Với mục tiêu giúp các bạn xây dựng trò chơi Dodgem, sau quá trình tìm tòi, tôi xin dịch và trích dẫn từ những tài liệu đã đọc qua, nhằm tóm lược lại một cách cô đọng nhất, giúp các bạn tiết kiệm thời gian, cũng như công sức trong qua trình làm trò chơi Dodgem, tôi xin giới thiệu tài liệu này với tiêu đề: ***Doing Dogem In 2 Hours.***

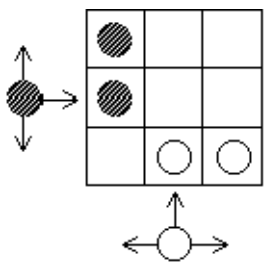
Mục lục:

Nội dung

Chương I: Giới thiệu Dodgem Game

Dodgem game, là một chess game giống như cờ caro, cờ tướng,… Mỗi game đều có 2 người chơi. Trong đó, một người được gọi là Max, người còn lại được gọi là Min. Hai người thay phiên nhau đưa ra các nước đi theo một quy luật nào đó.

Trò chơi Dodgen (được tạo ra bởi Colin Vout). Có hai quân Trắng và hai quân Đen, ban đầu được xếp vào bàn cờ 3*3 (Hình vẽ). Quân Đen có thể đi tới ô trống ở bên phải, ở trên hoặc ở dưới. Quân Trắng có thể đi tới trống ở bên trái, bên phải, ở trên. Quân Đen nếu ở cột ngoài cùng bên phải có thể đi ra khỏi bàn cờ, quân Trắng nếu ở hàng trên cùng có thể đi ra khỏi bàn cờ. Ai đưa hai quân của m.nh ra khỏi bàn cờ trước sẽ thắng, hoặc tạo ra t.nh thế bắt đối phương không đi được cũng sẽ thắng.



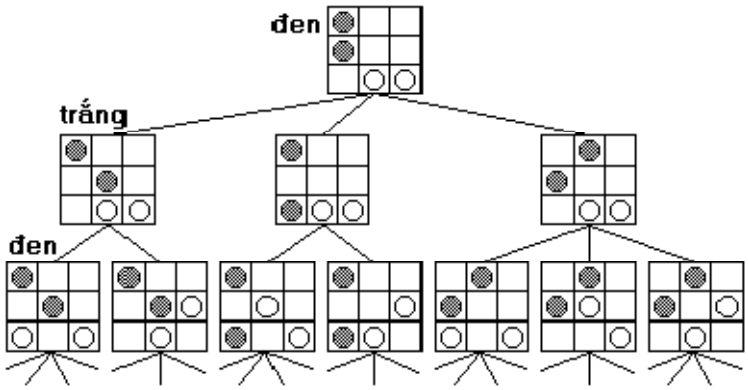
Vấn đề được đặt ra là: làm sao có thể lựa chọn được nước đi tối ưu nhất dành cho mình.

Chương II: Hàm đánh giá trong các thuật toán

Trong trò chơi Dodgem game có các thuật toán được sử dụng như: Minimax, Negamax, AlphaBeta…

Trong lần giới thiệu này tôi xin giới thiệu 2 thuật toán là Minimax và AlphaBeta, bạn đọc có thể tìm hiểu thêm thuật toán Negamax tại website:http://www.hamedahmadi.com/gametree/#minimax.

Các bạn theo dõi hình bên dưới



Như vậy để tìm nước đi tối ưu dành cho quân Đen, ta phải duyệt hết qua tất cả các đường đi có thể có của nó, và như thế ta sẽ tạo ra được một cây trò chơi như trên.

Như vậy, một câu hỏi được đặt ra là làm sao ta có thể lựa chọn được nước đi nào là thích hợp (dành cho người chơi là computer). Câu trả lời là ta cần phải xây dựng một hàm đánh giá như sau:

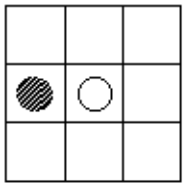
30	35	40
15	20	25
0	5	10

Giá trị quân Trắng.

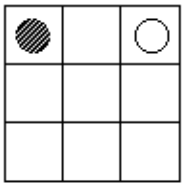
-10	-25	-40
-5	-20	-35
0	-15	-30

Giá trị quân Đen.

Ngoài ra, nếu quân trắng cản trực tiếp quân đen, **à** cộng thêm 40 điểm cho quân trắng, nếu cản gián tiếp thì được cộng 30 điểm, Tương tự cho quân đen, nếu quân Đen cản trực tiếp quân Trắng nó được thêm -40 điểm, cản gián tiếp nó được thêm -30 điểm.

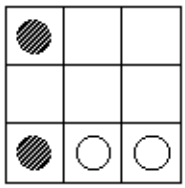


Trắng cản trực tiếp Đen được thêm 40 điểm.

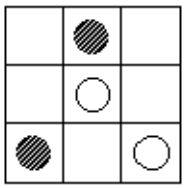


Trắng cản gián tiếp Đen được thêm 30 điểm.

Ví dụ, tính giá trị bàn cờ cho 2 hình vẽ bên dưới. Trong bàn cờ đầu, ta có các giá trị sau: Quân đen: -10 + 0 = -10, Quân trắng: 5 + 10 = 15, xét các trạng thái khác, ta thấy quân đen vừa bị cản trực tiếp, vừa bị cản gián tiếp, Quân trắng được thêm 70 điểm, Vậy KQ cuối cùng là: -10 + 15 +70 = 75, tương tự, cho hình số 2.



75



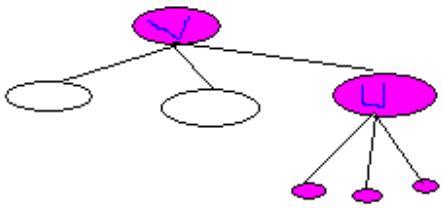
-5

Chương III: Xây dựng các thuật toán cho Dodgem Game

Như đã nói ở trên, trong lần giới thiệu này, tôi xin giới thiệu đến bạn 2 thuật toán là Minimax và AlphaBeta.

Thuật toán Minimax:

Nguyên lý: Nước đi tối ưu dành cho v, là nước đi duyệt qua tất cả các đỉnh con của u, là đỉnh tốt nhất trong số các đỉnh con của v.



Như vậy, vận dụng quy tắc trên, ta tìm hiểu hình bên dưới,

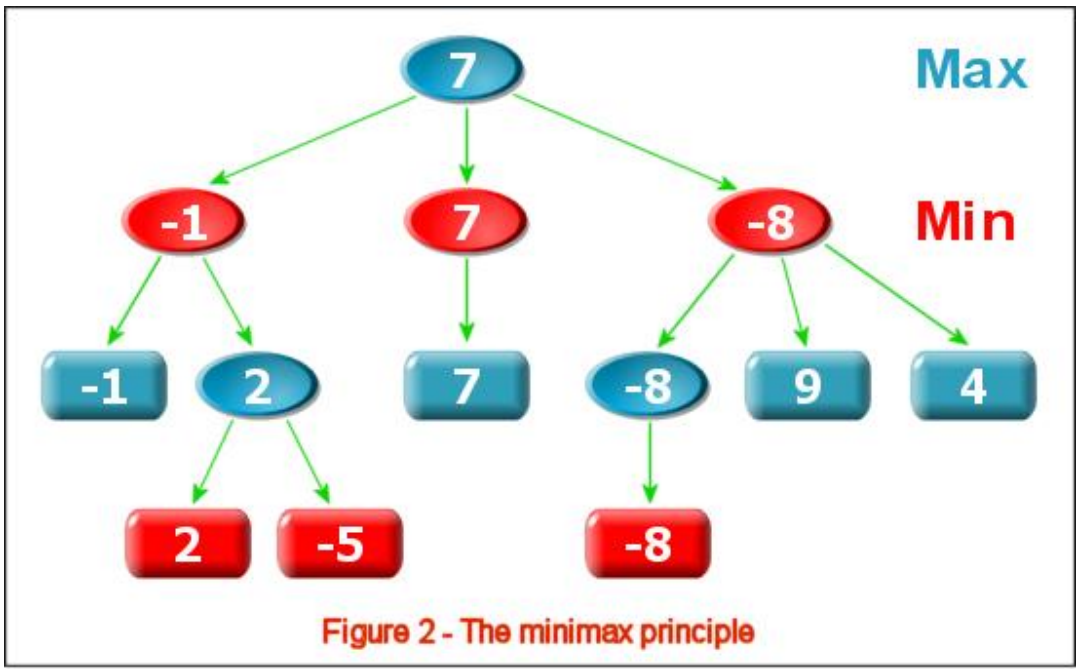


Figure 2 - The minimax principle

Với Blue (Quân Trắng), ta chọn Max(các children), Red (Quân Đen), ta tìm Min(các children). Từ hình vẽ trên, ta xây dựng các thủ tục đệ quy, gồm 2 hàm, BlueValue() đại diện cho Quân trắng, nhằm tìm nước đi tối ưu cho mình (Max(children)), ngược lại, RedValue() đại diện cho quân đen, tìm Miin(Children) cho mình.

```
int BlueValue(Board b, int depth)
{
      if ((GameOver(b) or depth>MaxDepth)
        return Analysis(b)
      int max = -infinity
      for each legal move m in board b
      {
            copy b to c
            make move m in board c
            int x = RedValue(c, depth+1)
            if (x>max) max = x
      }
      return max
}

int RedValue(Board b, int depth)
{
      if ((GameOver(b) or depth>MaxDepth)
      return Analysis(b)
      int min = infinity
      for each legal move m in board b
      {
            copy b to c
            make move m in board c
            int x = BlueValue(c, depth+1)
            if (x<min) min = x
```

```
        }
        return min
}
```

Trong đó, MaxDepth là độ sâu tối đa bạn duyệt đến, như vậy, duyệt càng sâu, bạn càng tiến gần đến thắng lợi. biến depth, được khởi tạo giá trị ban đầu là 0. Như vậy, tùy vào quân Trắng hay quân Đen mà bạn sẽ gọi hàm BlueValue() trước hay là RedValue() trước thông qua hàm Minimax() sau:

```
void Minimax(board u, int v)
{
        Val = - infinity;
        For Mỗi w là đỉnh con của U
        {
                If(val <= RedValue(w))
                {
                        Val = RedValue(w);
                        V = w;
                }
        }
}
```

Trong đó, v là đỉnh mà ta cần tìm (là đường đi tối ưu trong giới hạn nào đó).
Như vậy, cho đến bây giờ bạn đã có thể xây dựng thành công trò chơi Dodgame. Nhưng để trò chơi được diễn ra nhanh hơn, ta cần có các hàm nhằm loại bỏ những nước đi không cần thiết, vui lòng bạn xem thuật toán AlphaBeta bên dưới.

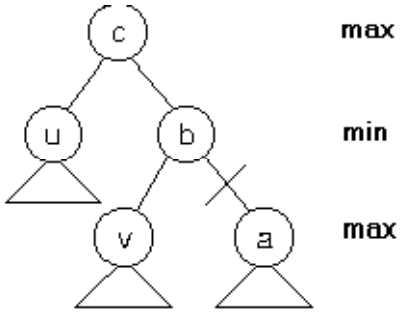
*Thuật toán AlphaBeta.

Trong chiến lược tìm kiếm Minimax, để tìm kiếm nước đi tốt cho Trắng tại trạng thái u, cho dù ta hạn chế không gian tìm kiếm trong phạm vi cây trò chơi gốc u với độ cao h, thì số đỉnh của cây trò chơi này cũng rất lớn với h >= 3.

Chẳng hạn, trong cờ vua, nhân tố nhánh trong cây trò chơi trung bình khoảng 35, thời gian đòi hỏi phải đưa ra nước đi là 150 giây, với thời gian này trên máy tính thông thường chương trình của bạn chỉ có thể xem xét các đỉnh trong độ sâu 3 hoặc 4.

Một người chơi cờ tr.nh độ trung bình cũng có thể tính trước được 5, 6 nước hoặc hơn nữa, và do đó chương trình của bạn mới đạt trình độ người mới tập chơi!

Khi đánh giá đỉnh u tới độ sâu h, một thuật toán Minimax đòi hỏi ta phải đánh giá tất cả các đỉnh của cây gốc u tới độ sâu h. Song ta có thể giảm bớt số đỉnh cần phải đánh giá mà vẫn không ảnh hưởng gì đến sự đánh giá đỉnh u. Phương pháp cắt cụt alpha-beta cho phép ta cắt bỏ các nhánh không cần thiết cho sự đánh giá đỉnh u.



Tư tưởng của kỹ thuật cắt cụt alpha-beta là như sau:

Nhớ lại rằng, chiến lược t.m kiếm Minimax là chiến lược tìm kiếm theo độ sâu.
Giả sử trong quá trình tìm kiếm ta đi xuống đỉnh a là đỉnh Trắng, đỉnh a có người anh em v đã được đánh giá.
Giả sử cha của đỉnh a là b và b có người anh em u d. được đánh giá, và giả sử cha của b là c (Xem hình trên).
Khi đó ta có giá trị đỉnh c (đỉnh Trắng) ít nhất là giá trị của u, giá trị của đỉnh b (đỉnh Đen) nhiều nhất là giá trị v.
Do đó, nếu eval(u) > eval(v), ta không cần đi xuống để đánh giá đỉnh a nữa mà vẫn không ảnh hưởng gì dến đánh giá đỉnh c. Hay nói cách khác ta có thể cắt bỏ cây con gốc a.
Lập luận tương tự cho trường hợp a là đỉnh Đen, trong trường hợp này nếu eval(u) < eval(v) ta cũng có thể cắt bỏ cây con gốc a.

Cụ thể hơn, ta có hình bên dưới

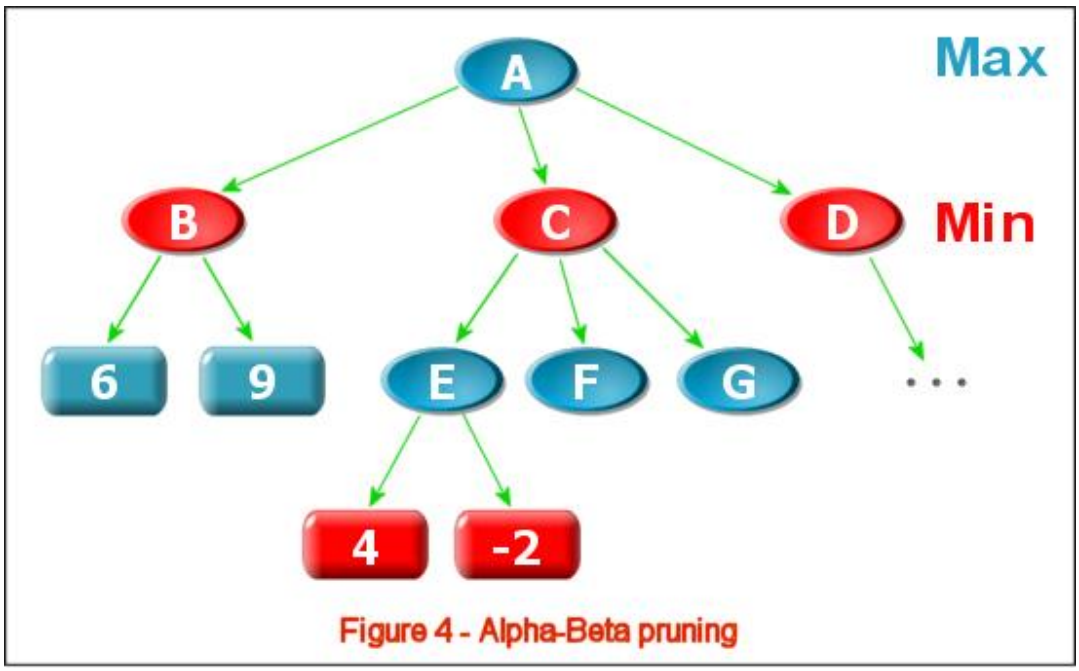


Figure 4 - Alpha-Beta pruning

Trong khi, ta đang cố tìm giá trị cho A, Chúng ta đi xuống B, và tìm được giá trị là 6 là Min(6,9). Sau đó, ta đi qua C. Từ đó, chúng ta tính được E là 4 là Max(4,-2). Do đó, C là nhỏ hơn,Ngay tại lúc này ta biết được giá trị của C là <=4, Do đó, C sẽ không được chọn bởi A là giá trị Maximum, vì A đã có B chứa giá trị 6, là giá trị chắc chắn lớn hơn bất kỳ giá trị nào của C (nếu ta tính tiếp). Do đó việc tìm giá tri Max cho F và G là không cần thiết nữa. Và lúc đó, ta sẽ dừng việc sử lý trên Node C, Và ta tiếp tục tìm trên Node D…

Như vậy, cho đến bây giờ ta có thể bổ sung kỹ thuật vừa nói trên vào thuật toán Minimax, ta có được chương trình hoàn thiện như sau:

Để cài đặt kỹ thuật cắt cụt alpha-beta, đối với các đỉnh nằm trên đường đi từ gốc tới đỉnh hiện thời, ta sử dụng tham số a để ghi lại giá trị lớn nhất trong các giá trị của các đỉnh con để đánh giá của một đỉnh Trắng, còn tham số b ghi lại giá trị nhỏ nhất trong các đỉnh con để đánh giá của một đỉnh Đen. Giá trị của a và b sẽ được cập nhật trong quá trình tìm kiếm. a và b được sử dụng như các biến địa phương trong các hàm MaxVal(u, a, b) (hàm xác định giá trị của đỉnh Trắng u) và Minval(u, a, b) (hàm xác định giá trị của đỉnh Đen u).

```
int BlueValue(Board b, int depth, int alpha, int beta)
{
    if ((GameOver(b) or depth>MaxDepth)
        return Analysis(b)
    int max = -infinity
    for each legal move m in board b
    {
        copy b to c
        make move m in board c
        int x = RedValue(c, depth+1, alpha, beta)
        if (x>max) max = x
        if (x>alpha) alpha = x
        if (alpha>=beta) return alpha
    }
    return max
}

int RedValue(Board b, int depth, int alpha, int beta)
{
    if ((GameOver(b) or depth>MaxDepth)
        return Analysis(b)
    int min = infinity
    for each legal move m in board b
    {
        copy b to c
        make move m in board c
        int x = BlueValue(c, depth+1, alpha, beta)
        if (x<min) min = x
        if (x<beta) beta = x
        if (alpha>=beta) return beta
    }
```

```
    return min
}
```

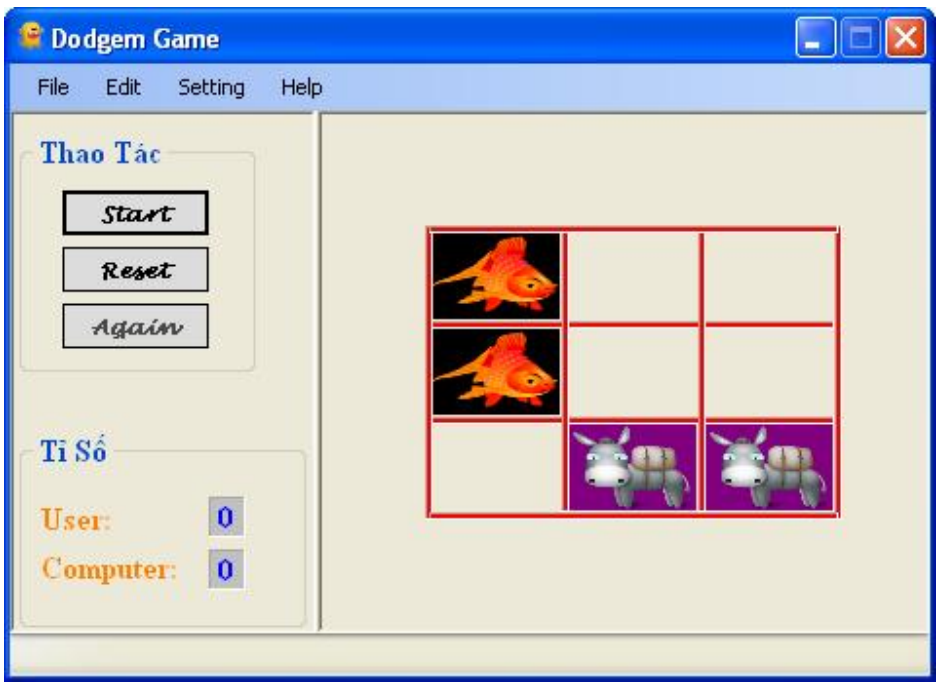
Lưu Ý: Chúng ta viết alpha>= beta thay vì alpha>beta, vì điều này làm tăng tốc độ cho chương trình. Các hàm này được khởi tạo các giá trị ban đầu như sau: depth=0; alpha=-infinity, beta=infinity.

Như vậy, tùy vào quân Trắng hay quân Đen mà bạn sẽ gọi hàm BlueValue() trước hay là RedValue() trước thông qua hàm AlphaBeta() sau:

```
Void Alpha_Beta(u, v)
{
Alpha = - infinity;
Beta = infinity;
For Mỗi đỉnh w là con của u
{
      If(val <= MinVal(w,depth,alpha,beta))
      {
            Val = MinVal(w,depth, alpha,beta);
            V=w;
      }
}
```

Trong đó, v là đường đi tối ưu cần tìm

Chương IV: Dodgem Game



*Đến đây, tôi xin tạm dừng nội dung, nếu có điều chi sơ sót, mong bạn đọc bỏ qua. Xin Cám Ơn.

References:

+ Giáo Trình Trí Tuệ Nhân Tạo – Trường ĐH SPKT TPHCM – Tháng 4-2003
+ Tài liệu kham khảo của tác giả Đinh Mạnh Tường
+ An Introduction to Game Tree Algorithms – Hamed Ahmadi