tac Toe with a gamepad made out of neopixels and a keypad to decide where the marker should be set. The aim was that the keypad should also have given the opportunity to switch between playing against another player and playing against a computer with different degrees of difficulty. Another implementation would also be a reset button to reset the gamepad.

To start with, the project involved collecting information about the hardware from the components datasheets. A draft of a circuit schematic was made from the collected information. With the help of the circuit diagram each component was wound or soldered onto a breadboard. Simultaneously the programming started using the IDE Atmel Studio 7, in which the focus was at first to make the neopixels glow in the chosen colour. After getting the desired colour combinations, the programming focus was switched to switch between two players. After the turn switching was programmed, the focus of the programming was then to create different AI:s with different difficulty levels that the player could choose to play against.

The project resulted in a Tic Tac Toe with a 3x3 size gamepad that is controlled by a keypad. By clicking the keypad the player chooses which neopixel they want to make their move on. Just beside the playing field there is a tenth neopixel which glows in the colour corresponding to which player is

supposed to make the next move. By using the keypad the player can select to play against three different AI by pressing the button corresponding to the different AI difficulty levels. In order to reset the playing field the player needs to press the keypads reset button

Keywords
ATmega1284, Tic-Tac-Toe, C, neopixel, AI

Innehållsförteckning

1. Inledning

1.1 Bakgrund

I kursen Digitala System på Lunds Universitet är projektarbete ett av kursmomenten. I grupp ska studenterna med hjälp av mikroprocessorn ATmega 1284 skapa en fungerande prototyp. Projektet ska innehålla en kravspecifikation, kopplingsschema, fysisk konstruktion och utveckling av mjukvara. Typ av projekt har valts av gruppen i samråd med lärare.

1.2 Syfte

Syftet med projektet är att bygga en fungerande konstruktion från idéstadie till färdig prototyp, öka problemlösningsförmågan och förståelsen för digitala system.

1.3 Kravspecifikation

I kravspecifikationen definierades ett antal mål för den slutgiltiga prototypen. Det ska gå att välja att spela mot datorn eller en annan person, där spelet för båda lägen fortsätter tills dess att någon antingen vunnit eller brädet är fullt. Spelar man mot datorn ska det finnas tre svårighetsgrader varav det svåraste innebär att datorn alltid vinner. En neopixel bredvid spelplanen ska indikera vems tur det är genom att byta färg. För knappsatsen som består av 4x4 knappar ska nio av dem representera spelplanen, som består av nio neopixlar som tänds i spelarens färg när den representerande knappen tryckts ned. För resterande knappar reserveras tre för svårighetsgrader, en för att välja att spela mot en person och en för att återställa spelet. Vid vinst ska neopixlarna blinka i vinnarens färg. Vid oavgjort ska neopixlarna lysa i rött.

1.4 Problemformulering

Vilka komponenter krävs och och hur ska de kopplas för att prototypen ska fungera? Hur ska de olika svårighetsgraderna kodas? Hur kodas neopixlarna för att de ska visa rätt?

2. Teknisk bakgrund

2.1 Hårdvara

- **Processor:** ATmega1284, 8-bitars AVR Microcontroller. Processor med 4 portar och 32 programmerbara I/O-pinnar.
- **Knappsats:** Grayhill 83BB1-001, 4x4 knappsats för att kunna ge inputs till programmet.
- **Nyckelkodare:** MM74C922, 4x4 key encoder, kopplades till knappsatsen för att kunna avkoda dess inputs.
- **J-TAG:** Atmel-ICE debugger som används för att överföra programmet till mikroprocessorn.
- **Neopixlar:** 10 stycken WS2812B RGB LEDs, kopplade till separata DATApins. 9 pixlar utgör spelplanen och 1 indikerar vems tur det är.
- **Kondensatorer:** Två kondensatorer på 0,1 µF respektive 1 µF kopplades till nyckelkodaren för att undvika att signalen från knapparna studsar.

2.2 Mjukvara

- **Atmel Studio 7**: Prototypen programmerades i Atmel Studio 7 i programspråken C och Assembly.

- **EAGLE:** Kretschemat skapades i Autodesks ritprogram EAGLE.

3. Metod

Gruppmedlemmarna för projektet beslutade sig för att skapa en egen digitaliserad version av spelet luffarschack. Därefter skapades ett första utkast av kravspecifikation och kopplingsschema som, med hjälp av handledare, bestämde komponenter och kopplingar för konstruktionen. Kopplingsschemat ritades i programmet Autodesk Eagle och fakta om behövda komponenter hämtades i komponenternas datablad. Möten och laborationstillfällen bokades via Facebook då detta ansågs vara det enklaste sättet att kommunicera.

Utefter kopplingsschemat började arbetet med att koppla och fästa komponenterna på mönsterkortet med virning och lödning. Under processens gång reviderades kopplingsschemat för att underlätta programmeringen som samtidigt påbörjades i programmet Atmel Studio 7. Detta innebar att vissa omkopplingar gjordes av det som redan virats.

När alla komponenter var kopplade och fästa på mönsterkortet började fasen som innehöll programmering parallellt med simulering och felsökning med hjälp av Atmel Studio 7´s debugger. Fokus låg först på att få varje enskild neopixel att lysa i önskad färg för att sen övergå i att kunna styra neopixlarna med hjälp av knappsatsen. När spel mot annan spelare fungerade övergick fokus till spel mot datorn. Programmeringen byggdes upp olika för olika svårighetsgrader där normal ansågs enklast och därmed var den som fokuserades först på. Vidare programmerades svårighetsgraden lätt som byggdes upp på liknande sätt som normal. Mot slutskedet skrevs programkoden för den svåraste svårighetsgraden, som fortsatte arbetas på fram till projektets sista dag.

3.1 Källkritik

Källorna som har använts för projektet är de olika komponenternas datablad där trovärdigheten kan anses vara hög eftersom dessa kommer direkt från leverantören av produkterna. Databladet för neopixlarna kan dock ifrågasättas då neopixlarna i verkligheten var mer toleranta än vad som visades i databladet.

4. Analys

Valet att använda neopixlar gjordes för att gruppmedlemmarna önskade lampor som kan lysa i flera olika färger. Gruppen ansåg att knappsatsen skulle användas för att få många knappar utan att behöva alla kopplingar som enskilda knappar behöver. För att på ett enkelt sätt avkoda knapptryckningarna valdes att använda en nyckelkodare som ser till att varje knapp på knappsatsen får ett enskilt värde som för att tydas läses av med hjälp av debuggern och sedan används för kodningen.

Assemblykod användes för att styra neopixlarna eftersom det färdiga biblioteket som var tänkt att användas ej lyckades. För att få neopixlarna att lysa krävdes det att ettor och nollor skickades i specifika tider enligt datablad. Dessa tider justerades i assemblykoden genom att ändra antal cykler som de olika värdena sändes och önskat antal cykler räknades ut med processorns klockfrekvens. Efter lång tid utan att få önskade färger föreslog handledare mätning av signaler med oscilloskop vilket visade helt andra tider än beräknat. Tiden för låg signal var mycket lång trots att den skickats i mycket få cykler men detta visade sig ej ha någon större betydelse. När neopixlarna testades med de nya justerade tiderna så stämde färgerna överens med datablad och önskat sken.

Vid testande av MiniMax koden så var det tydligt att något var mycket fel. AI:n försökte inte vinna när den hade 2 i rad och försökte inte heller hindra spelaren från att vinna när spelaren hade 2 i rad.
Det visade sig det inte räckte med själva MiniMax-metoden, det behövdes kod som gjorde att det första AI:n testade var ifall den kunde spela ett vinnande drag. Lösningen till problemet var att gå igenom hela spelplanen och leta upp vilka rutor som AI:n kunde ha som nästa drag, därefter kolla ifall något av dessa drag skulle göra att AI:n hade vunnit.Efter att sådan kod implementerades testades AI:n igen och nu spelade AI:n ett vinnande drag när det var möjligt. Problemet om att AI:n inte hindrade spelaren från att vinna fanns fortfarande kvar, detta löstes genom använda samma tänk som användes vid lösningen av förra problemet. En liknande algoritm som för att hitta det vinnande draget letades istället upp för motståndaren och en placering gjordes för att blockera.

Ett hårdvaru problem var att den första processor inte fungerade, detta blev självklart när det provades att köra ett tomt program med endast en while-loop med en breakpoint i. När programmet kördes så lästes aldrig koden i while-loopen av. Efter ett byte av processor så fungerade koden för att få neopixlarna att lysa upp.

5. Resultat

Detta projekt resulterade i ett luffarschacksspel uppbyggt av en ATmega1284 där 9 stycken neopixlar visar spelplanen och 1 styck neopixel visar med grön eller blå färg vem som ska göra nästa drag. Spelet har 3 olika svårighetsgrader där en undviker att lägga vinnande drag, en som slumpar sina drag och en som aktivt söker efter “bra” drag. Vid vinst blinkar samtliga 9 pixlar i spelplanen i vinnarens färg. Om spelplanen fyllts utan vinnare så blinkar spelplanen i rött. För att återställa spelplanen finns en resetknapp i knappsatsen som gör att samtliga lampor i spelbrädet släcks och spelet kan börja om.

Projektet har även resulterat i en HTML-baserad hemsida med information riktad till personer utan teknisk bakgrund. Hemsidan sammanfattar projektet och dess funktioner. Därutöver finns även åtkomst till rapporten, källkoden samt kretsschemat.

6. Slutsats

Vad gäller svårighetsgraderna så fungerar de som de ska, där “svår” inte går att vinna mot då datorn aktivt söker efter de bästa dragen samtidigt som den blockerar användarens potentiella vinnande drag, “normal” lägger slumpmässiga drag och “lätt” aktivt söker efter de sämsta dragen som kan läggas. Även läget för människa mot människa fungerar som planerat.

Neopixlarna för spelplanen fungerar som de ska, men pixeln bredvid planen som indikerar vems tur det är visar ibland fel i början av spelet.

Resetknappen uppfyller kraven att återställa spelplanen, men triggar till synes slumpmässigt då och då spelplanens pixlar, varefter man får trycka på en knapp och sedan reset för att återställa spelet igen. Övriga knappar, för att välja svårighetsgrad och spelläge, fungerar som tänkt.

7. Källförteckning

Datablad:
Nyckelkodare (MM74C922), 1999-01. [online]:
https://www.soemtron.org/downloads/disposals/74c922.pdf
Hämtad: 2022-05-02

Knappsats (Grayhill Series 87, 4x4), datum ej tillgängligt. [online]:
https://www.grayhill.com/documents/87-Datasheet
Hämtad: 2022-05-02

WS2812B (neopixlar), 2016-01. [online]:
https://voltiq.ru/datasheets/WS2812B_datasheet_EN.pdf
Hämtad: 2022-04-13

ATMEGA1284 (microprocessor), 2016-10. [online] :
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42718-ATmega1284_Datasheet.pdf
Hämtad: 2022-04-06.

8. Appendix

Assembler1.s klassen

```
.global send_one
.global send_zero
.global send_ret
.global read_buttons

#define PORTD 0x0B
#define DDRD 0x0A
#define PIND 0x09
#define PINA 0x20
#define DDRA 0x01
#define PORTB 0x05
#define DDRB 0x04

send_one:
        ldi r19, 0b11111111

        out PORTB, r19
        out PORTD, r19
        nop
        nop
        ldi r18, 0b00000000
        nop
        out PORTB, r18
        out PORTD, r18

        ret

send_zero:
        ldi r18, 0b11111111
        out PORTB, r18
        out PORTD, r18
        ldi r18, 0b00000000
        out PORTB, r18
        out PORTD, r18

        ret

send_ret:
        ldi r18,0b00000000
        out PORTB, r18
        out PORTD, r18
```

```
        ldi r19, 20
                loop1:
                        ldi r18, 10
                        loop2:
                                dec r18
                                brne loop2
                        dec r19
                        brne loop1

        ret
```

Board.c klassen

```
#include <stdbool.h>

#include <avr/io.h>

#include <util/delay.h>

// Sätter in ett drag i board-vektorn
extern int tur;
extern int board[9];
extern int keypad;
extern redbaby();
extern nollbaby();

bool checkMove(int drag);
int convertToCase(int n);
int convertFromCase(int n);
int checkWin(int board[9]);

// Metod sätter DDRen för en specifik pixel
void setDDR(int pixel) {

  switch (pixel) {

  case 4:

    DDRD = 0b00000100;
    DDRB = 0b00000000;

    break;

  case 6:

    DDRD = 0b00000010;
```

```
    DDRB = 0b00000000;

    break;

  case 7:

    DDRD = 0b00000001;
    DDRB = 0b00000000;

    break;

  case 8:

    DDRD = 0b00100000;
    DDRB = 0b00000000;

    break;

  case 11:

    DDRD = 0b00001000;
    DDRB = 0b00000000;

    break;

  case 10:

    DDRD = 0b00010000;
    DDRB = 0b00000000;

    break;

  case 15:

    DDRD = 0b01000000;
    DDRB = 0b00000000;

    break;

  case 14:

    DDRB = 0b00000001;
    DDRD = 0b00000000;

    break;

  case 12:
```

```
    DDRB = 0b00000010;
    DDRD = 0b00000000;

    break;
  }
}
// Metod konverterar vektor- till case-värde
int convertToCase(int n) {
  int newN;
  if (n == 0) {
    newN = 12;
  } else if (n == 1) {
    newN = 14;
  } else if (n == 2) {
    newN = 15;
  } else if (n == 3) {
    newN = 8;
  } else if (n == 4) {
    newN = 10;
  } else if (n == 5) {
    newN = 11;
  } else if (n == 6) {
    newN = 4;
  } else if (n == 7) {
    newN = 6;
  } else if (n == 8) {
    newN = 7;
  }
  return newN;
}
// metod konverterar case- till vektor-värde
int convertFromCase(int n) {
  int newN;
  if (n == 12) {
    newN = 0;
  } else if (n == 14) {
    newN = 1;
  } else if (n == 15) {
    newN = 2;
  } else if (n == 8) {
    newN = 3;
  } else if (n == 10) {
    newN = 4;
  } else if (n == 11) {
    newN = 5;
  } else if (n == 4) {
```

```
    newN = 6;
  } else if (n == 6) {
    newN = 7;
  } else if (n == 7) {
    newN = 8;
  }
  return newN;
}
// Denna metod tar alltid in case-värdet
void playMove(int plats) {
  // vektor-värdet av inputen placeras i vektorn
  board[convertFromCase(plats)] = tur;

  // metoden setDDR tar input som case-värde, och ingen konvertering behövs därför
  setDDR(plats);

  if (tur == 1) {
    bluebaby();
  } else if (tur == 2) {
    greenbaby();
  }

  int vinnare = checkWin(board);
  if (vinnare != 0) {
    winAnimation(vinnare);
  }
}

// Om någon har vunnit kommer brädet att resetas inför ett nytt spel och en spelanimation kommer
köras

// Denna metod kommer att ändra pixelfärgen på alla pixlar beroende på vem som van, och göra
någon form av animation
void winAnimation(int vinnare) {

  DDRB = 0b00000011;
  DDRD = 0b01111111;

  if (vinnare == 2) {
    while (PINA != 3) {
      greenbaby();
      _delay_ms(1000);
      nollbaby();
      _delay_ms(1000);
    }
  } else if (vinnare == 1) {
    while (PINA != 3) {
```

```
      bluebaby();
      _delay_ms(1000);
      nollbaby();
      _delay_ms(1000);
    }
  } else if (vinnare == -1) {
    while (PINA != 3) {
      redbaby();
      _delay_ms(1000);
      nollbaby();
      _delay_ms(1000);
    }
  }
  DDRB = 0b00000011;
  DDRD = 0b01111111;
  nollbaby();
  DDRD = 0b00000000;
  DDRB = 0b00000100;
  redbaby();
  // tur = 1;
  //_delay_ms(1000);
}

// Metod som kollar om någon har vunnit och returnerar isåfall vem. Har ingen vunnit returneras 0
int checkWin(int board[9]) {

  // Kollar den första rutan i varje x-axel, dvs rutor 0, 3 och 6, inte är tom och stämmer överens med
resten av raden
  // Om det är sant kommer värdet på den första rutan i raden att returneras
  for (int x = 0; x <= 2; x++) {
    if (board[3 * x] != 0 && board[3 * x] == board[3 * x + 1] && board[3 * x] == board[3 * x + 2]) {
      return board[3 * x];
    }
  }
  // Samma princip som innan fast för y-axeln. Lite annorlunda kod för att få dessa positioner
  for (int y = 0; y <= 2; y++) {
    if (board[y] != 0 && board[y] == board[y + 3] && board[y] == board[y + 6]) {
      return board[y];
    }
  }
  // Kod för att kolla diagonal 1 och diagonal 2
  if (board[0] != 0 && board[0] == board[4] && board[0] == board[8]) {
    return board[0];
  } else if (board[2] != 0 && board[2] == board[4] && board[2] == board[6]) {
    return board[2];
  }
```

```
  // Kollar oavgjort, returnerar -1 i så fall
  int cot = 0;
  for (int i = 0; i <= 8; i++) {
    if (board[i] != 0) {
      cot++;
    }
  }

  if (cot == 9) {
    return -1;
  }

  // Om inga linjer har samma värden kommer 0 att returneras
  return 0;
}

// Kollar om draget är lagligt, dvs om platsen är tillgänglig att lägga på
bool checkMove(int drag) {
  if (board[drag] == 0) {
    return true;
  }
  return false;
}

// Metod för att byta den publika variabeln 'tur', vilken bestämmer vems tur det är att spela (blå =
1/grön = 2)
void switchTur() {
  DDRD = 0b00000000;
  DDRB = 0b00000100;
  if (tur == 1) {
    tur = 2;
    greenbaby();
  } else if (tur == 2) {
    tur = 1;
    bluebaby();
  }
}

void resetBoard() {

  DDRB = 0b00000011;
  DDRD = 0b01111111;
  nollbaby();

  for (int i = 0; i <= 8; i++) {
    board[i] = 0;
  }
```

```
}

// Det ovanför här är main-metoden, och allt nedanför är funktioner

// Initierar pinsen, DDR är vilka som ska aktiveras, PORT är 0 för out, 1 för in
void init() {
  DDRA = 0b11110000;
  DDRB = 0b00000000;
  DDRC |= 0b00111100;
  DDRD = 0b00000000;
  PORTA = 0x00;
  PORTB = 0x00;
  PORTC = 0b00111100; // JTAG
  PORTD = 0b11111000;
}
```

CFile1.c klassen

```
extern void send_one();
extern void send_zero();
extern void send_ret();

void greenbaby() {
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
```

```
  send_zero();
  send_zero();
  send_zero();
  send_ret();
}

void bluebaby() {

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();

  send_ret();
}
void redbaby() {
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
```

```
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
}
void nollbaby() {
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_ret();
```

```
}

void ret() {

  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();

  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();

  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
}
```

Dator.c klassen

```
#include <util/delay.h>

#include <stdbool.h>
 // AI easy svargihet. Tanken är att den testar ett random drag, och så länge det inte är ett vinnande drag så placerar den det.
extern int board[];
extern int grad;
extern int tur;
extern int keypad;
extern int convertToCase(int n);
extern int checkWin(int board[9]);
extern int hittaTopDrag(int board[9]);
```

```
extern bool checkMove(int drag);
extern void switchTur();
extern void playMove();

int easy();
int normal();
void playAIMove();

int easy() {

  int drag;
  int board2[9]; // Nytt bräde var man sätter in och testar den nya positionen

  for (int i = 0; i <= 8; i++) {
    board2[i] = board[i];
  }

  int failSafe = 0; // En failsafe variabel som ser till att metoden gås igenom max 50 gånger

  while (1) {
    failSafe++;
    drag = rand() % 9;
    //       random();
    while (!checkMove(drag)) {
      drag = rand() % 9;
    }

    board2[drag] = tur; // Sätter in draget

    if (checkWin(board2) == 0) { // Testar om draget var vinnande
      board2[drag] == 0;
      return drag;
    }

    board2[drag] == 0; // återställer draget

    if (failSafe == 50) {
      return normal(); // Om failsafe aktiveras så läggs ett drag genom normal-boten
    }
  }
}

// AI normal svårighetsgrad, lägger ett slumpmässigt drag
int normal() {

  // Körs tills ett giltigt drag hittats
  while (1) {
```

```
    int drag = rand() % 9;
    if (checkMove(drag)) {
      return drag;
    }
  }
}

// Metod som spelar ett drag beroende på AIns svårighetsgrad
void playAIMove() {
  int drag;
  if (grad == 1) {
    drag = easy();
  } else if (grad == 2) {
    drag = normal();
  } else if (grad == 3) {
    drag = hittaTopDrag(board);
  }
  // playMove-metoden tar bara in case-värde, och en konvertering behövs därför.
  // !Notera att boten kommer skapa ett vektor-värde, medan en människa skapar ett case-värde
  playMove(convertToCase(drag));
  // Turen switchas återigen till människans tur
  switchTur();
}
```

Main.c klassen

```
// CPU klockad till 8MHz
#define F_CPU 8000000 UL

#include <util/delay.h>

#include <avr/io.h>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>


#include <stdbool.h>

extern void bluebaby();
extern void greenbaby();
extern void nollbaby();
```

```
extern void ret();

// Metoder från Board.c
extern void playMove(int plats);
extern void winAnimation(char vinnare);
extern void switchTur();
extern void resetBoard();
extern void init();
extern void playAIMove();
extern char checkWin(int board[9]);
extern int convertFromCase(int n);
extern bool checkMove(int drag);


// Anger vems tur det är. 1 = blå, 2 = grön
int tur = 1;
// Vektor som håller positioner i brädet. 0 = tom, 1 = blå, 2 = grön
int board[9] = {
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0
};

// 0 = human, 1 = easy bot, 2 = mellan bot, 3 = hard/miniMax
int grad = 0;
// keypad bestämmer vad som kommer göras i kommande switch-sats. Sätts till w/e, så länge det inte
är mellan 0 & 15
int keypad = 100;
// Bestämmer om boten ska göra ett drag eller om det är spelarens tur

//  knappsats ordning |  pixlar ordning
//  13 12 14 15    |  1  2  3
//  9  8 10  11      |  4  5  6    10
//  5  4  6  7       |  7  8  9
//  1  0  2  3       |
// P1: DDRD=0b01000000, P2: DDRD=0b00100000, P3: DDRD=0b00010000, P4:
DDRD=0b00001000, P5: DDRD=0b00000100,
// P6: DDRD=0b00000010,  P7: DDRD=0b00000001, P8: DDRB= 0b00000001, P9:
DDRB=0b00000010, P10: DRRB=0b00000100

// Main-metod följer
```

```
int main(void) {
  // Initierar pinsen
  init();

  DDRD = 0b00000000;
  DDRB = 0b00000100;
  bluebaby();

  // Spel körs medan strömmen är igång
  while (1) {

    // Om PINA har fått in en ny input som inte stämmer överens med gamla värdet så hanteras inputen
    if (PINA != keypad) {

      // keypad sätts till PIN A, så att if-satsen inte aktiveras förrän nytt värde på PIN A uppstår
      keypad = PINA;

      // Switch-sats som kommer att göra något beroende på vilken knapp som tryckts ner.
      switch (keypad) {

        // knappar 4, 6, 7, 8, 10, 11, 12, 14, 15 är olika dragen,
        // där värde 12 representerar knapp 0, 14 = 1, 15 = 2, 8 = 3, 10 = 4, 11 = 5, 4 = 6, 6 = 7 och 7 = 8
      case 4:
      case 6:
      case 7:
      case 8:
      case 10:
      case 11:
      case 12:
      case 14:
      case 15:
        // Jag kommer att referera till värden som "case-värden" & "vektor-värden" framöver, där
case-värdet är det talet som keypaden ger ifrån sig, och vektor-värdet är det som placeras i vektorn.
Pga att de är olika kommer konvertering mellan dessa att uppstå, och det är bra att ha i åtanke varför
detta görs.
        // Nedanför skickas case-värdet till playMove(), för att sedan byta tur.
        if (checkMove(convertFromCase(keypad))) {
          playMove(keypad);
          switchTur();
          // Om svårighetsgraden inte är pvp ska boten göra ett drag. Detta sker först med en delay.
          if (grad != 0) {
            _delay_ms(1000);
            playAIMove();
          }
        }

        break;
```

```
      case 3: // RESET
        resetBoard();
        DDRD = 0b00000000;
        DDRB = 0b00000100;
        nollbaby();
        bluebaby();
        tur = 1;
        grad = 0;
        break;

      case 13: // pvp
        grad = 0;
        break;

      case 9: // easy svarighet
        grad = 1;
        playAIMove();
        break;

      case 5: // normal svarighet
        grad = 2;
        playAIMove();
        break;

      case 1: // hard svarighet
        grad = 3;
        playAIMove();
        break;
      }
    }
  }
}
```

miniMax.c klassen

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

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <stdbool.h>
```

```
extern void switchTur();
extern int tur;
extern int checkWin(int board[9]);
extern int board[9];

int hittaTopDrag(int board[9]);
int max(int a, int b);
int min(int a, int b);
int miniMax(int board[9], int djup, bool isMax);

// metod för att hitta det största värdet av 2 tal - används för miniMax
int max(int a, int b) {
  if (a < b) {
    return b;
  } else if (a > b) {
    return a;
  }
// Om a = b
  return a;

}
// metod för att hitta det minsta värdet av 2 tal - används för miniMax
int min(int a, int b) {
  if (a < b) {
    return a;
  } else if (a > b) {
    return b;
  }

  return a;

}
// miniMax-algoritm - testar varje noden och returnerar p-värdet
int miniMax(int board[9], int djup, bool isMax) {
  int p = checkWin(board);
// Om en slutnod har nåtts
  if (p != 0) {
    return p;
  }
// Om det är maximizern
  if (isMax) {
    int top = -1000;

    for (int i = 0; i <= 8; i++) {
      if (board[i] == 0) {
        board[i] = 1;
        top = max(top, miniMax(board, djup + 1, !isMax));
```

```
        board[i] = 0;
      }
    }
    return top;
  }
// Om det är minimizern
else {
    int top = 1000;

    for (int i = 0; i <= 8; i++) {
      if (board[i] == 0) {
        board[i] = 2;
        top = min(top, miniMax(board, djup + 1, !isMax));
        board[i] = 0;
      }
    }
    return top;
  }

}

int hittaTopDrag(int board[9]) {

  int c = 0;
  for (int i = 0; i <= 8; i++) {
    if (board[i] != 0) {
      c++;
    }

  }
  if (c == 0) {
    while (true) {
      int drag = rand() % 9;

      if (drag == 0 || drag == 2 || drag == 6 || drag == 8) {
        return drag;
      }
    }

  }

  int board2[9]; // Nytt bräde var man sätter in och testar den nya positionen
//kopierar brädet till ett temporärt bräde
  for (int i = 0; i <= 8; i++) {
    board2[i] = board[i];
  }
```

```
//går igenom hela brädet och kollar vilka drag som finns kvar att göra, kollar därefter vilka av dragen som gör att AI:n vinner, om det fanns något sådant drag så lägger AI:n där
  for (int i = 0; i <= 8; i++) {
    if (checkMove(i)) {
      board2[i] = tur;
      if (checkWin(board2) == tur) {

        return i;
      }

      board2[i] = 0;

    }
  }
//kopierar brädet till ett temporärt bräde
  for (int i = 0; i <= 8; i++) {
    board2[i] = board[i];
  }
//går igenom hela brädet och kollar vilka drag som finns kvar att göra, kollar därefter vilka av dragen som gör att spelaren vinner, om det fanns något sådant drag så lägger AI:n där
  for (int i = 0; i <= 8; i++) {
    if (checkMove(i)) {
      board2[i] = 2;
      if (checkWin(board2) == 2) {

        return i;
      }

      board2[i] = 0;

    }
  }
 // Kollar varje nod i miniMax och väljer det bästa draget
  int topVal = -1000;
  int topDrag = -1;
  for (int i = 0; i <= 8; i++) {
    if (board[i] == 0) {
      board[i] = tur;
      int dragVal = miniMax(board, 0, false);
      board[i] = 0;

      if (dragVal > topVal) {
        topDrag = i;
        topVal = dragVal;
      }
    }
```

```
  }
  return topDrag;
}
```