**PRIRODOSLOVNO-MATEMATIČKI FAKULTET**
**SVEUČILIŠTE U SPLITU**

Josip Ante Cindrić

# Minimax i Alpha-Beta Pruning algoritmi

Projekt

| | |
|---|---|
| **Studij:** | Preddiplomski studij |
| **Studijska grupa:** | Informatika |
| **Predmet**: | Algoritmi u primjeni |
| **Ak. god.**: | 2019/2020. |
| **Nastavnik**: | Prof. dr. sc. Divna Krpan |

Split, 2020.

# Sadržaj

# Uvod

Odabrani algoritmi Minimax i Alpha-Beta Pruning algoritmi svojstveni su području proučavanja umjetne inteligencije. Minimax je algoritam fokusiran na minimiziranje mogućeg gubitka u najgorem mogućem slučaju, a Alpha-Beta Pruning je nadogradnja nad postojećim Minimax algoritmom čiji je cilj smanjiti količinu rekurzivnih poziva u pretraživanju stabla. Za izlaganje ovih algoritama izgrađena je jednostavna igra „Križić-Kružić“ ili na engleskom „Tic-Tac-Toe“.

# Razrada

## Minimax algoritam

Generalno Minimax je algoritam koji je svojstven „zero-sum“ igrama tj. igrama koje imaju inverznu vrijednost evaluacije stanja za igraće između poteza gdje je formalna definicija $v_x + v_y = 0$ te gdje su $v_x$ i $v_y$ evaluacije stanja za igraće $x$ i $y$ u igri između poteza). Minimax vrijednost poteza u igri za igraća je najmanja vrijednost poteza kojeg protivnik može isforsirati igraća da poduzme. Formalna definicija Minimax vrijednosti za potez igraća je $\overline{v}_i = \min_{a_{-i}} \max_{a_i} v_i(a_i, a_{-i})$ gdje je $i$ igrač za kojeg primjenjujemo Minimax funkciju, $-i$ protivnik, $v_i$ vrijednost Minimax funkcije, $a_i$ potez igraća $i$ te $a_{-i}$ potez igraća $-i$. Prostorna i vremenska kompleksnost Minimax algoritma jednostavna je za razmotriti ako pretragu Minimax-a konceptualiziramo kao pretragu stabla gdje je svaka rekurzivna evaluacija novi čvor koji seže do ili proizvoljne dubine ili do završnog stanja igre. Dakle u najgorem slučaju svaki čvor ima $x$ moguće djece od kojih svako dijete ima jednako toliko vlastite djece do dubine $d$. Prema tome očito je da je u najgorem slučaju vremenska kompleksnost algoritma $O(x^d)$. Što se tiče prostorne kompleksnosti sagledati ćemo istu konceptualnu strukturu te je onda evidentno da je prostorna složenost $O(x * d)$.

## Alpha-Beta Pruning

Modifikacija Minimax algoritma sa Alpha-Beta Pruning algoritmom u prosjeku uvelike poboljšava prostornu i vremensku složenost. Poanta Alpha-Beta Pruning algoritma jest to da već postojeće rekurzivne pozive Minimax algoritma dopunimo sa evaluacijskim vrijednostima alfa i beta koji služe za eliminiranje potrebe za potpunim „backtracking-om“ stabla u povoljnim slučajevima evaluacije određenih čvorova. Konkretnije alfa i beta predstavljaju minimalnu vrijednost koja se osigurava igraču koji maksimizira te maksimalnu vrijednost koja se osigurava igraču koji minimizira. Inicijalno alfa poprima minimalnu vrijednost koju u teoriji jest minus beskonačno, a beta poprima suprotnu vrijednost od plus beskonačno tj. oba igrača počinju sa najgorim mogućim vrijednostima. Tijekom realizacije algoritma kada se maksimalna vrijednost rekurzivno vrati igraču koji minimizira te je ta vrijednost evaluirana kao povoljnija od minimalne vrijednosti igrača koji maksimizira algoritam zaključuje da igrač koji maksimizira ne treba razmatrati daljnje čvorove stabla u pretrazi tj. sljedeće moguće poteze. Vremenska i prostorna kompleksnost u najgorem slučaju su jednake kao kod Minimax algoritma, ali u najboljem slučaju je napredak ogroman sa Minimax-ovog $O(x^d)$ na $O(\sqrt{x^d}$¿ Alpha-Beta Pruning-a.

# Pseudokod realizacije algoritama

Realizacija Minimax algoritma vrlo je jednostavna u inačici pseudokoda. Prvi blok koda je provjera potrebe za prekidanjem rekurzije algoritma za slučaj da je došlo do kraja igre ili da je dostignuta željena dubina. Drugi koji je ujedno i posljednji blok je blok koji izvršava poziv evaluacije mogućih koraka u ovisnosti o kojemu se igraču radi.

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

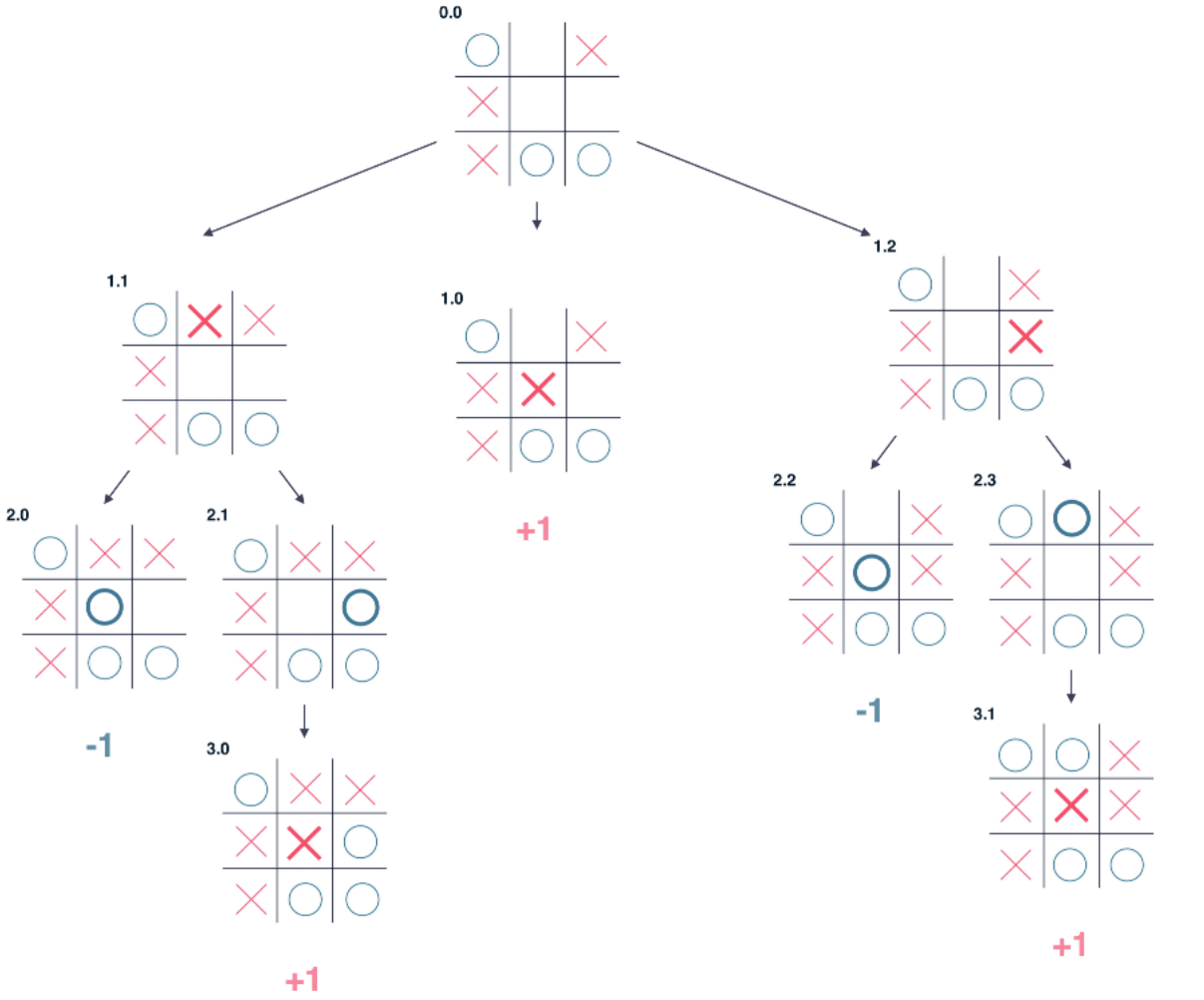
Realizacija Alpha-Beta Pruning algoritma se razlikuje samo po dodatnim vrijednostima alfa i beta koje se dodatno prosljeđuju u rekurzivne pozive te u samom njihovom korištenju. Naime u Minimax algoritmu nakon poziva evaluacije sljedećeg poteza slijedi nova iteracija petlje koja radi istu stvar, dok se u Alpha-Beta Pruning algoritmu u istoj iteraciji petlje uspoređuje maksimizirana tj. minimizirana vrijednost sa alfa ili beta vrijednostima te se u prikladnom slučaju prekida petlja te se vraća vrijednost evaluacije bez novih rekurzivnih poziva.

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β,
FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cut-off *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β,
TRUE))
            β := min(β, value)
            if β ≤ α then
                break (* α cut-off *)
        return value
```

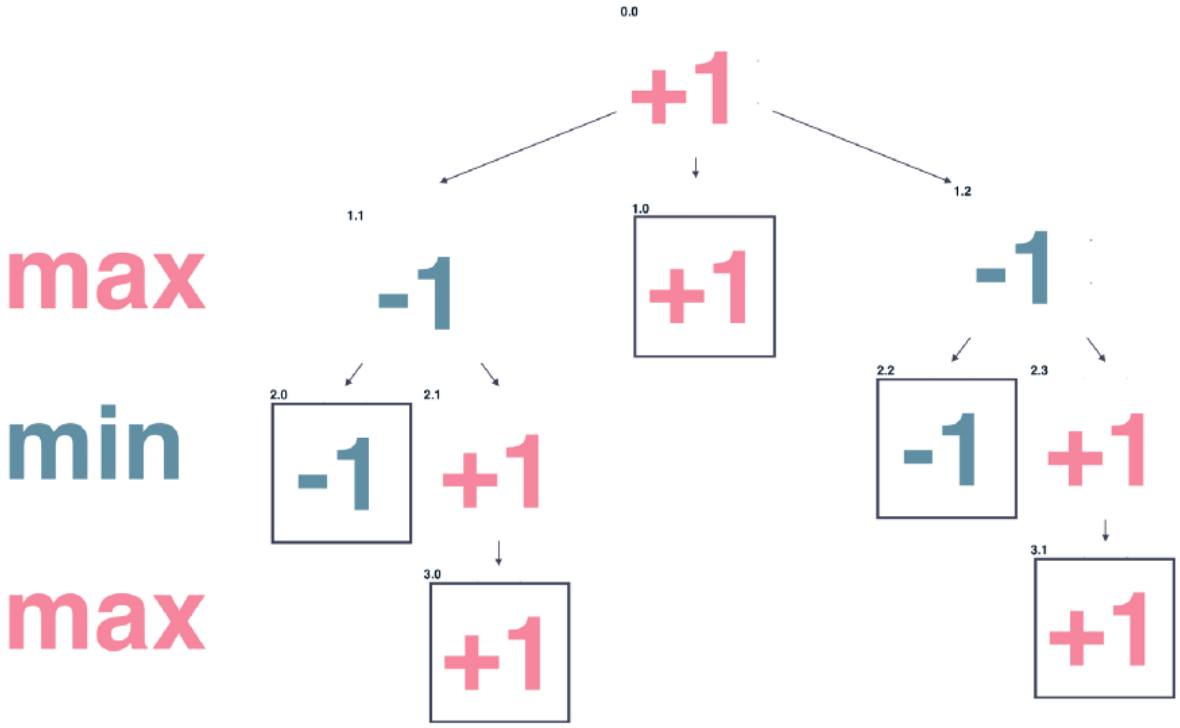
# Primjena Minimax-a i Alpha-Beta Pruning-a na Križić-Kružić

Križić-Kružić je igra za dva igraća kojima su dodijeljeni vlastiti simboli x ili O te se igra na 3x3 ploči gdje je uvjet pobjede postaviti tri vlastita simbola u redak, stupac ili dijagonalu. Igra može završiti i neriješeno ako se ne realizira pobjeda. Za razmatranje naših algoritama potrebno je uvidjeti da je broj mogućih poteza prije početka devet pošto polja na ploči ima točno devet, a sa svakim potezom broj polja se smanjuje za jedan. Time možemo vremensku kompleksnost najgoreg slučaja evaluirati na $O(x!)$ gdje je $x$=9 sa Minimax algoritmom, a u najboljem slučaju Alpha-Beta Pruning-a ona postaje O($\sqrt{x!}$¿ gdje je $x$=9.

U samoj realizaciji algoritama na Križić-Kružić njih možemo dodatno pojednostavniti time što ćemo samo provjeravati je li igra završena pošto želimo provjeriti stanja do krajnje dubine radi same jednostavnosti igre. Time smo eliminirali varijablu $d$ te je učinili konstantnom($d$=9). Još smo k tome evaluacijska stanja limitirali na elemente skupa {-1, 0, 1} prikladno krajnjim evaluacijama same igre.  Na primjeru sa slike ćemo prikazati konkretan rad algoritama.




Na slici vidimo stablo realizirano Minimax algoritmom. Rekurzivnim pozivima došli smo do krajnjih stanja. Na sljedećoj slici možemo vidjeti kako će se dobivena stanja rekurzivno vratiti na prve pozive Minimax funkcije.




Prema tome možemo vidjeti da nam je samo jedno stanje povoljno te je evidentno da će algoritam izabrati njega. Ovaj primjer je izrazito povoljan za prikaz rada Alpha-Beta Pruning algoritma. Ako razmotrimo obje slike slijed rekurzivnih poziva pratimo sa lijeva na desno te ako razmotrimo rad Alpha-Beta Pruning algoritma evidentno je da se kompletno stablo 1.2 neće izvršiti pošto smo našli zadovoljavajuće(pobjedničko tj. maksimizirano) stanje.

# Zaključak

Izabrali smo dva algoritma te ih formalno definirali i razmotrili. Prikladno smo prikazali na primjeru jednostavne igre sa konačnim krajem „Križić-Kružić“ te potvrdili premise postavljene u formalnim definicijama na konkretnim primjerima. U dijelu prakse ovog projekta uvidjeli smo konkretnu realizaciju u programskom jeziku C# te osigurali laku i sigurnu provjeru funkcionalnosti algoritama.

# Literatura


1. https://towardsdatascience.com/tic-tac-toe-creating-unbeatable-ai-with-minimax-algorithm-8af9e52c1e7d#14e6
2. https://en.wikipedia.org/wiki/Minimax
3. https://en.wikipedia.org/wiki/Tic-tac-toe
4. https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning