Not: Burada bahsedilen her mantığı ve kodu oturup saatlerce kağıt-kalem ile düşünüp kendim geliştirdim, Chat-GPT veya stackoverflow’dan alınmış tek bir kod veya algoritma yoktur.

Program hakkında tek bir kelime anlaşılmadan önce kuralların okunup anlaşılması şiddetle tavsiye edilir. Aksi takdirde programda oluşturulan ağaçlar, olasılıklar vs. anlamsız gelecektir. “Sadece algoritmalara, OOP mimarisine veya yapısına (kesinlikle mimari denecek kadar sofistike değil, ama gene de…), fonskiyonlara bakacağım” diyorsanız bunu yapmanıza pek gerek yok.

Belirtilmesi gereken çok önemli bir nokta şudur ki, programı geliştirirken belirli bir yere kadar tamamen matematik formülleri ve oyun teorisi kuramlarına sadık kalmaya çalışsam da ne yazık ki bazı noktalarda insiyatif kullanmam gerekmiştir. Zaten tamamiyle objektif kurallara göre hareket eden bot yazmak için oyun teorisinden çok iyi anlayan birinin oturup bu oyunu günlerce çözümlemesi gerekir. Günlerce kısmını halledebilirim ama oyun teorisinden bu oyunu mükemmel hamlelerle oynatacak bir bot yazacak kadar iyi anladığım kesinlikle söylenemez.

Glados, Portal göndermesidir ve botun ismidir. Program boyunca bota “Glados” olarak hitap edeceğim.

Defalarca kez “bilinen deste” terimini kullanacağım. Bu, Glados’un kurallar dahilinde bilebileceği desteyi ifade eder. Mesela, normal deste 6,3,8,10,5,7 olsun. Tur başladığında Glados’a gelen kartlar 10,6 ve insana gelen kartlar da 8 ve 7 olduğunu hayal edersek Glados’un bilebileceği deste 3,5 ve 7’dir. 10 ve 6’yı kendi kartları olduğu için eler, 8’i bilebilmesinin sebebi ise oyun kuralları gereği ilk kartın herkese açık olmasıdır.

Oyun başlar başlamaz, her turda botun yapması gereken ilk şey, kart çekmesi “mantıklı” olduğu sürece, kart çekmesi ve kazanma olasılığını arttırmasıdır. Burada “mantıklı”dan kasıt istatistik ve oyun teorisi dünyasının vazgeçilmezi beklenen değerin pozitif veya sıfır olmasıdır. Beklenen değer, farklı seçimlerin mevcut olduğu bir durumda, bireyin karar vermesini sağlayan bir formüldür.

Beklenen değer

Beklenen değer = P1\*V1 + P2\*V2 … PN\*VN

Veya daha havalı bir gösterim olmasını istiyorsak:

s, toplam durum sayısını; n, her bir durumu; P(n) n durumunun gerçekleşme olasılığını ve V(n) n durumunun “değerini” belirtir. Nedir bu “değer”?

Bizim oyunumuz için “değer” imajiner kazanma olasılığıdır. Yani Glados’un bilebileceği destede 6,9,3,10 kartları olsun ve Glados’un elinde 10+5 olduğunu ve insan oyuncunun açık kart değerinin 7 olduğunu düşünelim. Şuanki kazanma olasılığımız 2/4’dür çünkü 6 veya 3 gelmiş ise Glados galip gelecektir, ama 9 veya 10 gelmiş ise insan galip gelecektir. Bu olasılığa ilk olasılığın i’sinden yola çıkarak Pi diyelim.

Glados kart çekmeli mi? 4 karttan her birinin gelme olasılığı tamamen eşit kabul edilir, yani 1/4'dür. 6’yı çekersek elimiz 21 olur, bu da kazanma olasılığını 1’e çıkartır çünkü artık insanın eline gelebilecek olan 9,3 veya 10 vardır ve bunların hangisi gelmiş olursa olsun Glados galip gelecektir. Bu imajiner, var olabilecek olasılığa da düş kelimisinin d’sinden yola çıkarak Pd diyelim. Değer için Pd-Pi yapmamız gerek. Yani bu durumun değeri 1 – 2/4’den +1/2 gelir. Yani P(1) = 1/4 ve V(1) = 1/2'den ilk iterasyonumuzun değeri 1/8’dir.

Böylelikle V(n)’in Pdüş-Pilk olduğunu buluruz.

Bu durumda formülü açmamız daha anlaşılır olacaktır.

İkinci karta bakalım. 9 gelirse Glados’un eli 24 olur, bu el 21’den büyük olduğu için değeri 0 kabul edilir. Bu sefer de insana 6,3,10’dan hangi kart gelmiş olursa olsun, Glados kaybeder. Bu durumda değerimiz 0-2/4’den -1/2 gelir.

Diğer kartları tek tek yapmamıza gerek yok.

Bu formülün kod kısmında açıklanması gereken pek bir şey yoktur, basit bir for döngüsü ve her seferinde winProbabilityFunction’ı çağırmak yeterli.

Kazanma olasılığı

Basit ama çok önemli bir formüldür. İnsan oyuncunun açık kart değeri, Glados’un toplam el değeri ve insan oyuncunun elinde Glados’un tur bitmeden bilemeyeceği kaç kart olduğu bilgileri gereklidir.

Öncelikle Glados’un bilebileceği deste ve bilinmeyen kart sayısına göre olabilecek bütün kombinasyonları içeren bir vectör vectörü oluşturulur. Oluşabilecek kombinasyon sayısı (s, toplam kart sayısı ve d insanın bilinmeyen kart sayısı) bastiçe matematikteki kombinasyon fonksiyonu ile bulunabilir. Bu fonksiyonu hatırlamayanlar için:

C(s,d) = s!/((s-d)!\*d!)

Fonksiyon, kombinasyon vectör vectöründeki vectörleri sırasıyla alır, alınan vectördeki kartları toplar ve Glados’un elinden iyi mi değil mi, yoksa eşit mi karar verir. Burada belirtilmesi gereken çok önemli bir nokta şudur ki eşit ellerin olması, Glados için sanki kazanacak el gibi davranmasını gerektirir çünkü en kötü ihtimalle tur berabere biter ve herkese ortaya koyduğu parayı geri alır, en iyi ihtimalle ise insan oyuncu çekilir ve ortadaki parayı Glados kapar. Eğer Glados’un eli iyiyse bir değişken, kötüyse başka bir değişken ve eşit ise başka bir değişken +1 alır. Fonksiyon sonunda Glados’un eşit veya kazandığı değişkenler bölü bütün değişkenlerin toplamı alınır ve bu, bize güncel kazanma olasılığını belirtir.

Tabii ki eğer Glados’un eli 21’den büyükse fonksiyon içerisinde ayrı bir yer çalıştırılır, burada sadece olabilecek değerin 21’den küçük veya büyük olduğuna bakılır. Eşit veya küçük ise Glados’un kaybettiği değişken +1’e uğrar, büyük ise Glados’un kazandığı değişken +1’e uğrar.

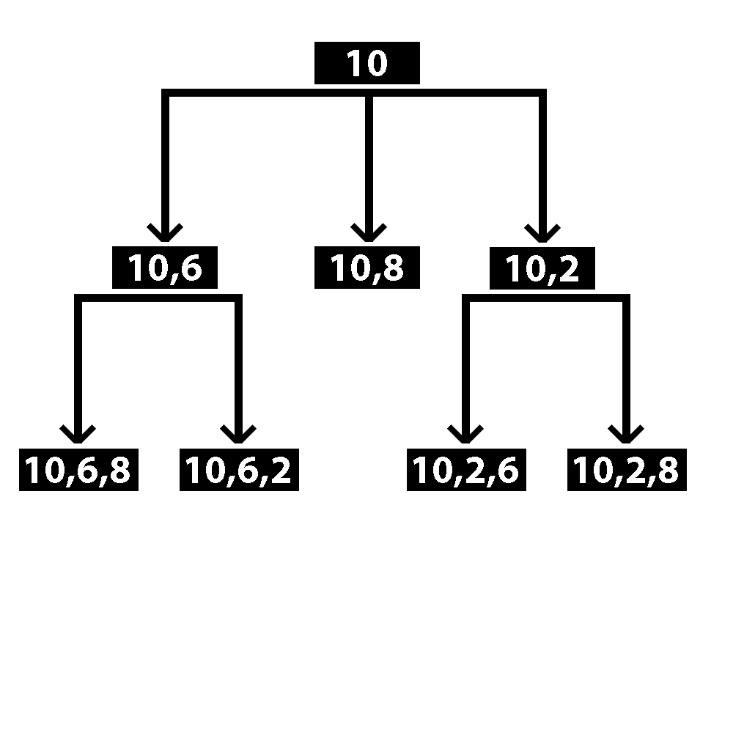
İnsan seçim ağacı

Not: Bu fonksiyonda işin hem mantık hem kod kısmı basit değildir. İlk olarak mantığını anlatacağım, sonra bunun recursive bir şekilde nasıl koda döktüğümü.

Glados her zaman olabildiğince yüksek bir el ile tura başlar, çünkü beklenen değer fonksiyonumuz var. Bu yüzden insan oyuncunun yapabileceği en mantıklı hamle, bir üst limite ulaşana kadar kart çekmektir.

Bu oyunu 5 senedir oynamış biri olarak söyleyebilirim ki, insanların “tamam, daha fazla kart verme” dediği nokta 18’dir. 18’e ulaşıncaya kadar da “genelde” kart çekerler.

Glados’un bilebileceği deste 9,8,2 olsun ve oyuncunun açık kartı 10 olsun. Böyle bir durumda insanın sahip olabileceği imajiner eller şu şekildedir:



Buradan itibaren bilgisayar mühendsiliğinde “ağaç” olarak bilinen veri yapısıyla ilgili terimler kullanmaya başlayacağım. Aşina değilseniz kısaca göz atmanız yeterli olacaktır.

Benim ağacımda her düğümün gerçekleşme olasılığı onun ebevyn düğümünün gerçekleşme olasılığı \* 1/kardeş sayısıdır. Bunun tek istisnası ilk düğümdür (bu durumda 10). Onun (Allah gibi) ne ebeveyni vardır, ne de kardeşten söz etmek mümkündür. Olasılığı 1’dir.

Mesela, 10,6,2 düğümünün gerçekleşme olasılığı 1/3 \* 1/2 yani 1/6’dır.

İlgilenmemiz gereken düğümler yaprak düğümleridir. Her bir yaprak düğümü, 18’e eşit veya büyüktür çünkü insanların 18’e ulaşıncaya kadar kart çektiğini varsayıyoruz. Şimdi yapılması gereken bu yaprakların hangilerinin Glados’un elinden üstün olacağını ve toplam olasılığını bulmak.

Glados’un el değeri 10+9 olsun. Sol alttaki 10,6,8 düğümüne baktığımızda bu düğümün 21’i aştığını görürüz, bu yüzden değeri 0’dır. Yani Glados’un leyhine bir olasılık bulunmuştur. Bu düğümün gerçekleşme olasılığı 1/6 olduğu için pozitif olasılık şuan 1/6’dır. Bunun gibi bütün yaprakları değerlendirirsek pozitif olasılığın 5/6 olduğunu buluruz. Yani bu demek olur ki, insan oyuncu beklenildiği gibi davranırsa 5/6 ihtimalle Glados galip gelecektir.

Beklenildiği gibi davranmayıp kart çekmezse zaten Glados her türlü galip gelir. Böyle bir durumda, Glados’un turun başında bahis arttırması gerek ki insan oyuncu patlamasına rağmen “en kötü ne olabilir ki, kör bahis dışında artış yok” deme lüksüne sahip olamasın. Bu, oyuncuyu çekilmeye zorlar.

Şimdi kod kısmına gelelim.

Fonksiyonu yazmadan önce yapılması gereken şey, düğüm adında bir sınıf oluşturmaktır çünkü her düğümün birden fazla özelliği bulunur: İçinde bulundurduğu kartlar, bu kartların değeri, gerçekleşme olasılığı, kardeş sayısı, ebevyninin gerçekleşme olasılığı, yaprak olup olmadığı.

Koda bakıldığında bu sınıf kolaylıkla anlaşılır, açıklama gereği duymuyorum.

Şimdi fonksiyona gelelim.

Her şeyden önce, oyun kuralları gereği her oyuncuya rızası dışında 2 kart dağıtıldığı için ilk katmanı oluşturmamız gerek. Bunun için humanTreeInitialFunction oluşturdum. Şimdi napıyor inceleyelim:

vector<handNode> humanTreeInitialFunction(Deck knownDeck, int playerOpenCardValue, vector<handNode> handNodeVector)  
{  
 vector<int> tempVector;  
 int numberOfSiblings = knownDeck.getNumberOfCards();  
  
 for(int i=0; i<numberOfSiblings; i++)  
 {  
 tempVector.push\_back(playerOpenCardValue);  
 tempVector.push\_back(knownDeck.getElementI(i));  
 handNodeVector.emplace\_back(numberOfSiblings,1.0,tempVector);  
 tempVector.clear();  
 }  
  
 return handNodeVector;  
}

tempVector, temporaryVector’ün kısaltılmış halidir. Bu vector, insan oyuncunun açık kartı ve for döngüsünde i indexinin denk geldiği sayıyı içerir. Her iterasyonda değiştiği için geçiçi vektör adını verdim.

Ağaç grafiğini ve bilinen desteyi hatırlayalım. Bilinen deste 6,8 ve 2 idi. Fonksiyondaki for döngüsü, sırasıyla sayıları yerleştirerek ilk katmandaki düğümleri oluşturmak için kullanılıyor. Kardeş sayısı, bilinen destedeki kart sayısına eşittir, ebeveynin olasılığı ise yukarıda bahsettiğimiz gibi 1’dir.

Böylelikle ilk katmanı oluşturmuş olduk. Şimdi recursive kısmına gelelim.

void humanTreeRecursiveFunction(vector<handNode> &handNodeVector, Deck knownDeck, int lastRunLastNodeID, int lastRunFirstNodeID, int limit)  
{  
 vector<int> tempVector;  
 vector<int> originalVector;  
 vector<int> knownDeckCopy;  
 knownDeckCopy = knownDeck.getCards();  
 originalVector = knownDeckCopy;  
 int nodesCreatedThisRun = 0;  
 int firstNodeID;  
 int lastNodeID;  
  
 for(int i=lastRunFirstNodeID; i<=lastRunLastNodeID; i++)  
 {  
 if(handNodeVector[i].value<limit)  
 {  
 knownDeckCopy = originalVector;  
  
 // Remove the cards that CAN'T be drawn, starting from the last element  
 for (auto it = handNodeVector[i].cardVector.begin() + 1; it != handNodeVector[i].cardVector.end(); ++it)  
 {  
 int cardThatWillBeRemoved = \*it; // Get the current card (excluding the open card value)  
 knownDeckCopy = eraseFunction(cardThatWillBeRemoved,knownDeckCopy);  
 }  
  
 for(int create=0; create<knownDeckCopy.size(); create++)  
 {  
 tempVector = handNodeVector[i].cardVector;  
 tempVector.push\_back(knownDeckCopy[create]);  
 handNodeVector.emplace\_back(knownDeckCopy.size(),handNodeVector[i].selfProbability,tempVector);  
 tempVector.clear();  
 if(nodesCreatedThisRun==0 && create==0)  
 {  
 firstNodeID = handNodeVector.size()-1;  
 }  
 nodesCreatedThisRun++;  
 }  
 knownDeckCopy = originalVector;  
 }  
 }  
 knownDeckCopy = originalVector;  
 lastNodeID = handNodeVector.size()-1;  
  
 if(nodesCreatedThisRun!=0)  
 {  
 humanTreeRecursiveFunction(handNodeVector,knownDeck,lastNodeID,firstNodeID,limit);  
 }  
  
}

nodesCreatedThisRun değişkeninin var olma sebebi, artık oluşturulabilecek düğüm kalmadığında fonksiyonu sonlandırmaktır.

lastRunFirstNodeID ve lastRunLastNodeID değişkenleri ise fonksiyon her çalıştığında nerde kaldığını ve hangi düğüme işlem yapması gerektiğini söyler.

Düğüm oluşturmak için, öncelikle düğümün içindeki kartların değerinin limit değerinden (bu değerin 18 olduğundan bahsetmiştik) küçük olması gerekir. Aksi takdirde oyuncu kart çekmeyecektir.

Şimdi remove the cards denen kısım ile başlayalım.

Her düğüm oluşturulurken, çekilebilecek kartların hesaplanması gerek. Bunun için, düğümdeki desteki kartları, bilinen desteden çıkarmamız gerekir. Auto iterator bu işlemi gerçekleştirir. +1’in sebebi, ilk kartın açık kart değeri olduğu ve bütün bu işlemlerden uzak tutulması gerektiğindendir.

(eraseFunction, vectörden istenilen sayıyı bulup silen, benim yazdığım bir fonksiyondur. Büyük ihtimalle c++’da bunu gerçekleştirmenin çok daha düzgün bir yolu vardır ama ben bulamadım)

Sonra değişkeni create olan döngüye geliriz. Create dememin sebebi, düğümlerin tam olarak bu döngü içerisinde oluşturulduğundandır. İlk olarak ebeveyn düğümün bütün kartlarını alırız, sonra da çekilebilecek kartlardan 0’dan başlayarak çekip yeni düğüm oluştururuz. Tabii ki bütün bu işlemlerde bilinen deste kopyası kullanırız ve her i iterasyonunda bilinen deste kopyasını orijinal haline getiririz.

İşlemler bittiğinde, lastNodeID’nin atamasını yaparız ve bütün bu işlemler fonksiyon yeniden çağrılarak tekrarlanır, ta ki oluşturacak düğüm bitene kadar.

Artık zor kısım bitmiştir. Bu iki fonksiyonu tek fonksiyona paketleyip yaprak düğümlerini bulmak kalmıştır.

vector<handNode> humanTreeFunction(Deck knownDeck, int playerOpenCardValue, vector<handNode> &handNodeVectorParameter, int satisfactionValue)  
{  
 vector<handNode> handNodeVector;  
 vector<handNode> finalNodeVector;  
 handNodeVector = humanTreeInitialFunction(knownDeck,playerOpenCardValue,handNodeVectorParameter);  
 humanTreeRecursiveFunction(handNodeVector,knownDeck,knownDeck.getNumberOfCards()-1,0,satisfactionValue);  
  
 for(int i=0; i<handNodeVector.size(); i++)  
 {  
 if(handNodeVector[i].finalHand)  
 {  
 finalNodeVector.push\_back(handNodeVector[i]);  
 }  
 }  
  
 return finalNodeVector;  
}

Ardından bu yaprakların olasılıkları ve değerleri göz önünde bulundurularak insan tatmin oluncaya kadar kart çekerse Glados’un kazanma olasılığı hesaplanır. Çalışma mantığı normal kazanma olasılığına aşırı benzerdir, sadece bu fonksiyonda her bir iterasyonda +1 yerine +DüğümünOlasılığı işlemi yapılır.

Bunların hepsi, Glados’un tur başında (tabii gerekiyorsa kart çektikten sonra) bahis arttırıp arttırmama seçimi içindir. İnsan oyuncu kart çekmezse zaten (muazzam yüksek ihtimalle) Glados’un elinden düşük bir ele sahip olacağı için Glados kazanır, çekmek isterse az önce anlattığım ağaç oluşur. Böylelikle Glados nadiren de olsa blöf yapabilir. Neden mi? İnsan oyuncu tur başında bahis artışını gördüğü anda iyi bir ele sahip olmak ister (çünkü Glados kazanma olasılığı düşük ise bahis arttırmaz), dolayısıyla kart çeker ve patladığı anda çekilmek ister, boş yere daha fazla para kaybetmenin anlamı yoktur.

İnsiyatif kullanılan alanlar

İdeal bir bot, tamamen objektif bir şekilde oyun teorisi ve olasılık formüllerine ve kavramlarına dayalı çalışır. Bu metnin başında söylediğim gibi, bunu ilkeyi tamamen tatmin edecek kadar bilgim yok, şimdilik bu bilgiyi edincek vaktim de yok.

Bu yüzden deneme-yanılma yöntemi ile koyduğum değişkenlerden ve oranlardan bahsetmek istiyorum. Bu subjektif değerleri mor renk ve kalın italik font ile belirteceğim.

Bunların ilki ve de belki de en önemlisi gladosStandFunction’dır.

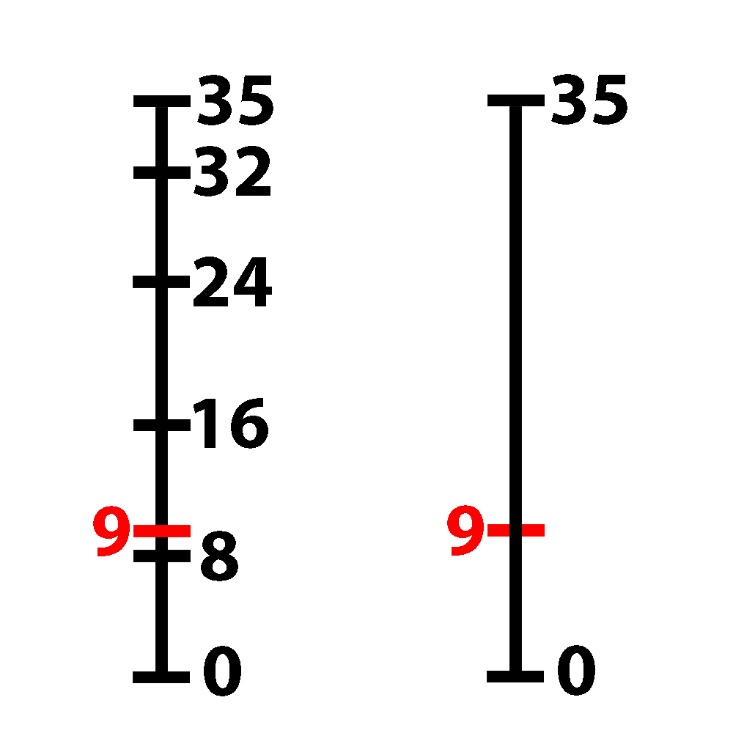
Glados, kendisinden ne kadar emin ise o kadar yüksek bahis görmelidir (***Kazanma olasılığı 0.5’ten yüksek olduğu sürece***). Bundan kastettiğim, olabilecek maksimum bahis artışı \* kazanma olasılığıdır. Yani kazanma olasılığı 0.69 ise ve maksimum bahis artışı 35 ise, Glados 35\*0.69 yani 24 para artışına kadar durmalı, bundan büyük bir bahis artışı görürse çekilmelidir.

İlk bakışta kusursuz gözükse de, bu Glados’un inanılmaz korkak olmasına yol açar, çünkü kazanma olasılığı çok düşük bir ihtimalle 1 olacağı için insan oyuncu her seferinde bahsi 35 arttırıp Glados’u çekilmeye zorlayabilir. Bunun önlenmesi için compensation adlı bir değişken ekledim. Compensation, Glados’un ne kadar ekstra cesarete sahip olması gerektiğini belirtir. Bu değişken olmazsa, kolaylıkla suistimal edilebilir. Değişkenin formülünü körBahis / başlangıçParası olarak belirledim. Fark ettiyseniz körBahisi yukarıya yazdım, çünkü körBahis ne kadar büyükse Glados’un “ama kazanma olasılığım yeterli değil” deme lüksü ortadan kalkar, çünkü mükemmel veya çok iyi bir kazanma olasılığını bekleyene kadar zaten iflas edecektir. başlangıçParası büyükse ise bu değişkene çok ihtiyacımız yoktur, çünkü oyun zaten uzun sürecek demektir.

Analog-dijital bahis limiti kavramı ise kafa karıştırmasın, basitçe Glados’un bahis görme toleransını eğer mümkünse yuvarlıyor. Şu şekilde çalışıyor:

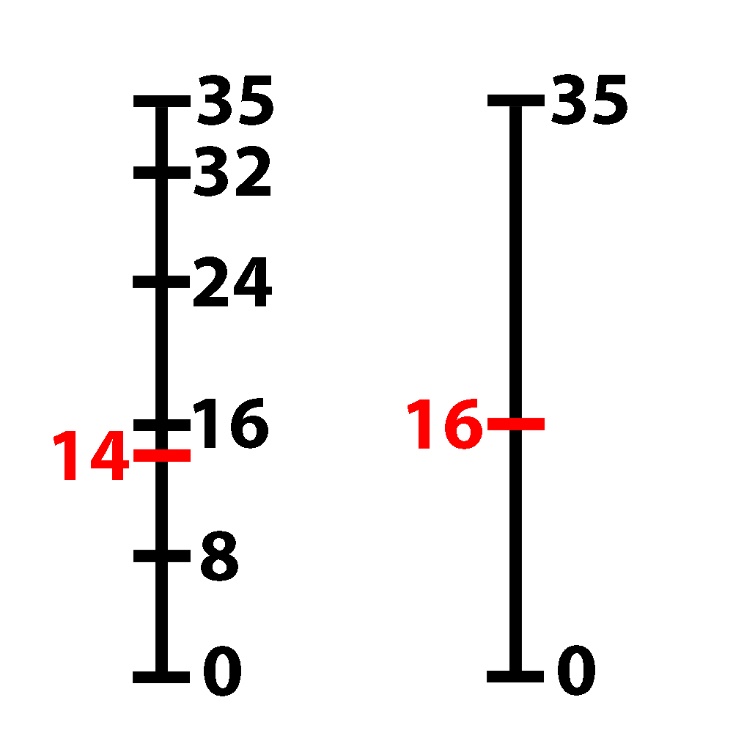
Bir sayı doğrusu hayal edelim. Başlangıç değeri 0, bitiş değeri oyundaki maxBahisArtışı-körBahis kadar olsun. Bu sayı doğrusunu ***4***’e bölelim. Glados’un ilk hesaplanılan bahis toleransına programda analogToleranceLimit diye belirttim. Eğer analogToleranceLimit, arasında bulunduğu aralığın yüksek olan değerine yakın ise yüksek olan değere yuvarlanmalı, değil ise olduğu gibi kalmalıdır.

Akılda daha kolay canlanması için:



Eğer analogToleranceLimit 9 ise, arasında bulunduğu 8-16 aralığının küçük olan değerine yakın olduğu için değeri 9 olarak kalacaktır.

Eğer 14 olsaydı:



8-16 aralığının 16 değerine yakın olduğu için 16 değerine yuvarlanırdı ve Glados’un tolere edebileceği maksimum bahis 16 olurdu.

Bir diğer mevzu ise Glados’un blöf yapıp yapmaması gerektiği. Üzerinde çok düşünsem de, iki taraf için de kesin ve net bir argümana varamadığım için olması gerektiğini ve bunun ***0.2*** ihtimalle olmasıdır. Bunun sağladığı şey, insan oyuncunun kafasını karıştırmak ve psikolojisiyle oynamaktır. İnsanlar (her ne kadar rasyonel olduğumuzu düşünmeyi sevsek de) genelde duygu ağırlıklı hareket ederler. İnsan oyuncu çekildiyse ve Glados’un patlamış elini gördüyse, kendisini kandırılmış hisseder ve bir dahaki bahis artışına (ki, bu bahis artışının gerçek olma ihtimali epey yüksektir), çekilmek istemez ve gene kaybedince (çünkü Glados kazanma olasılığı ***0.75***’den az ise bahis arttırmaz) bu sefer daha da öfkeye kapılır.