МОНГОЛ УЛСЫН ШИНЖЛЭХ УХААН,
ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
КОМПЬЮТЕРИЙН ТЕХНИК, МЕНЕЖМЕНТИЙН СУРГУУЛЬ

Лхагвадоржийн Идэртүвшин

ШАТРЫН ПРОГРАМ

Мэргэжлийн индекс: D480101
Мэргэжлийн нэр: Програм хангамж

Компьютерийн ухааны бакалаврын
зэрэг горилох бүтээл

УЛААНБААТАР 2005

**МОНГОЛ УЛСЫН ШИНЖЛЭХ УХААН,
ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
КОМПЬЮТЕРИЙН ТЕХНИК, МЕНЕЖМЕНТИЙН СУРГУУЛЬ**

Лхагвадоржийн Идэртүвшин

ШАТРЫН ПРОГРАМ

Мэргэжлийн индекс: D480101
Мэргэжлийн нэр: Програм хангамж

**Компьютерийн ухааны бакалаврын
зэрэг горилох бүтээл**

Эрдэм шинжилгээний удирдагч:
Магистр, багш Ц.Амарбат

Зөвлөгч:
Магистр, багш Ө.Сүх-Очир
Магистр, багш Б.Батзолбоо

Улаанбаатар 2005 он

Гарчиг

Оршил

Компьютер тоглоомын тухай

Тоглоом гэдэг ойлголт нь хүний бүхий л амьдралын туршид дагалдан хөгжиж байдаг. Ямар ч насны хүмүүс тоглоом тоглодог бөгөөд тоглоом нь төрөл бүрийн байдаг. Спортын , дайны , шатрын , шатрын төрлийн гэх мэт...

Техникийн хөгжил нь шинэ тоглоомын төрлийг бий болгосон. Энэ бол компьютер тоглоом юм.

Шатрын түүх

Шатар нь хүн төрөлхтний оюун ухааны гайхамшигт бүтээлүүдийн нэг билээ. Шатрыг хүн төрөлхтөн үүссэн цагаас нь эхлэн өргөнөөр тоглож ирсэн. Шатар нь хүний оюун ухааныг хөгжүүлэх , сэтгэн бодох чадварыг сайжруулдгаараа бусад тоглоомуудаас давуу талтай. Одоо үед шатар нь биеэ даасан шинжлэх ухаан , мөн спортын төрөл болтлоо хөгжжээ. Үүнийг дагаад шатрын програмчлал нь маш өндөр түвшинд хүрсэн. Үүний нэг тод жишээ бол шатрын програм шатрын дэлхийн аврага Г. Каспаровыг хожиж чадсан явдал юм.

Шатрын програмчлалын түүх

Хөлөгт тоглоомын програмчлал нь маш урт түүхтэй юм. Хамгийн анх компьютер зохиогдсоноос хойш тоглоомыг програмчлах оролдлого хийгдэж эхэлсэн. Иймээс энэ тал дээр нилээд туршлага хуримтлагджээ.

Анх 1950 – иад оны эхээр McCulloch , Turing хоёр шатрын програм бичиж эхэлсэн. 1952 онд Артур Самуэлийн бичсэн шатрын програм нь програм зөвхөн заасныг л хийдэг гэсэн үзлийг няцааж чадсан. Удалгүй зохиогчоо хождог болсон байна. 1956 онд анх шатрын програмыг телевизээр гаргасан нь хиймэл оюуныг нийтэд дэлгэн харуулсан анхны тохиолдол болжээ. Эдүгээ орчин үед компьютерын хүчин чадал сайжирч шатрын програм маш өндөр түвшинд хүрсэн. Хэдхэн жилийн өмнө шатрын програм (Deep Blue) дэлхийн аврага Г.

Каспаровыг хожиж шуугиан тарьж байсан. Харин одоо энэ бол нэг их гайхалтай сонин зүйл биш болжээ. Энэ нь хиймэл оюуны програмчлалын хөгжил , хүмүүс хиймэл оюуныг өдөр тутам хэрэглэдэг болсонтой холбоотой байх.

Хэрэглэгчийн шаардлага

Шатрын програм ямар байх ёстой вэ?

- Шатрын дүрмийн дагуу нүүдэл хийж тоглодог байх
- Нүүсэн нүүдлээ буцаж болдог
- Windows үйлдлийн систем дээр ажилладаг байх
- Ашиглахад хялбар ойлгомжтой байх
- Хөгжилтэй дуу дүрс тайлбартай байх

Архитектурын сонголт

Үйлдлийн системийн сонголт

Windows үйлдлийн системийг сонгож байна. Дэлхийд хамгийн түгээмэл ашиглагддаг хамгийн олон хэрэглэгчтэй ашиглахад хялбар шатрын програмын хувьд хамгийн тохиромжтой үйлдлийн систем юм.

Програмчлалын систем

Microsoft Visual Studio .Net

Програмчлалын хэлний сонголт

Visual C++ хэлийг ашиглаж хийх нь тохиромжтой.

Техник эдийн засгийн үндэслэл

Техник хангамж

Компьютер	:	1 ширхэг
Процессорын хурд	:	2000 MHz
Санах ойн хэмжээ	:	256 MByte
Хатуу дискний сул зай	:	10 MByte
Үйлдлийн систем	:	Windows 2000+

Програм хангамж

Уг програмыг Windows үйлдлийн систем дээр тулгуурлан Visual C# програмчлалын хэл дээр гүйцэтгэнэ. Visual C# нь бусад програмчлалын хэлнээс олон талын давуу талтай орчин үеийн програмчлалын хэл билээ. Мөн зураг боловсруулах програм шаардлагатай. Үүнд PhotoShop програм байхад хангалттай.

Шатрын дүрэм

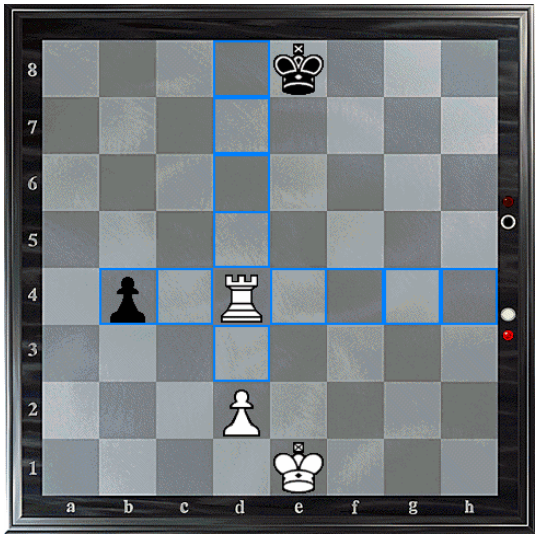
Шатрын хөлөг нь цагаан , хар өнгийн 64 буудлаас бүрдэнэ. Тоглогчийн баруун гар талын өнцөгт цагаан буудал байхаар хөлгийг байрлуулна. Хөлөг дээр тал тус бүр нэг ноён , нэг бэрс , хоёр тэмээ , хоёр морь , хоёр тэрэг , найман хүү бүгд 32 дүрс байрлана. Тоглогчдын гол зорилго бол эсрэг талын ноёныг мадлахад оршино. Энэ зорилгыг эхэлж биелүүлсэн нь хожно. Зорилгодоо хүрэхийн тулд эсрэг талын бод (бэрс , тэрэг , морь , тэмээ) , хүүг устгаж тоглоно.

Нүүдэл :

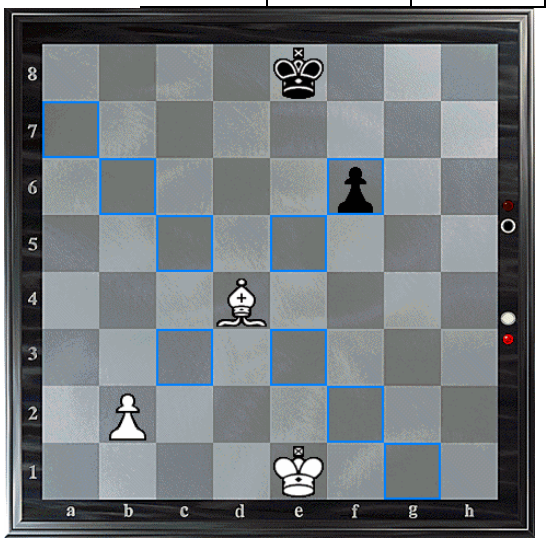
Бод болон хүүний хөлгийн нэг буудлаас нөгөө буудалд буух шилжилтийг нүүдэл гэнэ. Нүүдлийг тоглогчид ээлжлэн гүйцэтгэнэ. Цагаан тал анхны нүүдлийг хийнэ. Мориноос өөр аль ч бод дүрстэй буудлыг давж нүүдэггүй. Тэрэг босоо , хэвтээ шугамын дагуу нүүж иднэ. (Зураг 2) Тэмээ ташуу шугамын дагуу нүүж иднэ. (Зураг 3) Бэрс босоо , хэвтээ , ташуу шугамын дагуу нүүж иднэ. (Зураг 4) Бэрс бол тэрэг , тэмээний нүүдлийг хослуулан эзэмшсэн хамгийн хүчтэй бод. Ноён аль ч чиглэлээр зөвхөн нэг буудалд нүүж иднэ. (Зураг 6) Морь бол босоо ба хэвтээ шугамын аль нэгээр нэг нүд дамжин түүнээс цаашаа нэг буудал ташуу шугамаар нүүж иддэг. (Зураг 6) Морь өөрийн болон эсрэг талын дүрсүүд дээгүүр нүүдэг цорын ганц бод юм. Хүү нэг нүдээр урагшаа нүүнэ. Хэрвээ байрнаасаа хөдлөөгүй байгаа бол хоёр нүдээр нүүж болно. Ухарч нүүж болохгүй. Идэхдээ нэг нүд ташуулдаж иднэ. Хэрвээ идэх дүрс байхгүй бол ташуулдаж нүүж болохгүй. Мөн урд байгаа дүрсээ идэж чадахгүй. (Зураг 7) Хүүгийн хувьд бас нэг өвөрмөц нүүдэл байдаг. Энэ нь эсрэг талын хүү хоёр нүүж хажуугаар зөрж гарсан тохиолдолд тэр хүүний тал руу ташуулдаж нүүж идэж болдог. (Зураг 8) Зурган дээрээс харахад ойлгомжтой болох байх.

	Цагаан тал	Хар тал
Ноён		
Бэрс		
Тэрэг		
Морь		

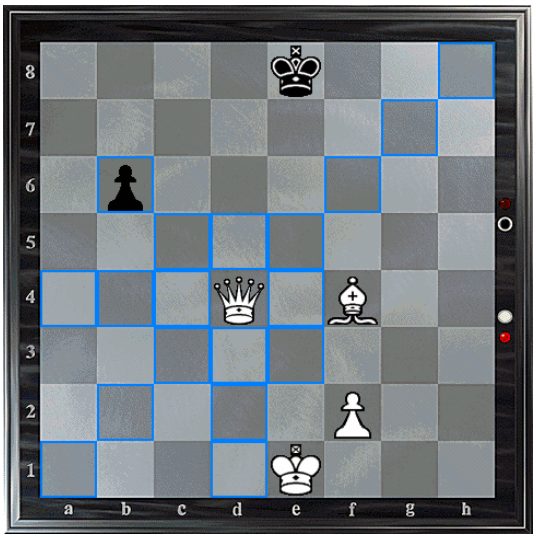
Тэмээ		
Хүү		



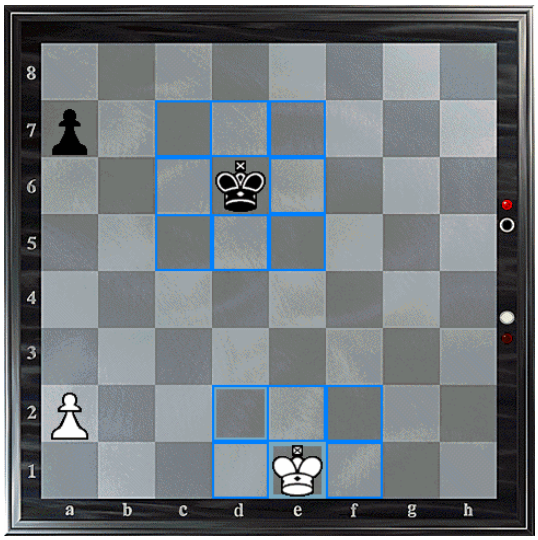
Зураг 2
Тэрэгний нүүдэл
(Хүрээлэгдсэн нь тэрэгний нүүх боломжтой нүднүүд)



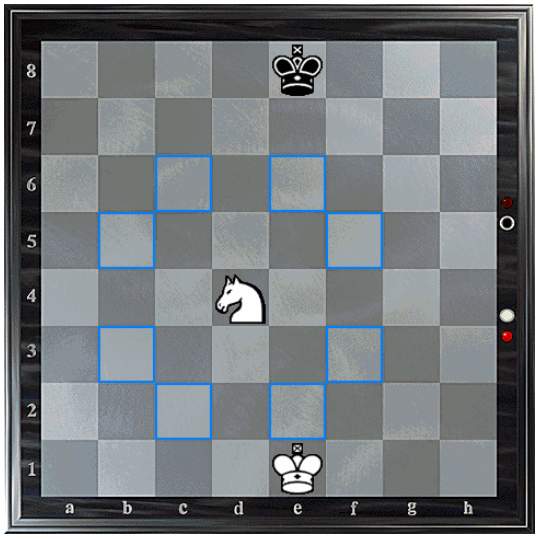
Зураг 3
Тэмээний нүүдэл
(Хүрээлэгдсэн нь тэмээний нүүх боломжтой нүднүүд)



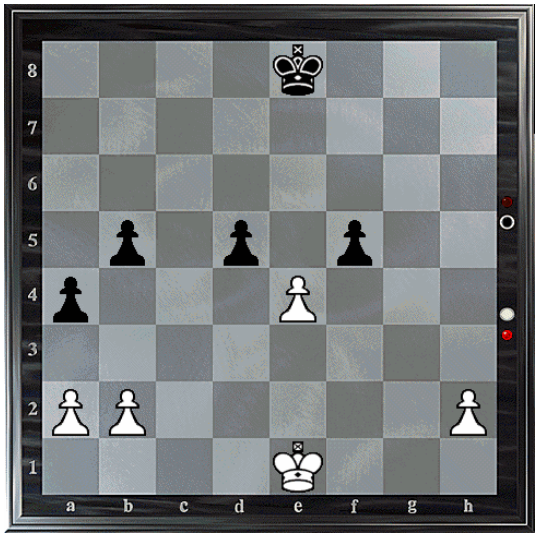
Зураг 4
Бэрсний нүүдэл
(Хүрээлэгдсэн нь бэрсний нүүх боломжтой нүднүүд)



Зураг 5
Ноёны нүүдэл
(Хүрээлэгдсэн нь ноёны нүүх боломжтой нүднүүд)



Зураг 6
Морьны нүүдэл
(Хүрээлэгдсэн нь морьны нүүх боломжтой нүднүүд)



Зураг 7
Хүүний нүүдэл
(Цагаан талын хүү e4 : d5 , e4 : f5 , хар талын хүү d5 : e4 , f5 : e4 гэж идэж болно. Цагаан талын хүү h2 : h4 гэж нүүж болно. Харин a2 : a4 гэж нүүж болохгүй учир нь эсрэг талын хүү саад болж байна. Хэрвээ цагаан талын хүү b2 : b4 гэж нүүвэл хар талын хүү a4 : b3 гэж b4 рүү нүүсэн хүүг идэж болно.)

Шалаа :

Аль нэг талын ноён эсрэг тоглогчийнхоо дүрсний довтолгоонд орохыг ноёныг шалах гэнэ. Шалааг хариулах 3 арга байдаг.

1. Довтолж байгаа (шалж байгаа) дүрсийг устгах (идэх).
2. Ноёноо зайлуулах. Өөрөөр хэлбэл эсрэг талын дүрсний довтолгоогүй буудалд буулгах.
3. Шалааг хаах. Довтолж буй бод ноён хоёрын хоорондох буудалд нүүж чадах хүү , бодоор халхлалт хийх. Харин морин шалааг хаах боломжгүй.

Мад :

Шалуулж байгаа үедээ түүнээс хамгаалах ямар ч нүүдэлгүй болсон ноёныг маданд орлоо гэнэ. Маданд орсон талыг хожигдсонд тооцно.

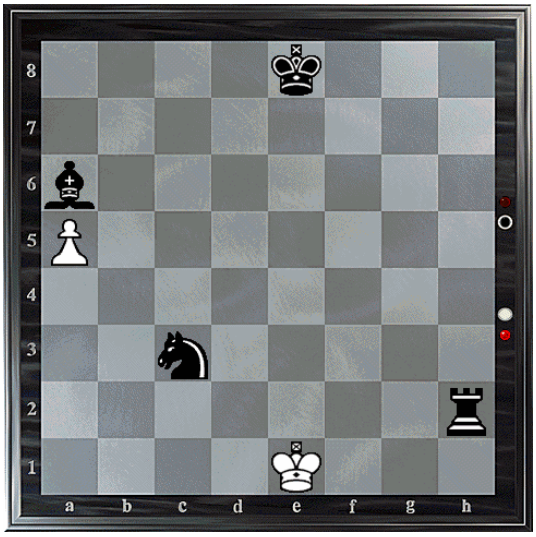
Жид :

Нэг тал нь нүүх ээлжиндээ бод , хүү ноёноор хийх ямар ч нүүдэлгүй болсон үед ноён шалуулаагүй байвал тийм байдлыг жид гэнэ. Энэ байдалд орсон өргийг тэнцэж төгссөнд тооцно. (Зураг 8)

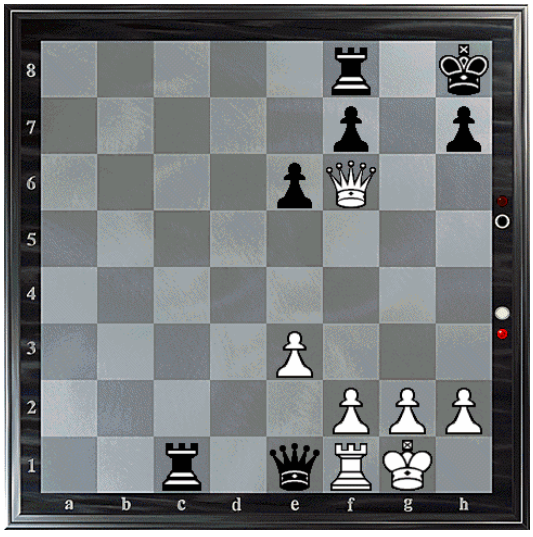
Тэнцээ :

Өрөг тэнцэх дөрвөн тохиолдол бий.

1. Хожил үл гарах нөхцөл байдал үүсэх (хоёр нүцгэн ноён үлдэх , нэг тал нь морь буюу тэмээтэй үлдэх , тус бүр ижил хөлийн тэмээтэй үлдэх)
2. Жид болох.
3. Нэг нь тэнцэх санал гаргасныг нөгөөдэх нь зөвшөөрөх.
4. Дандай шалах , байрлал гурван удаа давтагдах. (Зураг 9)



Зураг 8
Жид болох
(Цагаан талын нүүх ээлж. Шалаанд ороогүй боловч ямар ч нүүх нүүдэлгүй



Зураг 9
Тэнцээ
(Харын нүүх ээлж. Энэ тохиолдолд цагааны бэрс гурав дараалан шалахад

болсон байна.) тэнцэнэ.)

Сэлгээ :

Тоглогч нэг өрөгт нэг удаа ноён тэрэг хоёрын хамтарсан хөдөлгөөнөөр нэг нүүдэл хийхийг сэлгээ гэнэ. Сэлгээг бэрсний жигүүр рүү хийвэл уртын , ноёны жигүүр рүү хийвэл богины гэж нэрлэнэ. (Зураг 10 , 11)

Сэлгээг ноёноос эхэлж хийнэ.

Сэлгээ хийж болохгүй байх шалтгаан :

1. Уг өрөгт ноён , сэлгэх талын тэргээр нэгэнт нүүдэл хийсэн бол.
2. Ноён шалаанд байх үед.
3. Эсрэг талын дүрсний хөл довтолсон буудал дээгүүр ноён өнгөрөх бол.
4. Сэлгээ хийх үед ноёны буусан буудал дээр эсрэг талын дүрсний хөл довтолсон байвал.
5. Ноён тэрэг хоёрын хооронд ямар нэгэн дүрс байрласан байх

Зураг 10
Сэлгээ
(Уртын сэлгээ хийхийн өмнө)

Зураг 11
Сэлгээ
(Уртын сэлгээ хийсний дараа)

Зураг №12
Сэлгээ
(Богины сэлгээ хийхийн өмнө)

Зураг №13
Сэлгээ
(Богины сэлгээ хийсний дараа)

Шатрын хөлгийг програмд дүрслэх

Шатрын хөлгийг хэрхэн дүрслэх вэ?

Шатрын хөлгийг хэрхэн дүрслэх нь хамгийн чухал асуудал юм. Шатрын хөлгийг програмдаа хэрхэн дүрсэлсэн бэ гэдгээс хэр сайн програм болох вэ гэдэг шалтгаална.

Энд шатрын програмуудад хэрэглэгдэж байгаа хөлгийг дүрслэх ялгаатай хэдэн аргыг авч үзлээ.

1. 8x8 массив ашиглах

Шатрын хөлөг 8*8 хэмжээтэй массив байж болно.

Массивын нүд бүр нь ямар дүрс байгааг харуулсан утгыг агуулж байдаг.

Давуу тал нь ойлгомжтой энгийн, энэ хөлөг дээрээс бүх нүүдлийг олж авах нь хэцүү биш. Хэдий тийм боловч хөлгийн хязгаарыг тооцоолох нөхцлүүдийг шалгах шаардлагатай. Мөн массивт хоёр индексээр хандана. Энэ нь кодыг түвэгтэй програмыг илүү удаан ажиллагаатай болгодог. Материалын үнэлгээг тооцоход хялбар.

2. Өргөтгөсөн 10x10 массив ашиглах

Нэмэлт хоёр нүдэнд нь хөлгийн зах гэсэн мэдээллийг хадгалдаг. Ингэснээр эхний тохиолдолд гарч ирж байсан нэмэлт нөхцөл шалгах нь багасдаг.

3. Нэг хэмжээст массив ашиглах

64 нүдтэй нэг хэмжээст массив ашиглаж болно. Зөвхөн ганц индекс ашиглан хөлөгт хандаж байгаа учраас бага зэрэг хурдан болно.
Массивын 0 – р индекс шатрын хөлгийн зүүн дээд буланд таарч байхаар авъя.

	A	B	C	D	E	F	G	H	
8	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	8
7	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	7
6	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	6
5	[24]	[25]	[26]	[27]	[28]	[29]	[30]	[31]	5
4	[32]	[33]	[34]	[35]	[36]	[37]	[38]	[39]	4
3	[40]	[41]	[42]	[43]	[44]	[45]	[46]	[47]	3
2	[48]	[49]	[50]	[51]	[52]	[53]	[54]	[55]	2
1	[56]	[57]	[58]	[59]	[60]	[61]	[62]	[63]	1
	A	B	C	D	E	F	G	H	

Дүрсүүдийг массивт хэрхэн хадгалах вэ? Массивт дүрсүүдийг хадгалахдаа дүрсүүдийн үнэлгээгээр хадгалж болно. Ингэснээр материалын үнэлгээг бодоход маш хялбар болно.
Зарим нэг онцгой нүүдлийг хийхийн тулд нэг дүрсийг хоёр өөр байдлаар хадгалах шаардлага гарч ирдэг.

Жишээ нь:
Ноёны сэлгээ хийх нүүдэл. Ноён сэлгээ хийхийн тулд ямар нэг нүүдэл хийгээгүй мөн сэлгээ хийж байгаа талын тэрэг нүүгээгүй байх ёстой. Үүнээс шалтгаалж нүүсэн ноён нүүгээгүй ноён , нүүсэн тэрэг нүүгээгүй тэрэг гэсэн ойлголт гарч ирж байна.
Нүүсэн нүүгээгүй дүрсүүдийг ойролцоо үнэлгээгээр үнэлж массивтаа хадгалж болно.

Дүрсүүдийг үнэлж тодорхойлох.

Цагаан талын хүүд 100 оноо өгье. Ойролцоогоор гурван хүү нэг тэмээ болон морьтой тэнцдэг учраас цагаан талын морь , тэмээнд 350 , 352 оноо өгье. Хоёр хүү нэг морь (тэмээ) нэг тэрэгтэй тэнцдэг. Тэгэхээр тэрэгний оноо ойролцоогоор 500 болж байна. Бэрс хоёр тэрэгтэй тэнцдэг. 1000 оноо. Шатрын ноён хамгийн чухал дүрс учраас бүх дүрсүүдийн онооны нийлбэрээс илүү оноотой байх хэрэгтэй. Иймээс ойролцоогоор 10000 оноо болж байна.

Энэ оноонуудыг дараа үнэлгээний функцтээ ашиглана.

```
#define WKing       10000                   //Нүүгээгүй ноён
#define WKingM      10002                   //Нүүсэн ноён
#define WQueen      1000
#define WRook       500                 //Нүүгээгүй тэрэг
#define WRookM      502                 //Нүүсэн тэрэг
#define WKnight          352
#define WBishop          350
#define WPawn       100
#define BKing       -10000                  //Нүүгээгүй ноён
#define BKingM      -10002                  //Нүүсэн ноён
#define BQueen      -1000
```

```
#define BRook       -500                //Нүүгээгүй тэрэг
#define BRookM      -502                //Нүүсэн тэрэг
#define BKnight          -352
#define BBishop          -350
#define BPawn       -100
```

Материалын үнэлгээг тооцохдоо массивт байгаа бүх тоог хооронд нь нэмэхэд хангалттай.

```
int Board[64]=
{    BRook,BKnight,BBishop,BQueen,BKing,BBishop,BKnight,BRook,
     BPawn,BPawn,BPawn,BPawn,BPawn,BPawn,BPawn,BPawn,
     0,0,0,0,0,0,0,0,
     0,0,0,0,0,0,0,0,
     0,0,0,0,0,0,0,0,
     0,0,0,0,0,0,0,0,
     WPawn,WPawn,WPawn,WPawn,WPawn,WPawn,WPawn,WPawn,
     WRook,WKnight,WBishop,WQueen,WKing,WBishop,WKnight,WRook
}
```

Массиваас шаардлагатай нэмэлт мэдээллүүддийг яаж авах вэ?
Жишээ нь: 0 – р индекст байгаа дүрсийн мөр баганын дугаарыг яаж мэдэх вэ?
Үүний тулд нэмэлт массивуудыг ашиглая.
Мөр баганы дугаарыг хадгалъя.

```
const int Row[64]=
{    8,   8,   8,   8,   8,   8,   8,   8,
     7,   7,   7,   7,   7,   7,   7,   7,
     6,   6,   6,   6,   6,   6,   6,   6,
     5,   5,   5,   5,   5,   5,   5,   5,
     4,   4,   4,   4,   4,   4,   4,   4,
     3,   3,   3,   3,   3,   3,   3,   3,
     2,   2,   2,   2,   2,   2,   2,   2,
```

```
     1,   1,   1,   1,   1,   1,   1,   1
};
const int Col[64]=
{    1,   2,   3,   4,   5,   6,   7,   8,
     1,   2,   3,   4,   5,   6,   7,   8,
     1,   2,   3,   4,   5,   6,   7,   8,
     1,   2,   3,   4,   5,   6,   7,   8,
     1,   2,   3,   4,   5,   6,   7,   8,
     1,   2,   3,   4,   5,   6,   7,   8,
     1,   2,   3,   4,   5,   6,   7,   8,
     1,   2,   3,   4,   5,   6,   7,   8
};
```

Доорх массивт шатрын хөлгийн нүднүүдийн өнгөний талаарх мэдээллийг хадгалсан.

```
#define   white     1
#define   black     0
const int Color[64]=
{    white,    black,    white,    black,    white,    black,
     white,    black,
     black,    white,    black,    white,    black,    white,
     black,    white,
     white,    black,    white,    black,    white,    black,
     white,    black,
     black,    white,    black,    white,    black,    white,
     black,    white,
     white,    black,    white,    black,    white,    black,
     white,    black,
     black,    white,    black,    white,    black,    white,
     black,    white,
     white,    black,    white,    black,    white,    black,
     white,    black,
```

```
     black,    white,    black,    white,    black,    white,
     black,    white,
};
```

Мөн зүүн баруун диагоналиудыг дугаарлаж хадгалъя.

```
const int LeftDiagonal[64]=
{    8,   9,   10,  11,  12,  13,  14,  15,
     7,   8,   9,   10,  11,  12,  13,  14,
     6,   7,   8,   9,   10,  11,  12,  13,
     5,   6,   7,   8,   9,   10,  11,  12,
     4,   5,   6,   7,   8,   9,   10,  11,
     3,   4,   5,   6,   7,   8,   9,   10,
     2,   3,   4,   5,   6,   7,   8,   9,
     1,   2,   3,   4,   5,   6,   7,   8
};
const int RightDiagonal[64]=
{    1,   2,   3,   4,   5,   6,   7,   8,
     2,   3,   4,   5,   6,   7,   8,   9,
     3,   4,   5,   6,   7,   8,   9,   10,
     4,   5,   6,   7,   8,   9,   10,  11,
     5,   6,   7,   8,   9,   10,  11,  12,
     6,   7,   8,   9,   10,  11,  12,  13,
     7,   8,   9,   10,  11,  12,  13,  14,
     8,   9,   10,  11,  12,  13,  14,  15
};
```

Нүүдэл шалгах

Шатрын дүрмийг програмд хэрхэн тусгах вэ? Хэрэглэгч зөв нүүдэл хийсэн эсэхийг зайлшгүй шалгах хэрэгтэй. Шатрын дүрсүүдийн нүүдлийг массив доторх элементүүдийг шалгах замаар тодорхойлж болно.

Ноёны нүүдлийг шалгах.

m нь байрлалуудын ялгаварын абсолют утга.

```
int CheckKingMove(int f,int t,int m)
{    if(m==1){ return true; }
     if(m==8){ return true; }
     if(same_diagonal(f,t))
     {    if(m<10){ return true; }
     }
     return false;
}
```

Хэрвээ ноёны одоо байгаа мөн буух гэж байгаа байрлалуудын ялгаварын абсолют утга 1 буюу 8 байвал зөв нүүдэл. Хэрвээ ялгавар нь нэг байвал зүүн баруун талын зэргэлдээх буудал руу нүүж байна гэсэн үг. 8 байвал дээд доод талын зэргэлдээх буудал руу нүүж байна гэсэн үг.

```
int Board[64]=
{    0,   1,   2,   3,   4,   5,   6,   7,
     8,   9,   10,  11,  12,  13,  14,  15,
     16,  17,  18,  19,  20,  21,  22,  23,
     24,  25,  26,  27,  28,  29,  30,  31,
     32,  33,  34,  35,  36,  37,  38,  39,
     40,  41,  42,  43,  44,  45,  46,  47,
     48,  49,  50,  51,  52,  53,  54,  55,
     56,  57,  58,  59,  60,  61,  62,  63
};
```

Ташуу нүүдлийг шалгахдаа нэг диагонал дээр оршиж байгаа эсэхийг шалгаад дараа нь байрлалуудын ялгавар нь 10 – с бага байгаа эсэхийг шалгахад хангалттай. Нэг диагонал дээр байгаа эсэхийг шалгахдаа диагоналуудыг хадгалсан нэмэлт массив ашиглана.

```
const int LeftDiagonal[64]=
{    8,   9,   10,  11,  12,  13,  14,  15,
     7,   8,   9,   10,  11,  12,  13,  14,
     6,   7,   8,   9,   10,  11,  12,  13,
```

```
     5,   6,   7,   8,   9,   10,  11,  12,
     4,   5,   6,   7,   8,   9,   10,  11,
     3,   4,   5,   6,   7,   8,   9,   10,
     2,   3,   4,   5,   6,   7,   8,   9,
     1,   2,   3,   4,   5,   6,   7,   8
};
const int RightDiagonal[64]=
{    1,   2,   3,   4,   5,   6,   7,   8,
     2,   3,   4,   5,   6,   7,   8,   9,
     3,   4,   5,   6,   7,   8,   9,   10,
     4,   5,   6,   7,   8,   9,   10,  11,
     5,   6,   7,   8,   9,   10,  11,  12,
     6,   7,   8,   9,   10,  11,  12,  13,
     7,   8,   9,   10,  11,  12,  13,  14,
     8,   9,   10,  11,  12,  13,  14,  15
};
int same_diagonal(int first,int second)
{    if(LeftDiagonal[first]==LeftDiagonal[second]) return true;
     if(RightDiagonal[first]==RightDiagonal[second]) return
true;
     return false;
}
int Board[64]=
{    0,   1,   2,   3,   4,   5,   6,   7,
     8,   9,   10,  11,  12,  13,  14,  15,
     16,  17,  18,  19,  20,  21,  22,  23,
     24,  25,  26,  27,  28,  29,  30,  31,
     32,  33,  34,  35,  36,  37,  38,  39,
     40,  41,  42,  43,  44,  45,  46,  47,
     48,  49,  50,  51,  52,  53,  54,  55,
     56,  57,  58,  59,  60,  61,  62,  63
};
```

Байрлалуудынх нь ялгавар 9 – с хэтрэхгүй байна.

Тэрэгний нүүдлийг шалгах.

Эхлээд буудлууд нь нэг мөр буюу баганан дээр байгаа эсэхийг шалгана.

Row[] массивын харгалзах буудлуудын утга ижил байвал нэг мөрөнд байна.

Col[] массивын харгалзах буудлуудын утга ижил байвал нэг багананд байна.

Үүний дараа тухайн нүүж байгаа буудлуудын хооронд ямар нэгэн дүрс байгаа эсэхийг шалгана.

```
int CheckRookMove(int f,int t,int m)
{    if(Row[f]==Row[t])
     {    m=m-1;
          if(f>t)   //Left                    Зүүн тийшээ нүүж байна.
          {    while(m!=0)
               {    if(Board[f-m]!=0){ return false; }
                    m=m-1;
               }
               return true;
          }
          else //Right                    Баруун тийшээ нүүж байна.
          {    while(m!=0)
               {    if(Board[f+m]!=0){ return false; }
                    m=m-1;
               }
               return true;
          }
     }
     if(Col[f]==Col[t])
     {    m=m-8;
```

```
            if(f>t)    //Up                  Дээшээ нүүж байна.
            {     while(m!=0)
                  {     if(Board[f-m]!=0){ return false; }
                        m=m-8;
                  }
                  return true;
            }
            else //Down                Доошоо нүүж байна.
            {     while(m!=0)
                  {     if(Board[f+m]!=0){ return false; }
                        m=m-8;
                  }
                  return true;
            }
      }
      return false;
}
```

Тэмээний нүүдлийг шалгахдаа мөн адил нэг диагонал дээр байгаа эсэхийг шалгаад дараа нь буудлуудын хооронд дүрс байгаа эсэхийг шалгана.

Бэрсний нүүдэл бол тэмээ тэрэг хоёрын нүүдлүүдийн нийлбэр учраас тэрэг болон тэмээний нүүдэл шалгадаг функцээ ашиглаад шалгаж болно.

Морьны нүүдэл шалгах.

Морьны эргэн тойрны найман буудал дээр бууж байгаа эсэхийг шалгах хэрэгтэй.

```
int Board[64]=
{     0,    1,    2,    3,    4,    5,    6,    7,
      8,    9,    10,   11,   12,   13,   14,   15,
      16,   17,   18,   19,   20,   21,   22,   23,
      24,   25,   26,   27,   28,   29,   30,   31,
      32,   33,   34,   35,   36,   37,   38,   39,
```

```
     40,  41,  42,  43,  44,  45,  46,  47,
     48,  49,  50,  51,  52,  53,  54,  55,
     56,  57,  58,  59,  60,  61,  62,  63
};
```

Эндээс ажиглахад нэг зүй тогтол харагдаж байна.

28 – с 11 – р нүд рүү нүүж байгаа тохиолдолд буудлуудын ялгавар нь 17 байна.

28 – с 45 – р нүд рүү нүүж байгаа тохиолдолд буудлуудын ялгавар нь -17 байна.

28 – с 13 үед ялгавар нь 15 , 28 – с 43 үед -15 , 28 – с 18 үед 10 , 28 – с 38 үед -10 , 28 – с 22 үед 6 , 28 – с 34 үед -6 байна.

Эндээс буудлуудын ялгаврын абсолют утгыг шалгахад хангалттай байна.

m – нь буудлуудын ялгаврын абсолют утга.

```
int CheckKnightMove(int m)
{    if(m==17){ return true; }          //
     if(m==15){ return true; }          //
     if(m==10){ return true; }          //
     if(m==6){ return true; }      //
     return false;
}
```

Нүүлгэх

Нүүлгэх функцд нүүх гэж байгаа дүрсийн байрлал , буух гэж байгаа буудлын байрлалыг дамжуулна.

Ямар дүрс нүүж байгаагаас шалтгаалж өөр өөр байдлаар нүүлгэнэ.

Жишээ нь:

Ноён сэлгээ хийж байвал эхлээд ноёныг нүүлгээд дараа нь тэргийг нүүлгэнэ.

Нүүдэл буцаах

Нүүдлийг буцаах хэдэн арга байдаг.

1. Элемент бүр нь бүх хөлгийг агуулсан стек авах.

Нүүдэл хийхдээ түүндээ хөлгийг хадгалаад буцаахдаа стекээс хөлгийг авч сэргээнэ. Амар боловч энэ нь хэтэрхий удаан арга юм.

2. Зөвхөн хийгдсэн нүүдлийг хадгалах.

Зөвхөн хийгдсэн нүүдэл түүнтэй хамаатай (буцаж нүүхэд хэрэгтэй) мэдээллийг хадгалах. Шатар бол идүүлсэн дүрс нүүсэн буудал , буусан буудлыг хадгалах хэрэгтэй. Мөн сэлгээ буюу онцгой нүүдэл хийсэн эсэхийг хадгалах хэрэгтэй байдаг.

Боломжит нүүдлийн жагсаалт үүсгэх

Хэрэглэгчийн нүүдлийг шалгасны дараа компьютерын хийх нүүдлийг тодорхойлох хэрэгтэй. Үүнийг хийхийн тулд тухайн тоглолтын тухайн байрлалаас боломжит нүүдлүүдээр хайлтын мод үүсгэж түүнээсээ ашигтай нүүдлийг сонгож авч нүүдэл хийнэ.

Тухайн байрлалаас нүүх гэж байгаа талын хийх боломжтой байгаа бүх нүүдлийг тодорхойлно.

Үүний тулд хөлөг дээр байгаа бүх буудлыг шалгаж нүүх ээлжэнд байгаа дүрсүүдийн хувьд өөрсдийн нүүдлийн онцлогийг харгалзан нүүх боломжтой байгаа нүүдлүүдийг олж жагсаалт үүсгэнэ.

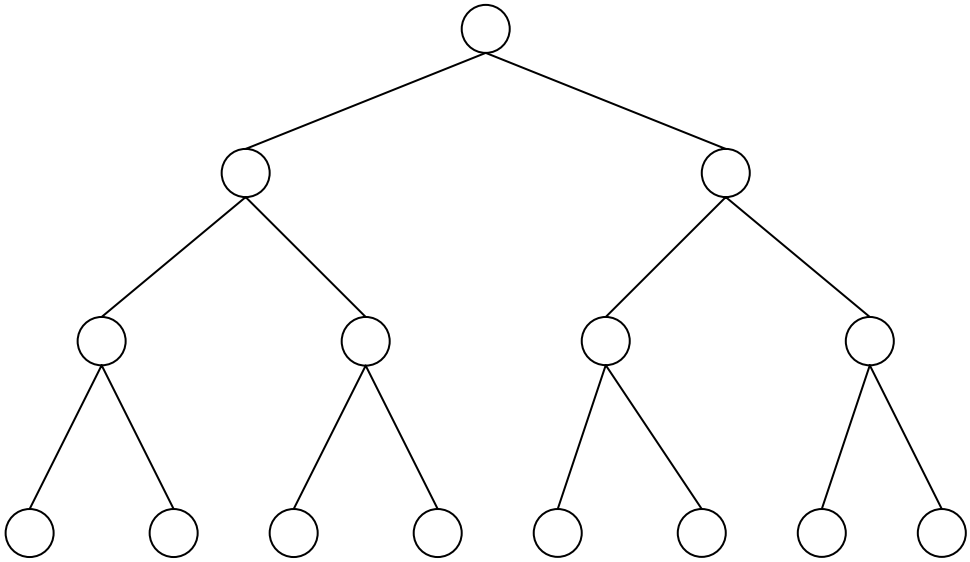
Жишээ нь:

Ноёны хувьд эргэн тойрных нь найман нүдийг шалгаад нүүх боломжтой байгаа буудлуудын хувьд жагсаалт үүсгэнэ. Мөн сэлгээ гэх мэт онцгой нүүдлүүдийг мөн шалгаж оруулах шаардлагатай.

Хайлтын модны тухай

Шатрын төрлийн тоглоомыг програмчлах зарчмыг анх 1950 онд Америкийн эрдэмтэн Шеннон томьёолжээ. Энэ арга нь модыг ашиглаж

шийдэлт боловсруулдаг юм. Ингээд хайлтын модны тухай авч үзье. Хайлтын мод бол тоглоомын програмчлалын гол аргуудын нэг юм. Модыг тоглоомын бүх боломжит нүүдлүүдийг ашиглаж үүсгэсэн тоглоомын тухайн байрлалыг хадгалахад ашигладаг.
Хайлтын модны жишээ авч үзье. (Зураг 14)



Зураг 14
Хайлтын мод

Програм нь тухайн агшинд нүүдэл хийж байгаа байрлал бол модны үндэс болдог. Орой бүрт байрлал , мөчир бүрт тал нүүдэл (тал нүүдэл бол нэг тоглогчийн нүүдэл) тус тус харгалзана. Боломжтой тал нүүдэл болгон нь мөчир үүсгэж эцэг оройг охин оройтой нь холбодог. Эдгээр охин оройнууд нь боломжтой нүүдэл хийх байрлалууд болно. Эсрэг тоглогчийн хариу нүүдлүүд нь нэгдүгээр түвшний зангилаа болгонд охин зангилаа (хоёрдугаар түвшний) болдог. Ийм дүрмээр доод түвшний бүх зангилаанууд үүсгэж байдаг. Хэрэв хайлтын мод нь гүйцэд үүссэн бол , төгсгөлийн зангилаануудад тоглоомын төгсгөлийн байрлалууд харгалздаг. Өөрөөр хэлбэл энэ байрлалууд дээр аль нэг тоглогч ялсан эсвэл тоглоомын дүрмийн дагуу хайнцсан байна.

Гэвч шатрын төрлийн тоглоомын байрлал ба нүүдлийн хувилбар маш их тул бүрэн гүйцэд хайлтын модыг байгуулах боломжгүй юм. Шатрын нэг байрлалд маш олон нүүдэл харгалздаг учир түвшин ихсэх тусам зангилааны тоо геометр прогрессоор ихсэнэ. Тоглолтын явцад дээрх модны хэмжээ маш том хэмжээтэй болдог. Иймээс бүх бодит шатрын тоглоомуудад тодорхой хязгаартай хайлтын мод зохиодог.

Үүний тулд бид ямар нэгэн зогсоох дүрмийг хэрэглэн тодорхой зангилаан дээр хүрээд цааш нь өргөтгөхөө зогсоох ба уг зангилааг навч болгон авч үздэг.

Тухайлбал 8 нүүдлийн дараа өргөтгөхөө зогсоох гэх мэт.

Мэдээж уг навчнууд дээр очоод тоглоом дуусчихгүй учраас үнэлгээний функцүүдийг ашиглан навчин дээрх байрлалуудын үнэлгээг гаргаж авах бөгөөд нэг тоглогч нь уг оноонуудаас хамгийн их оноотой байрлал руу нөгөө нь хамгийн бага руу нь тэмүүлэн тоглодог. Модон дахь уг байрлал хүрэх замыг хувилбар гэж нэрлэдэг.

Компьютерын хурд хүчин чадал , санах ойн хэмжээ ихсэх тусам хайлтын модыг бүрэн хайлтын модтой ойролцоо хэмжээнд дөхүүлж болно. Ихэнх хайлтын алгоритмуудын үр дүн нь бүрэн хайлтын модны үр дүнтэй дөхөж очдог.

Тухайн байрлалд харгалзах хамгийн оновчтой нүүдлийг яаж олох вэ? Үүнийг олохын тулд нилээд хэдэн төрлийн хайлтын алгоритмыг ашигладаг.

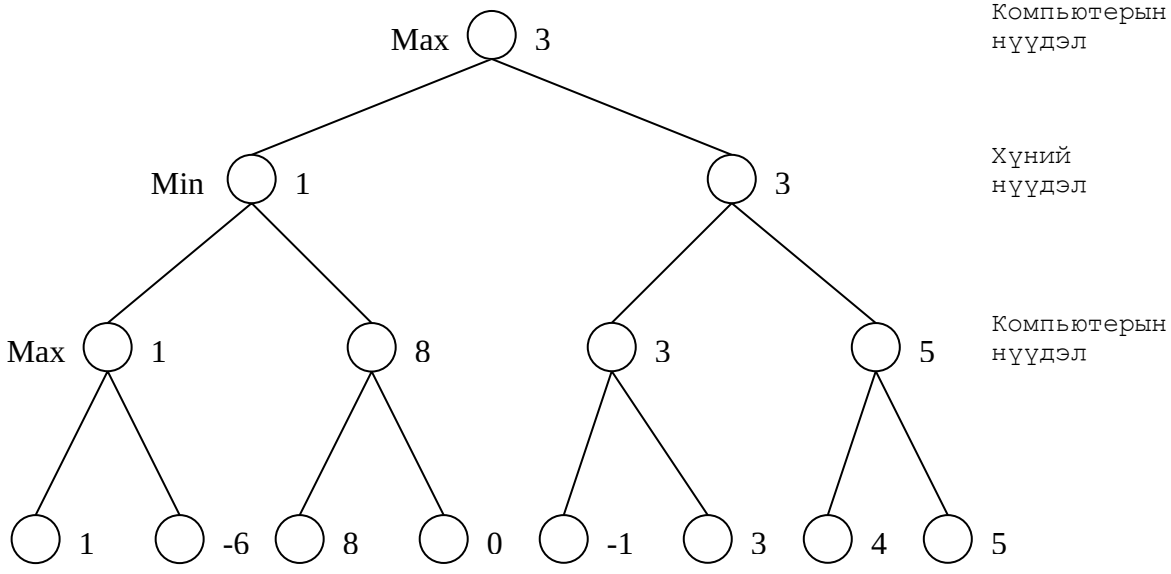
Хайлтын алгоритмууд

- MiniMax хайлтын алгоритм
- Итерац бүхий хайлт
- Альфа – Бета алгасалттай хайлтын алгоритм

MiniMax хайлтын алгоритм :

Энэ хайлтын алгоритм нь хоёр гарын нүүдлээр тоглодог тоглоомуудад зориулагдсан тэдгээрт ашиглахад илүү тохиромжтой юм. Шатар , даам гэх мэт. Эдгээр тоглоомуудын нийтлэг шинж нь тодорхой дүрэм зорилготой логик үйлдлүүд дээр тулгуурладагт байгаа юм. Энэ нь тухайн тоглоомд тодорхой үнэлт дүгнэлт өгч болно гэсэн үг. Мөн дараагийн бүх боломжтой нүүдлүүд нь тодорхой байдаг. Эсрэг тоглогчийн бүх зүйл тодорхой байдаг. Энэ арга нь хайлтын модыг үүсгэж түүнээсээ хамгийн сайн үнэлгээтэй нүүдлийг сонгодог. Модны үндэснээс эхлэн боломжит бүх нүүдлийг тооцоолж цаашид алхам алхамаар боломжит нүүдлүүдийг тооцоолж өргөтгөх замаар ажилладаг. Хайлтыг тодорхой түвшинд хүргээд зогсооно. Өөрийн болон эсрэг тоглогчийн боломжит нүүдлүүдийн үнэлгээнээс Min эсвэл Max – г сонгох замчмаар ажилладаг. Навчин дээрх үнэлгээнүүдээс Max утгатайг нь сонгон авч эцэг зангилаанд онооно. Харин дараагийн эцэг зангилаанд утга олгохдоо охин зангилаануудын үнэлгээнүүдээс Min – г сонгоно. Ингэж солбисон байдлаар ажиллана. Учир нь нэг тоглогчийн хувьд өндөр оноотой нүүдэл нь эсрэг тоглогчийн хувьд ашиггүй нүүдэл болдог.

Жишээ нь : (Зураг 15)



Зураг 15

Хайлтын мод

(MiniMax)

Энэхүү жишээн дээр 3 үнэлгээтэй нүүдэл хамгийн ашигтай болж байна.

Одоо энэ аргын псевдо алгоритмыг авч үзье.

```
MiniMax(GamePosition game)
{    if(GameEnded(game))
     {    return EvalGameState(game);
     }
     else
     {    best_move<-{};
          moves<-GenerateMoves(game);
          ForEach moves
          {    move<-MiniMax(ApplyMove(game));
               if(Value(move)>Value(best_move))
               {    best_move<-move;
               }
          }
          return best_move;
     }
}
```

Шатрын хувьд энэ тохиромжгүй. Шатрын төгсгөл маш олон нүүдлийн дараа байдаг учраас бүрэн хайлтын мод үүсгэх боломжгүй. Үүнийг сайжруулъя. Нэгэнт бүрэн мод үүсгэж чадахгүй учраас хайлтын гүнийг хязгаарлах хэрэгтэй.

Ингэж гүнийг хязгаарлаж өгснөөр тодорхой тооны нүүдлийн дараахыг харж чаддаг болно.

```
MiniMax(GamePosition game)
{    if(GameEnded(game) | | DepthLimitReached())
     {    return EvalGameState(game);
     }
     else
     {    best_move<-{};
```

```
            moves<-GenerateMoves(game);
            ForEach moves
            {     move<-MiniMax(ApplyMove(game));
                  if(Value(move)>Value(best_move))
                  {     best_move<-move;
                  }
            }
            return best_move;
      }
}
```

MiniMax хайлтын шинжилгээ.

Энэ хайлтын хувьд дараах нөхцлүүд биелнэ.

- ❖ Хайлтыг тодорхой гүнд явуулна.
- ❖ Модны зангилаа бүр нь ижил тооны хүүтэй гэж үзэж болно. (Дунджаар авч үзье)
- ❖ Хайлтын гүндээ хүрэхээс өмнө тоглоом дуусахгүй гэж үзэж болно.

D (depth) – Хайлтын гүн.

W (width) – Модны өргөн буюу нэг зангилаанаас үүсэх дундаж хүү зангилааны тоо. (Дундаж боломжит нүүдлийн тоо)

Энэ хайлтаар үүсэх навчны тоо нь геометр прогрессоор өсч байна. Иймээс хайж олсон байрлалын тоо нь W^D болно.

Шатрын хувьд дундаж боломжит нүүдлийн тоо нь ойролцоогоор 30 байдаг. Хэрвээ гурав гүнтэй мод үүсгэвэл навчны тоо нь 30^3 буюу 27000 болно. Гурав гүн шатрын хувьд төдийлөн хангалтгүй. Тав гүнтэй мод үүсгэвэл навчны тоо 30^5 буюу 24300000 болно. Энэ олон навч үүсгэхийн тулд маш их цаг зарцуулах хэрэгтэй болно. Ингэхээр энэ хайлт нь шатрын хувьд төдийлөн тохиромжгүй байна.

Итерац бүхий хайлтын алгоритм:

Энэ хайлтын алгоритмыг цагтай шатрын хувьд ашигладаг.

Хайлт хийх хугацааг хязгаарласан үед хайлтыг бага гүнээс эхэлж цаг хангалттай байгаа үед гүнийг аажмаар нэмэгдүүлж өөрт олгогдсон цагийг дуустал хайлтыг үргэлжлүүлнэ.

Псевдо алгоритм нь:

```
depth=0;
while(enough time left for another level of search)
{
     depth++;
     m=rootsearch(depth);
}
```

m нүүдлийг нүү.

rootsearch хайлтын функц нь:

```
move rootsearch(int depth)
{    double e=-infinity;
     Нүүдэл mm;
     for(тухайн байрлалд боломжтой байгаа нүүдэл m болгоны
хувьд)
     {    m нүүдлийг нүүнэ.
          double em=-MiniMax(depth - 1);
          if(e<em)
          {    e=em;
               mm=m;
          }
          m нүүдлийг буцах.
     }
     return mm;
}
```

Энэ нь илүү хайлт хийж цаг үрж байгаа мэт харагдаж болох юм. Гэвч энэ хайлтаар өргөтгөгдөх зангилааны тоо нь мөн л W^D – тэй ойролцоо байгааг дээрхтэй ижил тооцоогоор харж болно.

MiniMax – р тав гүнд 30^5 буюу 24300000 зангилаа үүсгэж байсан.

Харин итерацитай хайлтаар 30^1 + 30^2 + 30^3 + 30^4 + 30^5 = 30 + 900 + 27000 + 810000 + 24300000 буюу 25137930 зангилаа үүсгэж байна.
Эндээс харахад итерацитай хайлт нь ойролцоогоор 10 хувийн илүү хайлт хийж байна. Энд төдийлөн их хугацаа алдахгүй юм.
Энэ хайлт нь цагийн сайн хяналттай байдлыг олгоход тусалдаг. Мөн эхний хэдэн итерацийн бага гүн дэх хайлт нь том гүнд аль зангилаагаар хайхыг шийдэхэд тусалдаг ба альфа – бета хасах аргын үед энэ нь хурдтай хайлтыг явуулахад чухал ач холбогдолтой байдаг.

Alpha – Beta хайлтын алгоритм :

Өмнөх MiniMax хайлтын алгоритмыг хэрхэн сайжруулах вэ?. Хайлт хийх хугацааг яаж багасгах вэ? Үүний тулд ашиггүй нүүдлүүдийг хасч зангилааны тоог цөөлөх хэрэгтэй.
Үүний тулд хоёр арга хэрэглэдэг.

1) Шууд огтлол
2) Урвуу огтлол

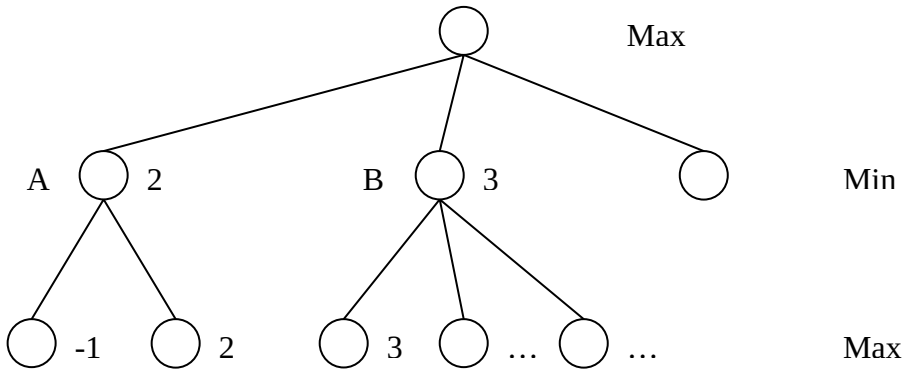
1)Шууд огтлол нь модны түвшин болгонд зөвхөн “ухаалаг” нүүдлүүдийн хувьд охин зангилаа үүсгэдэг. Шатар тоглодог хүн дандаа энэ зарчмаар тоглодог. Энэ аргыг хэрэгжүүлэх алгоритмыг Ботвинник дэвшүүлсэн. Тэрээр програм нь түвшин болгонд бүх боломжтой нүүдлийг биш харин дайралт хийх нүүдлүүдийн дагуу зангилаа үүсгэе гэжээ. Энэ нь төдийлөн сайн арга биш.
2)Урвуу огтлол буюу AlphaBeta процедур нь шууд огтлолыг бодвол төгс мод байгуулсан юм шиг үр дүнтэй байдаг.

Хайлтын моднуудыг байгуулах үед зангилаануудыг ямарч дарааллаар байгуулж болдог.

Жишээ нь :

Модыг үүсгэхдээ түвшин дараалан үүсгэнэ. Өөрөөр хэлбэл эхлээд тухайн түвшний бүх зангилааг үүсгээд , дараа нь дараагийн түвшний бүх зангилааг үүсгэх гэх мэт замаар явна. Үүнийг түвшин түвшинээр үүсгэх гэнэ.

Үүнээс гадна нэг талын мөчирлөлт ашиглаж үүсгэж болно. Энэ нь эхлээд эцэг зангилаа байгуулаад , дараа нь охин зангилаа байгуулаад тэрэндээ харгалзах охин зангилаа байгуулах замаар явна. Ингэсний дараа төгсгөлийн өмнөх түвшин бүрд төгсгөлийн охин зангилаанууд байгуулдаг , дараа нь нэг түвшин ахиад охин зангилаа мөн энэ түвшний хувьд байгуулдаг. Энэ аргыг нэг талын мөчирлөлт гэж нэрлэдэг.
Урвуу огтлол буюу AlphaBeta огтлолыг яаж хийхийг жишээн дээрээс харъя.
(Зураг 18)



Зураг 18

А зангилааны үнэлгээ 2 , В зангилааны охин зангилаануудын үнэлгээ 3 – с эхэлж байна. А , В зангилаанууд нь min түвшинд байгаа учраас энэ түвшинд байгаа үнэлгээнүүдээс хамгийн бага утгатайг нь сонгох ёстой. В зангилааны охин зангилаанууд max түвшинд байгаа. В зангилааны охин зангилаануудын ямарч утганд В зангилааны үнэлгээ 3 -с багасахгүй. Ийм учраас В зангилааг цааш

нь өргөтгөх шаардлагагүй. В зангилааны охин зангилаануудын үнэлгээ ямарч байсан А зангилааг сонгох нь ойлгомжтой.

AlphaBeta хайлтын гол зарчим нь:
Alpha , Beta гэсэн хоёр нэмэлт хувьсагч авна.
Alpha хувьсагчид зангилаануудын үнэлгээний max утгыг хадгална.
Beta хувьсагчид зангилаануудын үнэлгээний min утгыг хадгална.
Max түвшинд охин зангилаануудын үнэлгээг шалгахаас өмнө Beta хувьсагчийн утгыг өмнөх зангилаануудаас ирсэн үнэлгээтэй харьцуулна. Хэрвээ Beta их байвал шалгах шаардлагагүй тухайн зангилааг алгасна.
Min түвшинд охин зангилаануудын үнэлгээг шалгахаас өмнө Alpha хувьсагчийн утгыг өмнөх зангилаануудаас ирсэн үнэлгээтэй харьцуулна. Хэрвээ Alpha бага байвал хайлтыг үргэлжлүүлэх шаардлагагүй энэ зангилааг алгасна.
Ингэж MiniMax хайлтыг сайжруулснаар илүү хурдан их гүнд хайлт хийж чаддаг болно.

Альфа – Бета хайлтын псевдо алгоритм:

```
double AlphaBeta(int depth, double alpha, double beta)
{    if(depth<=0 | | тоглоом дууссан) return Evalution();
     Энэ байрлалд боломжтой байгаа нүүдлүүдийг үүсгэн эрэмбэлнэ.
     for(m нүүдэл бүрийн хувьд)
     {    m нүүдлийг гүйцэтгэ.
          double val= - AlphaBeta( depth – 1, - beta , -alpha );
          m нүүдлийг буцах.
          if( val >= beta ) return val;
          if( val > alpha ) alpha = val;
     }
     return alpha;
}
```

Үүнд беттад бид хүүхдүүдийн үнэлгээний хамгийн бага утгыг нь , альфад сондгой гүнд байгаа сайн зангилааг хадгалж явна. Энэ хоёр

параметрийг ашиглан алгасах алгоритмыг тун амархан кодлож байгааг харж байгаа бизээ. Урьд үзсэн хайлттай ижлээр өөр өөр гүнд байгаа үнэлгээнүүд нь бие биеэ үгүйсгэсэн шинж чанартай байна. Эрэмбэлэх нь юунд хэрэгтэйг бид удахгүй авч үзэх болно.
Энд нэг асуулт гарч ирнэ. Эх зангилаанаас анх хайлт эхлэх үед альфа болон беттагийн утга нь ямар байх вэ? Энэ хоёр параметр нь (альфа , бета) гэсэн бүхэл тоон завсрыг үүсгэх бөгөөд хэрэв үнэлгээний утга нь беттагаас их бол энэ нь үндсэн хувилбар биш болох учраас бид түүнийг алгасаад шууд үнэлгээгээ буцаана. Хэрэв утга нь альфагаас бага бол бид түүнийг алгасахгүй боловч түүнийг бас сонирхохгүй. Учир нь модны хаа нэгтээ түүнээс илүү сайн нүүдэл байгааг бид мэдэж байгаа. Гэвч модны эх зангилаан дээрээс энэ хоёр параметрт ямар утга өгвөл зохих вэ? Мэдээж бид ямар нэгэн ашигтай зангилааг санамсаргүй алгасахыг хүсэхгүй. Үүний тулд дараах утгыг өгөхөд хангалттай.

Альфа = - Их тоо; Бетта = - Их тоо;

Хэдийгээр тийм боловч бид ялангуяа итерацтай хайлт хэрэглэж байгаа үед илүү аятайхан арга ашиглаж болох юм. Энэ үед бид Их тоог x гэж үзье. Өөрөөр хэлбэл бид d гүнд энэ тоог ашиглах бөгөөд x нь d − 1 итерациар олдсон утга байх болно. Үүн дээр нэмэгдээд эпсилон хэмээх бага хэмжээний тоог ашиглах бөгөөд үнэлэхдээ дараах байдлаар дуудна.

```
AlphaBeta( depth , x-epsilon , x+epsilon )
```

Энэ үед дараах гурван тохиолдол үүснэ.

1. Хайлт нь (x-epsilon , x+epsilon) завсарт орших утга буцаах. Энэ тохиолдолд бид энэ утга нь зөв үнэлгээ болохыг мэдэх бөгөөд уг нүүдлийг хийхэд хамгийн зөв байх болно гэдэгт эргэлзэхгүй.

2. Хайлт нь v >= x+epsilon байх v утга буцаах. Энэ тохиолдолд үнэн үнэлгээний утга нь мөн л x+epsilon илэрхийллээс их утгатай байхыг мэдэх боловч яг ямар утгатай болохыг мэдэхгүй. Уг зөв нүүдэл нь алгасагдсан байж болзошгүй байна. Энэ байдлаас гарахын тулд бид x – н утгыг ихэсгэх (эсвэл эпсилоны утгыг ихэсгэж) дараа нь хайлтаа дахин эхлүүлэх явдал юм. Энэ байдлыг дээгүүрх алдаа гэж нэрлэнэ.
3. Хайлт нь v <= x+epsilon байх утга буцаах. Энэ тохиолдолд үнэн үнэлгээний утга нь мөн л x+epsilon илэрхийллээс бага утгатай байхыг мэдэх боловч яг ямар утгатай болохыг мэдэхгүй. Энэ байдлаас гарахын тулд бид x – н утгыг ихэсгэх (эсвэл эпсилоны утгыг ихэсгэж) дараа нь хайлтаа дахин эхлүүлэх явдал юм. Энэ байдлыг доогуурх алдаа гэж нэрлэнэ.

Хэдийгээр нь хайлт нь дээрх хоёр тохиолдолд алдаатай байгаа боловч энэ аргаар алгасалтыг сайн гүйцэтгэх тул илүү сайжруулсан алгоритм болж чадна.

Төгсгөл болон гарааг гүйцэтгэх

Төгсгөлд тоглох үед төгсгөлийн тоглолтын мэдлэгийг агуулсан тусгай үнэлгээний функцыг ашиглах хэрэгтэй болдог. Үүнээс бусад талаараа дунд үеийн энгийн тоглолттой ижлээр явагдана. Гарааны үед гараанд хэрэглэгддэг нүүдлүүд бүхий ном ашигладаг. Энэ өгөгдлийн сан дахь байрлалууд нь хэш хүснэгт хэлбэрээр бичигдсэн байх бөгөөд хандахад ч хурдан байна. Төгсгөлд дүрсүүд цөөн байдаг нь хувилбарыг илүү гүнд тооцоолох боломжийг олгодог тул ихэнх тохиолдолд хүнээс илүү сайн тоглодог. Үүнийг компьютер батласан юм. Эхнийх нь ноён ба тэрэг , ноён ба бэрсний эсрэг үед өргийг компьютероор шалгуулжээ. Үүнд хүмүүс тоглож байхад бэрстэй тал нь хялбар хождог байжээ. Харин компьютерыг үнэн хэрэгтээ тийм ч амархан хожиж чадахгүйг баталсан бөгөөд ихэнх тохиолдолд тэнцүүлж чадаж байсан байна. Дараагийн тохиолдол нь

ноён ба хоёр тэмээ , ноён ба морьны эсрэг тоглолт юм. Хүмүүс энэ тохиолдолд ихэвчлэн тэнцүүлдэг байсан бол компьютер нь тэмээтэй тал нь ихэвчлэн хождогийг харуулжээ. Харин үүний тулд хийж байсан нүүдлүүд нь хүн ойлгох аргагүй санамсаргүй юм шиг олон нүүдлүүд хийж байсны эцэст морийг гарах газаргүй болгон барьж байсан байна.

Үнэлгээний функц

Үнэлгээний функц бол програмын гол хэсэг юм. Энэ функц нь хайлтын модны навчнууд дээр дуудагдаж ажиллана. Ийм учраас маш олон удаа дуудагдаж ажиллана гэсэн үг. Хэрвээ програмаа хурдан ажилладаг , хайлтаа богино хугацаанд хийлгэдэг болгохын тулд энэ функцын код , ажиллах хугацааг бага байлгах хэрэгтэй. Гэхдээ хэт авсаархан хийвэл дутуу дулимаг үнэлгээг хийх тул мөн л тааруу тоглох болно.

Энэ функцыг аль тал илүү давуу талтай байгааг мэдэхэд ашиглана. Шатрын хувьд үүнийг хийхэд маш хэцүү. Хамгийн энгийн үнэлгээний функц нь шатрын хөлөг дээр байгаа бод богуудыг тоолж хэн нь олон хүү , бодтой байгаагаар үнэлгээ өгч болно. Гэхдээ шатрын хувьд энэ нь дэндүү өрөөсгөл болно. Хэдэн ч хүү бод дутуу байсан сайн байрлал эзэлж чадсан байвал ялалтанд хүрч болно.

Дараах зүйлүүдийг үнэлгээнд тусгаж өгөх хэрэгтэй.

- Материалын үнэлгээ. Өөрийн материал болон эсрэг талын материалын зөрүү байдлаар гарч ирнэ.
- Байрлалын үнэлгээ. Аль тал ашигтай байрлалд байгааг үнэлнэ. Үүнд хүүнүүдийн байрлал , төвийн байрлал эзэлсэн байдал , ноёны аюулгүй байдал гэх мэт орно.

- Хөллөгөөний үнэлгээ. Дүрсүүд аль болох олон нүүж болох буудалтай байвал сайн. Буланд шахагдсан нүүдэлгүй болсон бод нь муу үнэлгээтэй байна.

Харин төгсгөлд бол үнэлгээний функцыг өөрчлах зайлшгүй хэрэгтэй. Хуучин үнэлгээний функц дээр дараах хүчин зүйлүүдийг нэмж оруулж өгөх шаардлагатай.

- Нүүргүй хүү буюу цааш нүүсээр бод болох боломжтой хүүний үнэлгээ.
- Нүүргүй хүүг хамгаалж байгаа бод илүү үнэлгээтэй.
- Хэрэв хүүн бүтэц задгай бол тэрэг өндөр үнэлгээтэй байна.
- Төгсгөлд ноёны байрлал нь төв рүүгээ байх тусам үнэлгээ нь өндөрсөх болно.
- Ноёны хүүтэй хамтарсан байрлалуудын үнэлгээ.
- Хоёр талдаа хүү бүхий тэмээ нь төгсгөлд мориноос илүү үнэлгээтэй болно.
- Нэг талдаа хүүнүүдтэй бол морь нь тэмээнээс илүү үнэлгээтэй болно.

Тоглолт аль нэг талд ашигтай болох тусам үнэлгээ тэр талын хувьд өсөх ёстой.

Материалын үнэлгээ :

Ноён хамгийн үнэтэй. Бүх дүрсүүдийн үнэлгээнүүдийн нийлбэрээс үнэтэй. Хүүгийн үнэлгээг 100 – р авъя. Нэг тэмээ болон морь ойролцоогоор гурван хүүтэй тэнцүү байдаг. Нэг тэрэг ойролцоогоор хоёр хүү нэг морь буюу тэмээтэй тэнцдэг. Бэрс ойролцоогоор хоёр тэрэгтэй тэнцэнэ.

```
#define WKing       10000
#define WKingM      10001
#define WQueen      1000
#define WRook       500
```

```
#define WRookM      502
#define WKnight          355
#define WBishop          350
#define WPawn       100
#define BKing       -10000
#define BKingM      -10001
#define BQueen      -1000
#define BRook       -500
#define BRookM      -502
#define BKnight          -355
#define BBishop          -350
#define BPawn       -100
```

Байрлалын үнэлгээ :

Байрлалын үнэлгээний хувьд ашигтай байрлал эзлэхийн тулд хүү , бодоо золиослох нь тэдийлэн ашиггүй тул байрлалын үнэлгээ нь хүүний үнэлгээнээс бага байх хэрэгтэй.

Үнэлгээний функц дээр тухайн дүрсийн материалын үнэлгээг өгөөд харгалзах нүдний байрлалын үнэлгээг мөн нэмж өгнө.

Байрлал шалгах.

Зарим дүрсийн хувьд төвийн байрлал илүү ашигтай байдаг.

Бэрсний байрлалын үнэлгээ. Төвийн байрлал ашигтай.

```
const int QueenCenterScores[64] =
{
     0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,
     0,  0, 20, 30, 30, 20,  0,  0,
     0,  0, 30, 40, 40, 30,  0,  0,
     0,  0, 30, 40, 40, 30,  0,  0,
     0,  0, 20, 30, 30, 20,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0
```

```
};
```

Ноёны байрлалын үнэлгээ.

Төвийн байрлал ашиггүй боловч төгсгөлд ноён төв рүүгээ орох шаардлагатай байдаг.

```
const int KingCenterScores[64] =
{
      0,  0,  0,  0,  0,  0,  0,  0,
      0, 10, 20, 30, 30, 20, 10,  0,
      0, 15, 30, 40, 40, 30, 15,  0,
      0, 30, 40, 70, 70, 40, 30,  0,
      0, 30, 40, 70, 70, 40, 30,  0,
      0, 15, 30, 40, 40, 30, 15,  0,
      0, 10, 20, 30, 30, 20, 10,  0,
      0,  0,  0,  0,  0,  0,  0,  0,
};
```

Тэмээний хувьд мөн адил төвийн байрлал ашигтай.

```
const int BishopCenterScores[64]=       //Тэмээний төвийн
байрлалын үнэлгээ
{
     -70,  0,  0,  0,  0,  0,  0,-70,
       0,  0,  0,  0,  0,  0,  0,  0,
       0,  0,  0, 35, 35,  0,  0,  0,
       0, 40, 50, 45, 45, 50, 40,  0,
       0, 40, 50, 45, 45, 50, 40,  0,
       0,  0,  0, 35, 35,  0,  0,  0,
       0,  0,  0,  0,  0,  0,  0,  0,
     -70,  0,  0,  0,  0,  0,  0,-70
};
```

Цагаан хар талын тэмээнүүдийн хувьд ижилхэн тооцно.

Хар цагаан тэмээний хувьд бас байрлалын үнэлгээнүүд өөр өөр байна.

Гол диагональ дээр байвал илүү ашигтай.

```
const int BishopScoresW[64]=  //Цагаан талын тэмээний байрлалын үнэлгээ
{
       0,-20,-20,-20,-20,-20,-20,  0,
     -20, 30, 10, 10, 10, 10, 30,-20,
     -20, 20, 30,  0,  0, 30, 20,-20,
     -20,  0,  0,  0,  0,  0,  0,-20,
     -20,  0,  0,  0,  0,  0,  0,-20,
     -20,  0, 30,  0,  0, 30,  0,-20,
     -20, 25, 10, 20, 20, 10, 25,-20,
       0,-40,-40,-40,-40,-40,-40,  0
};
const int BishopScoresB[64] = //Хар талын тэмээний байрлалын үнэлгээ
{
       0,-20,-20,-20,-20,-20,-20,  0,
     -20, 25, 10, 20, 20, 10, 25,-20,
     -20,  0, 30,  0,  0, 30,  0,-20,
     -20,  0,  0,  0,  0,  0,  0,-20,
     -20,  0,  0,  0,  0,  0,  0,-20,
     -20, 20, 30,  0,  0, 30, 20,-20,
       0,-40,-40,-40,-40,-40,-40,  0
};
```

Морины хувьд булангийн байрлал хамгийн ашиггүй. Бүр хасах оноотой байсан ч болно. Морины хувьд байрлалын оноог иймэрхүй байдлаар өгч болно.

```
const int KnightScoresW[64]=       //Цагаан талын морьны төвийн үнэлгээ
{
```

```
      -50,-25,-25,-25,-25,-25,-25,-50
      -50, 25, 40, 30, 30, 40, 25,-50,
      -50, 20, 25, 40, 40, 25, 20,-50,
      -50, 25, 30, 45, 45, 30, 25,-50,
       20, 10, 30, 25, 25, 30, 10, 20,
      -50, 25, 50, 10, 10, 50, 25,-50,
      -50, 10, 20, 25, 25, 20, 10,-50,
      -70,  0,-50,-50,-50,-50,  0,-70,
};
const int KnightScoresB[64]=        //Хар талын морьны төвийн
үнэлгээ
{
      -70,  0,-50,-50,-50,-50,  0,-70,
      -50, 10, 20, 25, 25, 20, 10,-50,
      -50, 25, 50, 10, 10, 50, 25,-50,
       20, 10, 30, 25, 25, 30, 10, 20,
      -50, 25, 30, 45, 45, 30, 25,-50,
      -50, 20, 25, 40, 40, 25, 20,-50,
      -50, 25, 40, 30, 30, 40, 25,-50,
      -50,-25,-25,-25,-25,-25,-25,-50
};
```

Хүүний байрлалын үнэлгээ. Төвийн хүү мөн илүү үнэлгээтэй байна.

```
const int PawnCenterScoresW[64] =  //Цагаан хүүний төвийн
үнэлгээ
{
       80, 80, 80, 80, 80, 80, 80, 80,
       30, 30, 70, 50, 50, 70, 30, 30,
       25, 25, 60, 50, 50, 60, 25, 25,
       20, 20, 50, 50, 50, 50, 20, 20,
       15, 15, 40, 70, 70, 40, 15, 15,
       10, 10, 30, 50, 50, 30, 10, 10,
        0,  0,  0, 50, 50,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0
```

```
};
const int PawnCenterScoresB[64] =  //Хар хүүний төвийн үнэлгээ
{
       0,  0,  0,  0,  0,  0,  0,  0,
       0,  0,  0, 50, 50,  0,  0,  0,
      10, 10, 30, 60, 60, 30, 10, 10,
      15, 15, 40, 70, 70, 40, 15, 15,
      20, 20, 50, 50, 50, 50, 20, 20,
      25, 25, 60, 50, 50, 60, 25, 25,
      30, 30, 70, 50, 50, 70, 30, 30,
      80, 80, 80, 80, 80, 80, 80, 80
};
```

Мөн байрлалын үнэлгээнд орох нэмэлт зүйлс :

Үүнд:

Ноён:

Мөн ноён сэлгээ хийвэл ашигтай байрлалд орно. Ноёны талын сэлгээ бэрсний талынхаас илүү ашигтай байдаг.

```
#define KingSideCastled       75   //Ноёны талын сэлгээ
#define QueenSideCastled      50   //Бэрсний талын сэлгээ
```

Сэлгээ хийх боломжтой байдал үүсэж байвал ашигтай байрлалд байна гэж үзэж болно.

```
#define Can_KingSide_Castle   25   //Сэлгээ хийх боломжтой бол
#define Can_Queen_Side_Castle 25   //Сэлгээ хийх боломжтой бол
```

Сэлгээ хийх боломжгүй байдалд орвол ашиггүй байрлалд байна гэсэн үг.

```
#define Cant_Castle           -70  //Сэлгээ хийх боломжгүй бол
```

```
#define Cant_Castle_King_Side -50  //Ноёны сэлгээ хийх боломжгүй бол
#define Cant_Castle_Queen_Side     -25  //Бэрсний сэлгээ хийх боломжгүй бол
```

Хүү:

Давхардсан хүүний хувьд ашиггүй байрлалд байна гэж үзэж болно.

```
#define Doubled_Pawns              -50  //Давхардсан хүү
#define Blocked_Doubled_Pawns      -70  //Хашигдсан давхардсан хүү
```

Гурав давхардсан хүүний хувьд хоёр хүүнийхээс ч бага оноотой байж болох юм.

```
#define Trippled_Pawns             -99  //Гурав давхардсан хүү
```

Хүүний урд дүрс тээглэж байвал ашигггүй.

```
#define Blocked_Pawns              -40  //Хашигдсан хүү
```

Тэрэг:

Тэрэгний урд ямар нэг дүрс тээглээгүй урагшаа довтлоход бэлэн байвал ашигтай.

```
#define Rook_On_Open_Column        50   //
```

Бэрс:

Дээрх бас бэрсэнд хамаатай.

```
#define Queen_On_Open_Column       25   //
```

Хөллөгөөний үнэлгээ :

Байрлалаас гадна олон зүйлийн үнэлгээг тооцох хэрэгтэй.

Шатрын дүрснүүд аль болох нүүх боломжтой олон буух буудалтай байвал ашигтай. Үүнийг хөллөгөөний үнэлгээгээр тооцно.

```
#define Blocked_Queen              -    //Хашигдсан бэрс
#define Blocked_Rook               -    //
#define Blocked_Knight             -    //
#define Blocked_Bishop             -    //Хашигдсан тэмээ
```

Харилцан уялдаа :

Зэргэлдээ байрласан хоёр тэмээ илүү оноотой.

```
#define Two_Bishops           50   //Зэрэгцээ хоёр тэмээ
```

Холбоотой хоёр тэрэг илүү оноотой.

```
#define Two_Rooks             50   //Холбоотой тэрэг
```

Холбоотой хоёр морь илүү оноотой.

```
#define Two_Knights           50   //Холбоотой морь
```

Ноёны аюулгүй байдал :

```
//Ноёны эргэн тойронд нүх байвал
#define King_Near_Open_File       -100
//Ноёны ойролцоо давхардсан хүү байвал
#define Doubled_Pawn_Near_King    -150
```

Төгсгөлийн үнэлгээнүүд :

- Нүүргүй хүү буюу цааш нүүсээр бод болох боломжтой хүүний үнэлгээ.
- Нүүргүй хүүг хамгаалж байгаа бод илүү үнэлгээтэй.
- Хэрэв хүүн бүтэц задгай бол тэрэг өндөр үнэлгээтэй байна.
- Төгсгөлд ноёны байрлал нь төв рүүгээ байх тусам үнэлгээ нь өндөрсөх болно.
- Ноёны хүүтэй хамтарсан байрлалуудын үнэлгээ.
- Хоёр талдаа хүү бүхий тэмээ нь төгсгөлд мориноос илүү үнэлгээтэй болно.
- Нэг талдаа хүүнүүдтэй бол морь нь тэмээнээс илүү үнэлгээтэй болно.

```
//Нүүргүй хүү
#define Passed_Pawn           0
//Нүүргүй хүүний хамгаалалт
#define Support_Passed_Pawn       0
//Ноёны хүүтэй хамтарсан оноо
```

```
#define King_Pawn                    0
```

Хайлтын хурдыг ихэсгэх

1. Боломжит нүүдлийн жагсаалтыг эрэмбэлэх

Боломжит нүүдлийн жагсаалтыг эрэмбэлснээр хайлтын хурд ихэснэ. Альфа – Бета хайлтын алгасалтыг бүрэн ашиглахын тулд өргөтгөх нүүдлүүд нь эрэмбэлэгдсэн байх шаардлагатай. Өөрөөр хэлбэл бид эхлээд сайн нүүдлүүдийг өргөтгөн задална гэсэн үг юм. Хэрэв ингэж эрэмбэлэхгүй бол юу болох вэ? Тэгвэл бид алгасалтыг гүйцэтгэж чадахгүй бөгөөд илүү гүнд хайх боломжгүй болно.
Ийм учраас хайлтын модоо өргөтгөхдөө сайн нүүдлүүдийг эхэлж өргөтгөх хэрэгтэй байна.
Ямар нүүдлийг сайн гэх вэ?

a. Довтолгоо хийж байгаа буюу эсрэг талын дүрсийг идэж байгаа нүүдэл хамгийн сайн.
b. Боднуудын нүүдэл хүүний нүүдлээс сайн.

Ямар зангилааг өргөтгөх вэ?

Нүүдлийг сонгохдоо бүх нүүдлийг нүүлгэх, эсвэл сонголт хийн хэсэг нүүдлийг нүүлгэх гэсэн хоёр замын нэгийг сонгох болдог. Тодорхой байрлалаас сонголт хийн нүүхэд илүү хол илүү гүнд хайлтыг гүйцэтгэж чадах боловч хэзээ ч харахгүй нүүдлүүд түүнд гарч ирэх юм. Харамсалтай нь муу нүүдлүүд нь үргэлж тийм ч муу нүүдлүүд байдаггүй бөгөөд харин ч өргийн гол чимэг нь болсон уран гоё нүүдлүүд байх нь бий. Шатар тоглогч програмуудад хязгаарын эффект хэмээх нэгэн дутагдал байдаг.

Жишээ нь :

7 нүүдлийн цаана хайдаг програмыг авч үзье. 6 нүүдлийн цаана тэрээр мориор хүү идсэн нүүдэл хийсэн нь нилээд хэдэн нүүдлийн цаана гарцаагүй баригдах муу нүүдэл байв. Харин компьютер нь

үүнийг олж харахгүй учир нь тэр үүнээс цааш ганц ч нүүдлүүдийг авч үзээд ямар ч асуудалгүй гэж үзэх болно. Харин хүн нь 6 нүүдлийн цаана ийм нүүдэл байхыг харвал хэзээ ч үүнийг хийхгүй. Чухам энэ дутагдал нь шатар тоглодог програмуудыг хожих нөхцлийг хүнд олгож байгаа бөгөөд энэ дутагдлыг арилгах алгоритм олдтол энэ нь цааш үргэлжлэх нь тодорхой байна. Орчин үед яг энэ хоёрын нэгийг нь гэж хэрэглэдэггүй бөгөөд харин хоёуланг нь тодорхой хэмжээнд хэрэглэдэг юм. Бид зарим тодорхой түвшинд бүх зангилааг өргөтгөөд харин илүү гүнд заримыг нь өргөтгөдөг. Заримдаа альфа – бетад хийдгээс гадна алгасалтын аргуудыг ч ашигладаг.
Хайлтыг гүйцэтгэж байгаад хязгаарт тулан ирсэн үед нэмэлт маягийн хайлтыг хэрэглэж болно. Энэ хязгаарт байгаа зөвхөн идэх нүүдлүүдийг л өргөтгөж үзэх хэрэгтэй.

2. Хэш хүснэгт ашиглах

Хэш хүснэгт гэж юу вэ?

Хэш юунд хэрэгтэй вэ?

Хэш хийх нь маш энгийн шалтгаантай. Ихэнх тоглолтын явцад ялгаатай нүүдлүүдийн эцэст ижил байрлалд хүрэх тохиолдлууд элбэг гардаг. Тухайлбал шатарт дараах гараанууд нь ижил байрлалд хүрнэ.

1. d4 Nf6 2. c4 ба 1. c4 Nf6 2. d4 (Энэтхэг гараа)

Илүү олон нүүдэлтэй жишээ нь:

1. e4 c6 2. d4 d5 3. ed Qxd5 4. Nc3 Qd6 (Каро-Канны хамгаалалт)

1. e4 d5 2. ed Qxd5 3. Nc3 Qd6 4. d4 c6 (Скандинав гараа)

1. e4 Nf6 2. e5 Ng8 3. d4 d6 4. ed Qxd6 5. Nc3 c6 (Алехины хамгаалалт)

Эдгээр нүүдлээс болоод ижил байрлалууд хайлтын модны энд тэнд олон тааралддаг байна. Хэрэв бид байрлал бүрийг хайх явцад гарсан үр дүнг хадгалж явбал бид дахин тааралдах байрлалыг мэдэж авч чадна. Гэвч . . . бид бүх байрлалыг хадгалж хүрэхээр хангалттай санах ойгүй шүү дээ. Тэгээд ч хадгалсан байрлалуудаас хайх нь хайлт хийхээс хамаагүй хурдан цаг хэмнэдэг байх ёстой. Үүний хариулт нь хэш хүснэгт болох юм. Хэш хүснэгтийг физик санах ойгоос хальж гарахааргүй хэмжээтэйгээр сонгож авна. (Виртуал санах ой нь удаан тул ашиглана гэж саналтгүй)

```
struct Hash
{    long HashKey;       //
     int Depth;              //
     int Point;              //
} HashTable[Hash_Table_Size];
```

Хайж байгаа байрлал бүрийн хувьд хэш утга х – г хэш хүснэгтийг индекслэхэд , өөр нэг хэш утга у – г зөв байрлалыг олсон эсэхийг шалгахад хэрэглэнэ. Тэгээд байрлалыг хайхаасаа өмнө хэш хүснэгтэд шалгана. (HashTable[x])

Хэрэв олдож байвал түүнд хадгалсан оноог буцаана. Байрлалыг хайсны дараа бол у , тухайн гүн болон бодож олсон үнэлгээг HashTable[x] хүснэгтэд хадгалах хэрэгтэй.

Хэш утгыг хэрхэн бодож олох вэ?

Зобристын хэш аргачлал:

Тоглоомыг эхлэхийн өмнө санамсаргүй тоонууд бүхий Z[буудал, дүрсийн төрөл] массив үүсгэ. Хэш (хөлөг) утга нь энэ тохиолдолд ердөө Нийлбэр(Z[s,p]) буюу хөлөг дээр одоо байгаа дүрсүүдийн утгуудын нийлбэр байх юм. Харин шинэ нүүдэл хиймэгц шинэ хэш утгыг дахин тооцоолох хэрэггүй бөгөөд үүний тулд хуучин утгаас нь дүрсийн хуучин байрлал дахь утгыг хасаад шинэ байрлал дахь утгыг нэмснээр болох болно. Энэ аргыг хэш утга х болон нийлбэр у дээр хэрэглэнэ.

Хэш хийх зарим ашигтай зүйлс:

- Нүүдлийг хийсний дараа түүний хэсгийг бүү цэвэрлэ. Энэ нь ердөө цаг үрсэн ажил бөгөөд зарим хэш хийсэн байрлалууд нь нүүдэл хийсний дараа ч хэрэгтэй хэвээр байдаг.
- Хэрэв модонд өөр өөр түвшинд ижил байрлал тааралдвал энэ нь чамд илүү гүн хайх боломж олгох тул зүгээр хэрэг.
- Навчинд их ойролцоо байгаа байрлалуудыг хэшд хийсний хэрэггүй. Учир нь ийм байрлалууд хэтэрхий олон байх бөгөөд тэднийг хайснаас илүүгээр цаг хугацаа хэмнэж чадахгүй. Тэд хэш хүснэгтийг дүүргэн илүү чухал байрлалуудыг ашиглах аргагүй болгоно.

Хэрхэн хэш аргачлалыг альфа – бета хайлттай холбох вэ?

Шатрын програмын ихэнх дутагдал нь хэштэй холбоотой байдаг. Учир нь энэ нь альфа – бета хайлттай маш нарийн төвөгтэй холбогддог. Гэвч үүнээс зугтаах аргагүй бөгөөд учир нь сайн хайлтыг гүйцэтгэхийн тулд эдгээр хоёр хоёулаа чамд хэрэг болно. Бидний үнэлгээ нь альфа ба бета утгатай дараах байдлаар холбогдоно.

Альфа < үнэлгээ < бета

Хэрэв үнэлгээ нь бетагаас илүү их байвал бид түүнийг шууд буцаан дараах хувилбаруудыг нь алгасдаг. Бид альфа – бета хайлтыг ашиглах үед дараах гурван тохиолдлын нэг болдог. Дээгүүрх алдаа , энэ үед бид үнэлгээ нь ядаж бета болохыг мэдэх боловч яг ямар утгатай болохыг мэдэхгүй. Доогуурх алдаа, энэ үед бид үнэлгээг нь ядаж альфа болохыг мэдрэх боловч бас л яг ямар утгатай болохыг мэдэхгүй байна. Гурав дахь нь үнэлгээ нь альфа , бета завсарт байх. Энэ тохиолдолд бид яг утгыг нь мэдэж байна. Бид хэш хүснэгтэд зөвхөн яг мэдэгдэж байгаа утгыг л хадгалж чадна. Хэдийгээр тийм боловч дээгүүрх болон доогуурх алдаа нь бидэнд хувилбаруудыг дараа нь алгасах боломжийг олгосоор л байдаг.

Иймээс хэш хүснэгтэд хоёр өөр төрлийн үнэлгээ хадгалах болж байна.

1. Доогуурх хязгаар. Үнэлгээ нь хамгийн багадаа бета байна гэсэн үг.
2. Дээгүүрх хязгаар. Үнэлгээ нь хамгийн ихдээ бета байна гэсэн үг.

Бид хэш хүснэгтэд entry_type төрлийг ямар төрлийн үнэлгээ хадгалагдаж байгааг тэмдэглэхийн тулд авч байгаа юм. Хэрэв бид хүснэгтээс олж авсан үнэлгээ нь ийм төрөлтэй байвал бид уг мэдээлэл нь хайлтыг цааш нь үргэлжлүүлэлгүй хувилбарыг алгасахад хангалттай эсэхийг шалгана. Хэрэв тийм байвал бид уг үнэлгээг буцааж үгүй бол хайлтыг цааш нь үргэлжлүүлэх юм. Доор хэш хүснэгт хэрэглэсэн альфа – бета хайлтын алгоритмыг харууллаа. Хүснэгтийн индекс x болон y – г глобаль хувьсагчаар авсан.

Альфа – бета хайлт

```
int AlphaBeta(int depth, int alpha, int beta)
{    if(depth==0 | | тоглоом дууссан) return Evalution();
     if(HashTable[x].HashKey==y && HashTable[x].Depth >=depth)
     {    switch(HashTable[x].entry_type)
          {    case тодорхой :
               {    return HashTable[x].Point;
               }
               case доод хязгаар :
               {    if(HashTable[x].Point>=beta)
                         return HashTable[x].Point;
                    else break;
               }
               case дээд хязгаар :
               {    if(HashTable[x].eval<=alpha)
                         return HashTable[x].Point;
                    else break;
               }
          }
```

```
        }
        int үнэлгээ_тодорхой = 0;
        Энэ байрлалд боломжтой байга нүүдлүүдийг үүсгэн эрэмбэлнэ.
        for(m нүүдэл бүрийн хувьд)
        {    m нүүдлийг нүү;
             int val=-AlphaBeta(depth-1,-beta,-alpha);
             m нүүдлийг буцах;
             if(val>=beta)
             {    HashTable[x].HashKey = y;
                  HashTable[x].Depth=depth;
                  HashTable[x].entry_type=доод хязгаар;
                  HashTable[x].Point = val;
                  return val;
             }
             if(val>alpha)
             {    alpha=val;
                  үнэлгээ тодорхой = 1;
             }
        }
        HashTable[x].HashKey = y;
        HashTable[x].Depth=depth;
        if(үнэлгээ тодорхой)
        {    HashTable[x].entry_type = тодорхой;
        }
        else
        {    HashTable[x].entry_type = дээд хязгаар;
        }
        HashTable[x].Point=alpha;
        return alpha;
}
```

Энэ хайлтыг гүйцэтгэж байгаа үед үнэхээр алгасч болох бүх хувилбарыг алгасан ажиллуулбал модны гүнг хоёр дахин нэмэгдүүлэх чадалтай алгоритм юм.