

Lógica cuántica en Perl y su efecto sobre el juego organizado

Casino Cuántico

La superposición del mundo de la física cuántica estará pronto disponible en el núcleo de Perl 6, en las llamadas “yunciones”. En la actualidad, ya hay disponible un módulo. En el artículo de este mes le echaremos un vistazo a este revolucionario concepto haciendo uso de un script que juega al blackjack.

POR MICHAEL SCHILLI



Recientemente estuve en una librería ojeando algunos títulos cuando encontré uno titulado “Winning Casino Play” [2], donde un jugador profesional explicaba como jugar en los casinos de Las Vegas y de Atlantic City con una alta probabilidad de ganar. El Blackjack era uno de los temas tratados por el autor en detalle, ya que es uno de los juegos, dentro del casino, que ofrece una de las mejores probabilidades de ganar.

19, 20, 21... ¡Me pasé!

Decidí ponerlo en práctica antes de irme a probar suerte en el casino, así que me puse a escribir un script en Perl que simulara el juego. Las reglas son las siguientes: el juego lo juegan el jugador y el crupier. Se usan varias barajas de 52 cartas. Las cartas son extraídas de una en una del dispensador de cartas.

El objetivo del juego es tener todas las cartas que sean posibles para conseguir una mano de 21. Pero cuidado: si te pasas de 21, automáticamente pierdes. Esto es conocido como “busted”. Las cartas del 2 al 10 tienen como valor el número indicado, y las figuras (la Jota, la Reina y el Rey) valen

10 puntos cada una. Un As puede valer 1 u 11.

¡Cuidado con la Superposición!

¿1 u 11? De acuerdo, si le sale un 7, un 8 y un As, $7 + 8 + 11 = 26$, se pasa. Pero se puede quedar en el juego contando el As como un 1: $7 + 8 + 1 = 16$.

En otras palabras, es una mano que no tiene un valor fijo, sino dos estados solapados (26, 16), del que podemos quedarnos con el más favorable: 16. Imagínese que tiene cuatro Ases. Esto supondría 4 posibles estados (4, 14, 24, 34). Pero como 24 y 34 no son exactamente manos ganadoras, tan sólo estaremos interesados en dos de ellos: 4, la “mano suave” y 14 la “mano dura”.

Los estados solapados son conocidos como superposiciones en física cuántica y permite a una partícula ocupar dos lugares al mismo tiempo.

Después de instalar el módulo *Quantum::Superpositions* de Damian Conway (disponible en CPAN), se pueden crear una superposición de valores numéricos como los mostrados anteriormente, llamando a la función *any()*:

```
use Quantum::Superpositions;
my $count = any(4,14,24,34);
```

A partir de este punto, la variable *\$count* parecerá tener cuatro valores diferentes. Una expresión lógica como la siguiente

```
if($count == 4 and $count == 14){
    print "¡Correcto!\n";
}
```

devolverá verdadero y ejecutará el comando *print*. Además, comparaciones lógicas como *\$count <= 21* ya no serán más verdaderas o falsas en el contexto de *Quantum::Superpositions*, sino que devolverán una superposición de los estados que cumplan la condición. La sintaxis siguiente filtrará los estados que son irrelevantes mayores que 21, de los posibles resultados que dependen de como contemos el As (4, 14, 24, 34):

```
$count = ($count <= 21);
```

Ahora *\$count* contiene sólo *any(4, 14)*. ¡Qué ahorra un montón de tecleo! Las operaciones aritméticas basadas en la superposiciones son bastante simples de

formular ya que `Quantum::Superpositions` sobrecarga todos los operadores. Si el valor `any(4, 14)` está almacenado en `$count`,

```
$count += 10;
```

devolverá como resultado `any(14, 24)`.

Así pues, `any()` especifica la llamada superposición disyuntiva que puede ser consultada para todos los estados y responderá correctamente a las consultas lógicas de cualquiera de sus estados que la cumpla.

La segunda función presentada por `Quantum::Superpositions` es `all()`, que define una superposición conjuntiva que tiene todos los estados simultáneamente. La siguiente condición no es verdadera:

```
my $count = all(4,14,24,34);

if($count <= 21) {
    print "Todos válidos<\>\n";
}
```

al no tener todos los estados valores menores que 21. Por el contrario, `all(4, 14) <= 21` devolvería verdad.

¿Está el Gato Muerto?

Si un investigador comprueba la verdad de la física cuántica en el mundo real, los

estados múltiples colapsarían en un único estado, eliminando la ambigüedad: Esto significa que el gato de Schrödingers [3] debería estar muerto o vivo y coleando.

Esto es diferente en Perl: se puede jugar con los estados ambiguos sin destruir el sistema. Para descubrir los estados que una superposición contiene, `Quantum::Superpositions` importa la función `ownstates()`, que simplemente devuelve una lista de estados:

```
my @counts = ownstates($count);
```

Estas tres funciones `any()`, `all()` y `ownstates()` permiten al desarrollador escribir construcciones dentro del programa impresionantes. La siguiente sintaxis es lo único que se necesita para descubrir la “mano suave” para las cartas especificadas, o sea, el mínimo de `any(4, 14, 24, 34)`:

```
my $counts = any(4, 14, 24, 34);
my $soft = ($counts <= all(
    ownstates($counts)));
```

la comparación lógica usa `$counts` para devolver una superposición de todos los estados que son menores o iguales que

todos los estados de la superposición. Esta es la definición clásica de mínimo.

La Clase Shoe en Perl

La implementación: El listado `Blackjack.pm` (por motivos de espacio, este listado sólo está accesible a través de la web en [1]) contiene dos clases: `Blackjack::Shoe`, que representa el dispensador de cartas del que el repartidor extrae las cartas que van a ser jugadas, y `Blackjack::Hand`, que representa o bien la mano del jugador o del crupier.

La clase `Shoe` usa el módulo `CPAN Algorithm::GenerateSequence` para generar unas cuantas barajas de cartas. El constructor `new()` acepta referencias a un array, cuyos elementos combina. Si el primer array contiene todos los palos (Corazones/Diamantes/Picas/Tréboles) y el segundo todos los valores (A 2 3 4 5 6 7 8 9 10 J Q K) del juego de cartas, el método `as_list()` devolverá una lista de combinaciones que coincidirán con las cartas: As de Corazones, 2 de Corazones, ... Q de Trébol, K de Trébol. La lista creada es replicada por el operador `x` para reflejar el número de cartas que contendrá el dispensador (línea 29).

El módulo `Algorithm::Numerical::Shuffle` exporta el método `shuffle`. Este

Listado 1: blackjack

```
01 01 #!/usr/bin/perl                               $shoe);
02 #####                                           19 my $dealer =
03 # Jugar al blackjack con                         Blackjack::Hand->new(shoe =>
    crupier                                         $shoe);
04 # Mike Schilli, 2003                             20 $dealer->draw();
    (m@perlmeister.com)                          21 P(RED, "D", $dealer);
05 #####                                           22 $dealer->draw();
06 use warnings; use strict;                       23 $player->draw();
07 use Blackjack;                                  24 $player->draw();
08 use Term::ANSIColor                             25 while(!$player->busted()) {
09 qw(:constants);                                26 P(BLUE, "P", $player);
10 use Term::ReadKey;                              27 print
11 $| = 1; my $total = 0;                          28 "([Hit/[S]tand/[Q]uit) ";
12 my $shoe =                                       29 ReadMode 4;
    Blackjack::Shoe->new(nof_decks                30 my $move = ReadKey(0);
    => 4);                                         31 ReadMode 0;
13 {                                               32 print "\r";
14 if($shoe->remaining() < 52) {                   33 last if $move =~ /^s/i;
15 print "Shuffling ... \n";                       34 exit 0 if $move =~ /^q/i;
16 $shoe->reshuffle();                              35 $player->draw();
17 }                                               36 }
18 my $player =                                     37 P(BLUE, "P", $player);
    Blackjack::Hand->new(shoe =>                   38 while(!$dealer->busted() and
                                                $dealer->count("soft") < 17) {
                                                39 P(RED, "D", $dealer);
                                                40 $dealer->draw();
                                                41 }
                                                42 P(RED, "D", $dealer);
                                                43 $total +=
            $player->score($dealer);              44 print "Score: ",
                                                45 $player->score($dealer),
                                                46 ", Total: ", $total,
                                                47 "\n\n";
                                                48 redo;
                                                49 }
                                                50 sub P { # Print status in
            color                                  51 print(BOLD, $_[0],
                                                52 "$_[1]", "[",
                                                53 $_[2]->count_as_string(),
                                                54 "]",
                                                55 RESET, ": ",
            $_[2]->as_string(), "\n")              56 }
```

método baraja los elementos del array que se le pasa por el argumento, usando el método de Fisher-Yates. El método `reshuffle()` del objeto `Blackjack::Shoe` coloca las 52 cartas definidas en la variable de instancia `nof_decks` en el dispensador de cartas, `shoe`.

`remaining()` devuelve el número de cartas que quedan en el dispensador; esto permite al repartidor comprobar que el juego puede ser completado con las cartas restantes. `draw_card()` extrae una carta del dispensador y devuelve una referencia a un array, cuyo primer elemento es el palo (Corazones/Diamantes/Picas/Tréboles) y el segundo elemento es el valor de la carta (A, 2, 3, ..., J, Q, K).

La clase `Blackjack::Hand` representa la mano que está jugando un jugador. Trata con las cartas que se han sacado y sus valores. No importa si la mano pertenece al jugador o al repartidor. El constructor

```
Blackjack::Hand->new(shoe->
$shoe)
```

liga el dispensador del que se obtendrán las cartas con el jugador/repartidor. El método `draw()` añade una carta nueva a la mano.

Usando la superposición para contar la puntuación

El método `count()` de la línea 64 y siguientes cuenta la puntuación de la mano y devuelve el resultado, como una superposición, una “mano dura” `$hand->count("hard")` o una “suave” `$hand->count("soft")`.

Para permitir esto, el bucle `for` de la línea 68 y siguientes itera sobre las cartas, comprobando los valores y sumándolos. La variable `$count` almacena la superposición de las puntuaciones potenciales de la mano. Hace uso de la sobrecarga del operador `+` de `Quantum::Superpositions`, así que `$counts += 10` añadirá 10 a todas las superposiciones de `$counts`. Cuando un As es extraído, el número de superposiciones es doblado, sumándole un uno a la mitad de ellos y 11 a la otra mitad.

```
$counts = any($counts+1,
$counts+11);
```

La línea 79 elimina inmediatamente cualquier puntuación superior a 21 de las

superposiciones. Si una superposición no tiene valores después de esta operación, o sea, `ownstates($counts)` devuelve una lista vacía, significa que el jugador ha excedido la máxima puntuación de 21 y se ha pasado. El método `count()` de la línea 83 y siguientes descubre las manos “suaves” y “duras”, usando el truco visto anteriormente para encontrar los valores máximos y mínimos. La función `int()` convierte la superposición en un valor escalar.

Gana Blackjack

Si la mano de un jugador contiene exactamente un As y una carta cuya puntuación es 10, ha conseguido un Blackjack, y gana a cualquier otra mano que sume 21. El método `blackjack()` definido en la línea 94 y siguientes comprueba si se da esta situación, chequeando si el valor de la mano corresponde a una superposición con los valores 11 y 21, y si la mano contiene exactamente dos cartas.

El método `score()` averigua si la mano ha ganado o perdido contra el objeto `Blackjack::Hand` del crupier, que lo acepta como parámetro. Si el resultado es negativo, el jugador pierde. Si es positivo, gana. Echémosle ahora un vistazo a algunos casos especiales: El jugador de Blackjack paga 1:1.5, mientras que el repartidor sólo consigue ventajas frente a la apuesta del jugador. Si el jugador tiene más de 21, se ha pasado, y pierde incluso si el repartidor también excede después la puntuación de 21.

Texto en Color y Entradas por Teclado

El listado `blackjack` contiene un script que le permite jugar contra un repartidor computerizado al Blackjack, como en Las Vegas. Usa las clases definidas en `Blackjack.pm` para el dispensador de cartas, clase `Shoe`, y las manos del jugador y del repartidor, clase `Hand`. `Text::ANSIColor` se usa para proporcionar la salida coloreada; el módulo exporta constantes como `BOLD`, `RED`, `BLUE` o `RESET`, que está etiquetada como la secuencia de escape del terminal, basado en los valores de las constantes pasadas como `:constants`. Esto permite tener la salida en “negrita” o coloreada, o volver al modo normal.

`Text::ReadKey` en modo `Raw` (iniciado por `ReadMode 4`) permite capturar los

valores de las teclas pulsadas, sin la necesidad de que el usuario tenga que pulsar la tecla `Enter`. Pasando a `ReadMode 0` devuelve el terminal al modo `Cooked`, donde las entradas deben consistir en líneas completas, si el usuario pulsa `Enter` y ninguna otra tecla está pulsada provoca una reacción del programa.

Cuando es el turno del usuario, la siguiente salida es mostrada en pantalla:

```
[H]it/[S]tand/[Q]uit
```

El jugador puede optar por `Hit` (extrae otra carta), `Stand` (se planta) y `Quit` (se sale del juego), pulsando las teclas `H`, `S` o `Q`.

La Figura 1 muestra una jugada típica. El crupier empieza el juego sacando dos cartas, aunque tan solo una de ellas es mostrada boca arriba. Al jugador se le sirve dos cartas y se le pregunta si desea una carta nueva, para acumular puntos en su mano. Si opta por plantarse, el crupier seguirá un patrón fijo establecido para su propia mano. Si la puntuación total “suave” es menor que 17, tomará una carta nueva. 17 o más puntos, provocará que se plante, en ningún momento está influenciado por la mano del jugador y por ello el crupier tiene que pagar o recaudar.

El juego `Blackjack.pm` puede ser utilizado libremente. Podemos añadir una GUI o escribir un servidor TCP/IP que juegue al Blackjack a través de la red. Disfrútalo bien y como se dice en Las Vegas, “¡Qué tengas buena suerte!”. ■

RECURSOS

- [1] Listado de este artículo: <http://www.linuxmagazine.com.es/descargas>
- [2] Avery Cardoza, “Winning Casino Play”, Cardoza Publishing, 3rd Ed., 2003, 1-58042-090-7
- [3] Ilustración del experimento de Schrödinger con su gato: http://mist.npl.washington.edu/npl/int_rep/tiqm/Tl_fig_09.html

EL AUTOR

Michael Schilli trabaja como ingeniero Web para AOL/Netscape en Mountain View, California. Escribió “Perl Power” para Addison-Wesley y puede ponerse en contacto con él en mschilli@perlmeister.com. Su página web es <http://perlmeister.com>.