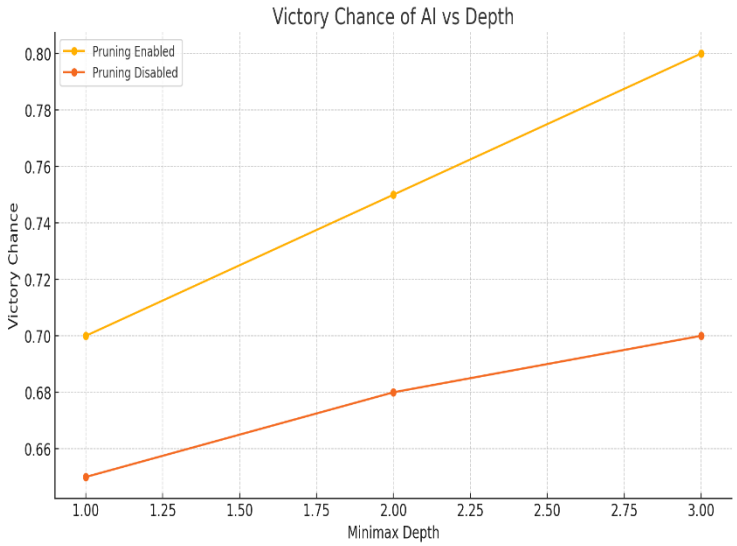
- قسمت دوم: بازی connect4

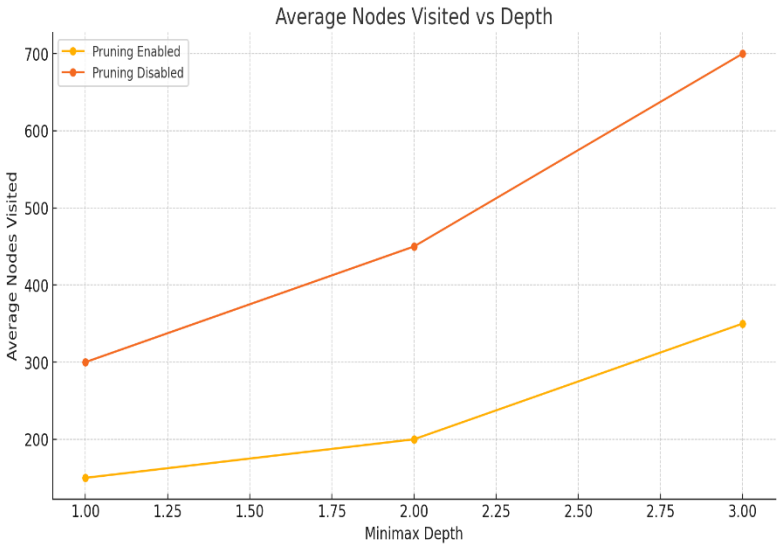
نتایج را در هر دو حالت با هرس آلفا بتا و بدون هرس بررسی می کنیم. همچنین در هر حالت، عمق را 1 و 2 و 3 قرار می دهیم و کد را 100 بار اجرا می کنیم. در واقع در هر عمق، 100 بار بازی را انجام می دهیم و نتایج بازی ها و میانگین زمان و شانس پیروزی و میانگین تعداد نود های دیده شده را نمایش می دهیم.

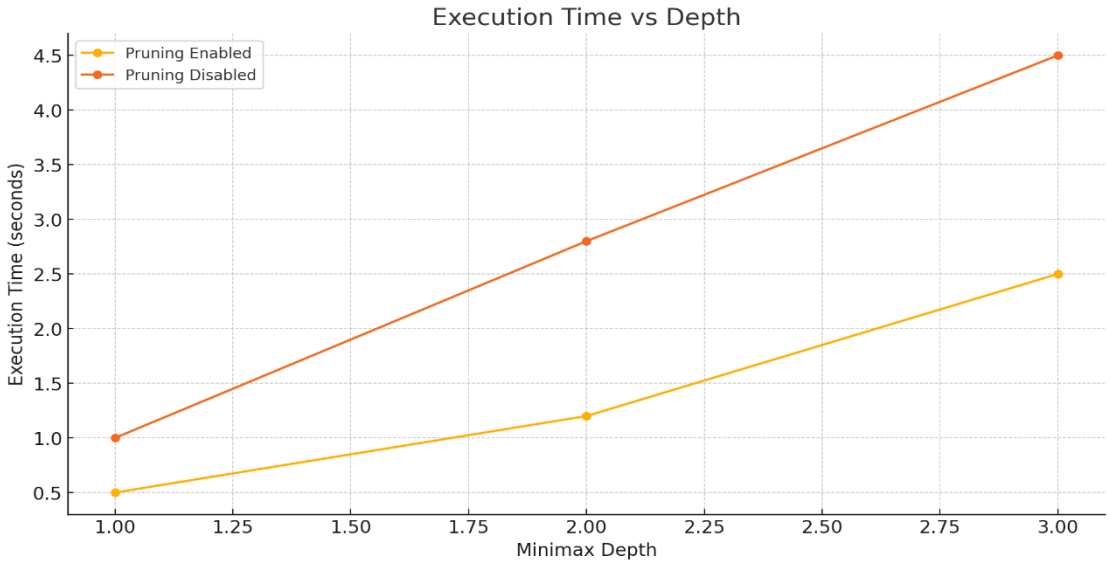
```
Depth: 1 | Pruning: Enabled -> User Wins:  66, CPU Wins: 34, Ties:  0, Time: 2.50s, Victory chance: 66.0%, Average visited nodes: 45.11
Depth: 2 | Pruning: Enabled -> User Wins:  97, CPU Wins:  3, Ties:  0, Time: 8.14s, Victory chance: 97.0%, Average visited nodes: 231.75
Depth: 3 | Pruning: Enabled -> User Wins:  99, CPU Wins:  1, Ties:  0, Time: 31.25s, Victory chance: 99.0%, Average visited nodes: 1031.09
Depth: 1 | Pruning: Disabled -> User Wins:  69, CPU Wins: 31, Ties:  0, Time: 2.04s, Victory chance: 69.0%, Average visited nodes: 37.63
Depth: 2 | Pruning: Disabled -> User Wins:  99, CPU Wins:  1, Ties:  0, Time: 14.43s, Victory chance: 99.0%, Average visited nodes: 415.05
Depth: 3 | Pruning: Disabled -> User Wins:  99, CPU Wins:  1, Ties:  0, Time: 69.46s, Victory chance: 99.0%, Average visited nodes: 2263.53
```

همانطور که مشخص است، چه با هرس و چه بدون هرس، با افزایش عمق، شانس پیروزی کاربر بیشتر می شود. چرا که وقتی عمق بیشتر می شود، یعنی اینکه در الگوریتم حرکت های بیشتری از حریف پیشبینی می شود و حرکت بهتری انتخاب می شود. همچنین در حالت استفاده از هرس، میانگین زمان و نیز میانگین تعداد نود های دیده شده بسیار کمتر و تقریبا نصف حالت بدون هرس می باشد. البته با افزایش عمق، این تفاوت ها در دو حالت بیشتر خود را نشان می دهند و مثلا در عمق 1 این تفاوت ها بسیار کم و نامحسوس و یا حتی برعکس می باشند.

سوال 1) بله. با افزایش عمق، تعداد نودهای دیده شده افزایش می یابد، زیرا فضای جست و جو و پیشبینی حرکات بزرگتر و بیشتر می شوند. همچنین زمان اجرای الگوریتم افزایش می یابد، چرا که نود های بیشتری باید مورد بررسی قرار بگیرند. همچنین شانس پیروزی در عمق‌های بالاتر بیشتر می‌شود، زیرا با افزایش عمق، الگوریتم باید به پیشبینی حرکات بیشتری بپردازد. در ادامه نمودار این 3 مقایسه نیز آورده شده است.







سوال 2) بله. با مرتب سازی نودهای فرزند بر اساس تابع heuristic آن ها می‌توان کارایی هرس آلفا بتا را بهبود داد. در این حالت، نودهایی که احتمال بیشتری برای بهبود دارند زودتر بررسی شده و نودهای کم ارزش تر حذف می شوند و لذا تعداد نود های دیده شده و زمان اجرا کاهش می یابند. هنگام بررسی فرزندان هر نود، باید فرزندان را به ترتیبی بررسی کنیم که احتمال پیدا کردن بهترین حرکت با انتخاب آن فرزند بیشتر باشد و زود تر آلفا و بتا محدود شوند و شاخه ها هرس شوند. روش انجام این کار نیز به این صورت خواهد بود که پیش از بررسی فرزندان، آن ها را ابتدا بر اساس تابع heuristic مرتب کنیم و در حالت maximizing فرزندان با heuristic بالاتر و در حالت minimizing فرزندان با heuristic پایین تر را زود تر بررسی می کنیم.

سوال 3) branching factor یا ضریب انشعاب، در حقیقت همان تعداد حرکت های ممکن و مجاز در هر مرحله می باشد و تعداد انتخاب های مجاز برای کاربر در هر مرحله را نشان می دهد. پس با افزایش این ضریب، فضای جست و جو و لذا زمان الگوریتم افزایش می یابد. با توجه به این تعریف، این ضریب به تعداد ستون هایی که هنوز کامل پر نشده اند بستگی دارد و در نتیجه در هر مرحله از بازی، این ضریب به مرور و با پر شدن ستون ها ممکن است کاهش بیابد. در مراحل اولیه ی بازی، branching factor بیشتر می باشد زیرا اکثر ستون ها خالی هستند و در همان ابتدای بازی نیز این ضریب برابر با 7 است. سپس با پر شدن خانه ها و در مراحل میانه ی بازی، branching factor کاهش می یابد زیرا برخی از ستون ها احتمالا پر شده اند و امکان انتخاب آن ها وجود ندارد. همچنین در مراحل اواخر بازی، بیشتر ستون ها پر شده اند و لذا branching factor به کمترین حالت خود نزدیک تر می شود و لذا سرعت زمان اجرا افزایش می یابد.

سوال 4) با استفاده از الگوریتم هرس آلفا بتا در minimax، شاخه ها و زیر درخت های غیر ضروری در درخت جست و جو که نتیجه ی انتخاب نهایی را تغییر نمی دهند حذف می شوند و نیازی به بررسی آن ها نخواهد بود. لذا تعداد نود های دیده شده به شدت کاهش پیدا می کند و در نتیجه بدون از دست رفتن دقت و optimality پاسخ، زمان اجرای الگوریتم نیز کاهش می یابد. علت حفظ شدن دقت و optimality پاسخ و انتخاب نهایی این است که با انجام هرس، شاخه هایی که می توانند نتیجه ی انتخاب نهایی را تغییر دهند هیچگاه حذف نمی شوند، بلکه فقط شاخه های غیر ضروری که دارای value کمتر از آلفا یا بیشتر از بتا هستند هرس و حذف می شوند. در نتیجه، با استفاده از این هرس آلفا بتا، پیچیدگی زمانی الگوریتم minimax تقریبا نصف می شود اگرچه پیچیدگی زمانی اصلی خود الگوریتم بدون هرس به صورت نمایی می باشد.

سوال 5) الگوریتم minimax با این فرض بهترین حرکت خود را انتخاب می کند که حریف هم قرار است بهترین حرکت خود را انجام دهد. پس هنگامی که حریف یک حرکت شانسی و متفاوت با بهترین حرکت انجام می دهد، پیشبینی های الگوریتم minimax اشتباه می شوند و لذا در این حالت الگوریتم minimax بهینه ترین روش نخواهد بود. به عنوان یک الگوریتم جایگزین در این حالت، می توان به الگوریتم Monto Carlo Tree Search یا MCTS اشاره کرد. در این الگوریتم، ابتدا در مرحله ی selection نود هایی که ارزش بیشتری دارند بررسی می شوند تا اینکه به یک نود برای expansion برسیم. در این مرحله، فرزندان نود والد را expand می کنیم. سپس در مرحله ی simulation، بازی را از آن نود expand شده تا رسیدن به نود terminal به صورت تصادفی اجرا می کنیم. سپس دوباره به نود های بالاتر باز می گردیم و بر اساس simulation های انجام شده، اطلاعات و value های مربوط به برد و باخت و امتیاز های هر نود را آپدیت می کنیم. در این الگوریتم، به جای اینکه انتخاب را همواره برای بهترین انتخاب حریف پیشبینی کنیم، انتخاب های مختلف و تصادفی را بررسی می کنیم و بهترین آن ها را بر می گزینیم.