Dots and Boxes: Un avversario "intelligente"

Roberto Falcone

r.falcone13@studenti.unisa.it

1 Introduzione

1.1 Contesto e Scopo del Progetto

Il presente report descrive il progetto finalizzato alla realizzazione di un algoritmo in grado di giocare in modo efficiente ad un gioco, con l'obiettivo di ottenere buone prestazioni contro un avversario umano. Il cuore dell'algoritmo è basato sulla tecnica di ricerca di alberi decisionali minmax e alpha-beta pruning. Questo approccio di intelligenza artificiale oggigiorno è ampiamente utilizzato per affrontare sfide riguardanti giochi e altre applicazioni che richiedono un ragionamento strategico ottimale.

1.2 La Rilevanza della Teoria dei Giochi

La teoria dei giochi è un campo fondamentale nell'intelligenza artificiale, in cui si studiano le strategie e i comportamenti di agenti razionali in situazioni competitive o cooperative. La teoria dei giochi offre un approccio matematico e formale per analizzare situazioni decisionali, identificare le migliori strategie e comprendere i risultati che emergono da interazioni complesse tra i giocatori. Questa teoria è di fondamentale importanza nello sviluppo di algoritmi intelligenti in quanto consente di modellare il comportamento razionale degli agenti e trovare le migliori azioni da intraprendere.

1.3 Applicazioni dei Giochi nei Problemi Complessi

I giochi non sono solo fonte di divertimento, ma costituiscono anche una preziosa piattaforma di studio per sviluppare e valutare algoritmi di intelligenza artificiale. La vasta gamma di giochi, dai deterministici a quelli non deterministici, da quelli perfettamente informati a quelli parzialmente informati, permette di affrontare una molteplicità di scenari che riflettono le sfide presenti in numerosi problemi reali.

1.4 La Scelta del Gioco

Come caso di studio per questo progetto è stato selezionato il gioco "Dots and Boxes", che offre una piattaforma semplice e allo stesso tempo ricca di complessità per testare l'efficacia dell'algoritmo di alpha-beta pruning.

2 Background

2.1 Dots and Boxes

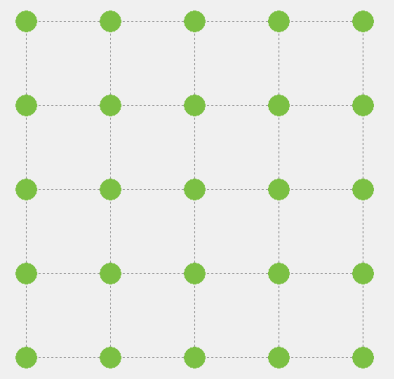


Figure 1: Tabellone di Dots and Boxes vuoto

"Dots and Boxes" è un gioco di carta e matita per due o più giocatori. Partendo da una griglia vuota, formata solo da punti, i giocatori possono, a turno, inserire una linea orizzontale o verticale tra due punti non ancora collegati. Quando uno dei due giocatori traccia il quarto lato di un quadrato ottiene un punto e marca il quadrato come conquistato. La partita termina quando non è più possibile disegnare linee all'interno della griglia e vince il giocatore che ha ottenuto più punti, ossia quello che ha conquistato più quadrati.

Non ci sono personaggi o oggetti in gioco, il che rende il gioco **astratto**. Poiché non ci sono elementi casuali o aleatori nel gioco, il risultato di ogni mossa è interamente prevedibile in base alla situazione attuale sulla griglia e alle azioni dei giocatori, perciò è **deterministico**. Ogni giocatore può vedere la griglia completa e le linee già disegnate. Ciò significa che entrambi i giocatori hanno una visione chiara e completa della situazione di gioco, il che influisce sulle loro decisioni tattiche e strategiche, da questo si evince che il gioco sia **ad informazione completa**.

Esistono due varianti di gioco: una prevede che, una volta completato un quadrato, il turno rimanga al giocatore che ha guadagnato il punto, che può quindi completare altri quadrati finché gli è possibile. L'altra invece, mantiene l'alternanza dei turni tra i giocatori, indipendentemente da chi conquista un punto durante la partita.

Si è scelto di prendere in analisi la prima variante perché ritenuta sicuramente più sfidante.

2.1.1 Il gioco nel dettaglio

La maggior parte delle argomentazioni di questa sezione deriva dagli studi del professore di matematica e computer science dell'Università della California, ***Elwyn Berlekamp***, che ha dedicato la propria carriera allo studio di giochi come "*Fox and Geese*", "*Go*" e, per l'appunto, "*Dots and Boxes*".

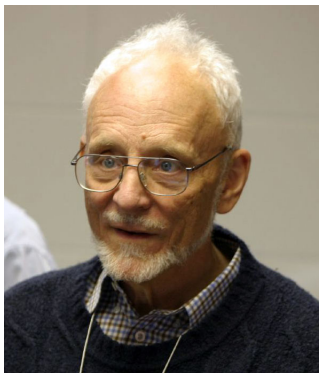


Figure 2: Elwyn Berlekamp

Il gioco, secondo il professore Berlekamp, ha un paio di meccaniche che lo rendono abbastanza unico e complesso da analizzare: la prima, la possibilità continuare a muovere una volta conquistato un quadrato: la seconda, i due giocatori hanno come obiettivo quello di fare più punti possibili e non di effettuare l'ultima mossa, tuttavia, come vedremo più avanti nel report, questi due obiettivi sono implicitamente collegati.

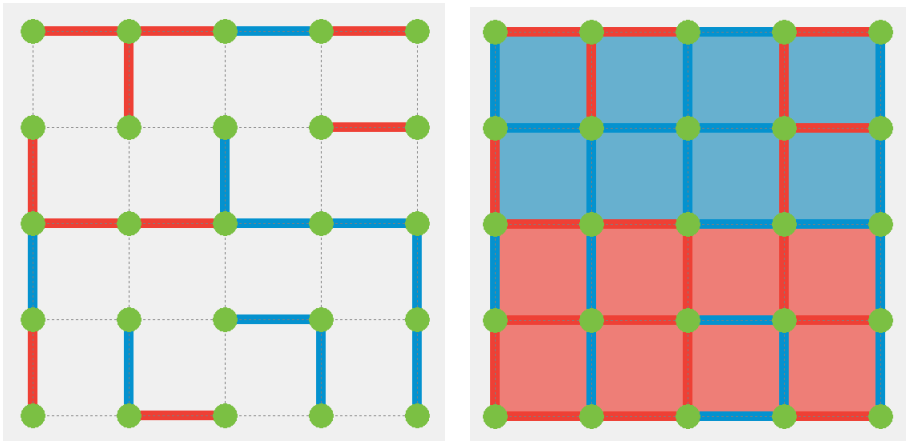
Prima di tutto c'è bisogno di dare la definizione di **catena**. Quando un giocatore completa un quadrato appartenente ad una catena, cattura quel quadrato e può continuare a catturare quadrati per tutta la lunghezza della catena. Le catene possono essere di lunghezza variabile, da una singola cella a una sequenza più lunga di quadrati adiacenti, e possono cambiare direzione o addirittura intersecarsi. L'obiettivo "nascosto" del gioco, come vedremo, è creare catene lunghe e complesse per catturare il massimo numero di caselle e segnare più punti rispetto all'avversario.

2.1.2 Le due fasi di gioco

La partita si divide in 2 fasi:

- La prima fase è la fase nella quale i giocatori cercano di creare catene ed evitare di regalare punti all'avversario (quindi marcando il terzo lato di un quadrato, mossa che porterà l'avversario a completare il quadrato stesso oltre che continuare con la consueta mossa).

- Durante la seconda fase invece i giocatori si occupano soltanto di completare le catene formate, cercando di aggiudicarsi quelle più lunghe.



(a) (b)

Figure 3: Figura (a) mostra la fine della prima fase; Figura (b) mostra la fine della seconda fase

Questa divisione in due fasi distinte non è facilmente riconoscibile da un giocatore alle prime armi.

La fase più "intuitiva" delle due risulta essere la prima. In questa fase l'obiettivo, agli occhi di un giocatore principiante, è cercare di non regalare punti all'avversario, segnando il terzo lato di un quadrato.

Le "buone maniere" in questa fase, sono quindi:

- Se possibile, segnare il quarto lato di un quadrato
- Evitare di segnare il terzo lato di un quadrato

Per un giocatore esperto invece, la fase più interessante è proprio la seconda, perché è proprio in vista di questa fase che quest'ultimo cerca di costruire più o meno catene lunghe.

2.1.3 Regola della catena lunga

Esiste una regola che, se sfruttata a dovere, può avvicinare alla vittoria il giocatore che la applica, ossia ***la regola della catena lunga***: supponiamo che il campo di gioco sia un rettangolo di m righe e n colonne e quindi sia composto da $m \cdot n$ quadrati. Se sia m che n sono pari, allora il primo giocatore dovrebbe cercare di rendere dispari il numero di catene lunghe. Se anche solo uno tra m o n è dispari, allora il primo giocatore dovrebbe cercare di rendere pari il numero di catene lunghe. Per catene lunghe si intende qualsiasi catena che superi la lunghezza di 3 quadrati.

Inoltre, in caso di una griglia con $m = n$, le due dimensioni dipendono strettamente dal numero di punti con i quali la si è costruita:

$$m = n = \#dots - 1$$

Quindi se il numero di punti è dispari allora m ed n saranno pari, così come se il numero di punti è pari allora m ed n saranno dispari.

Riassumendo formalmente il numero di catene lunghe desiderate è:

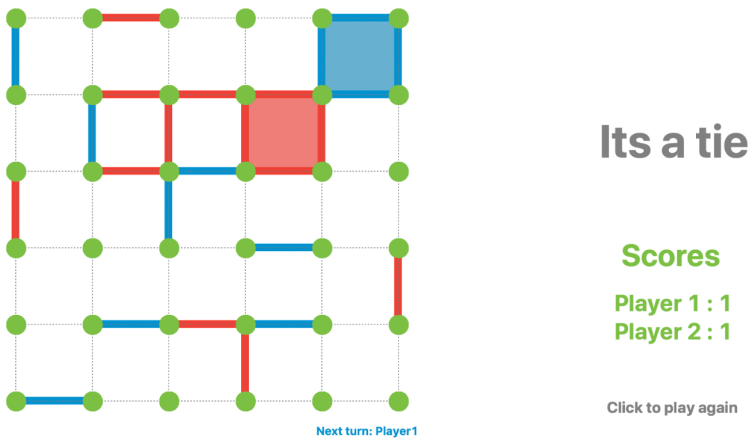
- PARI: per il primo giocatore se il numero di punti è pari, per il secondo giocatore se il numero di punti è dispari
- DISPARI: per il primo giocatore se il numero di punti è dispari, per il secondo giocatore se il numero di punti è pari

3 Implementazione

Il fine del presente progetto è stato l'ideazione e l'implementazione di un algoritmo che dimostri un'efficace abilità di gioco nel contesto specifico. L'algoritmo deve dimostrare la capacità di eseguire il gioco con precisione e competenza, evitando di cedere all'avversario vantaggi nel corso della fase di costruzione delle catene. Si richiede che l'algoritmo rispetti le convenzioni di gioco precedentemente delineate e che, al contempo, punti all'obiettivo di creare un numero ottimale di catene lunghe, avvicinandosi così alla vittoria della partita. Questi criteri sono dettagliatamente esposti nella sezione **2.1.2** e **2.1.3** del documento. Ovviamente, l'algoritmo deve infine dimostrare l'abilità di accumulare un quantitativo massimo di punti possibile.

3.1 GUI

Per la realizzazione dell'interfaccia grafica utente (GUI) nel contesto di questo progetto, è stata adottata la soluzione proposta da Aqueel Anwar, la quale è disponibile nella repository GitHub Dots-and-Boxes by aqueelanwar.



(a) (b)

Figure 4: Figura (a) mostra la GUI durante la fase di gioco; Figura (b) mostra come la GUI visualizza il vincitore della partita

L'interfaccia è molto semplice, permette al giocatore di cliccare sulla linea che vuole marcare e colora eventuali celle completate con il colore del giocatore che ha segnato il punto, blu per il primo giocatore, rosso per il secondo giocatore. Nel momento in cui tutte le celle sono state conquistate la GUI passa alla visualizzazione del risultato della partita, annunciando il vincitore e mostrando i punteggi di entrambi i giocatori. Per effettuare un'altra partita basterà cliccare ovunque sulla dashboard.

3.2 CPU solo alpha-beta

Il primo bot realizzato fa utilizzo soltanto dell'algoritmo di alpha-beta pruning. È implementato dalla classe E1_CPU() e per la valutazione di ogni mossa si limita a calcolare la differenza di punteggio tra il giocatore che sta giocando e l'avversario.

```
def evaluate_goodness(self, game, is_player1, player_1_stars):
    player1_score = len(np.argwhere(game.board_status == -4))
    player2_score = len(np.argwhere(game.board_status == +4))

    if is_player1:
        return player1_score - player2_score
    else:
        return player2_score - player1_score
```

Figure 5: Funzione per la valutazione di una mossa del primo bot

3.3 CPU alpha-beta e regola della catena lunga

Il secondo bot aggiunge all'algoritmo di alpha-beta pruning l'euristica della catena lunga, presentata nella sezione **2.1.3**. L'euristica è stata implementata pesando maggiormente le mosse che avrebbero avvicinato il giocatore ad un numero corretto di catene lunghe. Inoltre, è stato aggiunto un parametro CHAIN_POINTS per pesare correttamente l'influenza dell'euristica rispetto al comportamento iniziale del bot.

```
def evaluate_goodness(self, game, is_player1, player_1_stars):
    player1_score = len(np.argwhere(game.board_status == -4))
    player2_score = len(np.argwhere(game.board_status == +4))

    chain_points = 0

    chains = sorted(find_chains(game.board_status, game.row_status, game.col_status), key=len)
    even_points = sum(len(even_chain) for even_chain in chains[::2])
    odd_points = sum(len(odd_chain) for odd_chain in chains[1::2])

    if game.is_end_game() and check_no_4(game.board_status):
        if (is_player1 and player_1_stars) or (not is_player1 and not player_1_stars):
            chain_points = +odd_points - even_points
        else:
            chain_points = -odd_points + even_points

    if is_player1:
        return player1_score - player2_score + chain_points
    else:
        return player2_score - player1_score + chain_points
```

Figure 6: Funzione per la valutazione di una mossa del secondo bot

3.4 CPU con predizione di conquista delle catene

Il terzo bot invece cerca di predire quali catene verranno conquistate dai due giocatori. In questo caso alla differenza tra i punteggi dei giocatori vengono aggiunti i punteggi relativi al completamento di tutte le catene dispari o pari, in dipendenza da se il giocatore sarà il primo a iniziare il completamento delle catene o meno. Questo ragionamento si basa sul fatto che, una volta completata una catena, sia necessario cederne un'altra all'avversario (ovviamente lasciando a quest'ultimo le più piccole sulla griglia). Per l'implementazione infatti, è bastato ordinare le catene all'interno di una lista per poi attribuire il punteggio di quelle in posizioni dispari al primo giocatore che effettuasse la mossa e, di conseguenza, il punteggio di quelle pari al giocatore avversario.

```
rob_falc
def __init__(self, max_depth=3, chain_points=1):
    self.MAX_DEPTH = max_depth
    self.CHAIN_POINTS = chain_points * (NUMBER_OF_DOTS - 1) * (NUMBER_OF_DOTS - 1)

rob_falc
def evaluate_goodness(self, game, is_player1, player_1_stars):
    player1_score = len(np.argwhere(game.board_status == -4))
    player2_score = len(np.argwhere(game.board_status == +4))

    chain_points = 0

    long_chains = find_long_chains(game.board_status, game.row_status, game.col_status)
    number_of_long_chains = len(long_chains)

    if check_single_4(game.board_status):

        if NUMBER_OF_DOTS % 2 == 1:
            if (is_player1 and player_1_stars) or (not is_player1 and not player_1_stars):
                if number_of_long_chains % 2 == 1:
                    chain_points = +self.CHAIN_POINTS
            else:
                if number_of_long_chains % 2 == 0:
                    chain_points = +self.CHAIN_POINTS
        else:
            if (is_player1 and player_1_stars) or (not is_player1 and not player_1_stars):
                if number_of_long_chains % 2 == 0:
                    chain_points = +self.CHAIN_POINTS

            else:
                if number_of_long_chains % 2 == 1:
                    chain_points = +self.CHAIN_POINTS

    if is_player1:
        return player1_score - player2_score + chain_points
    else:
        return player2_score - player1_score + chain_points
```

Figure 7: Funzione per la valutazione di una mossa del terzo bot

3.5 Selezione dei giocatori

All'interno della classe main è possibile selezionare la tipologia dei due giocatori: PvP mode (giocatore contro giocatore); PvB mode (giocatore contro bot); BvB mode (bot contro bot)

4 Testing

Sono stati effettuati dei test per valutare le prestazioni dell'algoritmo e dell'euristiche sviluppate al variare di:

- Δ (MD): Maximum-Depth. Parametro dell'algoritmo di alpha-beta pruning che definisce quanto in basso scendere all'interno dell'albero di ricerca delle mosse.
- ρ: chain points. Definisce il peso che viene dato alla regola delle catene lunghe per il bot **B2** nella prima fase di gioco.
- Γ: grid_size. Definisce la grandezza della griglia in termini di quanti quadrati è formata.

Inoltre è stato effettuato un test per valutare i tempi necessari all'algoritmo per prendere una decisione su quale mossa effettuare.

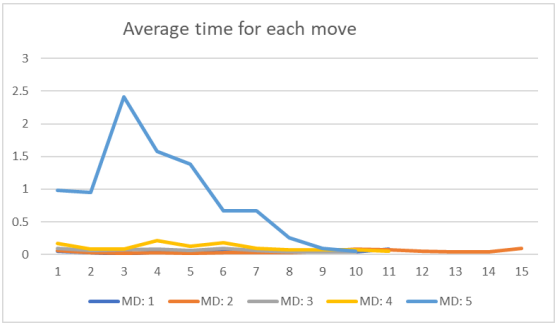
Infine, sono stati effettuati dei test facendo giocare i vari bot tra loro, allo scopo di capire se le euristiche aggiunte avessero un impatto sulle presetazioni dell'algoritmo.

4.1 B1 Δ - analisi temporale

È stata effettuata un'analisi temporale per capire quanto il primo bot impiegasse mediamente per effettuare una mossa, e come questo tempo cambiasse durante le varie fasi di gioco.

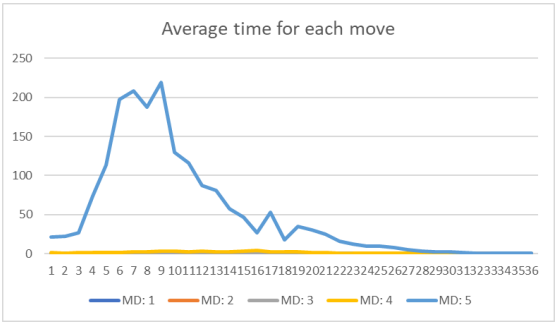
4.1.1 3x3 board

Δ: 1	**Δ: 2**	**Δ: 3**	**Δ: 4**	**Δ: 5**
0.0419	0.0442	0.0625	0.1102	0.9050



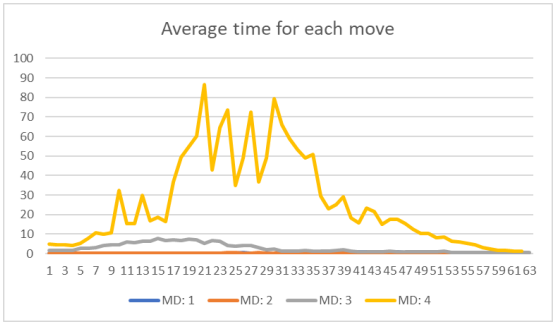
4.1.2 5x5 board

Δ: 1	Δ: 2	Δ: 3	Δ: 4	Δ: 5
0.1125	0.1281	0.3725	1.6655	51.3330



4.1.3 7x7 board

Δ: 1	Δ: 2	Δ: 3	Δ: 4
0.40417	0.37348	2.73431	25.9777



4.1.4 Considerazioni

I tempi di esecuzione per la scelta di una singola mossa sembrano essere abbastanza limitati per valori di $\Delta = \{1, 2, 3\}$ nel caso di tabelle piccole e medie, mentre iniziano a crescere per valori di $\Delta = \{4, 5\}$, sopratutto nel caso di tabelle grandi ($7x7$), caso in cui è impossibile effettuare test con $\Delta = 5$. Inoltre possiamo vedere come i tempi di esecuzione sembrano crescere con il proseguire della prima fase di gioco, per poi decrescere una volta che i giocatori iniziano il completamento delle catene (fattore che riduce notevolmente il numero di mosse da analizzare).

4.2 B1 Δ - analisi prestazionale

Γ	**Δ:1**	**Δ:2**
5x5	4	**96**

Γ	**Δ:2**	**Δ:3**
3x3	37	**63**
5x5	38	**62**

Γ	**Δ:3**	**Δ:4**
3x3	49	**51**
5X5	40	**60**

Γ	**Δ:4**	**Δ:5**
3X3	47	**53**
5X5	49	**51**

L'aumento di Δ comporta effettivamente delle migliorie sulle prestazioni dell'algoritmo nella totalità dei casi. Tuttavia, le differenze tendono ad assottigliarsi per valori di $\Delta = 3, 4, 5$ in base alla grandezza della griglia. Considerando anche il notevole aumento nei tempi di esecuzione relativi alla scelta di una singola mossa, è evidente che conviene mantenersi su valori di $\Delta = 3$ per griglie piccole e $\Delta = 4$ per griglie medie, per ottenere un migliore trade-off tra tempi di esecuzione e prestazioni.

4.3 B1 vs B2

Per valutare l'impatto dell'aggiunta dell'euristica sulle prestazioni dell'algoritmo i due bot **B1** e **B2** sono stati fatti giocare uno contro l'altro. Sono state valutate

le performance al variare del parametro ρ per **B2**, oltre al variare delle dimensioni della griglia come già detto in precedenza. Il parametro di max depth è stato invece fissato a $\Delta = 3$

Γ	**B1**	**B2(0.1)**
3x3	41	**59**
5x5	**52**	48

Γ	**B1**	**B2(0.2)**
3x3	50	50
5x5	**53**	47

Γ	**B1**	**B2(0.5)**
3x3	50	50
5x5	**51**	49

Γ	**B1**	**B2(1)**
3x3	50	50
5x5	45	**55**

Γ	**B1**	**B2(2)**
3x3	**51**	49
5x5	46	**54**

Dai risultati ottenuti possiamo notare come al crescere di ρ le performance di **B2** si avvicinino a quelle di **B1** per poi superarle leggermente, in casi di griglia media $(5x5)$, per valori di $\rho = 1, 2$, mentre nei casi di griglia piccola $(3x3)$ le performance sono quasi identiche.

4.4 B1 vs B3

Come nel caso precedete, per valutare le performance del bot **B3** lo si è fatto giocare contro il bot **B1** con fisso $\Delta = 3$.

Γ	**B1**	**B3**
3X3	**51**	49
5X5	46	**54**
7X7	47	**53**

Dai risultati è facile evincere come le prestazioni di **B3** superino quelle di **B1** per dimensioni crescenti della griglia di gioco.

4.5 B2 vs B3

Come test finale sono stati fatti giocare uno contro l'altro i bot **B2** e **B3**. Il valore di ρ per **B2** è stato fissato a 1.

Γ	**B2**	**B3**
3X3	42	**58**
5X5	47	**53**
7X7	**51**	49

Dalla tabella è facile notare come le prestazioni di **B3** siano leggermente maggiori rispetto quelle di **B2** al crescere della dimensione della griglia, mentre, per griglie piccole, i due bot sono quasi equivalenti. Questo probabilmente nasce dal fatto che B3 prende in considerazione anche i punti ottenuti da piccole catene (di lunghezza inferiore a 3 quadrati) che ovviamente risultano fondamentali per dimensioni ridotte della griglia.

5 Conclusione

Riassumento i risultati dei test, sia le modifiche aggiunte in **B2** che **B3** hanno portato ad miglioramenti per l'algoritmo di alpha-beta pruning presente in **B1**. Sono stati trovati i valori ottimali per i parametri Δ e ρ e analizzato le situazioni migliori per ognuno dei bot. Infine, valutando il comportamento dei bot contro avversari umani, è stato possibile notare come tutte le "buone regole" di gioco, illustrate nella sezione **2.1.2**, fossero rispettate.

6 Sviluppi futuri

In futuro si potrebbe pensare di inserire all'interno dell'algoritmo un modo per insegnare al bot come sacrificare celle per ottenere un punteggio maggiore. Esiste infatti una tecnica all'interno del gioco che permette a qualsiasi giocatore di vincere nella maggiorità delle situazioni. La tecnica, tanto più efficace quanto più e grande la board di gioco (e quindi le catene che vengono a formarsi), può essere applicata una volta che tutte le catene corte sono state conquistate (catene con lunghezza inferiore a 3 quadrati) e consiste nel sacrificare gli ultimi due quadrati di una catena che si sta conquistando, questo forzerà l'avversario a rivendicare quegli stessi quadrati (piuttosto che altre catene più lunghe) e cedere un'altra catena all'avversario (sicuramente più grande di 2 quadrati). Questa tecnica, se ripetuta ad ogni catena conquistata, porta sicuramente alla vittoria il giocatore che la utilizza, tuttavia, viene considerato dai giocatori un modo scorretto di giocare e quindi da evitare per non rovinare l'esperienza di gioco ai propri avversari.