# *Il gioco "Forza 4!"*

## *implementazione dell'algoritmo minimax in un ambiente grafico 3D*

Laureando: Francesco Andreussi
Relatore: Prof. Agostino Dovier

Università degli Studi di Udine

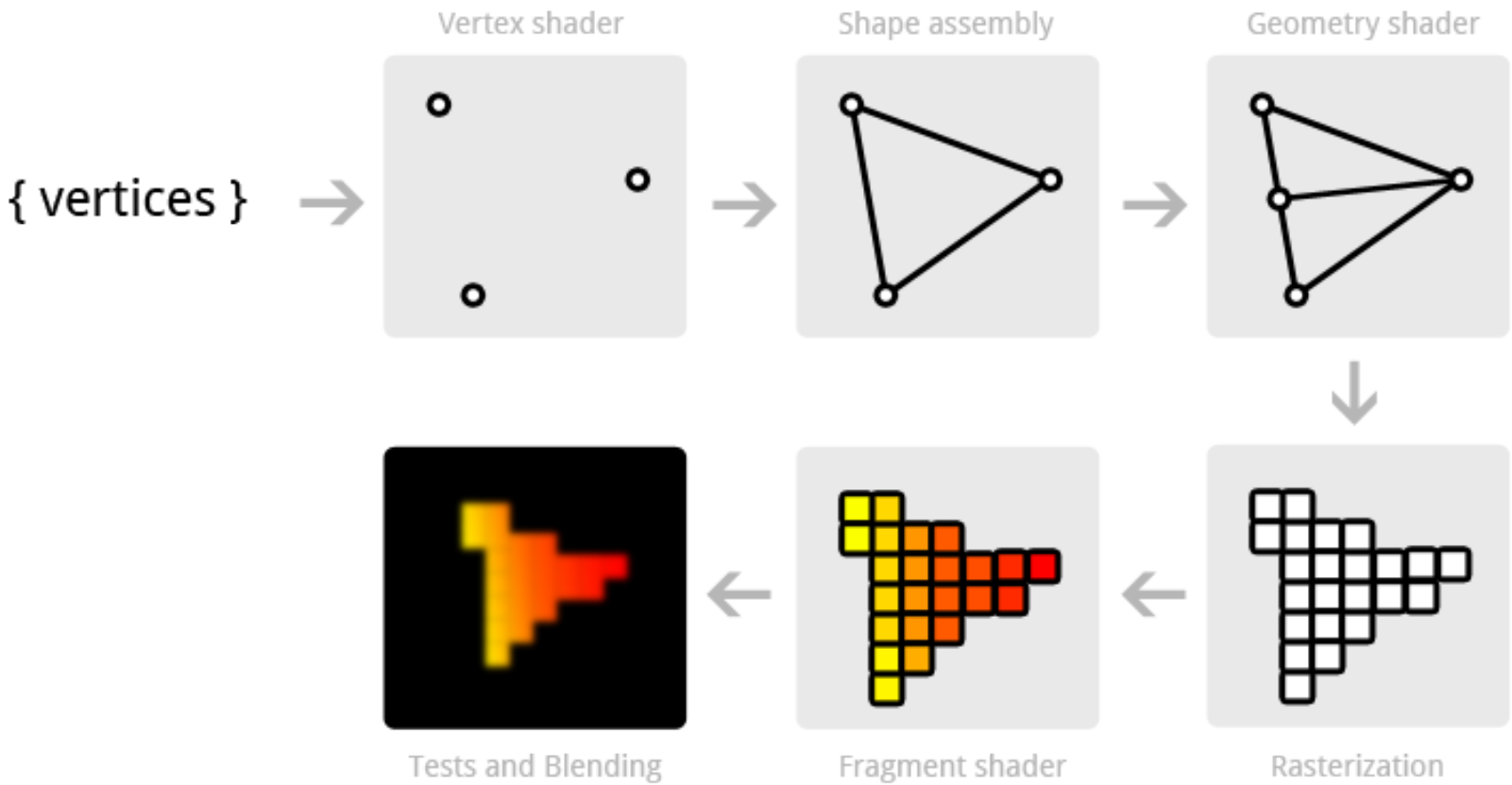
16 marzo 2017

# Indice

# Indice

# La Pipeline Grafica

Per poter creare un ambiente grafico interattivo bisogna scrivere un **ciclo di rendering**, che venga eseguito almeno 30 volte al secondo (meglio 60 o 120). Ogni esecuzione del ciclo si occuperà di visualizzare una specie di "istantanea" dell'ambiente (*frame*) che riprodotti sullo schermo velocemente creano, come nei film, l'impressione del movimento e più **fps** (frames per second) riesco a riprodurre più l'animazione sarà fluida e piacevole. Perciò le operazioni devono essere molto semplici altrimenti il processore grafico (GPU) non sarà in grado di eseguirle abbastanza velocemente.
Ad ogni ciclo di rendering vengono eseguiti gli step della cosiddetta **pipeline grafica**:


{ vertices }
Vertex shader
Shape assembly
Geometry shader
Rasterization
Fragment shader
Tests and Blending

# Geometry stage: assemblamento e posizionamento degli oggetti

Ogni oggetto 3D è rappresentato in memoria come un insieme di vertici.

# Geometry stage: assemblamento e posizionamento degli oggetti

Ogni oggetto 3D è rappresentato in memoria come un insieme di vertici.
Questi vengono processati dalla GPU che li "assembla" (*tassellation*) in triangoli che vanno a costituire la superficie dell'oggetto.

# Geometry stage: assemblamento e posizionamento degli oggetti

Ogni oggetto 3D è rappresentato in memoria come un insieme di vertici.
Questi vengono processati dalla GPU che li "assembla" (*tassellation*) in triangoli che vanno a costituire la superficie dell'oggetto. Ogni oggetto viene inserito in una struttura dati gerarchica ad albero che rappresenta la scena (lo *Scene Graph*) così, oltre alle sue proprietà (posizione, materiale...) specifiche, eredita quelle degli oggetti più in alto nella gerarchia.

# Geometry stage: assemblamento e posizionamento degli oggetti

Ogni oggetto 3D è rappresentato in memoria come un insieme di vertici.
Questi vengono processati dalla GPU che li "assembla" (*tassellation*) in triangoli che vanno a costituire la superficie dell'oggetto. Ogni oggetto viene inserito in una struttura dati gerarchica ad albero che rappresenta la scena (lo *Scene Graph*) così, oltre alle sue proprietà (posizione, materiale...) specifiche, eredita quelle degli oggetti più in alto nella gerarchia.
Il Geomerty Stage ritorna le superfici degli oggetti che sono inquadrati dalla camera in un certo momento.

# Rasterizer Stage e l'Equazione di Rendering

Il Rasterizer Stage ha il compito di assegnare ad ogni pixel dello schermo un colore.

# Rasterizer Stage e l'Equazione di Rendering

Il Rasterizer Stage ha il compito di assegnare ad ogni pixel dello schermo un colore.
Vengono identificati i pixel che "ricadono" in ogni triangolo, il loro colore si ottiene interpolando quelli dei tre vertici. Questo poi viene modificato calcolando gli effetti della luce: *(Fragment) Shading*.

# Rasterizer Stage e l'Equazione di Rendering

Il Rasterizer Stage ha il compito di assegnare ad ogni pixel dello schermo un colore.
Vengono identificati i pixel che "ricadono" in ogni triangolo, il loro colore si ottiene interpolando quelli dei tre vertici. Questo poi viene modificato calcolando gli effetti della luce: *(Fragment) Shading*.

## Equazione di Rendering

$$L_o(X, v) = \int_{||\mathbb{S}^2||} L_i(Y, w)|w \cdot n| f(w, v) dw$$

# Rasterizer Stage e l'Equazione di Rendering

Il Rasterizer Stage ha il compito di assegnare ad ogni pixel dello schermo un colore.
Vengono identificati i pixel che "ricadono" in ogni triangolo, il loro colore si ottiene interpolando quelli dei tre vertici. Questo poi viene modificato calcolando gli effetti della luce: *(Fragment) Shading*.

## Equazione di Rendering

$$L_o(X, v) = \int_{||\mathbb{S}^2||} L_i(Y, w)|w \cdot n| f(w, v) dw$$

## Equazione Semplificata

$$L_o(X, v) = A + \sum_{Y \in lights} |l \cdot n| f(l, v) \beta(Y, X)$$

# Sistemi Particellari e Postprocessing

*Sistemi Particellari* Si possono creare sistemi costituiti da oggetti molto semplici (*sistemi particellari*) tutti con le medesime caratteristiche e che si comportano coerentemente gli uni con gli altri.

# Sistemi Particellari e Postprocessing

*Sistemi Particellari* Si possono creare sistemi costituiti da oggetti molto semplici (*sistemi particellari*) tutti con le medesime caratteristiche e che si comportano coerentemente gli uni con gli altri.

*Postprocessing* Permette di applicare dei "filtri" alla scena. In pratica, l'immagine computata dalla pipline grafica viene passata ad un ulteriore shader il quale la modifica ad esempio cambiando i colori (invertendoli, mettendoli in scala di grigi o in toni seppia, saturandoli...) oppure sfocandola o ancora facendo risaltare maggiormente i bordi...le possibilità sono molte, basta che non venga appesantito troppo il ciclo di rendering!

# Indice

# Il Nucleo di Ragionamento

Ora passerò ad analizzare, invece, il cuore (o meglio il **cervello**) del progetto: il *nucleo di ragionamento*. Questo permetterà al software sviluppato di far sfidare un utente umano, oltre che con un altro utente, anche con il computer; il quale, specularmente, potrà anche giocare da solo. Il livello di bravura dei giocatori-computer è customizzabile e va da un minimo di 1 ad un massimo di 4 (su calcolatori sufficientemente performanti).

# La Modellazione con Linguaggi Dichiarativi

MiniZinc e Answer Set Programming, che sono linguaggi di modellazione (il primo a vincoli e il secondo logico). I modelli vengono risolti da programmi appositi (*solver*).
Questi strumenti permetto la risoluzione di problemi di soddisfacibilità e di **ottimizzazione** con vincoli (*CSP/COP*).

## Pro

- Efficienza
- Modularità
- Eleganza del formalismo

## Contro

# La Modellazione con Linguaggi Dichiarativi

MiniZinc e Answer Set Programming, che sono linguaggi di modellazione (il primo a vincoli e il secondo logico). I modelli vengono risolti da programmi appositi (*solver*).
Questi strumenti permetto la risoluzione di problemi di soddisfacibilità e di **ottimizzazione** con vincoli (*CSP/COP*).

## Pro

- Efficienza
- Modularità
- Eleganza del formalismo

## Contro

- Non si tiene conto che i due giocatori si ostacolino

# La Modellazione con Linguaggi Dichiarativi

MiniZinc e Answer Set Programming, che sono linguaggi di modellazione (il primo a vincoli e il secondo logico). I modelli vengono risolti da programmi appositi (*solver*).
Questi strumenti permetto la risoluzione di problemi di soddisfacibilità e di **ottimizzazione** con vincoli (*CSP/COP*).

## Pro

- Efficienza
- Modularità
- Eleganza del formalismo

## Contro

- Non si tiene conto che i due giocatori si ostacolino
- Non si possono gestire le strutture dati
- Il problema "esplode"!

# La Modellazione con Linguaggi Dichiarativi

MiniZinc e Answer Set Programming, che sono linguaggi di modellazione (il primo a vincoli e il secondo logico). I modelli vengono risolti da programmi appositi (*solver*).
Questi strumenti permetto la risoluzione di problemi di soddisfacibilità e di **ottimizzazione** con vincoli (*CSP/COP*).

**Pro**

- Efficienza
- Modularità
- Eleganza del formalismo

**Contro**

- Non si tiene conto che i due giocatori si ostacolino
- Non si possono gestire le strutture dati
- Il problema "esplode"!

Non adatti per il "Forza 4!"

# L'Algoritmo Minimax: cos'è e perché si usa

L'algoritmo minimax è stato pensato proprio per permettere di trovare la miglior **strategia** in giochi a due persone (*two-ply games*) ad informazione perfetta (scacchi, tria, dama, Forza 4...).

# L'Algoritmo Minimax: cos'è e perché si usa

L'algoritmo minimax è stato pensato proprio per permettere di trovare la miglior **strategia** in giochi a due persone (*two-ply games*) ad informazione perfetta (scacchi, tria, dama, Forza 4...). Bisogna fornirgli

- Uno stato iniziale

# L'Algoritmo Minimax: cos'è e perché si usa

L'algoritmo minimax è stato pensato proprio per permettere di trovare la miglior **strategia** in giochi a due persone (*two-ply games*) ad informazione perfetta (scacchi, tria, dama, Forza 4...). Bisogna fornirgli

- Uno stato iniziale
- Un insieme di mosse nel quale scegliere la migliore

# L'Algoritmo Minimax: cos'è e perché si usa

L'algoritmo minimax è stato pensato proprio per permettere di trovare la miglior **strategia** in giochi a due persone (*two-ply games*) ad informazione perfetta (scacchi, tria, dama, Forza 4...). Bisogna fornirgli

- Uno stato iniziale
- Un insieme di mosse nel quale scegliere la migliore
- Un'euristica (*utility/payoff function*) che permetta all'algoritmo di valutare quanto buona sia la situazione per un giocatore o per l'altro.

# L'Algoritmo Minimax: cos'è e perché si usa

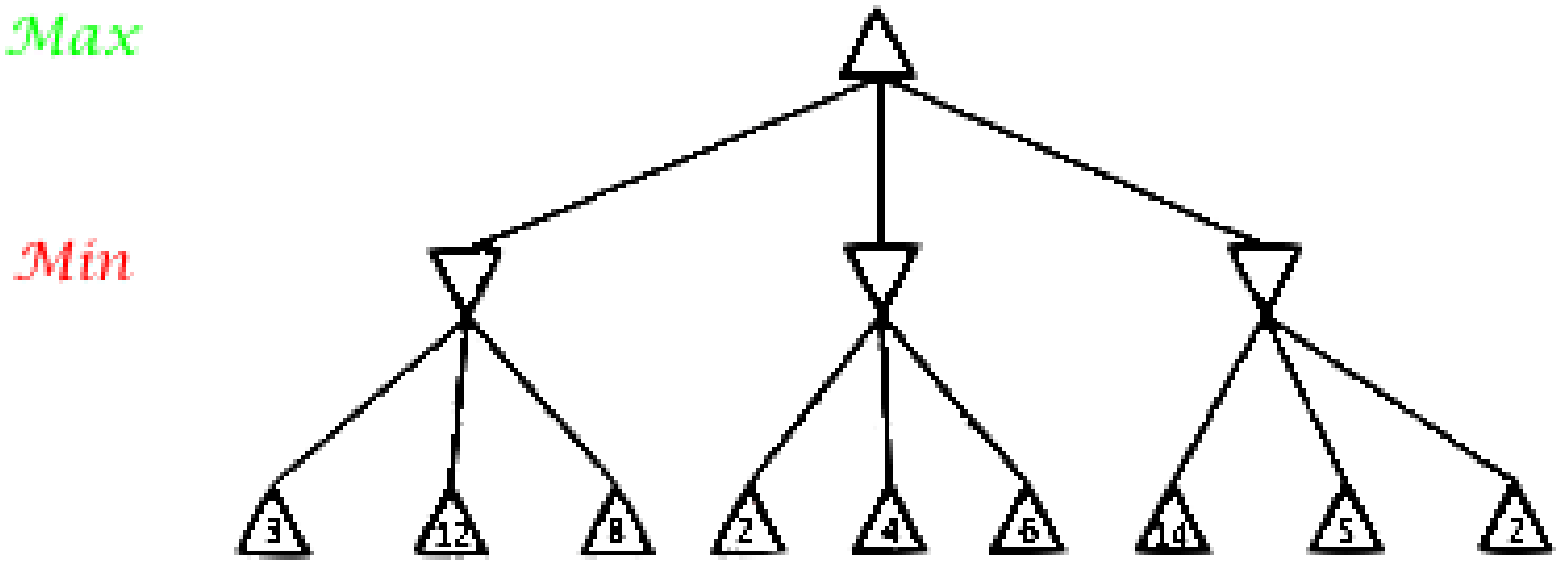
L'algoritmo minimax è stato pensato proprio per permettere di trovare la miglior **strategia** in giochi a due persone (*two-ply games*) ad informazione perfetta (scacchi, tria, dama, Forza 4...). Bisogna fornirgli

- Uno stato iniziale
- Un insieme di mosse nel quale scegliere la migliore
- Un'euristica (*utility/payoff function*) che permetta all'algoritmo di valutare quanto buona sia la situazione per un giocatore o per l'altro.

Per giochi "interessanti" scandire tutto l'albero di gioco diventa impensabile (per il Forza 4 avrebbe $7^{42}$ foglie e altrettanti nodi interni!) perciò siamo cotretti a limitare la nostra ricerca a pochi livelli.

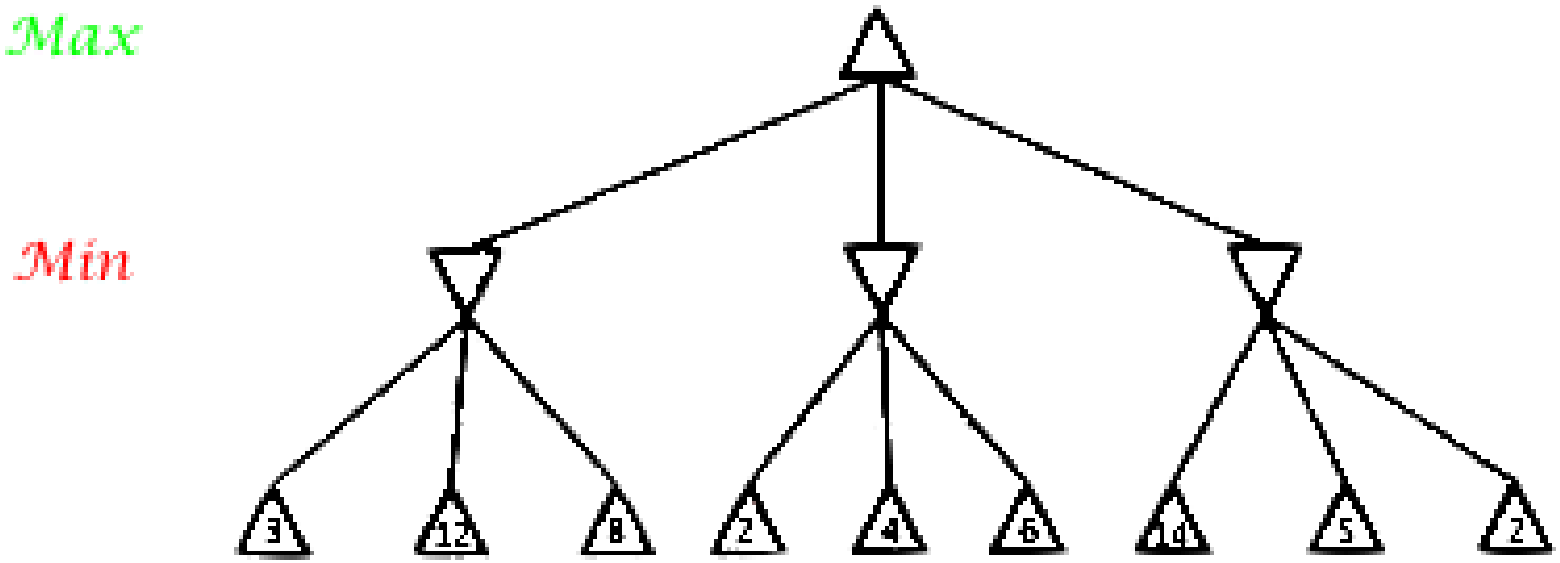
# L'Algoritmo e la sua Complessità

- Si genera l'albero di ricerca



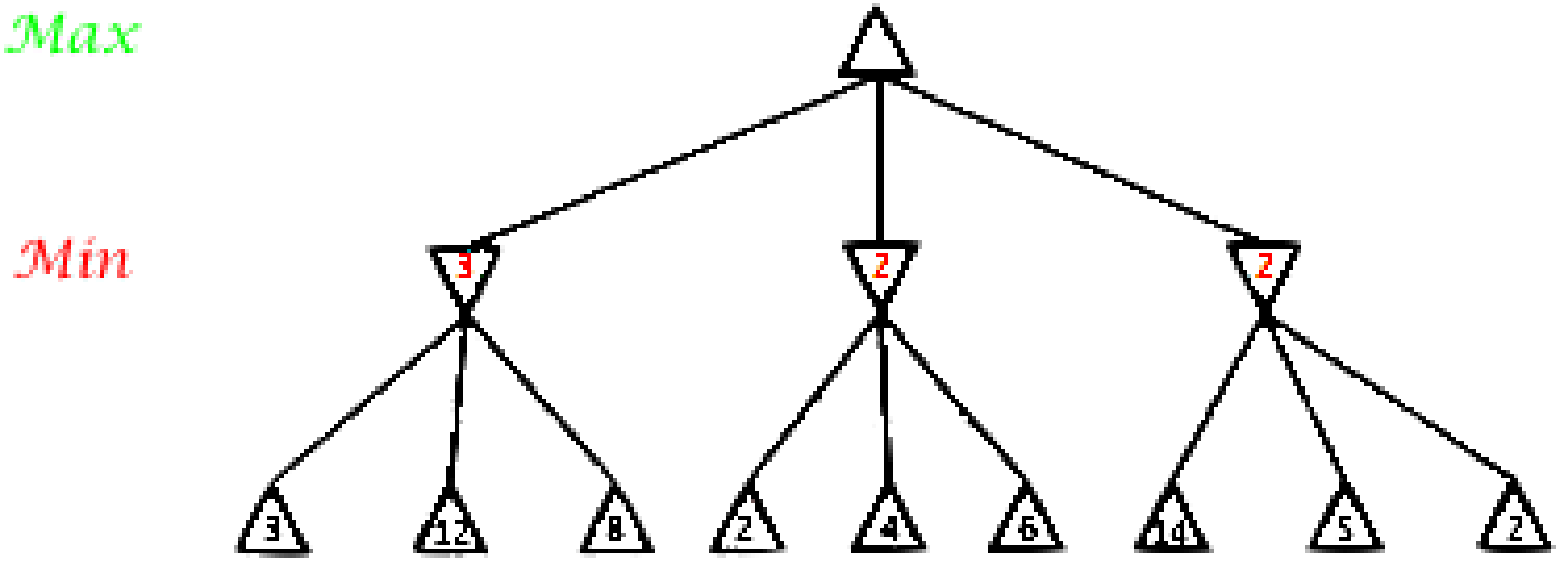
# L'Algoritmo e la sua Complessità

- Si genera l'albero di ricerca
- Si valutano le foglie con la funzione euristica



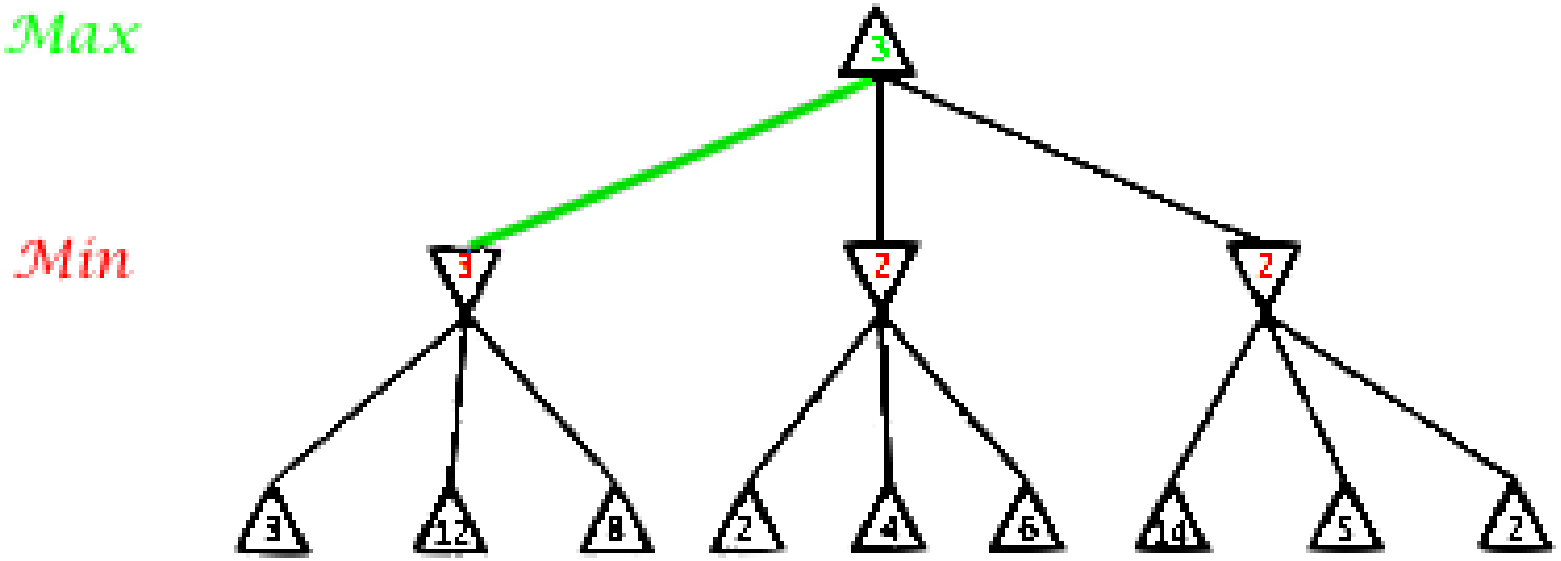
# L'Algoritmo e la sua Complessità

- Si genera l'albero di ricerca
- Si valutano le foglie con la funzione euristica
- Nei nodi *MIN* si sceglie il valore **minimo** fra quelli dei figli



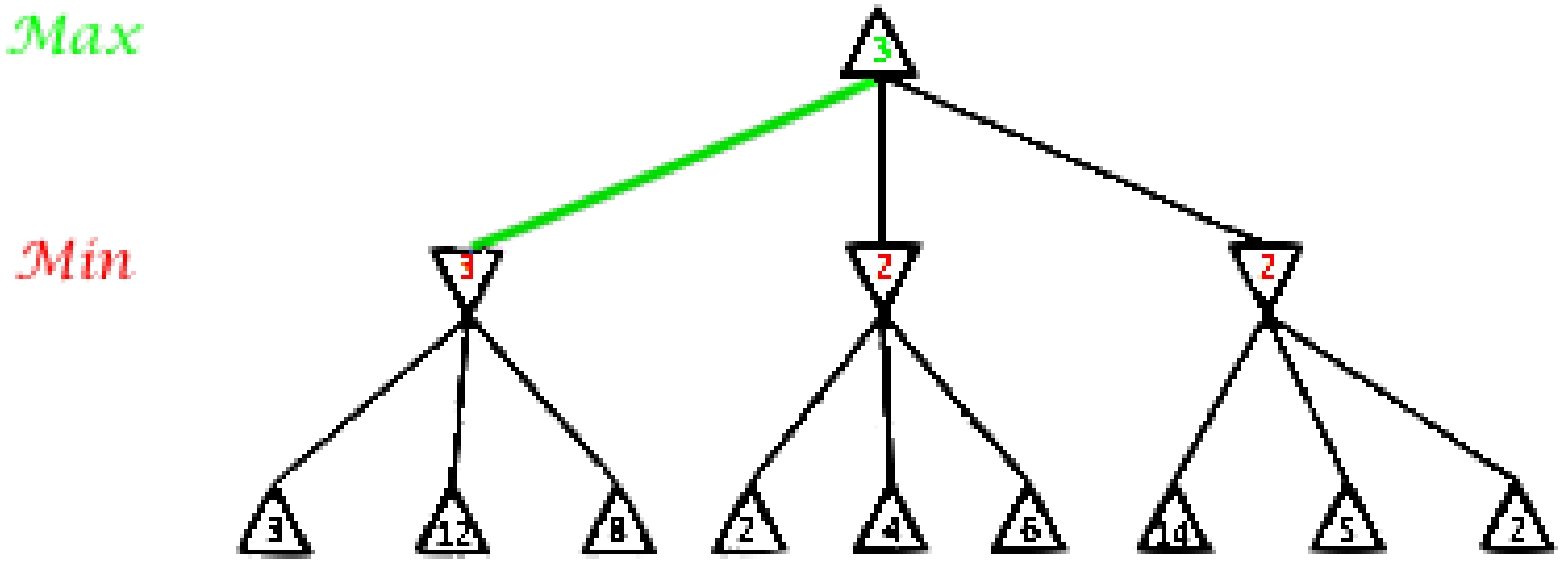
# L'Algoritmo e la sua Complessità

- Si genera l'albero di ricerca
- Si valutano le foglie con la funzione euristica
- Nei nodi *MIN* si sceglie il valore **minimo** fra quelli dei figli
- Nei nodi *MAX* si sceglie il valore **massimo** fra quelli dei figli



# L'Algoritmo e la sua Complessità

Complessità: $O(m^n)$ dove $m$ è il numero di possibili scelte che si hanno ad ogni livello (nel "Forza 4!" $m = 7$) e $n$ è il numero di livelli che si vogliono esplorare.



# Accorgimenti

Ho aggiunto un controllo iniziale che esegue l'algoritmo minimax su un solo livello (due mosse).
Questa modifica permette di:

# Accorgimenti

Ho aggiunto un controllo iniziale che esegue l'algoritmo minimax su un solo livello (due mosse).
Questa modifica permette di:

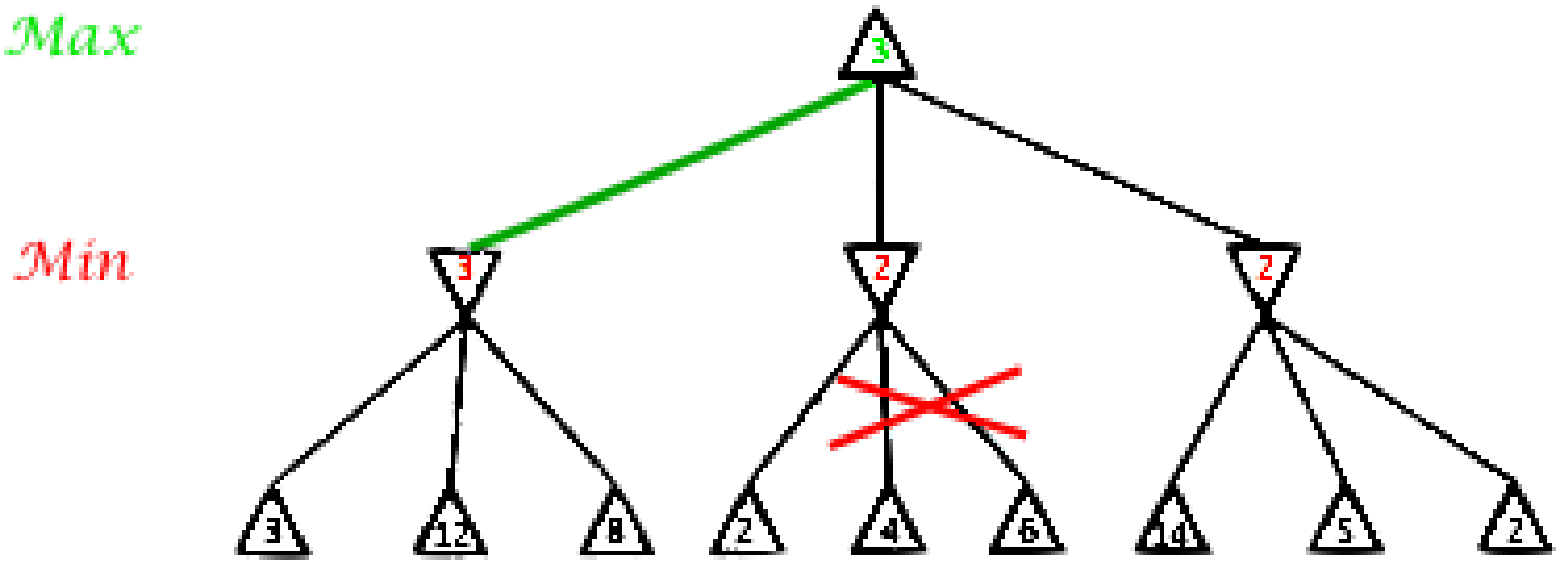
- Scovare sempre le "situazioni critiche", cioè una possibile vittoria o sconfitta in un passo e quindi permette di agire di conseguenza;

# Accorgimenti

Ho aggiunto un controllo iniziale che esegue l'algoritmo minimax su un solo livello (due mosse).
Questa modifica permette di:

- Scovare sempre le "situazioni critiche", cioè una possibile vittoria o sconfitta in un passo e quindi permette di agire di conseguenza;
- Evitare di esplorare un ramo quando la mossa da fare è individuabile già nel primo livello, cioè nei casi sopra citati.

# Possibili Miglioramenti




*Alfa-Beta Pruning* Permette di non visitare rami dell'albero il cui valore non sarà di sicuro scelto dai nodi superiori.

# Possibili Miglioramenti

*Alfa-Beta Pruning* Permette di non visitare rami dell'albero il cui valore non sarà di sicuro scelto dai nodi superiori.

*Approfondimento Progressivo* Si esegue il minimax su un numero crescente di livelli utilizzando le conoscenze apprese nelle esecuzioni precedenti per ordinare preliminarmente le mosse, così il minimax (con alfa-beta pruning) analizzerà prima le mosse migliori, scartando gran parte dei rami.

# Possibili Miglioramenti

*Alfa-Beta Pruning* Permette di non visitare rami dell'albero il cui valore non sarà di sicuro scelto dai nodi superiori.

*Approfondimento Progressivo* Si esegue il minimax su un numero crescente di livelli utilizzando le conoscenze apprese nelle esecuzioni precedenti per ordinare preliminarmente le mosse, così il minimax (con alfa-beta pruning) analizzerà prima le mosse migliori, scartando gran parte dei rami.

*Potatura Euristica* Una funzione euristica adeguata può permettere di "potare" l'albero in modo più sostanzioso anche se c'è il rischio di trascurare qualche buona combinazione e di non calcolare il valore minimax in modo corretto.

# *Indice*

# La Demo

Ora mostrerò il risultato del lavoro di tesi.
Vi chiedo solo ancora un attimo di pazienza...

Grazie dell'attenzione!