

# ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΠΕΞΕΡΓΑΣΤΩΝ ΗΡΥ 411

**LAB41140566**

**Milestone 2: Othello Game Project**

Φλέγγας Γεώργιος 2014030161  
Χατζηπέτρος Αλέξανδρος 2013030151

**09/05/2019**



HMMY

## Εισαγωγή:

Στόχος του δεύτερου milestone αποτέλεσε, βασισμένοι πάνω στην διεπαφή που δημιουργήσαμε για το milestone 1, να υλοποιήσουμε την πλήρη λειτουργία του παιχνιδιού othello. Το avr θα λειτουργεί ως ένας «χαζός» παίκτης, που βρίσκει σε κάθε περίπτωση τις «νόμιμες» κινήσεις και παίζει μία από αυτές, εντοπίζει εάν ο αντίπαλος έπαιξε παράνομη κίνηση και το υποδεικνύει, βρίσκει εάν ο αντίπαλος ξεπέρασε τον επιτρεπτό χρόνο, κλπ. Ο παίχτης δεν θα ακολουθεί κάποια συγκεκριμένη στρατηγική.

## Επιλογές και σχεδίαση:

Η υλοποίηση της πλήρης λειτουργίας του παιχνιδιού βασίστηκε πάνω σε 2 κυρίως διαφορετικά κομμάτια : α) Διεπαφή επικοινωνίας χρήστη-AVR, β) Κεντρικό do-while loop το οποίο φροντίζει την ομαλή εκτέλεση των κινήσεων των παιχτών.

Με το α) ασχοληθήκαμε κατά την διεκπεραίωση του Milestone 1 και προσθέσαμε τα απαραίτητα κομμάτια κώδικα, ώστε να βεβαιώνουμε την ορθή επικοινωνία των 2 παιχτών. Αυτό επιτεύχθηκε κυρίως με την χρήση while-loops, τα οποία “κολλάνε” τον κώδικα μέχρι η αντίστοιχη wait μεταβλητή πάρει την τιμή 0, αφού το AVR λάβει το κατάλληλο μήνυμα. Ιδιαίτερο ενδιαφέρον έχει το πώς θα αντιμετωπίσει το avr, την κίνηση του αντίπαλου παίχτη όταν λάβει το μήνυμα MV<SP>{[A-H][1,8]}<CR>, κάτι στο οποίο θα αναφερθούμε παρακάτω.

Το β) αποτελεί τον πυρήνα του παιχνιδιού και σε μορφή ψευδοκώδικα η κύρια ιδέα είναι η εξής:

```
board_init;
Get Players Color
do:
    if (white player):
        if (enemy's turn):
            if (valid_moves('W')):
                Passes = 0;
                PrintBoard(moves)
                read player's moves
            else:
                passes++
                if (passes<2):
                    Ask player to pass
                else:
                    Neither Player got a move, Game over
        if (avr's turn):
            if (valid_moves('W')):
                Passes = 0;
                avr_move('w')
                Moves_Done++
            else:
                passes++
                if (passes<2):
                    Avr passes
                else:
                    Neither Player got a move, Game over
        black player next round
    if (black player):
        (. . . )
while ((Moves_Done<64) && (Passes<2) && (End_Game!=1) && (New_Game!=1))
calculate_score()
announce winner
```

Κάθε φορά θα παίζει πρώτος ο άσπρος παίχτης. Αν αυτός είναι ο αντίπαλος, τότε με την προϋπόθεση ότι έχει διαθέσιμες κινήσεις, οι οποίες ανιχνεύονται μέσω της συνάρτησης `int valid_moves(char turn)` και αποθηκεύονται στον πίνακα `moves`, το `avr` περιμένει από τον παίχτη να του στείλει την κίνηση του, ενώ σε περίπτωση που δεν έχει κίνηση, θα περιμένει PASS. Αν και οι 2 παίχτες κάνουν PASS ο ένας μετά τον άλλον σημαίνει ότι δεν υπάρχουν άλλες διαθέσιμες κινήσεις και το παιχνίδι τερματίζει. Όταν το `avr` ανιχνεύσει την κίνηση θα την επεξεργαστεί μέσω του ακόλουθου κώδικα:

```
Y=(Data[reader+3]- '0')-17;
X=(Data[reader+4]- '0')-1;
if(timeout==0){
    if (moves[X][Y]==86) {
        make_move(X,Y,OtherPlayer);
        PrintBoard(board);
        Moves_Done++;
    }else
    {
        //invalid move
        USART_Transmit_Str("IL\r");
        LastTrasmit[0]=73;
        LastTrasmit[1]=76;
        LastTrasmit[2]=13;
        badmove=1;
        while (badmove!=0) {}
    }
}else{
    //time is up. my move
    USART_Transmit_Str("IT\r");
    LastTrasmit[0]=73;
    LastTrasmit[1]=84;
    LastTrasmit[2]=13;
    badmove=1;
    while (badmove!=0) {}
}
USART_Transmit_Str("OK\r");
//restart timer here, next players move
wait=0;
t_count=0;
timeout=0;
```

Αρχικά θα διαβάσει τις συντεταγμένες σε `ascii` (μετατρέποντας το γράμμα σε αριθμό) και θα τις μεταφέρει σε δεκαδική μορφή ώστε να εξυπηρετούν την υλοποίηση μας. Στην συνέχεια, αν ο χρήστης δεν έχει υπερβεί τον επιτρεπόμενο χρόνο και αν σε αυτές τις συντεταγμένες του πίνακα `moves`, υπάρχει η τιμή `V(86 σε ascii)`, τότε προχωράει στην εκτέλεση της πράξης. Με τον τρόπο αυτό εξασφαλίζεται ότι έχουμε `valid` κίνηση και χρόνο. Αν υπάρχει παραβίαση σε μια από αυτές τις προϋποθέσεις, το `avr` στέλνει το κατάλληλο μήνυμα και περιμένει την απάντηση του χρήστη. Για την εκτέλεση κάποιας κίνησης υλοποιήσαμε την συνάρτηση `void make_move(int row, int col, char turn)`, η οποία ανάλογα με τις συντεταγμένες και το χρώμα του παίχτη κάνει τις κατάλληλες αλλαγές και τις αποθηκεύει στο `board` και αυξάνει το `moves_done`. Εν τέλει στέλνει `Ok`, τερματίζει το `loop` στο οποίο είχαμε κολλήσει περιμένοντας την κίνηση του παίχτη και επανεκκινεί τον χρονομετρητή. Αν το `avr` είναι ο άσπρος παίχτης, και έχει διαθέσιμες κινήσεις τότε μέσω της συνάρτησης `int avr_move(char turn)`, θα εκτελέσει την κίνηση του. Προς το παρόν θα παίζει την 1η διαθέσιμη κίνηση που θα βρει ψάχνοντας τον πίνακα `moves`.

Με τον ίδιο τρόπο θα ενεργεί και στην περίπτωση που παίζει ο μαύρος παίχτης, αντικαθιστώντας το 'W' με 'B', όπου χρειάζεται.

Παρακάτω ακολουθεί από μια σύντομη περιγραφή για τις συναρτήσεις `valid_moves` και `make_move`:

```
int valid_moves(char turn)
```

Η συνάρτηση αυτή, βοηθάει στο να ανιχνεύσουμε τις διαθέσιμες κινήσεις του παίχτη που παίζει στον εκάστοτε γύρο. Αρχικά αδειάζει τον πίνακα moves, γεμίζοντας τον με κενά (Space=32 ascii). Στην συνέχεια ξεκινάει αναζήτηση μέσα στον πίνακα board. Ελέγχει ένα-ένα τα κουτιά, σε περίπτωση που έχει ένα κουτί κάποιο πούλι, τότε τοποθετεί στον moves στην ίδια θέση το πούλι αυτό. Αν όμως το κουτί είναι κενό, ελέγχει εάν κάποιο από τα γειτονικά του κουτιά είναι πούλι του αντιπάλου. Σε περίπτωση που βρει αντίπαλο, ξεκινάει αναζήτηση προς κάθε κατεύθυνση “πατώντας” πάνω σε αντίπαλα πούλια, μέχρι να βρει κενό ή πούλι του παίχτη. Αν βρει κενό, τότε τερματίζει την αναζήτηση προς την κατεύθυνση αυτή, ενώ αν βρει πούλι του παίχτη θέτει το κουτί από το οποίο ξεκίνησε ως διαθέσιμη κίνηση του παίχτη, μαρκάροντας το στον πίνακα moves με V.

```
void make_move(int row, int col, char turn)
```

Η συνάρτηση αυτή, τοποθετεί το πούλι του παίχτη στις δοσμένες συντεταγμένες και ξεκινάει αναζήτηση προς κάθε κατεύθυνση “πατώντας” πάνω σε αντίπαλα πούλια, μέχρι να βρει πούλι του παίχτη. Όταν βρει ένα, ξεκινάει να προχωράει ανάποδα μέχρι να βρει την αρχική θέση και αλλάζει ένα-ένα τα πούλια του αντιπάλου.

Στις παρακάτω εικόνες φαίνεται το τελικό αποτέλεσμα μια εκτέλεσης του παιχνιδιού:

Valid Moves για άσπρο παίχτη στην αρχή:

	A	B	C	D	E	F	G	H	Cr
1									Cr
2									Cr
3					U				Cr
4				W	B	U			Cr
5			U	B	W				Cr
6				U					Cr
7									Cr
8									Cr
Type your move									Cr

Τελικός πίνακας και score:

```

      A      B      C      D      E      F      G      H Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
1 |      | B |      |      |      |      |      |      | Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
2 | W | B | B | W | W | W |      |      | Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
3 |      | B |      | B |      |      |      |      | Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
4 | B | B | B | W | W | W |      |      | Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
5 |      | B |      | W | W |      |      |      | Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
6 | B |      |      | W |      |      |      |      | Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
7 |      |      |      |      |      |      |      |      | Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
8 |      |      |      |      |      |      |      |      | Cr
+-----+-----+-----+-----+-----+-----+-----+-----+ Cr
OK Cr
You have to pass Cr
OK Cr
The final score is: 14 Cr
Cr
AUR: 12 User: 9WNCr

```

Εμφάνιση νικητή στο ανρ μέσω led:

