TIC TAC TOE

creazione di una A.I. che gioca contro un player umano

realizzato da Gabriele Quagliarella

Introduzione

- ***Che cos'è il "Tic Tac Toe" ?***
  - un popolarissimo gioco di strategia meglio conosciuto in Italia con il nome di "Tris".
  - si gioca utilizzando come campo di gioco una matrice quadrata 3 x 3 .

- ***Come funziona ?***
  - A turno, due giocatori si sfidano inserendo in una cella vuota il proprio simbolo (di solito una "X" o un "O"). Vince chi dei due riesce per primo a disporre tre dei propri simboli in linea retta orizzontale , verticale o diagonale.
  - Se la griglia viene riempita senza che nessuno dei giocatori sia riuscito a completare una linea retta di tre simboli, il gioco finisce in parità.

In cosa consiste il progetto ?

- Il progetto da me realizzato ha come obiettivo quello di creare un calcolatore "intelligente" in grado di giocare una partita di tic tac toe contro un player umano e riuscire a pareggiare sempre o addirittura vincere la partita.

Una sfida impossibile ?

- Almeno in linea teorica, realizzare un calcolatore in grado di giocare una partita regolamentare di tic tac toe richiederebbe la realizzazione di un circuito sequenziale in grado di implementare l'algoritmo "MiniMax". Questo però , proprio per com'è strutturato l'algoritmo , necessiterebbe di implementare un piccolo "PC" dedicato , completo di Stack ed altri strumenti necessari per poter implementare un algoritmo ricorsivo.

Come risolvere il problema ?

- La soluzione da ma ideata per risolvere il problema dell'implementazione dell'algoritmo ricorsivo è stata quella di realizzare una macchina che sapesse giocare ad una variante del gioco classico : l'unica differenza consiste nel fatto che è sempre il PC a fare la prima mossa.

- A primo impatto questa scelta può sembrare del tutto ininfluente nella risoluzione del problema ma in realtà si rivela essere fondamentale. Ecco il perché …

Da dove partire a ragionare ?

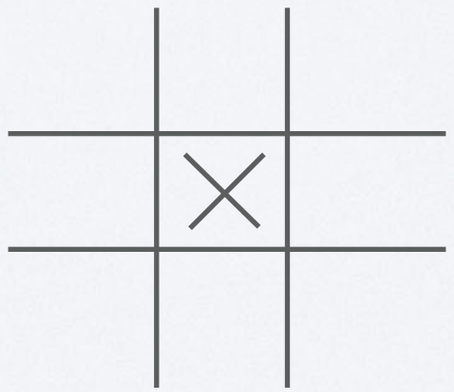
- Introducendo la variante secondo cui è il PC a fare la prima mossa, oltre ad evitare il problema marginale dello stabilire in maniere random quale dei due giocatori debba iniziare la partita, essa ci permette di stabilire a priori una strategia "artigianale" vincente ragionando non più sull'idea di valutare tutte le possibili mosse e poi scegliere quella più favorevole (come prevede l'algoritmo "Minimax") bensì sulle possibili mosse lecite che il giocatore avversario è costretto a fare e su di esse costruire tutte le mosse seguenti.

Un esempio concreto…

- Proviamo ora a costruire la nostra strategia vincente sulla base delle premesse precedentemente descritte.

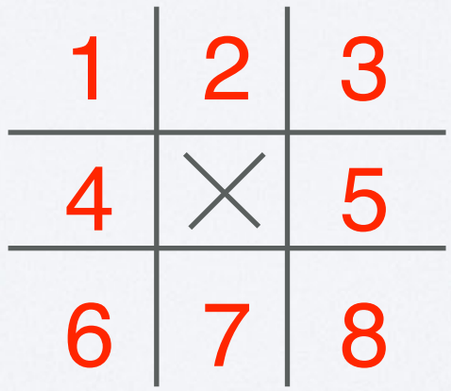
- Supponiamo di voler assegnare il simbolo "X" alla mossa del PC e il "O" a quella del giocatore.

  - Poiché è il PC a fare la prima mossa inseriamo la "X" al centro. Come vedremo più avanti , oltre che a semplificare l'implementazione circuitale della scelta della prima mossa (che in questo modo è annullata) , stabilire a priori la prima mossa , ed in particolare decidere di posizionarla al centro, si rivelerà fondamentale per la risoluzione del problema.



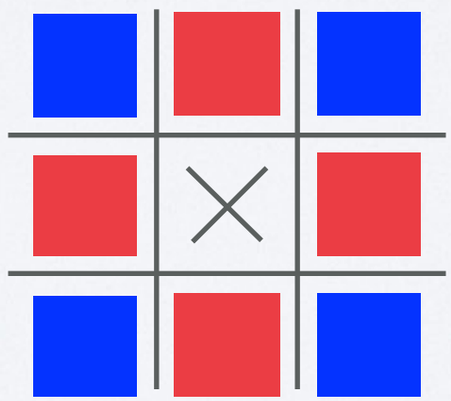
Prima mossa del giocatore

- A questo punto sarà il turno del giocatore scegliere quale mossa fare.
- L'idea è quella di stabilire la mossa successiva che prederà il calcolatore in base all'ultima mossa fatta del giocatore.
- ***Quante sono le possibili mosse lecite che il giocatore è costretto a fare ?***
  - A questo livello le possibili mosse sono solamente 8, infatti :



Dividere il problema in sottoproblemi

- L'ulteriore osservazione che ho fatto è stata quella di dividere in due grandi categorie le possibili mosse che il giocatore avversario può fare al primo turno. In particolare ho chiamato :
  - "Caso Lato" : le quattro mosse di colore **rosso**
  - "Caso Spigolo" : le quattro mosse di colore **blue**



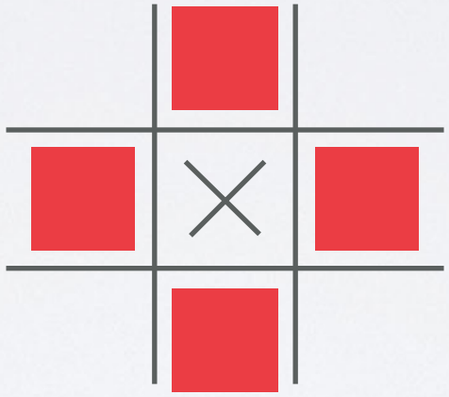
Da questo momento in poi i due casi (Lato e Spigolo) verranno analizzati in maniera differente per poi essere riuniti solo alla fine.

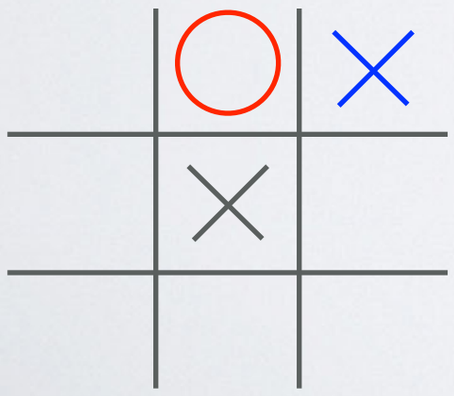
Il "Caso Lato" slide 1

- A questo punto per ogni situazione appartenente al "Caso Lato" andiamo ad esplicitare la migliore mossa da fare.
- Come è facile intuire , ciascuna delle quattro combinazioni è in realtà il medesimo caso ruotato di 90°. A seguito di questa considerazione possiamo quindi stabilire una strategia per un solo caso (ad esempio il primo) e poi ripetere le stesse scelte per tutti gli altri 3 casi ma tenendo conto della rotazione di 90°.

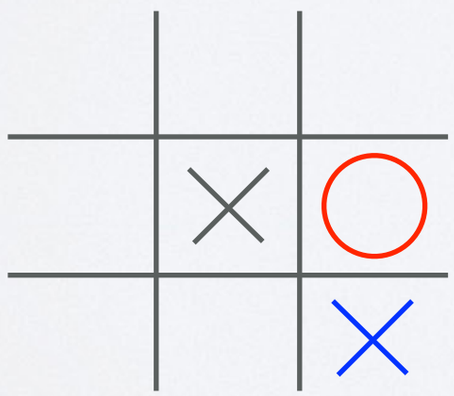
X = mossa PC

O = mossa PLAYER

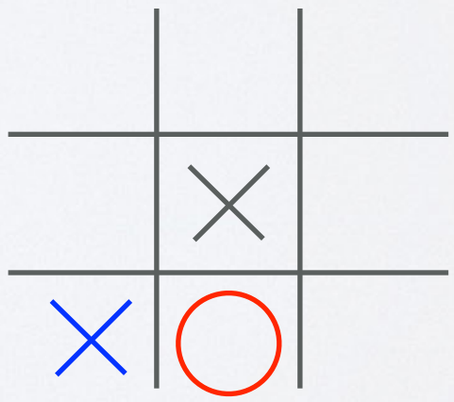




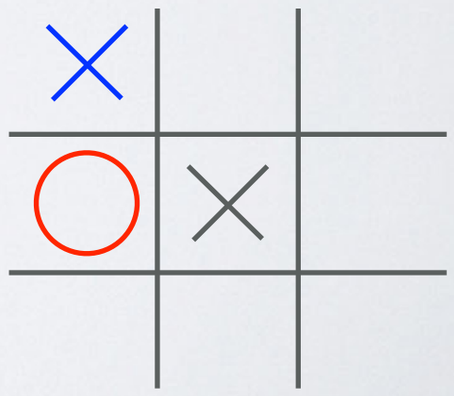
1° Caso



2° Caso



3° Caso

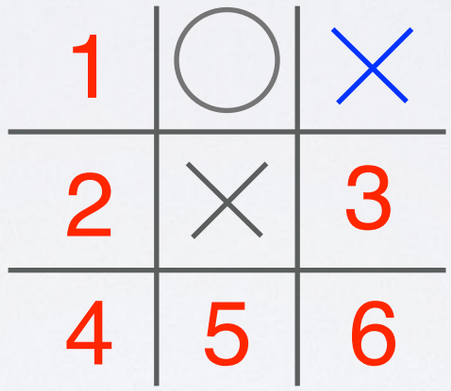


4° Caso

Il "Caso Lato" slide 2

- Come già accennato nella precedente slide, poiché le diverse quattro situazioni sono la medesima a meno di una rotazione di 90° , per semplicità andremo ad analizzare il caso più "semplice" per poi ripetere la medesima strategia sugli altri tre casi.

- A questo punto , dopo aver stabilito come seconda mossa per il PC quella colorata in **blu**, andiamo a prevedere le possibili mosse che il giocatore potrà fare :



- Utilizzando il ragionamento analogo a quello fatto per il "Caso Lato" e "Caso Spigolo" , possiamo anche qui distinguere in due gruppi le possibili mosse che il giocatore può fare : il primo gruppo sarà composto dalle mosse che non porteranno subito alla sconfitta e quindi faranno si che il gioco prosegua con il turno del PC. Al contrario il secondo gruppo contiene le mosse che porteranno alla sconfitta del giocatore e quindi alla vittoria del PC.

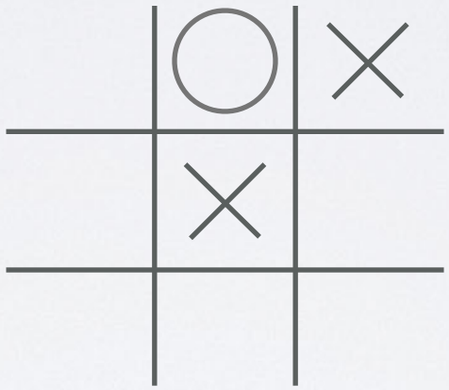
- In particolare le mosse numero 1 , 2 , 3 , 5 e 6 se fatte dal giocatore permetteranno al PC di posizionare al turno successivo il proprio segno nella casella della mossa 4 e quindi vincere la partita.

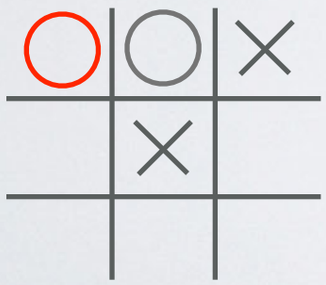
- Al contrario il gruppo delle mosse che faranno continuare il gioco è composto solamente dalla mossa 4.

Una visione ad albero

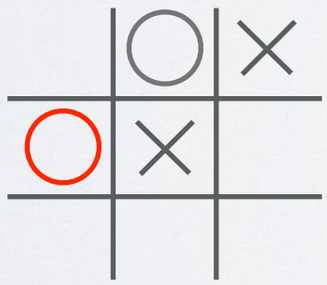
- Se volessimo rappresentare le possibili partite che nascono dalle varie scelte fatte dal giocatore, la rappresentazione risultate è quella di una struttura ad albero.

- Tra queste solo il caso 4 permette la continuazione della partita.Tutte le altre invece permettono al PC di posizionare il proprio segno nella casella vincente.

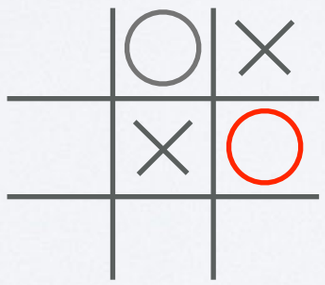




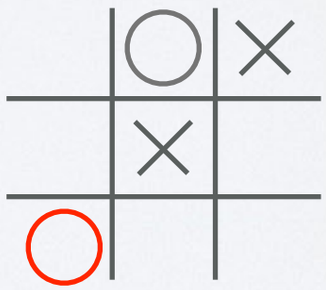
1° Caso



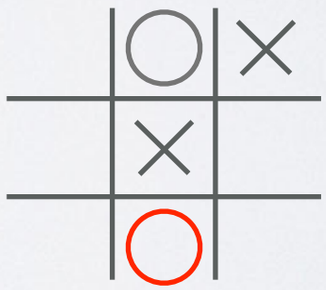
2° Caso



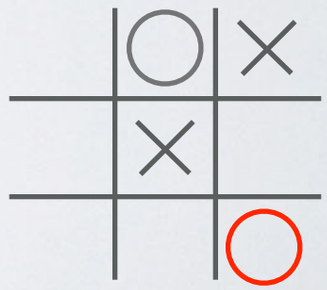
3° Caso



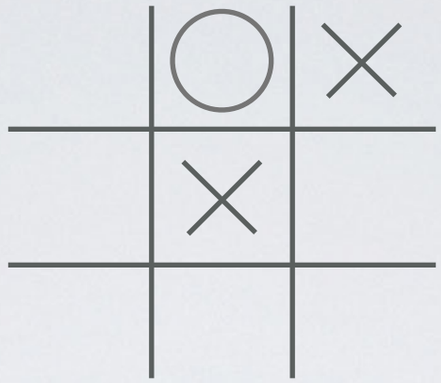
4° Caso

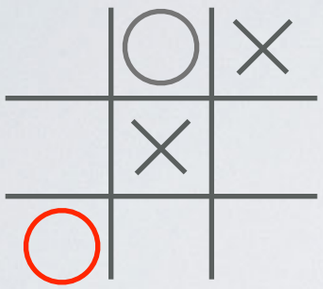


5° Caso

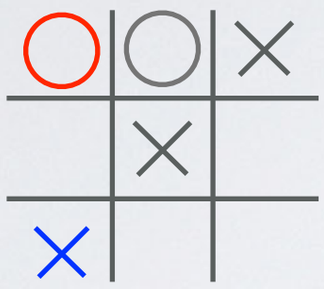


6° Caso

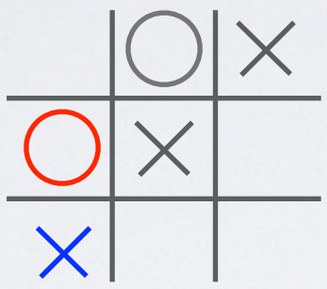




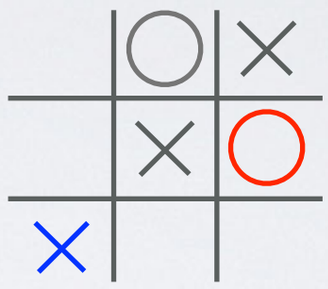
4° Caso



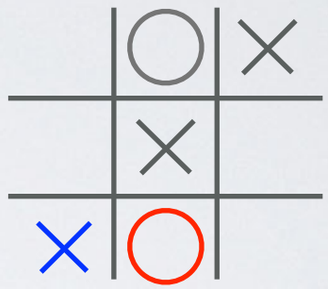
1° Caso



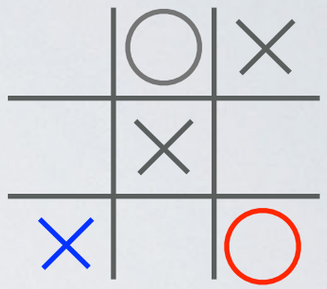
2° Caso



3° Caso



5° Caso



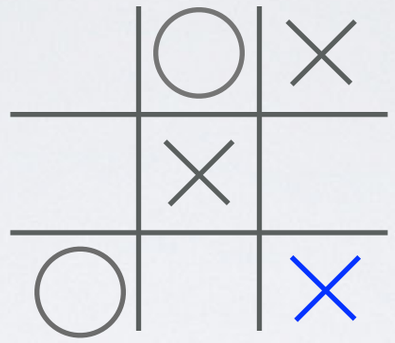
6° Caso

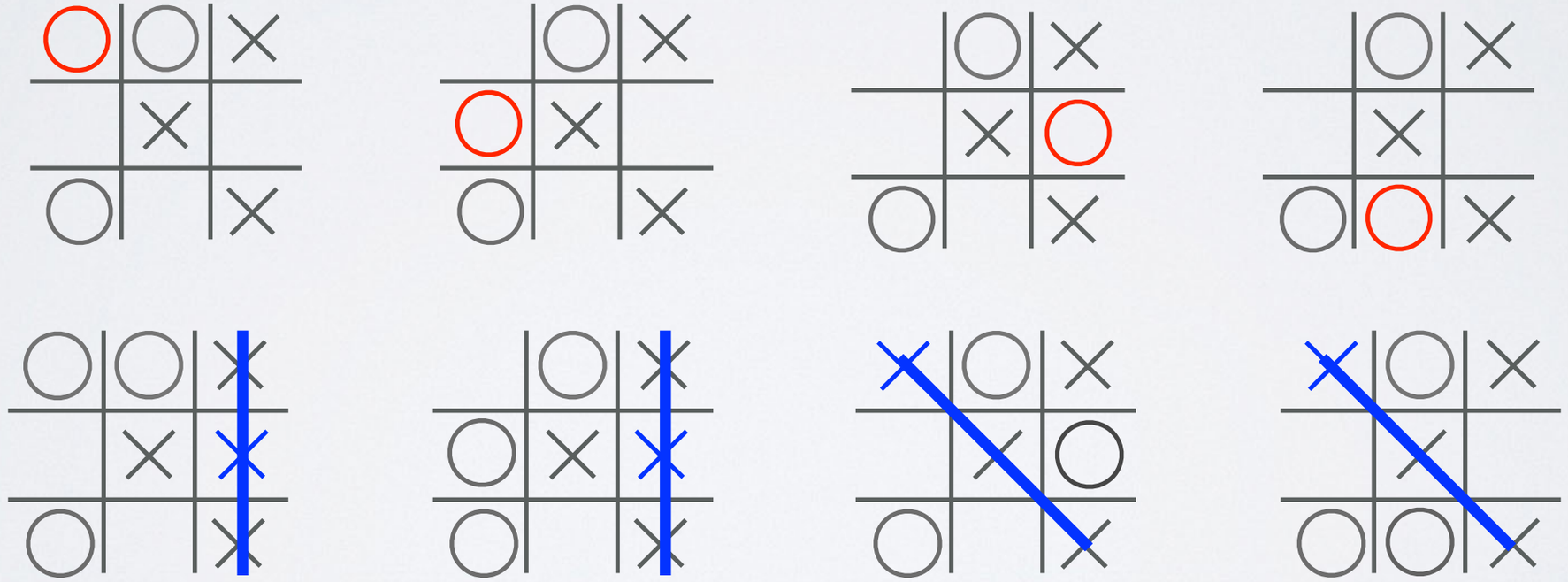
Questo invece è l'unico caso in cui la partita prosegue con la mossa del PC

In tutti questi cinque casi la partita termina con la vittoria del PC

Conclusione del "Caso Lato" parte I

- A questo punto il proseguiamo con la scelta della strategia per la nostra macchina, in particolare scegliamo di posizionare la "X" nell'ultima casella in basso a destra.



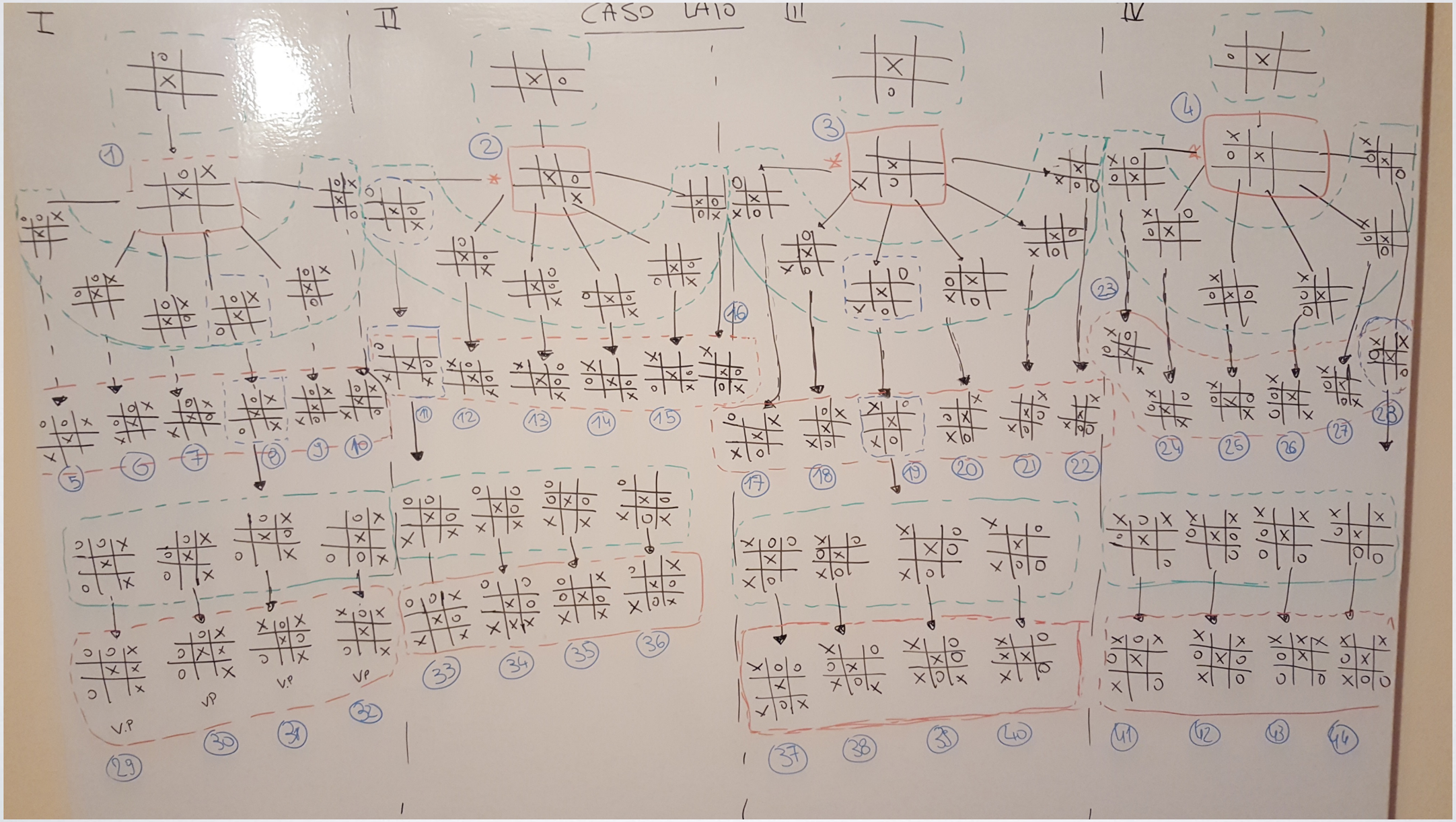


- Come si può notare dal grafico , qualsiasi mossa faccia il giocatore, il PC riuscirà comunque a trovare una strategia vincente che lo porterà alla vittoria.

Conclusione del "Caso Lato" parte 2

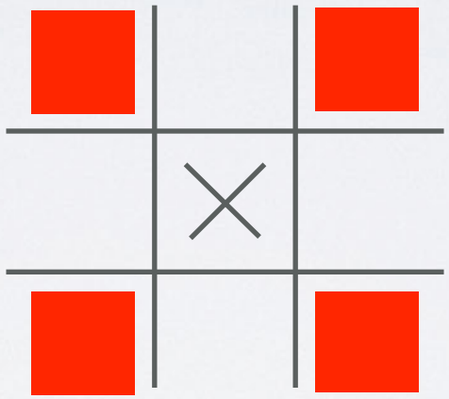
- Poiché tutte le altre possibili partite appartenenti al gruppo "Caso Lato" non sono altro che una traslazione del caso precedentemente analizzato, possiamo concludere in definitiva che esse daranno anche loro luogo ad una vittoria certa per il PC.

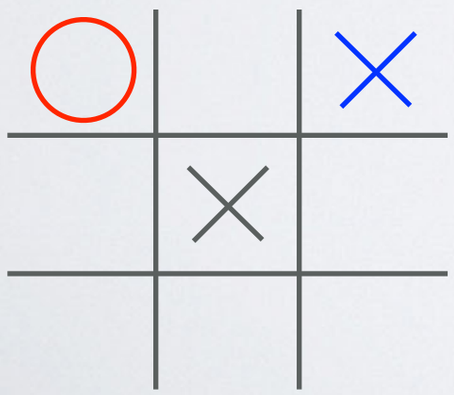
- Per avere una stima della complessità del grafico possiamo dare un'occhiata al seguente schema :



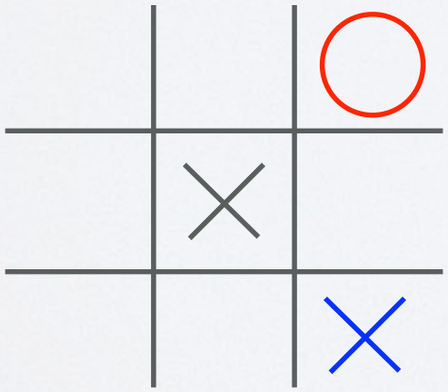
Il "Caso Spigolo"

- Anche per questo caso l'idea usata per affrontare il problema è stata quella di elaborare una strategia vincente solo per un caso e poi andarla a riproporre sugli altri tre in maniera analoga a meno di una rotazione di 90°

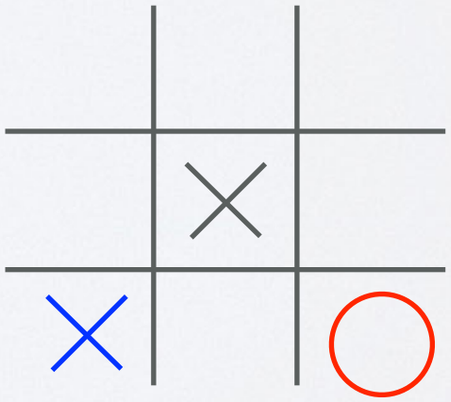




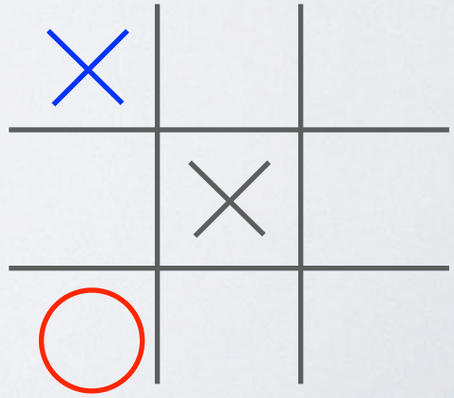
1° Caso



2° Caso



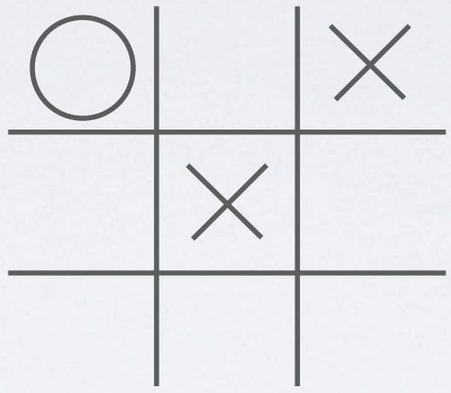
3° Caso



4° Caso

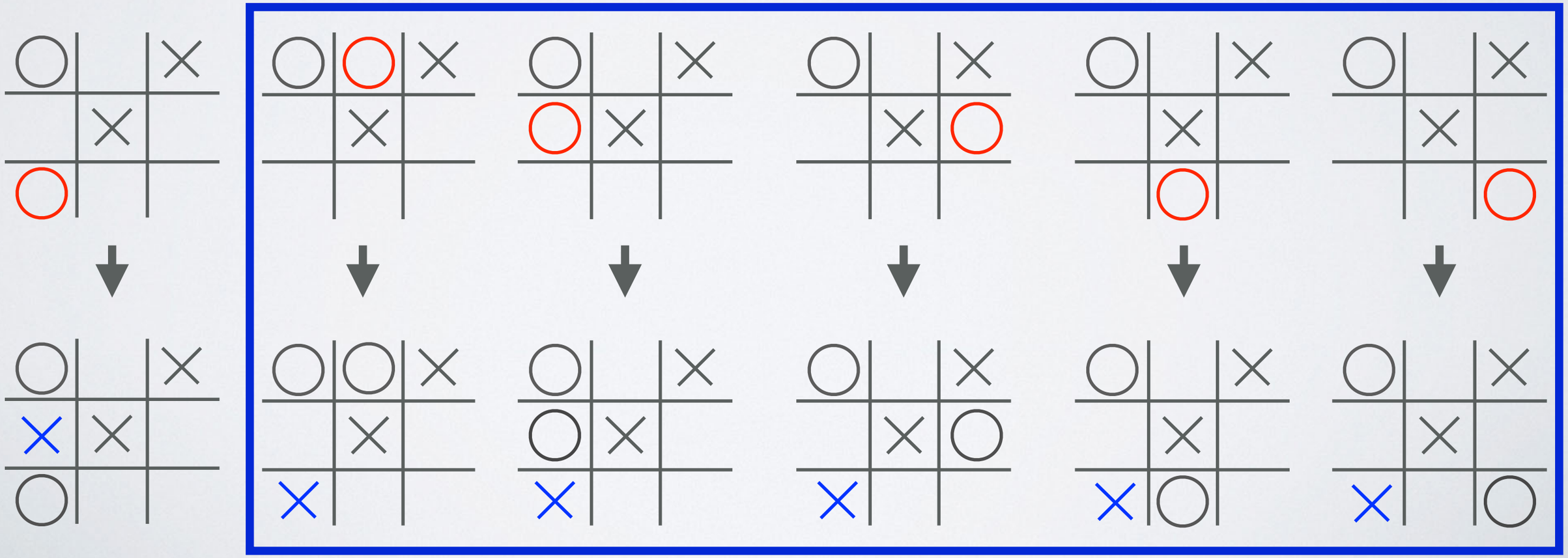
La prima scelta del "Caso Spigolo"

- Di seguito sono riportate rispettivamente le sei possibili mosse che il giocatore può fare e la mossa con cui il PC risponde. Come si può notare, anche qui come nel "Caso Lato" l'unica mossa che permette di far continuare il gioco è la prima sulla sinistra, tutte le altre invece permettono al PC di chiudere subito la partita.



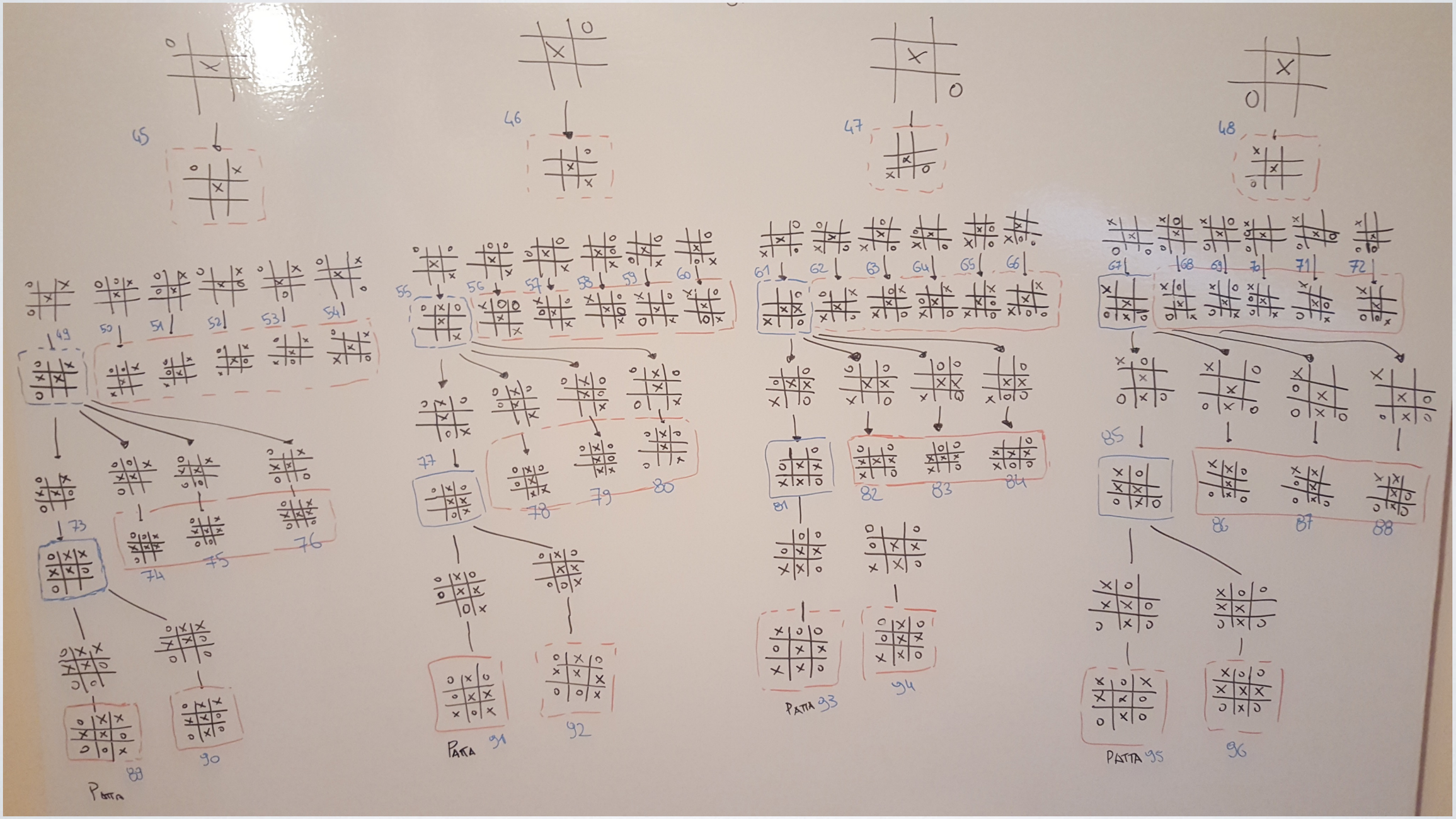
Partita continua

PC vince



Conclusione del "Caso Spigolo" parte 2

- Procedendo in maniera analoga si giunge all'ultimo livello dell'albero. Qui al contrario del "Caso Lato" possono accadere due situazioni : se il giocatore non commette errori la partita terminerà con un pareggio, mentre se così non fosse il PC riuscirà a vincere anche in questo caso.



In conclusione…

- Avendo costruito due alberi rappresentativi delle possibili partite che possono nascere dalla situazione iniziale, abbiamo esplicitato non solo i possibili scenari ma anche codificato la migliore strategia da applicare a tutte le possibili situazioni. Ovviamente tutto questo si basa sull'assunzione di far cominciare sempre per primo il PC.

- Come avevamo già scoperto poco fa , nel "Caso Lato" tutte le partite appartenenti a questo gruppo si concludono in favore del PC.

- Nell'altro caso invece, la percentuale di vittoria si dimezza (il PC riesce a vincere solo in metà dei casi) ma non si verificano comunque situazioni critiche in cui il giocatore ha la possibilità di vincere.

- Possiamo così concludere di aver trovato una strategia sub-ottima che ci permette di raggiungere sempre la vittoria o al più un pareggio , ma mai una sconfitta.

Come passare dall'idea al circuito ?

- In realtà la risoluzione del problema è già stata trovata. Come vedremo più avanti infatti , aver creato una strategia vincente che tenga conto di tutte le possibili mosse del giocatore ci permetterà di sintetizzare una Macchina a stati finiti (poiché a questo punto siamo in grado non solo di conoscere il numero esatto ma anche la loro natura ovvero come sono tutte le possibili partite che possono nascere) in grado di passare ad uno stato futuro (che qui è rappresentato nello schema ad albero come uno scenario intermedio o finale cerchiato in rosso) sulla base della mossa compiuta dal giocatore.

- A questo punto si tratta solamente di suddividere il problema in tanti sotto-problemi più semplici e di creare per ognuno di essi un circuito in grado di soddisfare la nostra esigenza.

Che cosa ci serve ?

- Volendo creare una macchina che sappia affrontare un player umano in una partita di Tris abbiamo sicuramente bisogno di :

1. uno "schermo" su cui visualizzare la partita.
2. una tastiera composta da bottoni che permetterà al giocatore di selezione la propria mossa.
3. bottone di reset per ricominciare la partita in qualsiasi momento.
4. e ovviamente il "cervello" della nostra intelligenza artificiale che sia capace di analizzare la mossa fatta dal giocatore ed in base a quella rispondere seguendo la strategia da noi prima trovata.

Uno sguardo al circuito

- Ecco una vista dall'alto di quello che sarà poi il circuito finale.

- Come detto prima , l'idea è stata quella di suddividere il problema in sotto-circuiti più semplici in particolare:

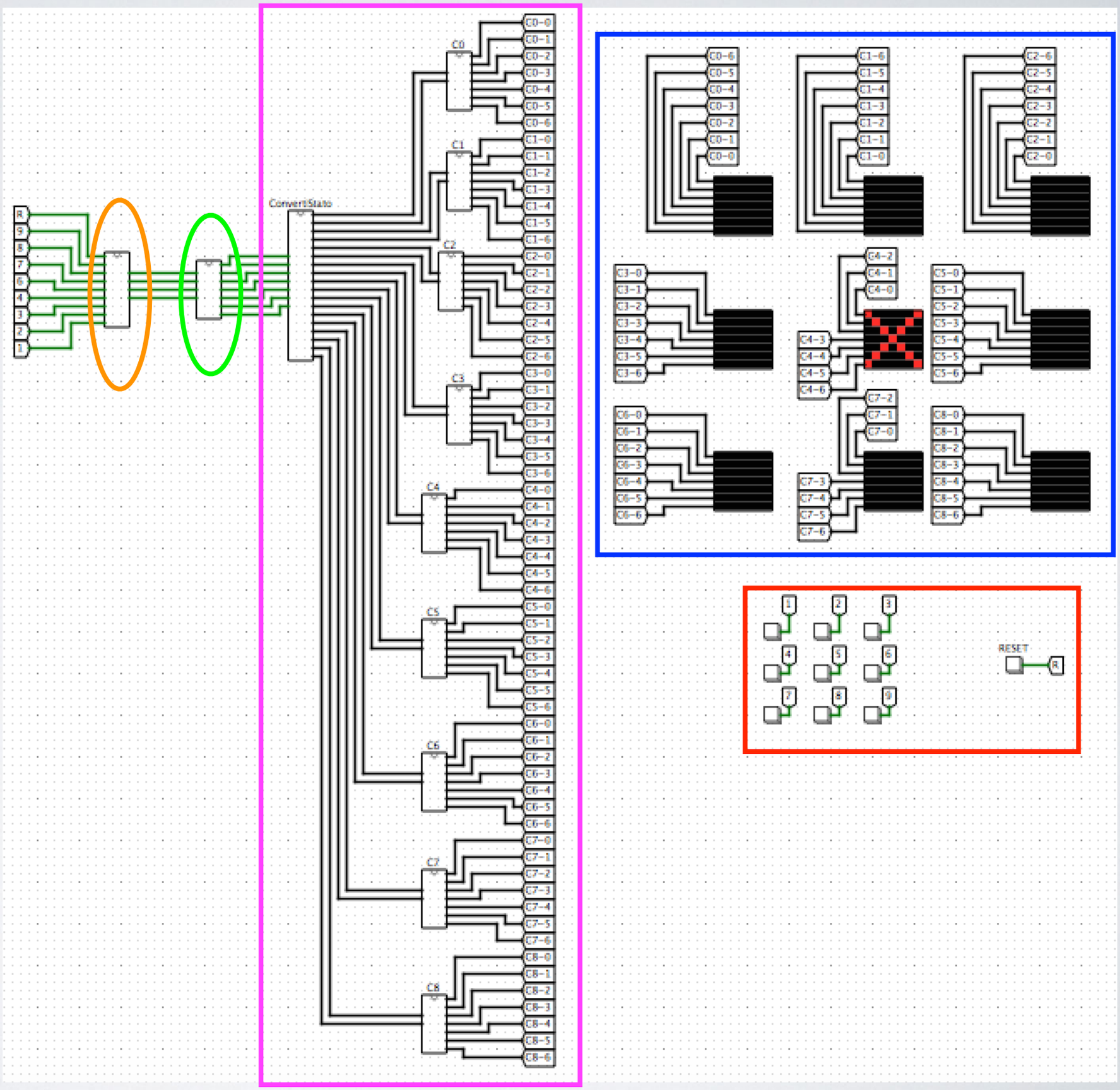
1. **Matrici LED** per stampare lo stato della partita

2. **Una tastiera** di selezione della mossa per il giocatore

3. Un circuito chiamato **"Codifica Ingresso"** che converte il tasto premuto sulla tastiera in una stringa di 4 bit.

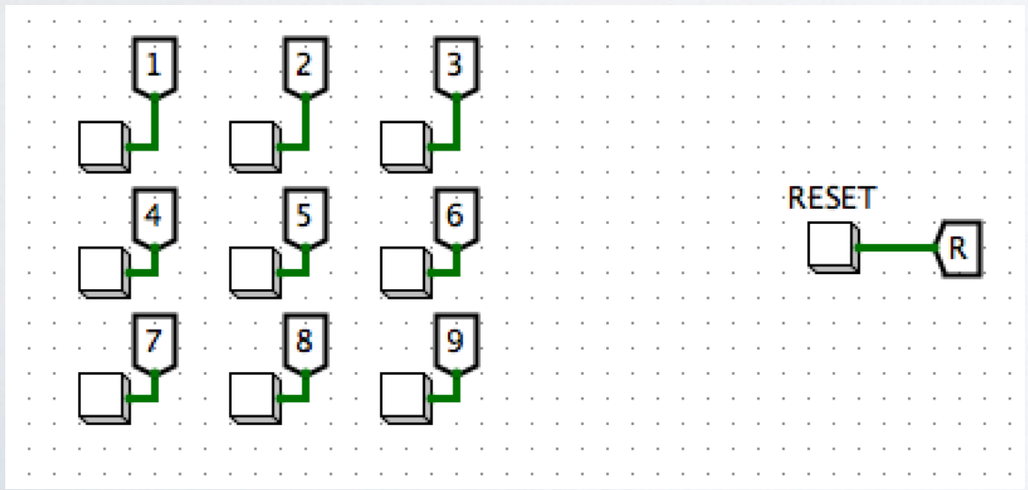
4. La **MSF** o macchina a stati finiti che elabora il tasto premuto e in base a quello fa avanzare lo stato della partita.

5. Un circuito chiamato **"Converti Stato"** che sulla base dello stato della partita stampa sulla matrice LED la partita stessa.



“Codifica Ingresso”

- Esso è il primo componente che entra in gioco nel momento in cui il giocatore clicca un qualsiasi pulsante della tastiera.
- Il suo compito è quindi quello di codificare ciascun tasto della tastiera in una stringa di 4 bit, in particolare :
  - Chiamati rispettivamente T1 il tasto 1, T2 il tasto 2 , … e R il tasto reset ecco la tabella di verità e le corrispettive stringhe di 4 bit in uscita.
  - Da notare che gli ingressi della tabella sono 9 e non 10 come nella tastiera : questo perché poiché la casella centrale della nostra matrice è in realtà già occupata dalla prima mossa del PC(assunzione di partenza) per semplificare la gestione degli errori ho preferito eliminare il tasto 5 dai possibili tasti premuti, tenendolo sulla tastiera , ma di fatto escludendolo dagli input del circuito.
  - In definitiva i tasti dall’ 1 al 4 corrispondono ai T1 / T4. Il tasto 5 è escluso e quindi dal tasto 6 in poi essi corrispondono a 6 -> T5, 7-> T6 , 8->T7 , 9 ->T8 e Reset -> R

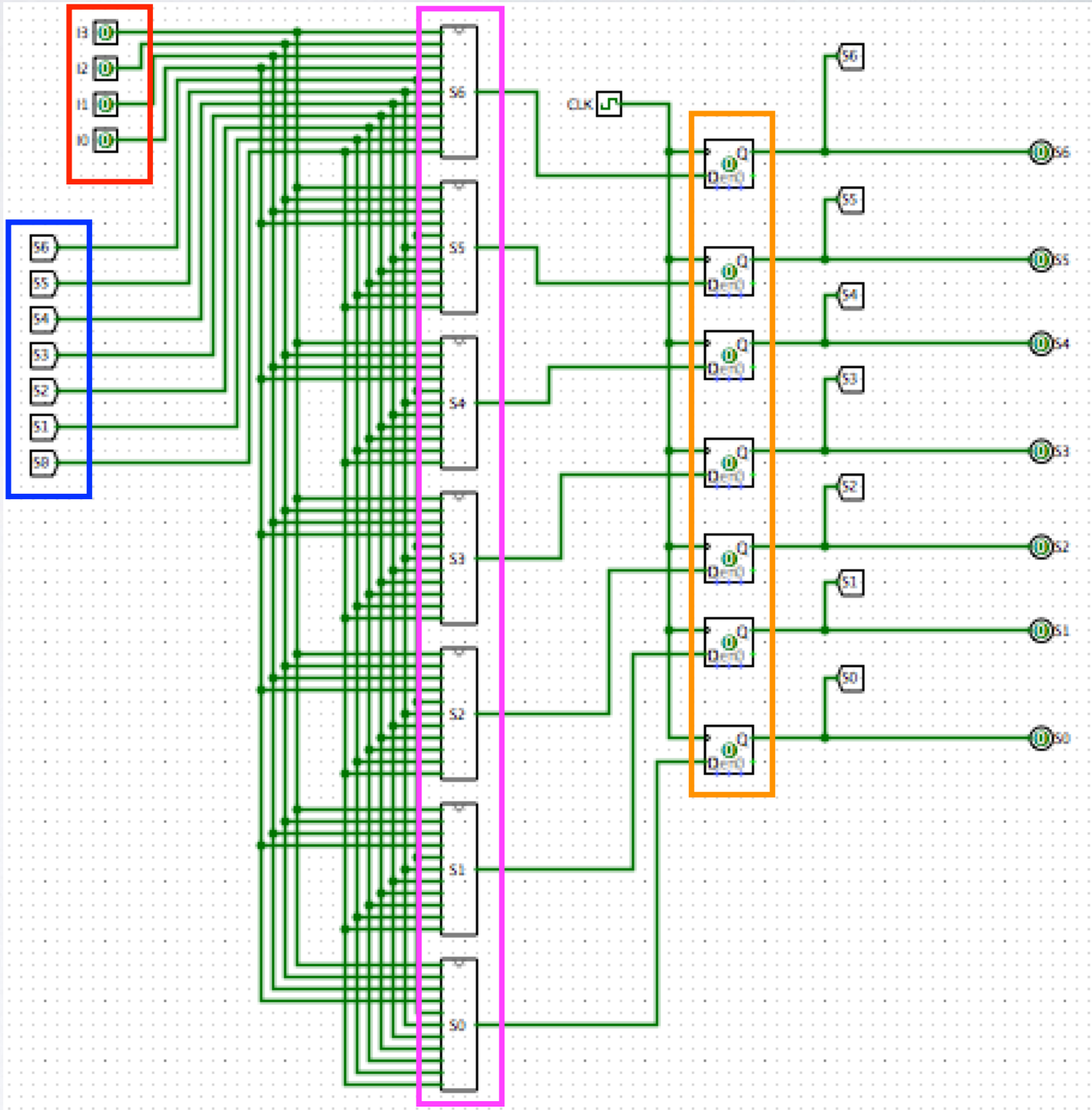


R	T8	T7	T6	T5	T4	T3	T2	T1	U3	U2	U1	U0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	1

MSF

- La MSF è , come abbiamo detto, il cuore dell'intelligenza artificiale. Sulla base dello stato in cui si trova la partita e la mossa del giocatore fa avanzare la partita al suo stato futuro. Essa è a sua volta divisa in diverse parti tra cui :

1. Input :
    - **Mossa giocatore** (4 BIT).
    - **Stato della partita** (7 BIT)
2. Area di **elaborazione Stato Futuro***:
    1. Composta da 7 sotto-circuiti, ognuno di essi collegato con gli 11 bit di ingresso (Mossa giocatore + Stato partita), che calcolano il bit corrispettivo dello stato futuro.
3. Area di **memorizzazione** :
    1. Composta da 7 Flip-flop. Ognuno di essi memorizza il bit dello stato futuro finché il giocatore non preme un altro tasto.



Sintetizzazione circuito di elaborazione Stato Futuro

parte I

- Il primo passo fatto è stato quello di codificare lo schema ad albero precedentemente mostrato , nella STT. Come stati (prima colonna di sinistra) troviamo tutte le possibili partite che nello schema sono cerchiate in rosso. Esse contengono la mossa fatta dal PC. Come input troviamo invece la mossa del giocatore, ovvero come vedremo successivamente, una stringa di 4 bit (quella codificata dal circuito "Codifica Ingresso").

				CASO LATO										
		NULL	1	2	3	4	5	6	7	8	RESET			
	STATI	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001			S = Caso Spigolo
	0	0	45	1	46	4	2	48	3	47	0			L= Caso Lato
	1	1	5	1	1	6	7	8	9	10	0			
	2	2	11	12	13	14	2	15	16	2	0			FINE PARTITA :
	3	3	17	18	19	20	21	3	3	22	0			RESET
	4	4	1	23	24	4	25	26	27	28	0			VINCE PC
	5	5	5	5	5	5	5	5	5	5	0			PATTA
	6	6	6	6	6	6	6	6	6	6	0			
	7	7	7	7	7	7	7	7	7	7	0			
	8	8	29	8	8	30	31	8	32	8	0			
	9	9	9	9	9	9	9	9	9	9	0			
	10	10	10	10	10	10	10	10	10	10	0			
	11	11	11	33	34	35	11	11	36	11	0			
	12	12	12	12	12	12	12	12	12	12	0			
	13	13	13	13	13	13	13	13	13	13	0			
	14	14	14	14	14	14	14	14	14	14	0			
	15	15	15	15	15	15	15	15	15	15	0			
	16	16	16	16	16	16	16	16	16	16	0			
	17	17	17	17	17	17	17	17	17	17	0			
	18	18	18	18	18	18	18	18	18	18	0			
	19	19	19	37	19	38	39	19	19	40	0			
	20	20	20	20	20	20	20	20	20	20	0			

Sintetizzazione circuito di elaborazione Stato Futuro

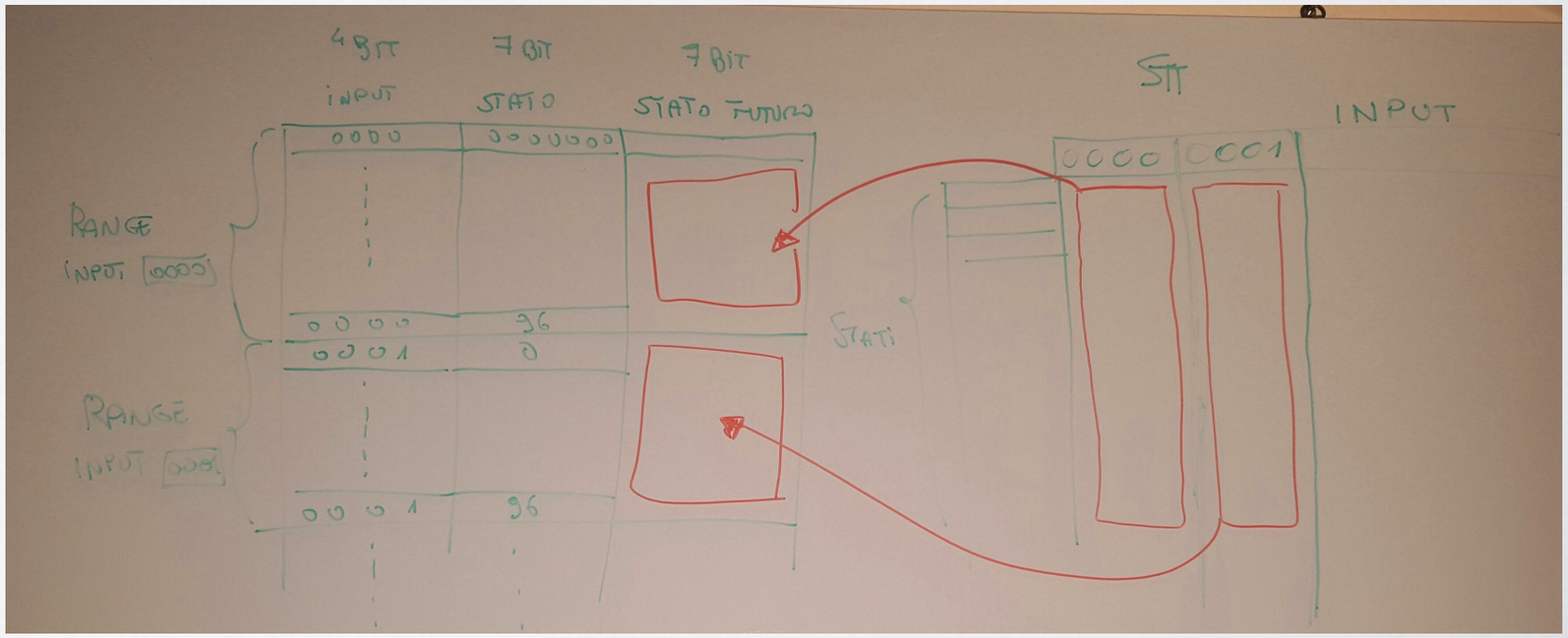
parte 2

- Il secondo passo è stato quello di codificare in binario tutti gli stati futuri presenti nella STT.

- Per sintetizzare in maniera più rapida la tabella degli stati futuri ho preferito riorganizzare la STT binaria in una tabella più compatta nel seguente modo :

- per ogni input ho creato un range di valori che copre tutti i 97 stati. Ripetendo questa operazione per tutti i 9 input ho così creato una tabella avente come range quelli riportati sullo schema in alto a destra. La nuova tabella creata riporta come valori di uscita i corrispettivi 7 bit dello stato futuro della STT binaria, corrispondenti al proprio range di appartenenza.

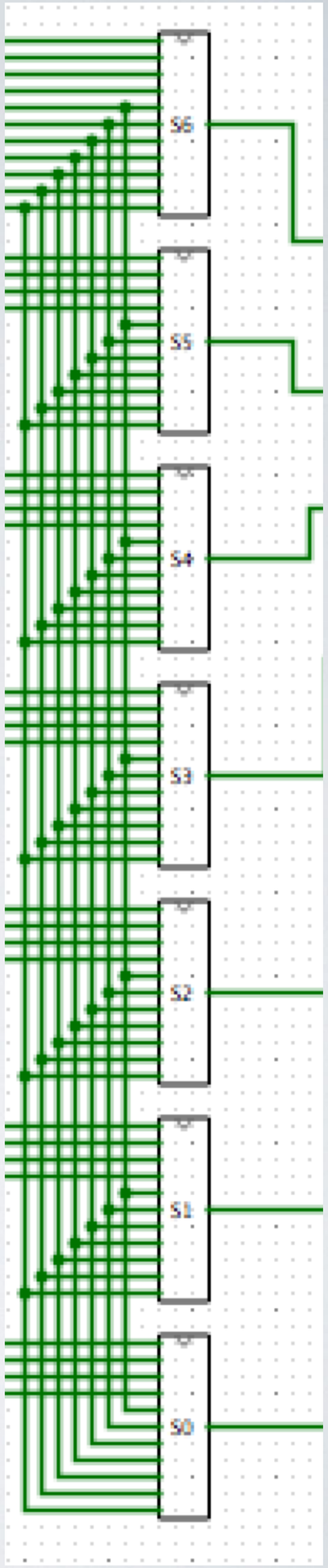
Sintetizzazione Macchina a Stati Finiti				
	RANGE		RANGE(DEC)	
INPUT	da	a	da	a
0000	0000-0000000	0000-1100000	0	96
0001	0001-0000000	0001-1100000	128	224
0010	0010-0000000	0010-1100000	256	352
0011	0011-0000000	0011-1100000	384	480
0100	0100-0000000	0100-1100000	512	608
0101	0101-0000000	0101-1100000	640	736
0110	0110-0000000	0110-1100000	768	864
0111	0111-0000000	0111-1100000	896	992
1000	1000-0000000	1000-1100000	1024	1120
1001	1001-0000000	1001-1100000	1152	1248



- Ecco la tabella risultante dopo la riorganizzazione della STT binaria.
- Come si può notare, in input troviamo i 4 bit relativi alla mossa fatta dal giocatore e i 7 bit relativi allo stato attuale.
- I valori di uscita di questa tabella corrispondono ai valori dello stato futuro della STT.
- A questo punto lo step successivo è stato quello di passare questa tabella ad un software di sintetizzazione di tabelle di verità (Espresso) che mi ha automaticamente calcolato la funzione generatrice per ogni uscita.
- Per ciascuna uscita è stata calcolata la corrispettiva funzione generatrice che mi ha quindi permesso di creare il corrispettivo circuito. Ecco quindi spiegato a cosa corrispondono i 7 circuiti che compongono al MSF.

I3	I2	I1	I0	S6	S5	S4	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	1	0	0	0	0

F0	F1	F2	F3	F4	F5	F6
0	0	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	0	1	0	1
0	0	0	0	1	1	0
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	0	1	0	0	1
0	0	0	1	0	1	0
0	0	0	1	0	1	1
0	0	0	1	1	0	0
0	0	0	1	1	0	1
0	0	0	1	1	1	0
0	0	0	1	1	1	1
0	0	1	0	0	0	0



Converti Stato

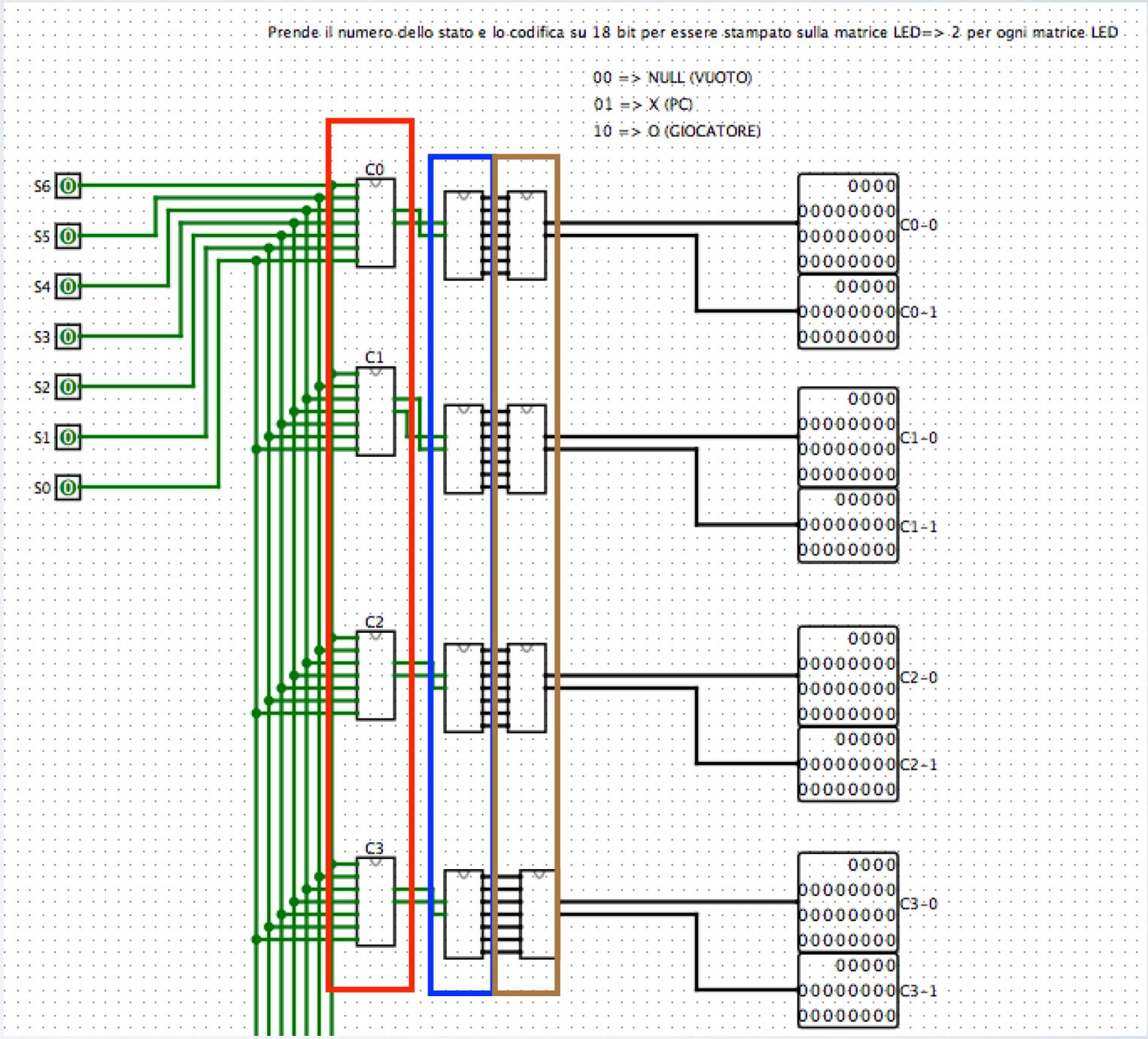
- Esso è l'ultimo sotto-circuito che compone il nostro progetto. Il suo compito è quello di ricevere in input i 7 bit di stato prodotti dalla MSF e codificarli sul 18 bit (2 per ogni ogni casella che compone la nostra griglia 3x3).

- E' composto dalle seguenti sottoparti :

1. **9 sottocircuiti** chiamati C0, C1,…che hanno il compito di produrre in output 2 bit ciascuno rappresentante il simbolo che dovrà essere poi inserito nella corrispettiva casella della griglia 3x3.

2. 9 componenti chiamati **"Converti Simbolo"** uno per ciascun "Cn". Essi hanno il compito di produrre in output 7 stringhe da 7 bit che saranno poi utilizzate nelle matrici LED per poter stampare il corrispettivo simbolo

3. 9 ulteriori circuiti chiamati **"Comprimi Stringhe"** necessari per compattare le stringhe del "Converti Simbolo".



Creazione dei "Cn"

- Per poter sintetizzare i sotto circuiti "Cn" ho codificato la sottostante tabella che si va a leggere in questa maniera : sulla sinistra troviamo tutti i possibili input ovvero tutti gli stati prodotti dalla MSF , mentre come output troviamo appunto i 18 bit che vanno a disegnare la griglia 3x3 corrispondente a quel determinato stato.

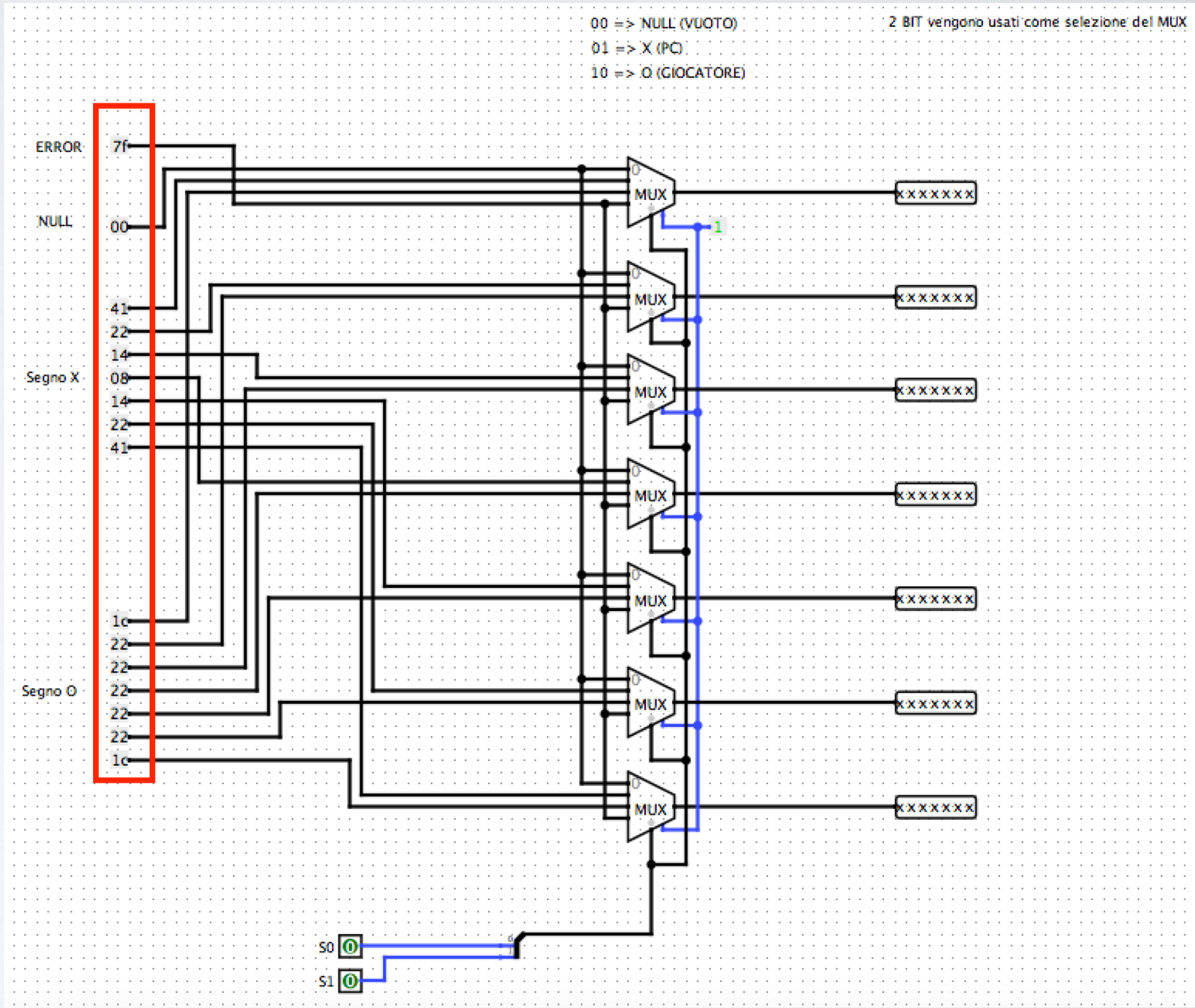
- A questo punto ho sviluppato 18 tabelle di verità una per ciascun "Yn" , ognuna delle quali ha prodotto un sottocircuito. Successivamente ho raggruppato a coppie di due i circuiti "Yn" per creare il corrispettivo "Cn".

	STATO 7 BIT							FUNZIONI															
								C0		C1		C2		C3		C4		C5		C6		C7		C8	
	S6	S5	S4	S3	S2	S1	S0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15	Y16	Y17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1
3	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0
4	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0
6	0	0	0	0	1	1	0	0	0	1	0	0	1	1	0	0	1	0	0	0	1	0	0	0	0
7	0	0	0	0	1	1	1	0	0	1	0	0	1	0	0	0	1	1	0	0	1	0	0	0	0
8	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1
9	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0
10	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0
11	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1
12	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1
13	0	0	0	1	1	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1
14	0	0	0	1	1	1	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1
15	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1
16	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1
17	0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0
18	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0
19	0	0	1	0	0	1	1	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1	0	0	0
20	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0
21	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0
22	0	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	0	1	0
23	0	0	1	0	1	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1
24	0	0	1	1	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1
25	0	0	1	1	0	0	1	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1
26	0	0	1	1	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1
27	0	0	1	1	0	1	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1
28	0	0	1	1	1	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	0	0	0	1	0
29	0	0	1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	0	1	1	0	0	0	0	1
30	0	0	1	1	1	1	0	0	0	1	0	0	1	1	0	0	1	1	0	1	0	0	0	0	1
31	0	0	1	1	1	1	1	0	1	1	0	0	1	0	0	0	1	1	0	1	0	0	0	0	1

Legenda Colori		
NULL		
01	X	PC
10	O	PLAYER

Converti Simbolo

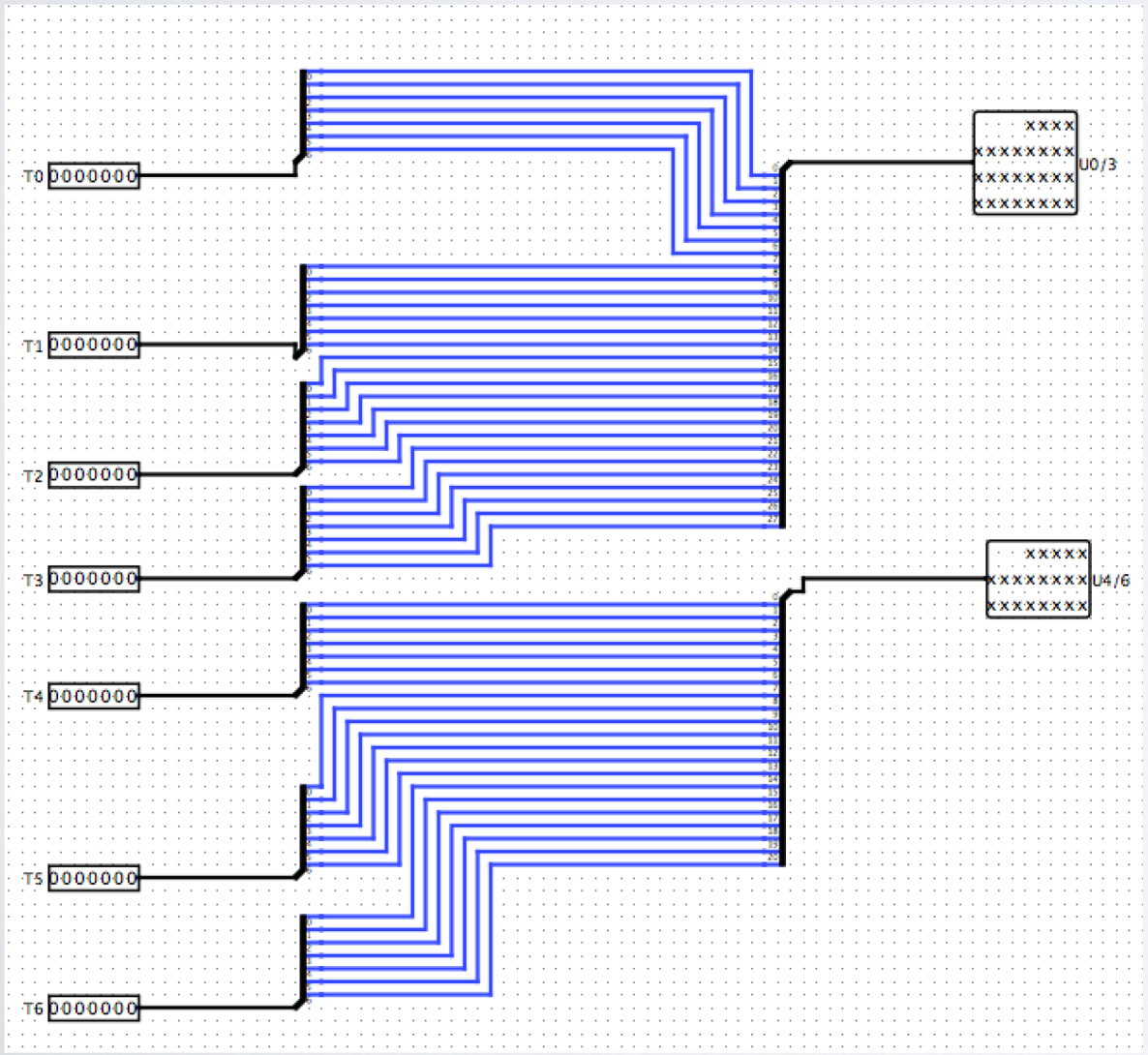
- Il seguente circuito ha il compito di produrre in output le 7 stringhe da 7 bit ciascuna necessarie per poter stampare il corrispettivo segno in una delle 9 matrici LED.
- Dopo aver stabilito le celle della matrice 7x7 da colorare per poter ottenere il simbolo corrispettivo, qui definite mediante delle **costanti**, i 2 bit in input al circuito andranno a pilotare i 7 MUX che avranno come propria uscita una stringa binaria da 7 bit corrispondente alla riga della matrice LED. L'unione di tutte le righe prodotte dai 7 MUX andrà quindi a comporre il simbolo da disegnare.

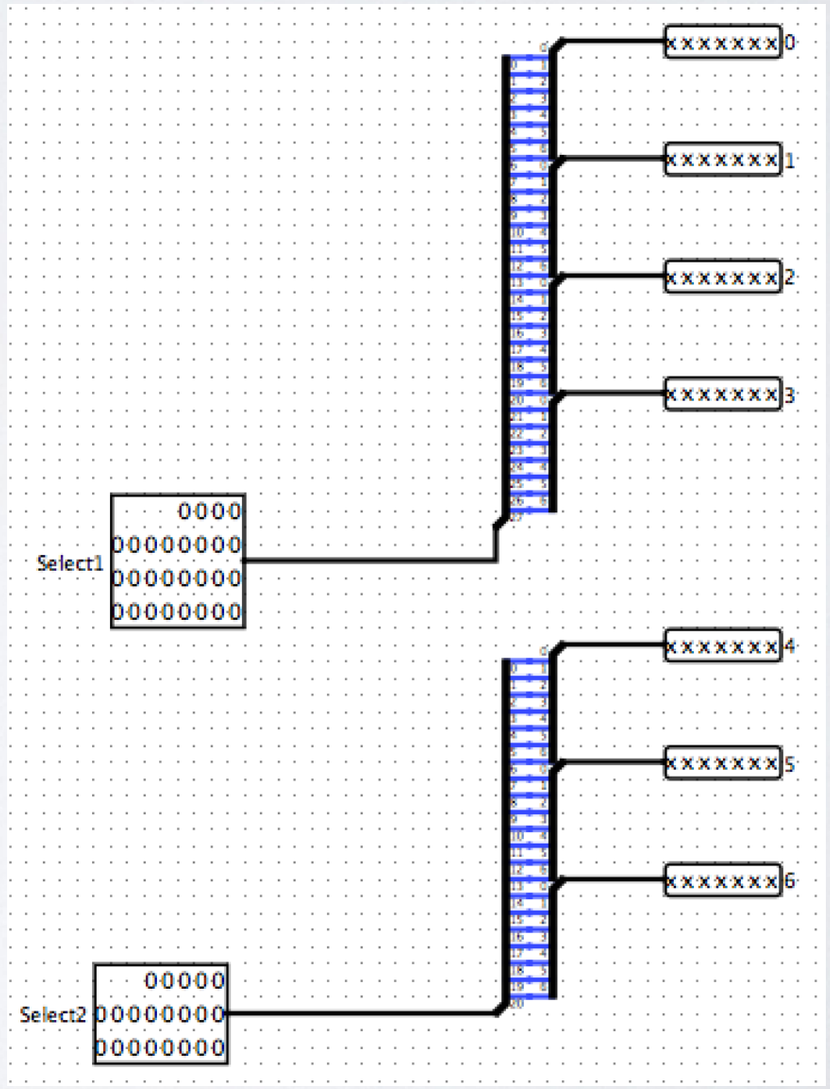


Comprimi Stringhe ed Espandi Stringhe

- Questi due circuiti sono uno l'inverso dell'altro e servono per avere degli output più compatti per il "Converti Stato". Il "Comprimi Stringhe" prende in input le 7 stringhe dei "Converti Simbolo" e le compatta in due stringhe da 28 e 21 bit rispettivamente.

- "Espandi Stringhe" è invece il circuito complementare e viene utilizzato nel circuito "main" per ricostruire le 7 stringhe da dare in input alle rispettive matrici LED.





Esempio di una possibile partita

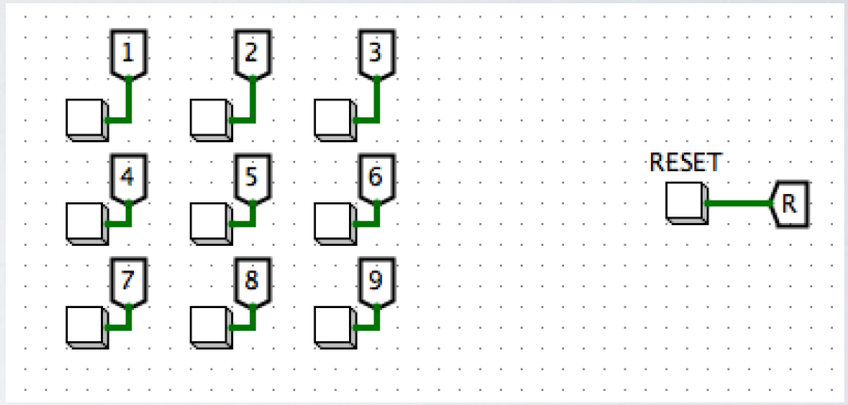
- In conclusione possiamo portare come esempio un ciclo completo di esecuzione della macchina, analizzando tutte le fasi che compongono un turno di gioco.

I° Fase : Il Giocatore fa la sua mossa

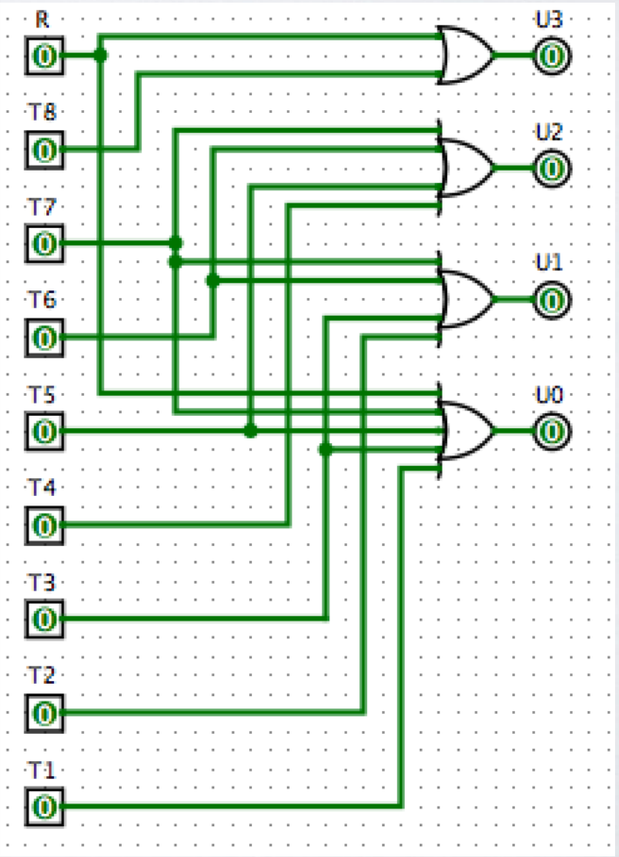
- Partendo dalla situazione iniziale , ovvero quella in cui il PC ha già posizionato il suo segno nella casella centrale, la macchina rimane nello stato iniziale finché il giocatore non avrà premuto un tasto sulla tastiera. Una volta premuto , come abbiamo già visto nella slide del Codifica Ingresso, verrà generata una stringa di 4 bit corrispondente al tasto premuto.

- Supponiamo ad esempio che il giocatore prema il tasto 2.

- A questo punto il circuito "Codifica Ingresso" produrrà in output la seguente stringa : "0010"



"Codifica Ingresso"

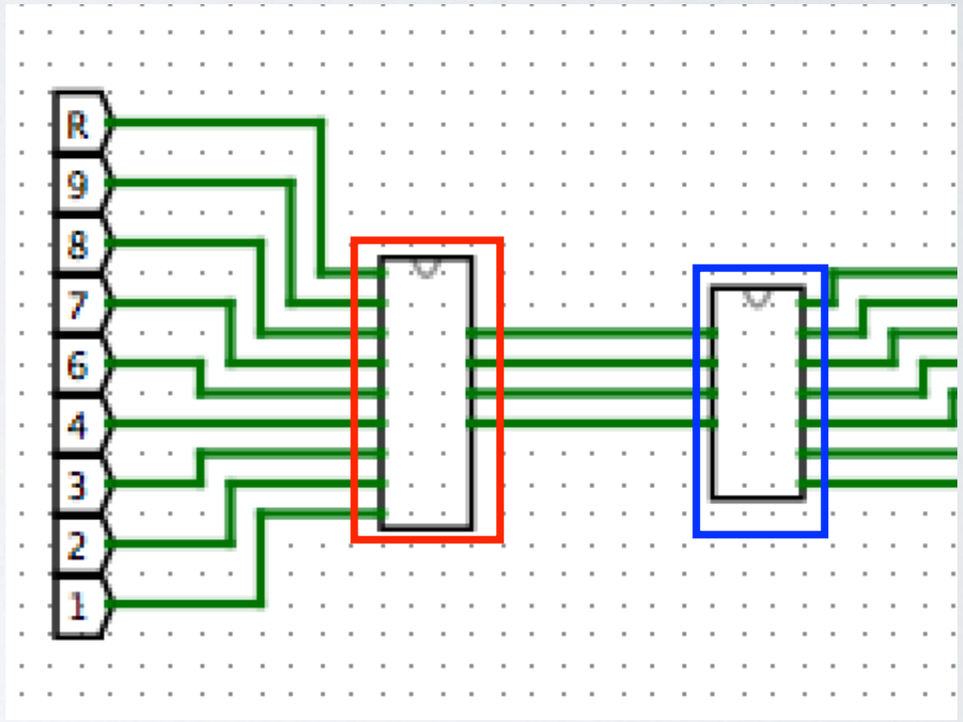


2° Fase : La MSF elabora la mossa

- A questo punto la stringa "0010" viene passata alla MSF che sulla base dello stato attuale , ovvero quello iniziale, e sull'input appena ricevuto elabora lo stato futuro.

- Dando un'occhiata alla STT scopriremo che lo stato futuro corrisponde al numero 1, quindi la MSF produce in output la stringa "0000001".

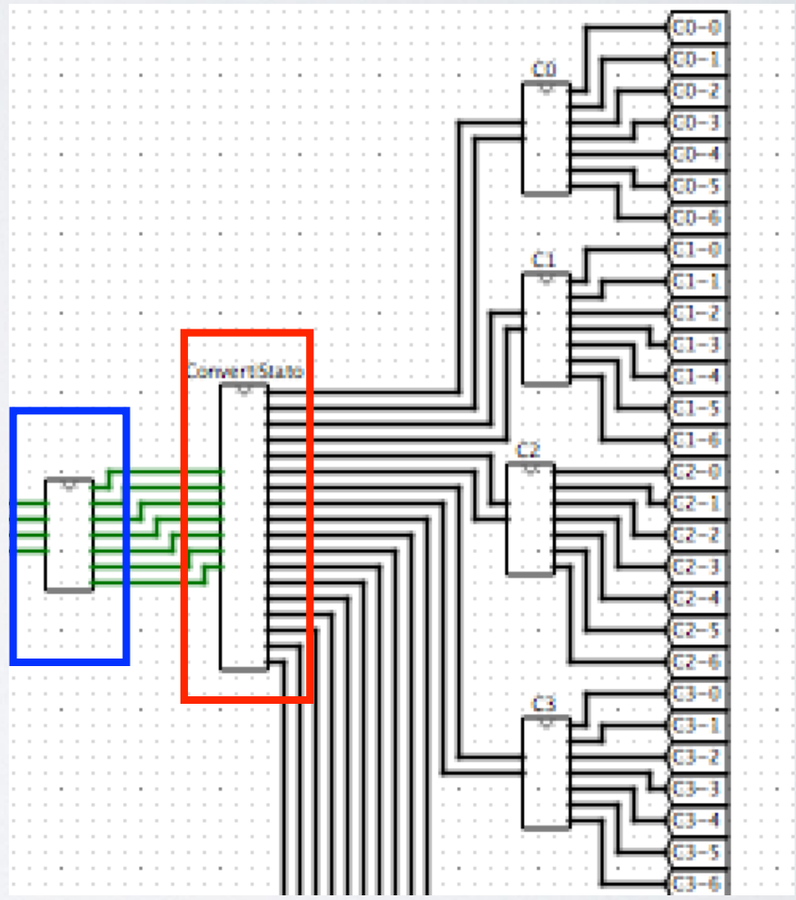
- Per vedere quali sia la sua corrispondente rappresentazione grafica sulla matrice LED dovremmo passare alla fase successiva ovvero quella di stampa dello stato futuro.



Codifica Ingresso MSF

3° Fase : Stampa dello Stato

- A questo punto la stringa "00000001" prodotta dalla MSF viene passata a "ConvertiStato". Essa elabora lo stato e produce 7 stringhe per ciascuna delle 9 matrici LED. Ogni stringhe coincide con la sequenza di bit necessari per accendere i LED della corrispondente riga della matrice.

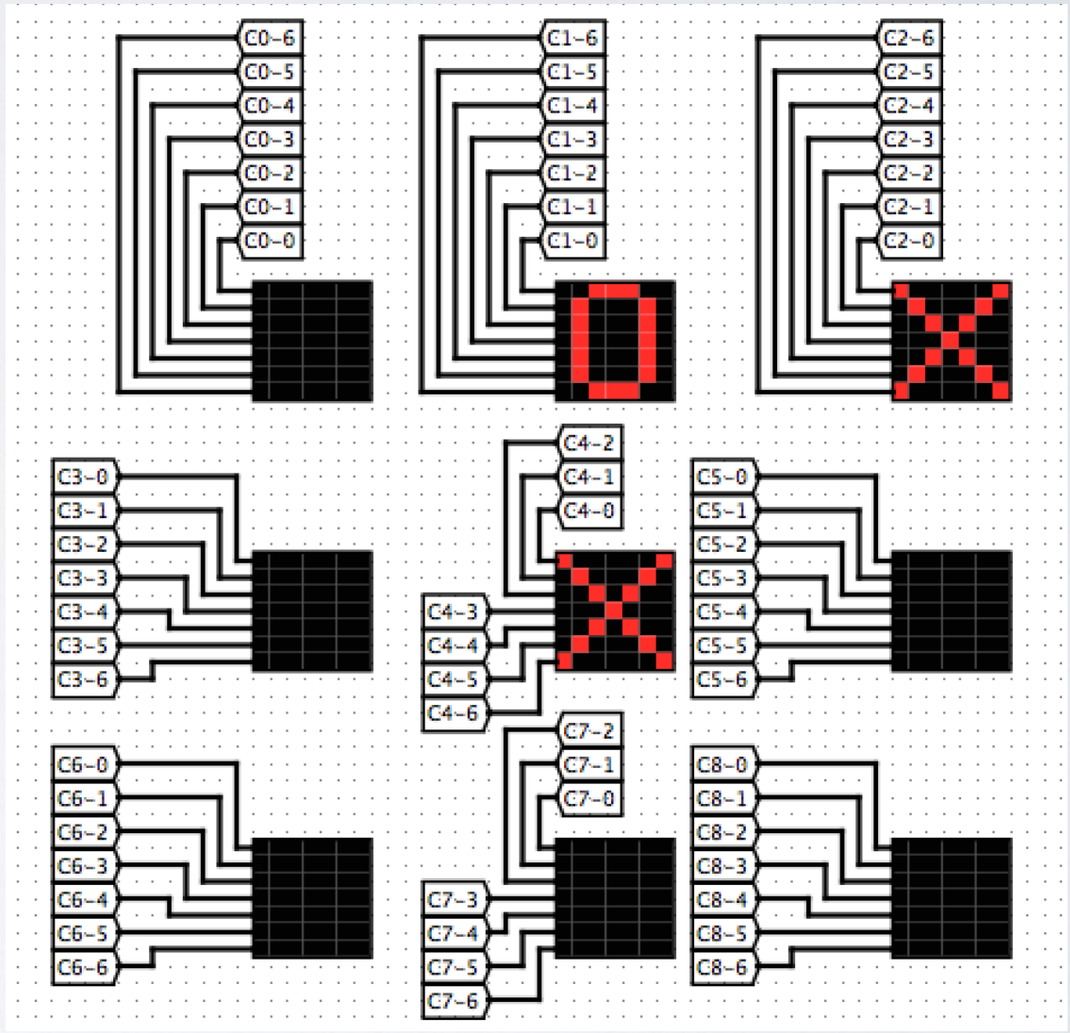


MSF

ConvertiStato

4° Fase :Visualizzazione dello Stato

- In definitiva lo stato viene rappresentato sulla matrice LED 3x3.
- Lo stato futuro, come si può notare , comprende già la mossa del PC.
- A questo punto la macchina rimane nel seguente stato, grazie alla memorizzazione dello stato dei Flip flop della MSF, finché il giocatore non farà la successiva mossa.
- La partita si conclude quando si raggiunge uno stato finale , ovvero una stato in cui l'unico modo per procedere è quello di cliccare il pulsante Reset e quindi ricominciare la partita.



Fine