C source

Battleships (main)

```
#include <avr/io.h>
#include <string.h>
#include "draw.h"
#include "displayCommands.h"
#include "keyboard.h"
#include "game.h"
#include "gameUtil.h"
#include "highscore.h"
#define F_CPU 8000000UL  // 8 MHz
#include <util/delay.h>
//#include "playSound.h"
void printLetters();
unsigned char choice;


int main(void)
{
    //set PORT A for out
    DDRD = 0xE0;
        DDRA = 0xFF;
        DDRB = 0x0F;
        startUp();
        setUpHighscore();
        while (1) {
                greetMe();
                choice = readChoice();
                if (choice == 1) {
                        setUpSingle();
                        while(playGame() == 1) {


                        }
                        clearScreen();
                        if (getWinner() == 0) {
                                write("you won", 3);
                                if (writeToHighscore(getNbrOfGuesses())) {
                                        clearScreen();
                                        write("new highscore", 3);
                                        _delay_ms(2000);
                                        clearScreen();
                                        write("enter initials", 2);
                                        addInitials(waitForInitials());
                                        displayHighscore();
                                        _delay_ms(3000);
                                }
                        } else {
                                write("you lost", 3);
                        }
                        _delay_ms(3000);
                } else if (choice == 2) {
                        setUpMulti();
                        while(playGame() == 1) {

                        }
                        clearScreen();
                        if (getWinner() == 0) {
                                write("you won", 3);
                        } else {
                                write("you lost", 3);
                        }
                        _delay_ms(3000);
                } else if (choice == 3) {
                        displayHighscore();
                        _delay_ms(3000);
                }
        }
        return 0;
}
```

Game

```
#define F_CPU 8000000UL  // 8 MHz
#include <util/delay.h>
#include "displayCommands.h"
#include "keyboard.h"
#include "highscore.h"
#include "draw.h"
#include "game.h"
#include "gameUtil.h"
#include <string.h>
#include "SPI.h"
```

```
unsigned char player[6][10];
unsigned char opponent[6][10];
unsigned char guessOpponent[6][10];
unsigned char smartGrid[6][10];
unsigned char totalNbrOfBoats = 4;
unsigned char total3Boats = 1;
unsigned char total2Boats = 3;
unsigned char totalBoatSize;
unsigned char oppBoatCoord[4][7];               //Change according to total nbr of boats.
unsigned char playerBoatCoord[4][7];    //Change according to total nbr of boats.
unsigned char turn;
unsigned char gameOver;
unsigned char hitCountPlayer;
unsigned char nbrOfGuesses;
unsigned char hitCountOpp;
unsigned char winner;
unsigned char difficulty = 0;
unsigned char multi = 0;


/* Sets up singleplayer game. Clears both player and opponent matrix.
Calls on prepareBattlefield() and randomizeOpponent() to set up matrisies.*/
void setUpSingle() {
        totalBoatSize = total2Boats*2 + total3Boats*3;
        resetAll();
        chooseDifficulty();
        clearScreen();
        write("place your ", 3);
        write("boats", 4);
        _delay_ms(2000);
        prepareBattlefield();
        _delay_ms(1500);
        randomizeOpponent();
        createSmartGrid();
        randomTurn();
}


void chooseDifficulty() {
        clearScreen();
        write("difficulty", 1);
        write("1. easy  ", 3);
        write("2. hard", 4);
        write("3. very hard  ", 5);
        difficulty = readChoice();
        while (difficulty < 1 || difficulty > 3) {
                difficulty = readChoice();
        }
}


/*Resets all variables of the game*/
void resetAll() {
        for (unsigned char i = 0; i < 6; i++) {
                for (unsigned char k = 0; k < 10; k++) {
                        opponent[i][k] = 0;
                        player[i][k]= 0;
                        guessOpponent[i][k] = 0;
                        smartGrid[i][k] = 0;
                }
        }
        for (unsigned char i = 0; i < totalNbrOfBoats; i++) {
                for(unsigned char k = 0; k < 7; k++) {
                        playerBoatCoord[i][k] = 0;
                        oppBoatCoord[i][k] = 0;
                }
        }
        nbrOfGuesses = 0;
        hitCountPlayer = 0;
        hitCountOpp = 0;
        gameOver = 0;
        multi = 0;
}


/* Randomizes turn in beginning of new game*/
void randomTurn() {
        turn = rnd(2);
}


void setUpMulti() {
        totalBoatSize = total2Boats*2 + total3Boats*3;
        resetAll();
        clearScreen();
        multi = 1;
        write("place your ", 3);
        write("boats", 4);
        _delay_ms(2000);
        prepareBattlefield();
```

```
        randomTurn();
        SPI_MasterInit();
        clearScreen();
        _delay_ms(2000);
        SPI_Send((turn<<3) | 0x01);             //Bit 3 signals turn and bit 0 is ready.
}


/* Greets player and prints choice menu*/
void greetMe() {
        clearScreen();
        write("greetings human", 1);
        write("1. singleplayer", 3);
        write("2. multiplayer ", 4);
        write("3. highscore   ", 5);
}


unsigned char getNbrOfGuesses() {
        return nbrOfGuesses;
}


/*Calls on ourTurn() or aiTurn() depending on variable turn.
Returns 1 if game is over, else returns 0*/
unsigned char playGame() {
        if(turn == 0) {
                //ourTurn();
                turn = 1;
        } else {
                if (multi == 1) {
                        opponentTurn();
                } else {
                        aiTurn();
                }
                turn = 0;
        }
        if (gameOver == 1) {
                return 0;
        } else {
                return 1;
        }
}


void opponentTurn() {
        clearScreen();
        write("their turn", 2);
        _delay_ms(2000);
        updateDisplay(player);
        unsigned char data = SPI_Send(0x00);
        unsigned char y = getOppY(data);
        unsigned char x = getOppX(data);
        unsigned char sunk = 0;
        unsigned char hit = 0;


        if(player[y][x] == 1) {
                player[y][x] = 3;
                hit = 1;
                sunk = placeHit(y, x, playerBoatCoord);
                hitCountOpp++;
        } else if (player[y][x] == 0) {
                player[y][x] = 2;
        }


        SPI_Send((sunk<<2) | (hit<<1) | 0x01);

        updateDisplay(player);
        _delay_ms(2000);
        if (hitCountOpp == totalBoatSize) {                     //If all boats are hit -> game over
                gameOver = 1;
                winner = 1;
        }
}


/* Reads input from keyboard and sets our guessing matrix to the correct value.*/
void ourTurn() {
        unsigned char boatSunk = 0;
        clearScreen();
        write("your turn", 2);
        _delay_ms(2000);
        updateDisplay(guessOpponent);
        unsigned char value = readKeyboard();
        unsigned char x = getX(value);
        unsigned char y = getY(value);
        if (multi == 0) {
/*----------------------------SINGLEPLAYER--------------------------------*/
                /*If boat and no prev. guess -> places boat*/
```

```
                if(opponent[y][x] == 1 && guessOpponent[y][x] == 0) {
                        guessOpponent[y][x] = 3;
                        hitCountPlayer++;
                        boatSunk = placeHit(y, x, oppBoatCoord);
                } else if (guessOpponent[y][x] == 0) {          // Places miss
                        guessOpponent[y][x] = 2;
                }
        } else {
/*-----------------------------MULTIPLAYER--------------------------------*/
                processCoord(y, x);
                unsigned char hit = isHit();
                boatSunk = sunk();
                if(hit == 1) {
                        if (guessOpponent[y][x] != 3) {
                                guessOpponent[y][x] = 3;
                                hitCountPlayer++;
                        }
                } else {
                        guessOpponent[y][x] = 2;
                }
        }
        updateDisplay(guessOpponent);
        if (boatSunk == 1) {
                write("boat sunk", 7);
        }
        _delay_ms(2000);
        if (hitCountPlayer == totalBoatSize) {                  //If all boats are hit -> game over
                gameOver = 1;
                winner = 0;
        }


        nbrOfGuesses++;
}


/* ============================================================================
                Game AI - Dolores         v. 1.0 - Random guesses all the time
                                          v. 1.1 - Able to guess around prev. hits
                                          v. 1.2 - No longer removes boats from screen
                                          v. 1.3 - Knows when a boat is already sunk
                                          v. 1.4 - Randomizes guessing direction around prev. hits
                                          v. 1.5 - Uses smart grid to never place two random guesses
next to each other.
                                          v. 1.6 - Able to recognize 3 boat and guess accordingly
============================================================================*/


/* Checks square to the right of previous hit.
Makes guess and returns 1 if guess was possible.*/
unsigned char checkRight(unsigned char y, unsigned char x) {
                if(x == 9) {                              //If prev. hit on right edge -> no square to the right ->
return 0.
                        return 0;
                } else if (player[y][x+1] == 0) {       //If square to the right no boat -> set guess = miss (2).
                        player[y][x+1] = 2;
                        return 1;
                        /*If square to the right boat -> set guess = hit (3). Add hit count.*/
                } else if (player[y][x+1] == 1)  {
                        hitCountOpp++;
                        player[y][x+1] = 3;
                        placeHit(y, x+1, playerBoatCoord);      //Places hit in boat coord matrix
                        return 1;
                } else {                                  //Square to the right already guessed -> no guess made.
                        return 0;
                }
}


/* Checks square to the left of previous hit.
Makes guess and returns 1 if guess was possible.*/
unsigned char checkLeft(unsigned char y, unsigned char x) {
                if(x == 0) {                              //If prev. hit on left edge -> no square to the left ->
return 0.
                        return 0;
                } else if (player[y][x-1] == 0) {       //If square to the left no boat -> set guess = miss (2).
                        player[y][x-1] = 2;
                        return 1;
                        /*If square to the left boat -> set guess = hit (3). Add hit count.*/
                } else if (player[y][x-1] == 1)  {
                        hitCountOpp++;
                        player[y][x-1] = 3;
                        placeHit(y, x-1, playerBoatCoord);      //Places hit in boat coord matrix
                        return 1;
                } else {                                  //Square to the left already guessed -> no guess made.
                        return 0;
                }

}
```

```
/* Checks square above of previous hit.
Makes guess and returns 1 if guess was possible.*/
unsigned char checkUp(unsigned char y, unsigned char x) {
		if(y == 0) {				//If prev. hit on top -> no square above -> return 0.
			return 0;
		} else if (player[y-1][x] == 0) {	//If square above no boat -> set guess = miss (2).
			player[y-1][x] = 2;
			return 1;
			/*If square above boat -> set guess = hit (3). Add hit count.*/
		} else if (player[y-1][x] == 1)  {
			hitCountOpp++;
			player[y-1][x] = 3;
			placeHit(y-1, x, playerBoatCoord);	//Places hit in boat coord matrix
			return 1;
		} else {				//Square above already guessed -> no guess made.
			return 0;
		}


}

/* Checks square below of previous hit.
Makes guess and returns 1 if guess was possible.*/
unsigned char checkDown(unsigned char y, unsigned char x) {
		if(y == 5) {				//If prev. hit on bottom -> no square below -> return 0.
			return 0;
		} else if (player[y+1][x] == 0) {	//If square below no boat -> set guess = miss (2).
			player[y+1][x] = 2;
			return 1;
			/*If square below boat -> set guess = hit (3). Add hit count.*/
		} else if (player[y+1][x] == 1)  {
			hitCountOpp++;
			player[y+1][x] = 3;
			placeHit(y+1, x, playerBoatCoord);	//Places hit in boat coord matrix
			return 1;
		} else {				//Square below already guessed -> no guess made.
			return 0;
		}
}

void aiTurn() {
	//clearScreen();
	//write("ai turn", 2);
	//_delay_ms(2000);
	updateDisplay(player);		//Updates display to show player matrix while AI guesses.
	_delay_ms(2000);
	unsigned char x = 0;
	unsigned char y = 0;
	unsigned char random = 0;
	unsigned char counter[4];
	unsigned char direction = 0;

	if (difficulty > 1) {
		 /*Scans player matrix for prev. hits*/
		for (unsigned char i = 0; i < 6; i++) {
			 for (unsigned char k = 0; k < 10; k++) {
				if (player[i][k] == 3) {		//If prev. hit found -> try to guess
around this hit.
					if (isSunk(i, k) != 1) {	//Check if boat on coord is not already
sunk

						for (int n = 0; n < 4; n++) {	//Clears guessing counter array
							counter[n] = 0;
						}
						direction = checkDirection(i,k);
						while(1) {
							if (direction == 0) {
								random = rnd(4);
							} else if (direction == 1) {
								random = rnd(2);
							} else {
								random = rnd(2)+2;
							}
							/*Randomizes direction to check around prev. hit based on
random variable*/
							switch (random) {
								case 0:
									if (checkRight(i, k)) {		//If guess
made to the right -> break with return
										return;
									}
									counter[0] = 1;			//Guess
not made -> set array[0] = 1 to show that guess was tried
									break;


								case 1:
									if (checkLeft(i, k)) {		//If guess
```

```
made to the left -> break with return
                                                                        return;
                                                                }
                                                                counter[1] = 1;         //Guess not made
-> set array[1] = 1 to show that guess was tried
                                                                break;


                                                        case 2:
                                                                if (checkUp(i, k)) {            //If guess
made above -> break with return
                                                                        return;
                                                                }
                                                                counter[2] = 1;         //Guess not made
-> set array[2] = 1 to show that guess was tried
                                                                break;


                                                        case 3:
                                                                if (checkDown(i, k)) {          //If guess
made below -> break with return
                                                                        return;
                                                                }
                                                                counter[3] = 1;         //Guess not made
-> set array[3] = 1 to show that guess was tried
                                                                break;
                                                }


                                                /*If all possible directions tried -> break loop and try
random guess.*/
                                                if ((counter[0] == 1) && (counter[1] == 1) && (counter[2]
== 1) && (counter[3] == 1)) {
                                                        break;
                                                } else if(direction == 1 && counter[0] == 1 && counter[1]
== 1) {
                                                        break;
                                                } else if(direction == 2 && counter[2] == 1 && counter[3]
== 1) {
                                                        break;
                                                }
                                        }
                                }
                        }
                }
        }
        /*No "smart guess" made -> place random guess.
        "Smart guess" = Guess near prev. hit.*/
        x = rnd(10);
        y = rnd(6);


        /*Random x and y until square in player[][] not prev. guess or hit.
        Uses smartGrid to place random guesses one square apart.*/
        while ((player[y][x] > 1) || (smartGrid[y][x] == 0)) {
                x = rnd(10);
                y = rnd(6);
        }
        if (player[y][x] == 1) {        //If guess = hit -> set guess = hit (3). Add hit count.
                player[y][x] = 3;
                hitCountOpp++;
                placeHit(y, x, playerBoatCoord);
        } else {                                        //If guess = miss -> set guess = miss (2). Add hit count.
                player[y][x] = 2;
        }


        updateDisplay(player);                  //Updates display to show AI guess.
        _delay_ms(2000);
        if (hitCountOpp == totalBoatSize) {     //If all boats are hit -> game over.
                gameOver = 1;
                winner = 1;
        }
}


unsigned char checkDirection(unsigned char y, unsigned char x) {
        unsigned char row = 0;
        for (int i = 0; i < totalNbrOfBoats; i++) {
                if (playerBoatCoord[i][5] != 10) {
                        row = i;
                }
        }
        for (int k = 1; k < 7; k=k+2) {
                if (playerBoatCoord[row][k] == y && playerBoatCoord[row][k+1] == x) {
                        if (playerBoatCoord[row][0] == 2) {
                                if (playerBoatCoord[row][1]==playerBoatCoord[row][3]) {
                                        return 1;
                                } else {
```

```
                                        return 2;
                                }
                        }
                }
        }
        return 0;
}

/* Generates AI matrix. */
void randomizeOpponent() {
        unsigned char nbrOfBoats = 0;
        unsigned char x[3];             //Prev. parts of boat placed.
        unsigned char y[3];
        while (nbrOfBoats < totalNbrOfBoats) {          //Generates boats until totalNbrOfBoats fulfilled.
                y[0] = rnd(6);          //Random coordinates for part 1 of boat.
                x[0] = rnd(10);
                unsigned char rd  = rnd(2);     //Place boat horisontal (0) or vertical (1)?
                if (rd == 0) {
                        if (nbrOfBoats < total2Boats) {         //Places boats of size 2.
                                if (x[0] == 9) {        //If part 1 on right edge:
                                        x[1] = x[0]-1;  //Place part 2 left of part 1.
                                } else {
                                        x[1] = x[0]+1;  //Else: place part 2 right of part 1.
                                }

                                /*If all squares are empty -> Place boat in AI matrix. Add number of boats.
                                Else no boat places -> redo loop = new start coordinates.*/
                                if ((opponent[y[0]][x[0]] == 0) && (opponent[y[0]][x[1]] == 0)) {
                                        opponent[y[0]][x[0]] = 1;
                                        opponent[y[0]][x[1]] = 1;
                                        oppBoatCoord[nbrOfBoats][1] = y[0];
                                        oppBoatCoord[nbrOfBoats][2] = x[0];
                                        oppBoatCoord[nbrOfBoats][3] = y[0];
                                        oppBoatCoord[nbrOfBoats][4] = x[1];
                                        oppBoatCoord[nbrOfBoats][5] = 10;               //Be able to check if boat
is 2 or 3
                                        nbrOfBoats++;
                                }
                        } else {                        //All boats of size 2 placed -> gen. boat of size 3.
                                if (x[0] >= 8) {        //If no room to the right of part 1:
                                        x[1] = x[0]-1;  //Place part 2 & 3 to the left of part 1.
                                        x[2] = x[0] - 2;
                                } else {                //Else: place part 2 & 3 to the right of part 1.
                                        x[1] = x[0]+1;
                                        x[2] = x[0] + 2;
                                }

                                /*If all squares are empty -> place boat in AI matrix. Add number of boats.
                                Else no boat places -> redo loop = new start coordinates.*/
                                if ((opponent[y[0]][x[0]] == 0) && (opponent[y[0]][x[1]] == 0) &&
                                (opponent[y[0]][x[2]] == 0)) {
                                        opponent[y[0]][x[0]] = 1;
                                        opponent[y[0]][x[1]] = 1;
                                        opponent[y[0]][x[2]] = 1;
                                        oppBoatCoord[nbrOfBoats][1] = y[0];
                                        oppBoatCoord[nbrOfBoats][2] = x[0];
                                        oppBoatCoord[nbrOfBoats][3] = y[0];
                                        oppBoatCoord[nbrOfBoats][4] = x[1];
                                        oppBoatCoord[nbrOfBoats][5] = y[0];
                                        oppBoatCoord[nbrOfBoats][6] = x[2];
                                        nbrOfBoats++;
                                }
                        }
                } else {                                //Place boats vertical.
                        if (nbrOfBoats < total2Boats) { //Places boats of size 2.
                                if (y[0] == 5) {        //If part 1 on bottom:
                                        y[1] = y[0]-1;  //Place part 2 above part 1.
                                } else {
                                        y[1] = y[0]+1;  //Else: place part 2 below part 1.
                                }

                                /*If all squares are empty -> Place boat in AI matrix. Add number of boats.
                                Else no boat places -> redo loop = new start coordinates.*/
                                if ((opponent[y[0]][x[0]] == 0) && (opponent[y[1]][x[0]] == 0)) {
                                        opponent[y[0]][x[0]] = 1;
                                        opponent[y[1]][x[0]] = 1;
                                        oppBoatCoord[nbrOfBoats][1] = y[0];
                                        oppBoatCoord[nbrOfBoats][2] = x[0];
                                        oppBoatCoord[nbrOfBoats][3] = y[1];
                                        oppBoatCoord[nbrOfBoats][4] = x[0];
                                        oppBoatCoord[nbrOfBoats][5] = 10;               //Be able to check if boat
is 2 or 3
                                        nbrOfBoats++;
                                }
                        } else {                        //All boats of size 2 placed -> gen. boat of size 3.
                                if (y[0] >= 4) {        //If no room below part 1:
```

```
                                        y[1] = y[0]-1;  //Place part 2 & 3 above part 1.
                                        y[2] = y[0] - 2;
                                } else {                //Else: place part 2 & 3 to the right of part 1.
                                        y[1] = y[0]+1;
                                        y[2] = y[0] + 2;
                                }

                                /*If all squares are empty -> place boat in AI matrix. Add number of boats.
                                Else no boat places -> redo loop = new start coordinates.*/
                                if ((opponent[y[0]][x[0]] == 0) && (opponent[y[1]][x[0]] == 0) &&
                                (opponent[y[2]][x[0]] == 0)) {
                                        opponent[y[0]][x[0]] = 1;
                                        opponent[y[1]][x[0]] = 1;
                                        opponent[y[2]][x[0]] = 1;
                                        oppBoatCoord[nbrOfBoats][1] = y[0];
                                        oppBoatCoord[nbrOfBoats][2] = x[0];
                                        oppBoatCoord[nbrOfBoats][3] = y[1];
                                        oppBoatCoord[nbrOfBoats][4] = x[0];
                                        oppBoatCoord[nbrOfBoats][5] = y[2];
                                        oppBoatCoord[nbrOfBoats][6] = x[0];
                                        nbrOfBoats++;
                                }
                        }
                }
        }
}


/*Generates smart grid = matrix of alternating 1 and 0 to be used by AI when randomly
guessing boats to never place two guesses next to each other.*/
void createSmartGrid() {
        if (difficulty == 3) {
                unsigned char random = rnd(2);
                for (unsigned char i = 0; i < 6; i++) {
                        for (unsigned char k = random; k < 10; k = k+2) {
                                smartGrid[i][k] = 1;
                                random++;
                                random = random % 2;            //Next row starts on other column than current.
                        }
                }
        } else {
                for (unsigned char i = 0; i < 6; i++) {
                        for (unsigned char k = 0; k < 10; k = k+2) {
                                smartGrid[i][k] = 1;
                        }
                }
        }
}


/*Finds boat corresponding to input coord. and checks if hit counter equals
size of boat -> return 1 (boat sunk)*/
unsigned char isSunk(unsigned char y, unsigned char x) {
        unsigned char sunk = 0;
        for (unsigned char i = 0; i < totalNbrOfBoats; i++) {
                for (unsigned char k = 1; k < 4; k++) {
                        if (playerBoatCoord[i][k*2-1] == y && playerBoatCoord[i][k*2] == x) {
                                if (playerBoatCoord[i][5] == 10) {      //2-boat, has no third coord
                                        if(playerBoatCoord[i][0] == 2) {
                                                sunk = 1;
                                        }
                                } else {                                //3-boat, third coord != 0
                                        if(playerBoatCoord[i][0] == 3) {
                                                sunk = 1;
                                        }
                                }
                        }
                }
        }
        return sunk;
}


/*================================================================================
End of AI
================================================================================*/


/*Returns winner of game. 0=player, 1=AI.*/
unsigned char getWinner() {
        return winner;
}


/*Finds boat corresponding to input coord. of either player or AI basedon input matrix
 and adds 1 to hit counter. If hit counter equals size of boat -> return 1 (boat sunk)*/
unsigned char placeHit(unsigned char y, unsigned char x, unsigned char matrix[4][7]) {
        unsigned char sunk = 0;
```

```
                for (unsigned char i = 0; i < totalNbrOfBoats; i++) {
                        for (unsigned char k = 1; k < 4; k++) {
                                if (matrix[i][k*2-1] == y && matrix[i][k*2] == x) {
                                        matrix[i][0] = matrix[i][0] + 1;
                                        if (matrix[i][5] == 10) {       //2-boat, third coord = 10
                                                if(matrix[i][0] == 2) {
                                                        sunk = 1;
                                                }
                                        } else {                                //3-boat, third coord != 10
                                                if(matrix[i][0] == 3) {
                                                        sunk = 1;
                                                }
                                        }
                                        return sunk;
                                }
                        }
                }


        return sunk;
}


/* Prepares player matrix according to number of boats of different size.
Calls on placeBoat(size) to wait for keyboard input and place boats*/
void prepareBattlefield() {
        updateDisplay(player);
        for (unsigned char i = 0; i < total2Boats; i++) {
                placeBoat(2);
        }
        for (unsigned char i = 0; i < total3Boats; i++) {
                placeBoat(3);
        }
}


/*Waits for keyboard input and places boats according to received size and input.*/
void placeBoat(unsigned char size) {
        unsigned char boatSize = 0;
        unsigned char x[size];
        unsigned char y[size];
        unsigned char x1;
        unsigned char y1;
        unsigned char counter ;
        unsigned char boatNumber = 0;
        for (unsigned char i = 0; i < totalNbrOfBoats; i++) {
                if(playerBoatCoord[i][1] == 0 && playerBoatCoord[i][2] == 0 &&
                playerBoatCoord[i][3] == 0 && playerBoatCoord[i][4] == 0) {
                        boatNumber = i;
                        break;
                }
        }
        unsigned char value = isVacant();       //Gets keyboard input from isVacant().
        x[0] = getX(value);
        y[0] = getY(value);
        placeDot(value);                        //Places first part of boat.
        boatSize++;
        while (boatSize < size) {
                value = isVacant();
                x1 = getX(value);
                y1 = getY(value);
                counter = 0;
                for (unsigned char i = 0; i < boatSize; i++) {                  //Checks new value to
                see if it is in line with prev.
                        if(x1 == x[i] && absolute(y[0], y1) <= boatSize) {
                                counter++;
                        } else if(y1 == y[i] && absolute(x[0], x1) <= boatSize) {
                                counter++;
                        }
                }
                if (counter == boatSize) {              //If value in line with prev. -> new boat part
                placed. Incr. boat size
                        placeDot(value);
                        x[boatSize] = x1;
                        y[boatSize] = y1;
                        boatSize++;
                }
        }
        playerBoatCoord[boatNumber][1] = y[0];
        playerBoatCoord[boatNumber][2] = x[0];
        playerBoatCoord[boatNumber][3] = y[1];
        playerBoatCoord[boatNumber][4] = x[1];
        playerBoatCoord[boatNumber][5] = 10;            //Be able to check if boat is 2 or 3
        if (boatSize > 2) {
                playerBoatCoord[boatNumber][5] = y[2];
                playerBoatCoord[boatNumber][6] = x[2];
        }
}
```

```
/*Polls keyboard input and returns value when input is not already occupied in player
matrix.*/
unsigned char isVacant() {
	unsigned char value = readKeyboard();
	while (1) {
		if (player[getY(value)][getX(value)] == 0) {
			return value;
		} else {
			value = readKeyboard();
		}
	}
}


/*Places part of boat (dot) on the received value in the player matrix.*/
void placeDot(unsigned char value) {
	player[getY(value)][getX(value)] = 1;
	updateDisplay(player);
}


void displayHighscore() {
	clearScreen();
	write("highscores", 0);
	chooseLeft();
	for(unsigned char i = 0; i < 5; i++) {
		setStartCoord((i+2), 3);
		drawNumbers(i+1);
		drawNumbers(10);
		for (unsigned char k = 0; k < 3; k++) {
			drawLetters(getInitials((4-i), k));
		}
	}


	chooseRight();
	for(unsigned char i = 0; i < 5; i++) {
		setStartCoord((i+2), 2);
		drawBigNumbers(getScoreValue(4-i));
	}
	turnOnDisplay();
}
```

GameUtil

```
#include "gameUtil.h"
#include <stdlib.h>
#include <avr/eeprom.h>
#define SEED_ADDR		0x02


uint8_t rndSeed;	//Seed stored in EEPROM used to generate random numbers.


/*Retrieves value of rndSeed from EEPROM, adds 1 to this value and writes it back
to the EEPROM for use in the next start up.*/
void startUp() {
	rndSeed = eeprom_read_byte((uint8_t*)SEED_ADDR);
	rndSeed += 1;
	eeprom_write_byte((uint8_t*)SEED_ADDR, rndSeed);
}


/*Uses rndSeed to generate a random number. Returns random number between 0 and max*/
unsigned char rnd(unsigned char max) {
	unsigned char rnd = rand() / (max*rndSeed);
	return rnd % max;
}


/*Returns the absolute value of the difference between v1 and v2.*/
unsigned char absolute(unsigned char v1, unsigned char v2) {
	if (v1>v2) {
		return v1-v2;
	} else {
		return v2-v1;
	}
}
```

Keyboard

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


/*Returns Y value from keyboard input (the 4 most significant bits)*/
int getY(unsigned char value) {
	return (((value & 0xF0)>>4)-10);
}
```

```
/*Returns X value from keyboard input (the 4 lest significant bits)*/
int getX(unsigned char value) {
	return ((value & 0x0F));
}


/*Waits for 1 key to be pressed on the keyboard and returns this value.*/
unsigned char readChoice() {

	unsigned char x = 0;
	unsigned char value = 0;

	/*State machine that waits for a button to be pressed and released.*/
	while(x != 2) {
		switch (x) {
			case 0:
				if ((PIND & 0x10) == 0x10) {		//Checks dataAvailable (PD4)
					value = ((PIND & 0x0F));	//Gets data from keyboard (PD0-PD3)
					x = 1;
				}
				break;
			case 1:
				if ((PIND & 0x10) == 0x00) {		//Checks if button released
(dataAvailable=0)
					x = 2;
				}
				break;
		}
	}
	return value;


}


/*Waits for 2 buttons to be pressed and returns these two values as 4 most and least
 significant bits of an 8 bit char.*/
unsigned char readKeyboard() {

	unsigned char x = 0;
	unsigned char value = 0;
	unsigned char temp = 0;

	/*State machine that waits for two buttons to be pressed and released.*/
	while(x != 4) {
		switch (x) {
			case 0:
				if ((PIND & 0x10) == 0x10) {		//Checks dataAvailable (PD4)
					value = ((PIND & 0x0F));	//Gets data from keyboard (PD0-PD3)
					if (value >= 0x0A) {		//Checks if data is a letter (>0x0A)
						value = value << 4;	//Shifts data of first button 4 bits.
						x = 1;
					}
				}
				break;
			case 1:
				if ((PIND & 0x10) == 0x00) {		//Checks if button released
(dataAvailable=0)
					x = 2;
				}
				break;
			case 2:
				if ((PIND & 0x10) == 0x10) {		//Checks dataAvailable (PD4)
					temp = PIND & 0x0F;		//Gets data from keyboard (PD0-PD3)
					if (temp < 0x0A) {		//Checks if data is a number (<0x0A)
						value = value | temp;	//Adds this 4 bit data to end of 8 bit
value.
						x = 3;
					}
				}
				break;
			case 3:
				if ((PIND & 0x10) == 0x00) {		//Checks if button released
(dataAvailable=0)
					x = 4;
				}
				break;
		}
	}
	return value;


}


unsigned long waitForInitials() {
	unsigned long name = 0;
	unsigned long value = 0;
	unsigned char x = 0;
```

```
        while(x != 6) {
         switch (x) {
                     case 0:
                          if ((PIND & 0x10) == 0x10) {            //Checks dataAvailable (PD4)
                          value = ((PIND & 0x0F));        //Gets data from keyboard (PD0-PD3)
                               if (value >= 0x0A) {
                                     name |= ((value-10) << 24);
                                           x = 1;
                                }
                          }
                          break;


                          case 1:
                         if ((PIND & 0x10) == 0x00) {            //Checks if button released (dataAvailable=0)
                              x = 2;
                          }
                          break;


                  case 2:
                          if ((PIND & 0x10) == 0x10) {            //Checks dataAvailable (PD4)
                              value = ((PIND & 0x0F));        //Gets data from keyboard (PD0-PD3)
                               if (value >= 0x0A) {
                                     name |= ((value-10) << 16);
                                     x = 3;
                                }
                           }
                           break;


                          case 3:
                          if ((PIND & 0x10) == 0x00) {            //Checks if button released (dataAvailable=0)
                               x = 4;
                          }
                          break;


                  case 4:
                          if ((PIND & 0x10) == 0x10) {            //Checks dataAvailable (PD4)
                              value = ((PIND & 0x0F));        //Gets data from keyboard (PD0-PD3)
                              if (value >= 0x0A) {
                                           name |= ((value-10) << 8);
                                    x = 5;
                              }
                           }
                          break;

                          case 5:
                                   if ((PIND & 0x10) == 0x00) {            //Checks if button released
(dataAvailable=0)
                                   x = 6;
                                    }
                        break;
         }
        }
        return name;
}
```

Highscore

```
#include "highscore.h"
#include <avr/eeprom.h>


uint32_t highscore[5];
unsigned char rowOfCurrent;


unsigned char writeToHighscore(unsigned char value) {
        getHighscore();
        unsigned char temp = 0;
        unsigned long initials = 0;
        if (isHighscore(value)) {
                for (unsigned char i = 0; i < 5; i++) {
                        if (getScoreValue(i) > value) {
                                temp = getScoreValue(i);
                                initials = showInitials(i);
                                highscore[i] = (uint32_t)value;
                                if(i > 0) {
                                        highscore[i-1] = (uint32_t)(temp + initials);
                                }
                                rowOfCurrent = i;
                        }
                }
                return 1;
        }
        return 0;
```

```
}

unsigned char isHighscore(unsigned char value) {
        if (getScoreValue(0) > value) {
                return 1;
        }
        return 0;
}


unsigned int getScoreValue(unsigned char i) {
        return (unsigned int)(highscore[i] & 0x000000FF);
}


unsigned long showInitials(unsigned char i) {
        return highscore[i] & 0xFFFFFF00;
}
unsigned char getInitials(unsigned char i, unsigned char initial) {
        return (unsigned char)(highscore[i] >> ((3-initial)*8));
}


void writeArray() {
        for (unsigned char i = 4; i < 9; i++) {
                eeprom_write_dword((uint32_t*)(i*4), highscore[i-4]);
        }
}


void addInitials(unsigned long initials) {
        unsigned long name = initials | (unsigned long)getScoreValue(rowOfCurrent);
        highscore[rowOfCurrent] = (uint32_t)name;
        writeArray();
}


void getHighscore() {
        for (unsigned char i = 4; i < 9; i++) {
                highscore[i-4] = eeprom_read_dword((uint32_t*)(i*4));
        }
}


void setUpHighscore() {
        getHighscore();
}
```

DisplayCommands

```
#include <avr/io.h>
#include "displayCommands.h"
#include "draw.h"


void setDisplayCode(unsigned char rs, unsigned char dispCode) {
        while(displayReady() == 0) {    //Waits for display to be ready
        }
        PORTB |= rs<<3;                                 // R/S
        PORTA = dispCode;                               //CB0-DB7
        PORTB |= 1<<2;                                  //Enable high
        PORTB ^= 1<<2;                                  //Enable low => writes signal
}


unsigned int displayReady() {
        unsigned char dataRead = 0;
        DDRA = 0x4F;                                    //Set PA0-3 and PA6 as output
        PORTB &= 0xF7;                                  //R/S low
        PORTD |= 1<<6;                                  //R/W high
        PORTB |= 1<<2;                                  //Enable high
        PORTB ^= 1<<2;                                  //Enable low
        dataRead = PINA & 0x80;                 //Busy
        if(dataRead == 0x80) {
                return 0;                                               //Check if busy
        }
        PORTD ^= 1<<6;                                  //R/W low
        DDRA = 0xFF;                                    //PORTA output
        return 1;
}


void turnOnDisplay() {
        PORTB |= 1<<0;                          //Choose right display
        PORTB &= 0xFD;
        setDisplayCode(0, 0x3F);        //Turn on right display
        setDisplayCode(0, 0xC0);
        PORTB |= 1<<1;                          //Choose left display
```

```
        PORTB &= 0xFE;
        setDisplayCode(0, 0x3F);        //Turn on left display
        setDisplayCode(0, 0xC0);
}


void chooseLeft() {
                PORTB &= 0xFE;          //Chooses left side of screen
                PORTB |= 1<<1;
}


void chooseRight() {
                PORTB |= 1<<0;          //Chooses right side of screen
                PORTB &= 0xFD;
}


//Sets the start coordinates for writing on the screen
void setStartCoord(unsigned char y, unsigned char x) {
        setDisplayCode(0, (184 + y));
        setDisplayCode(0, (64 + x*8));
}


//Updates the display on both sides with a matrix as input telling where to draw what
void updateDisplay(unsigned char matrix[6][10]) {
        chooseRight();  //Chose the right side of the screen
        drawBlank();    //Make that side of the screen draw blank


        //Draws up the right side of the playing field
        for (unsigned char i = 0; i < 6; i++) {
                setStartCoord(i+1, 0);  //Starts with 1 padding on x and 0 on y then
                increments x for every line
                for (unsigned char k = 5; k < 10; k++) {
                        if (matrix[i][k] == 1) {
                                drawCircle();   //Draws a boat
                        } else if (matrix[i][k] == 2) {
                                drawX();        //Draws a miss
                        } else if (matrix[i][k] == 3) {
                                drawHitBoat();  //Draws a hit
                        } else {
                                drawSquare();   //Draws an empty square
                        }
                }
        }
        //Draws the numbers marking the different columns on the bottom of the screen
        setStartCoord(0, 0);
        for (unsigned char i = 0; i < 5; i++) {
                drawNumbers(i+5);
        }
        chooseLeft();   //Chose the left side of the screen
        drawBlank();    //Make that side of the screen blank


        //Draws up the left side of the playing field
        for (unsigned char i = 0; i < 6; i++) {
                setStartCoord(i+1, 2);  //Starts with 1 padding on x and 2 on y then
                increments x for every line
                drawLetters(i);         //Draws A-F
                for (unsigned char k = 0; k < 5; k++) {
                        if (matrix[i][k] == 1) {
                                drawCircle(); //Draws boat
                        } else if (matrix[i][k] == 2) {
                                drawX();        //Draws miss
                        } else if (matrix[i][k] == 3) {
                                drawHitBoat();  //Draws hit
                        } else {
                                drawSquare();   //Draws an empty square
                        }
                }
        }
        //Draws the numbers marking the different columns on the bottom of the screen
        setStartCoord(0, 3);
        for (unsigned char i = 0; i < 5; i++) {
                drawNumbers(i);
        }
        turnOnDisplay();
}
```

Draw

```
#include "displayCommands.h"
#include "draw.h"
#include <string.h>
#define F_CPU 8000000UL  // 8 MHz
#include <util/delay.h>
```

```
//Draws a circle on an 8x8 pixel square
void drawCircle() {
        setDisplayCode(1,0xFF);
        setDisplayCode(1, 0x99);
        setDisplayCode(1, 0xA5);
        setDisplayCode(1, 0xC3);
        setDisplayCode(1, 0xC3);
        setDisplayCode(1, 0xA5);
        setDisplayCode(1, 0x99);
        setDisplayCode(1, 0xFF);
}


//Draws a square on an 8x8 pixel square
void drawSquare() {
        setDisplayCode(1,0xFF);
        for (unsigned char i=0; i<6; i++) {
                setDisplayCode(1, 0x81);
        }
        setDisplayCode(1, 0xFF);
}


//Draws a X on an 8x8 pixel square
void drawX() {
        setDisplayCode(1, 0xFF);
        setDisplayCode(1, 0xC3);
        setDisplayCode(1, 0xA5);
        setDisplayCode(1, 0x99);
        setDisplayCode(1, 0x99);
        setDisplayCode(1, 0xA5);
        setDisplayCode(1, 0xC3);
        setDisplayCode(1, 0xFF);
}


//Draws a circle with a X in it on an 8x8 pixel square
void drawHitBoat() {
        setDisplayCode(1, 0xFF);
        setDisplayCode(1, 0xDB);
        setDisplayCode(1, 0xA6);
        setDisplayCode(1, 0xDB);
        setDisplayCode(1, 0xDB);
        setDisplayCode(1, 0xA6);
        setDisplayCode(1, 0xDB);
        setDisplayCode(1, 0xFF);
}


//Clears active screen
void drawBlank() {
        for (unsigned char i = 0; i < 8; i++) {
                setDisplayCode(0, 0xC0);
                setDisplayCode(0, 184 + i);
                setDisplayCode(0, 0x40);
                for (unsigned char k=0; k < 64; k++) {
                        setDisplayCode(1, 0x00);
                }
        }
}


//Makes the screen blank
void clearScreen() {
        chooseLeft();
        drawBlank();
        chooseRight();
        drawBlank();
}


//Function of epicness
void drawLetters(unsigned char nbr) {
        if (nbr == 0) {                         // A
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0xFE);
                setDisplayCode(1, 0x12);
                setDisplayCode(1, 0x12);
                setDisplayCode(1, 0xFE);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
        } else if (nbr == 1) {          // B
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0xFE);
                setDisplayCode(1, 0x92);
                setDisplayCode(1, 0x92);
                setDisplayCode(1, 0xEE);
                setDisplayCode(1, 0x00);
```

```
        setDisplayCode(1, 0x00);
} else if (nbr == 2) {          // C
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 3) {          // D
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x7C);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 4) {          // E
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 5) {          // F
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x12);
        setDisplayCode(1, 0x12);
        setDisplayCode(1, 0x02);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 6) {          // G
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFC);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x74);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 7) {          // H
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x10);
        setDisplayCode(1, 0x10);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 8) {          // I
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 9) {          // J
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x42);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x7E);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 10) {         // K
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x10);
        setDisplayCode(1, 0x28);
        setDisplayCode(1, 0x44);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x00);
} else if (nbr == 11) {         // L
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x80);
        setDisplayCode(1, 0x80);
        setDisplayCode(1, 0x80);
```

```
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 12) {         // M
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x04);
        setDisplayCode(1, 0x08);
        setDisplayCode(1, 0x10);
        setDisplayCode(1, 0x10);
        setDisplayCode(1, 0x08);
        setDisplayCode(1, 0x04);
        setDisplayCode(1, 0xFE);
} else if (nbr == 13) {         // N
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x04);
        setDisplayCode(1, 0x18);
        setDisplayCode(1, 0x30);
        setDisplayCode(1, 0x40);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x00);
} else if (nbr == 14) {         // O
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x7C);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0x7C);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 15) {         // P
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x12);
        setDisplayCode(1, 0x12);
        setDisplayCode(1, 0x0C);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 16) {         // Q
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x7C);
        setDisplayCode(1, 0x82);
        setDisplayCode(1, 0xC2);
        setDisplayCode(1, 0x7C);
        setDisplayCode(1, 0x80);
        setDisplayCode(1, 0x00);
} else if (nbr == 17) {         // R
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x12);
        setDisplayCode(1, 0x32);
        setDisplayCode(1, 0x4C);
        setDisplayCode(1, 0x80);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 18) {         // S
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x8C);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x66);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 19) {         // T
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x02);
        setDisplayCode(1, 0x02);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x02);
        setDisplayCode(1, 0x02);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 20) {         // U
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x7E);
        setDisplayCode(1, 0x80);
        setDisplayCode(1, 0x80);
        setDisplayCode(1, 0x7E);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 21) {         // V
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x3E);
        setDisplayCode(1, 0x40);
        setDisplayCode(1, 0x80);
        setDisplayCode(1, 0x40);
```

```
                setDisplayCode(1, 0x3E);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
        } else if (nbr == 22) {         // W
                setDisplayCode(1, 0x0E);
                setDisplayCode(1, 0x38);
                setDisplayCode(1, 0xE0);
                setDisplayCode(1, 0x10);
                setDisplayCode(1, 0x10);
                setDisplayCode(1, 0xE0);
                setDisplayCode(1, 0x38);
                setDisplayCode(1, 0x0E);
        } else if (nbr == 23) {         // X
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x84);
                setDisplayCode(1, 0x48);
                setDisplayCode(1, 0x30);
                setDisplayCode(1, 0x30);
                setDisplayCode(1, 0x48);
                setDisplayCode(1, 0x84);
                setDisplayCode(1, 0x00);
        } else if (nbr == 24) {         // Y
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x02);
                setDisplayCode(1, 0x04);
                setDisplayCode(1, 0xF8);
                setDisplayCode(1, 0x04);
                setDisplayCode(1, 0x02);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
        } else  if (nbr == 25) {        // Z
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0xC2);
                setDisplayCode(1, 0xA2);
                setDisplayCode(1, 0x92);
                setDisplayCode(1, 0x8A);
                setDisplayCode(1, 0x86);
                setDisplayCode(1, 0x82);
                setDisplayCode(1, 0x00);
        } else {                                        // " "
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
        }
}

//Draws numbers with more than one digit.
void drawBigNumbers(unsigned int nbr) {
        unsigned int numbers[3];
        for (unsigned char i = 0; i < 3; i++) {
                numbers[i] = 0;
        }
        unsigned char flag = 0;
        unsigned char i = 0;


        while (nbr != 0) {
                numbers[i] = nbr%10;
                nbr = nbr/10;
                i++;
        }


        for (unsigned char i = 0; i < 3; i++) {
                if (numbers[2-i] != 0) {
                        flag = 1;
                }
                if (flag == 1) {
                        drawNumbers(numbers[2-i]);
                }
        }
}


//Function of not equal epicness but some
void drawNumbers(unsigned char nbr) {
        if (nbr == 0) { //Draws 0
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0xFE);
                setDisplayCode(1, 0x82);
                setDisplayCode(1, 0x82);
                setDisplayCode(1, 0xFE);
                setDisplayCode(1, 0x00);
```

```
        setDisplayCode(1, 0x00);
} else if (nbr == 1) {  //Draws 1
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 2) {  //Draws 2
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0xF2);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x9E);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 3) {  //Draws 3
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 4) {  //Draws 4
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x1E);
        setDisplayCode(1, 0x10);
        setDisplayCode(1, 0x10);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 5) {  //Draws 5
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x9E);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x72);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 6) {  //Draws 6
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x7C);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x62);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 7) {  //Draws 7
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x02);
        setDisplayCode(1, 0x02);
        setDisplayCode(1, 0x02);
        setDisplayCode(1, 0xFE);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 8) {  //Draws 8
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x6C);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x6C);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else if (nbr == 9) {  //Draws 9
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x0C);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x92);
        setDisplayCode(1, 0x6C);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
} else {        //Draws dot
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x80);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
        setDisplayCode(1, 0x00);
```

```
                setDisplayCode(1, 0x00);
                setDisplayCode(1, 0x00);
        }
}


//Function of most epicness that writes text on the screen, with the maximum of 16
chars, on row row
void write(char string[], unsigned char row) {
        unsigned char coord = 0;
        unsigned char flag = 0;
        if (strlen(string) <= 16) {     //Check if the string is to big to fit
                chooseLeft();
                coord = 8 - (strlen(string)/2); //Calculates where to put the text
                setStartCoord(row,coord);               //Centers the text on the screen
                char alphabet[27] = "abcdefghijklmnopqrstuvwxyz ";
                char numbers[11] = "0123456789.";
                //Draws the text on the screen
                for (unsigned char i = 0; i < strlen(string); i++) {
                        flag = 0;
                        //Checks if we are going to write on the right side of the screen, if not
                        then we write on the left side
                        if (i == 8-coord) {
                                chooseRight();
                                setStartCoord(row, 0);
                        }
                        //Check if the char is a letter then draws that letter on the screen
                        for (unsigned char n = 0; n < strlen(alphabet); n++) {
                                if (string[i] == alphabet[n]) {
                                        drawLetters(n);
                                        flag = 1;
                                }
                        }
                        //If it wasnt a letter we check if it's a number and draws that number on
                        the screen
                        if (flag == 0) {
                                for (unsigned char k = 0; k < strlen(numbers); k++) {
                                        if (string[i] == numbers[k]) {
                                                drawNumbers(k);
                                        }
                                }
                        }


                }
                turnOnDisplay(); //Update the display with the text
        }
}
```

SPI

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


unsigned char info = 0;


/* Concatenates boat coordinates and adds a 1 to the most significant bit. MSB is used
as a flag, all data sent has this bit set to 1. Calls on SPI_Send to send the data. */
void processCoord(unsigned char y, unsigned char x) {
        char data = (y << 4) + x;
        data |= 0x80;                   //Sets MSB = 1 to signal data.
        info = SPI_Send(data);
}


unsigned char isHit() {
        return info & 0x02 >> 2;
}


unsigned char sunk() {
        return (info & 0x04) >> 3;
}


unsigned char getOppX(unsigned char data) {
        return (data & 0x0F);
}


unsigned char getOppY(unsigned char data) {
        return (data & 0x70) >> 4;
}


void SPI_MasterInit() {
        /*Set SS, MOSI and SCK output, all other input */
```

```
        DDRB |= (1<<PB4)|(1<<PB5)|(1<<PB7);

        /* Enable SPI, Master, set clock rate fck/16 */
        SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);

}

unsigned char SPI_MasterTransmit(char data) {
        /* Start transmission */
        SPDR = data;

        /* Wait for transmission complete */

        while(!(SPSR & (1<<SPIF))) {
        }

        return SPDR;

}

unsigned char SPI_Send(char data) {
        unsigned char input = SPI_MasterTransmit(data);
        while(input == 0x00) {
                input = SPI_MasterTransmit(data);
        }
        return input;
}
```

Headerfiles

Game

```
void prepareBattlefield();
void placeBoat(unsigned char size);
void placeDot(unsigned char value);
unsigned char isVacant();
void randomizeOpponent();
void createSmartGrid();
void setUpSingle();
void setUpMulti();
void resetAll();
unsigned char playGame();
void greetMe();
void randomTurn();
void ourTurn();
void opponentTurn();
unsigned char getNbrOfGuesses();
unsigned char checkDirection(unsigned char y, unsigned char x);
unsigned char checkRight(unsigned char y, unsigned char x);
unsigned char checkLeft(unsigned char y, unsigned char x);
unsigned char checkUp(unsigned char y, unsigned char x);
unsigned char checkDown(unsigned char y, unsigned char x);
void aiTurn();
unsigned char getWinner();
unsigned char placeHit(unsigned char y, unsigned char x, unsigned char input[4][7]);
unsigned char isSunk(unsigned char y, unsigned char x);
void displayHighscore();
void chooseDifficulty();
```

GameUtil

```
void startUp();
unsigned char rnd(unsigned char max);
unsigned char absolute(unsigned char v1, unsigned char v2);
```

Keyboard

```
unsigned char readKeyboard();
unsigned char readChoice();
int getX(unsigned char value);
int getY(unsigned char value);
unsigned long waitForInitials();
```

Highscore

```
unsigned char writeToHighscore(unsigned char value);
unsigned char isHighscore(unsigned char value);
unsigned int getScoreValue(unsigned char i);
void writeArray();
void addInitials(unsigned long initials);
unsigned char getInitials(unsigned char i, unsigned char initial);
void getHighscore();
void setUpHighscore();
unsigned long showInitials(unsigned char i);
```

DisplayCommands

```
unsigned int displayReady();
void setDisplayCode(unsigned char, unsigned char);
void turnOnDisplay();
void setStartCoord(unsigned char x, unsigned char y);
void chooseLeft();
void chooseRight();
//void resetCoord();
void updateDisplay(unsigned char matrix[6][10]);
```

Draw

```
void drawNumbers(unsigned char nbr);
void drawBigNumbers(unsigned int nbr);
void drawLetters(unsigned char nbr);
void drawCircle();
void drawSquare();
void drawX();
void drawHitBoat();
void clearScreen();
void drawBlank();
void write(char string[], unsigned char row);
```

SPI

```
void processCoord(unsigned char y, unsigned char x);
unsigned char isHit();
unsigned char sunk();
void SPI_MasterInit();
unsigned char getOppX(unsigned char data);
unsigned char getOppY(unsigned char data);
unsigned char SPI_MasterTransmit(char data);
unsigned char SPI_Send(char data);
void signalReady();
```