# Recorrido del Caballo

Generado por Doxygen 1.8.1

Viernes, 15 de Junio de 2012 13:03:58

# Índice



## 1. Índice de clases

### 1.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:



## 2. Documentación de las clases

### 2.1. Referencia de la Clase Caballo

#### Métodos públicos

- **Caballo** (int fila_inicial, int col_inicial, **Tablero** &tablero)
- void **reset** (int fila_inicial, int col_inicial)
- bool **salto** ()
- int **recorridoCompleto** ()
- void **recorridosExitosos** (int intentos, int &exitos, int &media_mov)
- void **imprimeCasillasVisitadas** ()

#### Métodos privados

- int **eligeMov** (int indices_mov[], int num_mov_validos)
- bool **movPosible** (int i)
- int **calculaMinimiAccesibilidad** (int fila, int col)

**Atributos privados**

- **Tablero** ∗ **tab**
- int **num_mov**
- int **fila_actual**
- int **col_actual**

**Atributos privados estáticos**

- static const int **mov_col** [8] = {2,1,-1,-2,-2,-1,1,2}
- static const int **mov_fila** [8] = {-1,-2,-2,-1,1,2,2,1}

### 2.1.1. Descripción detallada

Clase **Caballo** (p. 1)

### 2.1.2. Documentación del constructor y destructor

#### 2.1.2.1. Caballo::Caballo ( int *fila_inicial,* int *col_inicial,* Tablero & *tablero* )

Constructor de la clase. Se inicializa la semilla de rand() con la hora del sistema para generar numeros aleatorios.

**Parámetros**

| | |
|---|---|
| *fila_inicial* | fila inicial donde se situara el caballo |
| *col_inicial* | columna inicial donde se situara el caballo |
| *&tablero* | objeto de la clase tablero |

```
{
   tab = &tablero;

   reset(fila_inicial,col_inicial);

   time_t t;
   srand ((int) time(&t));
}
```

### 2.1.3. Documentación de las funciones miembro

#### 2.1.3.1. int Caballo::eligeMov ( int *indices_mov[],* int *num_mov_validos* ) [private]

Metodo para eligir el salto del caballo dependendiendo de la accesibilidad de las casillas. NOTA: si esta desactivada la heuristica de mirada hacia delante se deshace el empate de forma aleatori.

**Parámetros**

| | |
|---|---|
| *indices_mov[]* | array con los indices de los movimientos validos |
| *num_mov_validos* | contador con el numero de indices validos |

**Devuelve**

indice del salto elegido

```
{
      int mov_valido = indices_mov[0];
      int nueva_fila = fila_actual + mov_fila[mov_valido];
      int nueva_col = col_actual + mov_col[mov_valido];
      int accesibilidad = tab->casillas_accesibilidad[nueva_fila][nueva_col];
```

```
      for (int i = 1; i < num_mov_validos; ++i)
      {
         int posible_nueva_fila = fila_actual + mov_fila[indices_mov[i]];
         int posible_nueva_col = col_actual + mov_col[indices_mov[i]];
         int nueva_accesibilidad = tab->casillas_accesibilidad[
      posible_nueva_fila][posible_nueva_col];
         bool salto_nuevo_valido = false;

         if (accesibilidad == nueva_accesibilidad)
         {

            if(tab->mirada_hacia_delante)
            {
               int acces_nueva = calculaMinimiAccesibilidad(posible_nueva_fila,
      posible_nueva_col);
               int acces_vieja = calculaMinimiAccesibilidad(nueva_fila,
      nueva_col);

               if (acces_nueva < acces_vieja)
               {
                  salto_nuevo_valido = true;
               }
               else if (acces_nueva == acces_vieja)
               {
                  salto_nuevo_valido = rand() % 2;
               }
            }

            else
            {
               salto_nuevo_valido = rand() % 2;
            }
         }

         if (nueva_accesibilidad < accesibilidad || salto_nuevo_valido)
         {
            accesibilidad = nueva_accesibilidad;
            nueva_fila = posible_nueva_fila;
            nueva_col = posible_nueva_col;
            mov_valido = indices_mov[i];
         }
      }

      return mov_valido;
}
```

#### 2.1.3.2. bool Caballo::movPosible ( int *i* ) [private]

Metodo para determinar si el mov[i] es posible

**Parámetros**

| | |
|---|---|
| *i* | indice del movimiento |

**Devuelve**

exito o fracaso del posible salto

```
{
   bool mov_posible = false;

   int nueva_fila = fila_actual + mov_fila[i];
   int nueva_col = col_actual + mov_col[i];

   if (nueva_col>=0 && nueva_col<=7 && nueva_fila>=0  && nueva_fila<=7
      && tab->casillas_visitadas[nueva_fila][nueva_col]==0)
   {
      mov_posible = true;
   }

   return mov_posible;
}
```

#### 2.1.3.3. int Caballo::calculaMinimiAccesibilidad ( int *fila,* int *col* ) [private]

Metodo para calcular la casilla con menor accesibilidad desde una fila y columna cualquiera

**Parámetros**

| | |
|---|---|
| *fila* | fila desde donde se calcula la accesibilidad |
| *col* | columna desde donde se calcula la accesibilidad |

**Devuelve**

minima accesibilidad

```
{
   /* 1º Almacenamos la posicion del caballo antes de saltar
    *    a la casilla desde donde queremos calcular la accesibildad
    * 2º Calculamos los indices de movimientos posibles de dicha casilla
    * 3º Elegimos aquel movimiento que tenga menor accesibilidad.
    *    (si al comparar dos movimientos tienen la misma, deshacemos
    *    el empate de forma aleatoria)
    */

   int fila_actual_guardada = fila_actual;
   int col_actual_guardada = col_actual;

   fila_actual = fila;
   col_actual = col;

   int indices_mov[tab->DIM];
   int num_mov_validos = 0;

   for (int i = 0; i < tab->DIM; ++i)
   {
      if (movPosible(i)){
         indices_mov[num_mov_validos] = i;
         num_mov_validos++;
      }
   }

   int nueva_fila = fila_actual + mov_fila[indices_mov[0]];
   int nueva_col = col_actual + mov_col[indices_mov[0]];
   int menor_accesibilidad = tab->casillas_accesibilidad[nueva_fila][nueva_col]
      ;

   for (int i = 1; i < num_mov_validos; ++i)
   {
      nueva_fila = fila_actual + mov_fila[indices_mov[i]];
      nueva_col = col_actual + mov_col[indices_mov[i]];
      int nueva_accesibilidad = tab->casillas_accesibilidad[nueva_fila][
      nueva_col];
      bool nueva_access_valida = false;

      if (nueva_accesibilidad == menor_accesibilidad)
      {
         nueva_access_valida = (int)(2.0*rand()/(RAND_MAX+1.0));
      }
      else if (nueva_accesibilidad < menor_accesibilidad || nueva_access_valida
      )
      {
         menor_accesibilidad = nueva_accesibilidad;
      }
   }

   fila_actual = fila_actual_guardada;
   col_actual = col_actual_guardada;

   return menor_accesibilidad;
}
```

### 2.1.3.4. void Caballo::reset ( int *fila_inicial*, int *col_inicial* )

Metodo para establecer la posicion inicial del caballo, resetear la matriz de casillas visitas y reinicia el contador de movimientos a cero

**Parámetros**

| | |
|---|---|
| *fila_inicial* | fila inicial donde se situara el caballo |
| *col_inicial* | columna inicial donde se situara el caballo |

```
{
   num_mov = 0;
```

```
   fila_actual = fila_inicial;
   col_actual = col_inicial;

   for (int i = 0; i < tab->DIM; ++i)
   {
      for (int j = 0; j < tab->DIM; ++j)
      {
         tab->casillas_visitadas[i][j]=0;
      }
   }
}
```

#### 2.1.3.5. bool Caballo::salto ( )

Metodo para desplazar el caballo un salto

**Devuelve**

exito o fracaso del salto

Almacenamos en un array los indices de los movimientos validos y elegimos uno de esos indices segun la heuristica

```
{
   bool salto_valido = true;
   int indices_mov[tab->DIM];
   int num_mov_validos = 0;

   for (int i = 0; i < tab->DIM; ++i)
   {
      if (movPosible(i)){
         indices_mov[num_mov_validos] = i;
         num_mov_validos++;
      }
   }

   if (num_mov_validos == 0)
   {
      salto_valido = false;
   }
   else
   {
      int mov_valido;

      if (tab->accesibilidad_por_casillas)
      {
         mov_valido = eligeMov(indices_mov,num_mov_validos);
      }
      else
      {
         mov_valido = indices_mov[rand() % num_mov_validos];
      }

      tab->casillas_visitadas[fila_actual][col_actual] = 1;
      fila_actual += mov_fila[mov_valido];
      col_actual += mov_col[mov_valido];

      num_mov++;
   }

   return salto_valido;
}
```

#### 2.1.3.6. int Caballo::recorridoCompleto ( )

Metodo para realizar un recorrido completo por todas las casillas pasando una sola vez

**Devuelve**

numero de movientos realizados

```
{
   int movimientos = 0;

   while (salto() && movimientos<64)
   {
      movimientos++;
```

```
   }

   return movimientos;
}
```

#### 2.1.3.7. void Caballo::recorridosExitosos ( int *intentos,* int & *exitos,* int & *media_mov* )

Metodo que averigua el numero de recorridos exitosos y la media de los movimientos realizados en un numero de intentos

**Parámetros**

| | |
|---|---|
| *intentos* | numero de intentos a provar |
| *&exitos* | numero de exitos conseguidos, devuelto por referencia |
| *media_mov* | media de los desplazamientosa, devuelto por referencia |

```
{
   exitos = 0;
   media_mov = 0;

   int fila_inicial = fila_actual;
   int col_inicial = col_actual;

   for (int i = 0; i < intentos; ++i)
   {
      reset(fila_inicial,col_inicial);
      int mov_realizados = recorridoCompleto();

      if (mov_realizados == 63)
      {
         exitos++;
      }

      media_mov += mov_realizados;
   }

   media_mov = media_mov/intentos;
}
```

#### 2.1.3.8. void Caballo::imprimeCasillasVisitadas ( )

Metodo que imprime el tablero mostrando las casillas por donde ha pasado el caballo

```
{
   cout « " ·) Posicion final del caballo: (" « fila_actual « "," «
      col_actual « ")" « endl;
   cout « endl;

   for (int i = 0; i < tab->DIM; ++i)
   {
      cout « "  ( ";
      for (int j = 0; j < tab->DIM; ++j)
      {
         if (i==fila_actual && j==col_actual)
            cout « "  ";
         else
            cout « tab->casillas_visitadas[i][j] « " " ;
      }
      cout « ")" « endl;
   }

   cout « endl;
}
```

### 2.1.4. Documentación de los datos miembro

#### 2.1.4.1. const int Caballo::mov_col = {2,1,-1,-2,-2,-1,1,2} [static],[private]

Vector con los posibles movimientos horizontales

#### 2.1.4.2. const int Caballo::mov_fila = {-1,-2,-2,-1,1,2,2,1} [static],[private]

Vector con los posibles movimientos verticales

#### 2.1.4.3. Tablero* Caballo::tab [private]

Puntero a la casillas visitadas del tablero y a la heuristica

#### 2.1.4.4. int Caballo::num_mov [private]

Contador con el numero de desplazamientos del caballo

#### 2.1.4.5. int Caballo::fila_actual [private]

Fila donde se encuentra el caballo

#### 2.1.4.6. int Caballo::col_actual [private]

Columna donde se encuentra el caballo

## 2.2. Referencia de la Clase Tablero

**Métodos públicos**

- **Tablero** ()
- void **setHeuristicaAccesibilidad** ()
- void **setHeuristicaMirada** ()

**Atributos privados**

- int **casillas_visitadas** [**DIM**][**DIM**]
- bool **accesibilidad_por_casillas**
- bool **mirada_hacia_delante**

**Atributos privados estáticos**

- static const int **DIM** = 8
- static const int **casillas_accesibilidad** [**DIM**][**DIM**]

**Amigas**

- class **Caballo**

### 2.2.1. Descripción detallada

Clase **Tablero** (p. 7)

### 2.2.2. Documentación del constructor y destructor

#### 2.2.2.1. Tablero::Tablero ( )

Constructor de la clase

```
                {
   accesibilidad_por_casillas = false;
   mirada_hacia_delante = false;
}
```

### 2.2.3. Documentación de las funciones miembro

#### 2.2.3.1. void Tablero::setHeuristicaAccesibilidad ( )

Metodo para activar la heuristica de accesibilidad

```
                                        {
   accesibilidad_por_casillas = true;
}
```

#### 2.2.3.2. void Tablero::setHeuristicaMirada ( )

Metodo para activar la heuristica de mirada hacia delante y comprueba que la heuristica de accesibilidad este activada

```
                                    {
   if(!accesibilidad_por_casillas){
      setHeuristicaAccesibilidad();
   }

   mirada_hacia_delante = true;
}
```

### 2.2.4. Documentación de los datos miembro

#### 2.2.4.1. const int Tablero::DIM = 8 [static], [private]

Dimension del tablero.

#### 2.2.4.2. int Tablero::casillas_visitadas[DIM][DIM] [private]

Matriz que muestra si la casilla se ha visitado (1) o no se ha visitado (0)

#### 2.2.4.3. const int Tablero::casillas_accesibilidad [static], [private]

**Valor inicial:**

```
{
   {2,3,4,4,4,4,3,2},
   {3,4,6,6,6,6,4,3},
   {4,6,8,8,8,8,6,4},
   {4,6,8,8,8,8,6,4},
   {4,6,8,8,8,8,6,4},
   {4,6,8,8,8,8,6,4},
   {3,4,6,6,6,6,4,3},
   {2,3,4,4,4,4,3,2}
}
```

Matriz que muestra la heuristica de accesibilidad: en cada casilla se escribe desde cuantas posiciones se puede llegar hasta ella

#### 2.2.4.4. bool Tablero::accesibilidad_por_casillas [private]

Activa o desactiva la heuristica de accesibilidad: elegir primero las casillas mas inaccesibles

#### 2.2.4.5. bool Tablero::mirada_hacia_delante [private]

Activa o desactiva la heuristica de mirada hacia delante: deshacer el empate para casillas con la misma inaccesibilidad

## Índice alfabético