

Abejas y tetris

1

Generated by Doxygen 1.8.13



# Contents



## Chapter 1

# README

Jose Ricardo Rodriguez Abreu



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">abeja</a>	La estructura abeja . . . . .	??
<a href="#">param</a>	Parametros de threads . . . . .	??
<a href="#">pieza</a>	La estructura PIEZA contiene datos basicos de ella . . . . .	??
<a href="#">punto</a>	La estructura punto . . . . .	??
<a href="#">tablero</a>	La estructura TABLERO contiene datos basicos de el . . . . .	??





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

lib/abc.c	File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/abc.h	File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/abejas.c	File containing the Artificial bee definitions for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/abejas.h	File containing the Artificial bee definitions for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/dump.c	File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/dump.h	File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/funcion.c	File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/funcion.h	File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/lock.h	File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/pieza.c	File containing the struct and funtions to simulate Tetris pieces for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/pieza.h	File containing the struct and funtions to simulate Tetris pieces for the "Combinatorial Optimization Heuristics" class . . . . .	??
lib/tablero.c	File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class . . . . .	??

lib/ <a href="#">tablero.h</a>	File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class . . . . .	??
src/ <a href="#">interfaz-grafica.c</a>	File containing the body for openGL GUI functions for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
src/ <a href="#">interfaz-grafica.h</a>	File containing the openGL GUI for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
src/ <a href="#">main.c</a>	File containing the final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
src/ <a href="#">tetromino.c</a>	File containing the openGL GUI for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??
src/ <a href="#">tetromino.h</a>	File containing the openGL GUI for a final project for the "Combinatorial Optimization Heuristics" class . . . . .	??

## Chapter 4

# Class Documentation

### 4.1 abeja Struct Reference

La estructura abeja.

```
#include <abejas.h>
```

#### Public Attributes

- double `funcion`
- bool `waggle_bee`
- `TABLERO` \* `solucion`

#### 4.1.1 Detailed Description

La estructura abeja.

Su funcion es mantener una ruta (tablero) y una cantidad a la que llamaremos "polen" que marca que tan buena es su ruta.

#### 4.1.2 Member Data Documentation

##### 4.1.2.1 funcion

```
double abeja::funcion
```

Representa al polen, lo bueno de su ruta.

#### 4.1.2.2 solucion

`TABLERO* abeja::solucion`

Es la ruta que tomo la abeja.

#### 4.1.2.3 waggle\_bee

`bool abeja::waggle_bee`

Nos dice si es la abeja principal.

The documentation for this struct was generated from the following file:

- `lib/abejas.h`

## 4.2 param Struct Reference

Parametros de threads.

### Public Attributes

- `int argc`
- `char ** argv`
- `TABLERO ** tablero`
- `int size_colonia`
- `int distancia`

#### 4.2.1 Detailed Description

Parametros de threads.

Usamos esta estructura para mantener unidos los datos que contiene el programa y necesitamos para correr la heuristica.

#### 4.2.2 Member Data Documentation

##### 4.2.2.1 argc

`int param::argc`

Es el id que representa el # de argumentos.

#### 4.2.2.2 argv

```
char** param::argv
```

Son los argumentos del programa original.

#### 4.2.2.3 distancia

```
int param::distancia
```

Es la distancia que recorrera una abeja.

#### 4.2.2.4 size\_colonia

```
int param::size_colonia
```

Es el tamaño de la colonia que correremos.

#### 4.2.2.5 tablero

```
TABLERO** param::tablero
```

Es el apuntador al apuntador del tablero.

The documentation for this struct was generated from the following file:

- [src/main.c](#)

## 4.3 pieza Struct Reference

La estructura PIEZA contiene datos basicos de ella.

```
#include <pieza.h>
```

### Public Attributes

- [int id](#)
- [FORMA tipo](#)
- [int orientacion](#)
- [int x](#)
- [int y](#)
- [bool fija](#)
- [bool activo\\_centro](#)
- [PUNTO \\*\\* bloques](#)

### 4.3.1 Detailed Description

La estructura PIEZA contiene datos basicos de ella.

Usamos esta estructura para mantener unidos los datos que contiene una pieza como tipo, orientacion, ..., y le asignamos un id para identificarla de manera univocua.

### 4.3.2 Member Data Documentation

#### 4.3.2.1 activo\_centro

```
bool pieza::activo_centro
```

Nos dice si el centro ya hizo tetris.

#### 4.3.2.2 bloques

```
PUNTO** pieza::bloques
```

Un arreglo de puntos para los bloques.

#### 4.3.2.3 fija

```
bool pieza::fija
```

Nos dice si la pieza se encuentra en juego ahora.

#### 4.3.2.4 id

```
int pieza::id
```

Un id que la identifica de forma unica.

#### 4.3.2.5 orientacion

```
int pieza::orientacion
```

La orientacion de la pieza. Existen 4.

#### 4.3.2.6 tipo

```
FORMA pieza::tipo
```

Cada pieza debe tener una de 7 formas.

#### 4.3.2.7 x

```
int pieza::x
```

La posicion x del centro.

#### 4.3.2.8 y

```
int pieza::y
```

La posicion y del centro.

The documentation for this struct was generated from the following file:

- [lib/pieza.h](#)

## 4.4 punto Struct Reference

La estructura punto.

```
#include <pieza.h>
```

### Public Attributes

- int [x](#)
- int [y](#)
- bool [activo](#)

#### 4.4.1 Detailed Description

La estructura punto.

Su funcion es mantener una pieza ubicada por cada uno de sus bloques que contienen una posicion.

#### 4.4.2 Member Data Documentation

##### 4.4.2.1 activo

```
bool punto::activo
```

Nos dice si un bloque se encuentra actvo.

#### 4.4.2.2 x

```
int punto::x
```

Es el punto x.

#### 4.4.2.3 y

```
int punto::y
```

Es el punto y.

The documentation for this struct was generated from the following file:

- [lib/pieza.h](#)

## 4.5 tablero Struct Reference

La estructura TABLERO contiene datos basicos de el.

```
#include <tablero.h>
```

### Public Attributes

- int [alto](#)
- int [ancho](#)
- int [size](#)
- int [max\\_size](#)
- int [num\\_piezas\\_totales](#)
- int [piezas\\_actuales](#)
- int [num\\_tetris](#)
- bool [game\\_over](#)
- [PIEZA](#) \*\*\* [piezas](#)
- [PIEZA](#) \* [actual](#)

#### 4.5.1 Detailed Description

La estructura TABLERO contiene datos basicos de el.

Usamos esta estructura para mantener unidos los datos que contiene un juego como piezas, actual.

#### 4.5.2 Member Data Documentation



#### 4.5.2.1 actual

```
PIEZA* tablero::actual
```

Es la pieza actual que se juega.

#### 4.5.2.2 alto

```
int tablero::alto
```

Es el alto del tablero.

#### 4.5.2.3 ancho

```
int tablero::ancho
```

Es el ancho del tablero.

#### 4.5.2.4 game\_over

```
bool tablero::game_over
```

Nos dice si el tablero termino y perdio.

#### 4.5.2.5 max\_size

```
int tablero::max_size
```

Es el maximo nuero de piezas que puede tener.

#### 4.5.2.6 num\_piezas\_totales

```
int tablero::num_piezas_totales
```

Es el numero de piezas jugadas.

#### 4.5.2.7 num\_tetris

```
int tablero::num_tetris
```

Es el numero de tetris que ha hecho.

#### 4.5.2.8 piezas

```
PIEZA*** tablero::piezas
```

Es el arreglo[][] de pointer a piezas.

#### 4.5.2.9 piezas\_actuales

```
int tablero::piezas_actuales
```

Es el numero de piezas jugadas actual.

#### 4.5.2.10 size

```
int tablero::size
```

Es la cantidad de piezas que posee el tablero.

The documentation for this struct was generated from the following file:

- [lib/tablero.h](#)

## Chapter 5

# File Documentation

### 5.1 lib/abc.c File Reference

File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class.

```
#include "abc.h"
```

#### Macros

- #define **INFINITO** 1\*INFINITY
- #define **COMPARADOR**(x, y) (x < y)

#### Functions

- void **ABC** (**TABLERO** \*\*tablero\_pointer, int empleadas, int distancia, bool lag)
  - *Artificial bee colony algorithm.*

#### 5.1.1 Detailed Description

File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 En este archivo se implementa la unica funcion que dara vida al algoritmo de la colonia de abejas.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/abejas-tetris>

## 5.1.2 Macro Definition Documentation

### 5.1.2.1 INFINITO

```
#define INFINITO 1*INFINITY
```

Redefine el infinito para usarlo negativo.

## 5.1.3 Function Documentation

### 5.1.3.1 ABC()

```
void ABC (
    TABLERO ** tablero_pointer,
    int empleadas,
    int distancia,
    bool lag )
```

- Artificial bee colony algorithm.

El algoritmo funciona de la siguiente manera:

1. Inicializamos a nuestras abejas con nuestro tablero.
2. Cada abeja tendra un clon de nuestro tablero inicial.
3. Mientras que no hayamos perdido ejecutamos los 4,5,6,7,8 y 9.
4. Cada abeja empleada sale a una propuesta.
5. Busca soluciones a tu clon de tablero y sus juegos.
6. Regresa y hace el baile "waggle dance" para ver si ella es la "waggle bee" que guiara el tablero.
7. Si su nectar (funcion) es mejor, avisa a la colmena.
8. Al final la mejor "waggle bee" marca el camino y todas las abejas deben copiar su camino.
9. Revisamos que no hayamos perdido en el siguiente turno.
10. Terminamos.

## 5.2 lib/abc.h File Reference

File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class.

```
#include <math.h>
#include <stdbool.h>
#include <unistd.h>
#include <stdio.h>
#include "abejas.h"
#include "dump.h"
#include "funcion.h"
#include "tablero.h"
```

### Functions

- void [ABC](#) ([TABLERO](#) \*\*tablero\_pointer, int empleadas, int distancia, bool lag)
  - Artificial bee colony algorithm.

#### 5.2.1 Detailed Description

File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class.

##### Author

Jose Ricardo Rodriguez Abreu

##### Date

14 May 2017 En este archivo se define la unica funcion que dara vida al algoritmo de la colonia de abejas.

El programa usa el estandar de documentacion que define el uso de doxygen.

##### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/abejas-tetris>

#### 5.2.2 Function Documentation

##### 5.2.2.1 ABC()

```
void ABC (
    TABLERO ** tablero_pointer,
    int empleadas,
    int distancia,
    bool lag )
```

- Artificial bee colony algorithm.

Artificial bee colony algorithm o colonia de abejas. Esta funcion ejecuta un set del juego completo de tetris, teniendo como objetivo la optimizacion de ganar la mayoria de puntos evitando perder y manteniendo el tablero lo mas limpio posible.

## Parameters

<i>tablero_pointer</i>	- Es la posicion se encuentra el tablero.
<i>empleadas</i>	- Es el numero de abejas que viviran.
<i>distancia</i>	- Es la "distancia" en tiempo que recorreran.
<i>lag</i>	- Para ver si lo ejecuta lento o no.

El algoritmo funciona de la siguiente manera:

1. Inicializamos a nuestras abejas con nuestro tablero.
2. Cada abeja tendra un clon de nuestro tablero inicial.
3. Mientras que no hayamos perdido ejecutamos los 4,5,6,7,8 y 9.
4. Cada abeja empleada sale a una propuesta.
5. Busca soluciones a tu clon de tablero y sus juegos.
6. Regresa y hace el baile "waggle dance" para ver si ella es la "waggle bee" que guiara el tablero.
7. Si su nectar (funcion) es mejor, avisa a la colmena.
8. Al final la mejor "waggle bee" marca el camino y todas las abejas deben copiar su camino.
9. Revisamos que no hayamos perdido en el siguiente turno.
10. Terminamos.

### 5.3 lib/abejas.c File Reference

File containing the Artificial bee definitions for a final project for the "Combinatorial Optimization Heuristics" class.

```
#include "abejas.h"
#include "funcion.h"
#include <stdlib.h>
```

#### Functions

- [ABEJA \\* init\\_abeja \(TABLERO \\*tablero\)](#)  
*Inicia una nueva abeja.*
- void [free\\_abeja \(ABEJA \\*abeja\)](#)  
*Libera de memoria una abeja.*
- void [busca\\_fuente\\_alimento \(ABEJA \\*abeja\)](#)  
*Busca nuevos origenes de "polen".*
- double [do\\_waggle\\_dance \(ABEJA \\*abeja\)](#)  
*Realiza el "waggle dance".*
- void [set\\_tablero\\_abeja \(TABLERO \\*tablero, ABEJA \\*abeja\)](#)  
*Cambia el origen de la solucion de la abeja.*

### 5.3.1 Detailed Description

File containing the Artificial bee definitions for a final project for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 En este archivo de c, implementamos las estructuras y las funciones que tendran nuestras abejas artificiales para la ejecucion del ABC.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/abejas-tetris>

### 5.3.2 Function Documentation

#### 5.3.2.1 busca\_fuente\_alimento()

```
void busca_fuente_alimento (
    ABEJA * abeja )
```

Busca nuevos origenes de "polen".

La busqueda de caminos a palem es en realidad encontrar una solucion al tablero actual de la abeja.

#### See also

[busca\\_solucion\\_actual](#)

#### 5.3.2.2 do\_waggle\_dance()

```
double do_waggle_dance (
    ABEJA * abeja )
```

Realiza el "waggle dance".

Manda a llamar a dentro de el archivo para ajustar sus datos.

### 5.3.2.3 free\_abeja()

```
void free_abeja (
    ABEJA * abeja )
```

Libera de memoria una abeja.

Liberamos primero al tablero que contiene a la abeja. despues ya liberamos a la abeja de la memoria.

### 5.3.2.4 init\_abeja()

```
ABEJA* init_abeja (
    TABLERO * tablero )
```

Inicia una nueva abeja.

Creamos en memoria una abeja nueva y iniciamos tu tablero y la funcion con el valor del .

### 5.3.2.5 set\_tablero\_abeja()

```
void set_tablero_abeja (
    TABLERO * tablero,
    ABEJA * abeja )
```

Cambia el origen de la solucion de la abeja.

Al cambiar el de la , tambien hay que realizar la nueva modificacion a la de la abeja.

## 5.4 lib/abejas.h File Reference

File containing the Artificial bee definitions for a final project for the "Combinatorial Optimization Heuristics" class.

```
#include "tablero.h"
#include "dump.h"
```

### Classes

- struct [abeja](#)  
*La estructura abeja.*

### Typedefs

- typedef struct [abeja](#) [ABEJA](#)  
*La estructura abeja.*



## Functions

- `ABEJA * init_abeja (TABLERO *tablero)`  
*Inicia una nueva abeja.*
- `void free_abeja (ABEJA *abeja)`  
*Libera de memoria una abeja.*
- `void busca_fuente_alimento (ABEJA *abeja)`  
*Busca nuevos origenes de "polen".*
- `double do_waggle_dance (ABEJA *abeja)`  
*Realiza el "waggle dance".*
- `void set_tablero_abeja (TABLERO *tablero, ABEJA *abeja)`  
*Cambia el origen de la solucion de la abeja.*

### 5.4.1 Detailed Description

File containing the Artificial bee definitions for a final project for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 En este archivo encabezado, definimos las estructuras y las funciones que tendran nuestras abejas artificiales para la ejecucion del ABC.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/abejas-tetris>

### 5.4.2 Typedef Documentation

#### 5.4.2.1 ABEJA

```
typedef struct abeja ABEJA
```

La estructura abeja.

Su funcion es mantener una ruta (tablero) y una cantidad a la que llamaremos "polen" que marca que tan buena es su ruta.

### 5.4.3 Function Documentation

#### 5.4.3.1 busca\_fuente\_alimento()

```
void busca_fuente_alimento (  
    ABEJA * abeja )
```

Busca nuevos origenes de "polen".

Juega una partida dentro del tablero solucion que contiene, lo que puede hacerse la analogia de que "viaja por polen".

**Parameters**

<i>abeja</i>	- La abeja que queremos "viaje".
--------------	----------------------------------

La busqueda de caminos a palem es en realidad encontrar una solucion al tablero actual de la abeja.

**See also**

[busca\\_solucion\\_actual](#)

**5.4.3.2 do\_waggle\_dance()**

```
double do_waggle_dance (
    ABEJA * abeja )
```

Realiza el "waggle dance".

Esta funcion hace que la abeja realice el waggle dance para si misma, lo que implica que modifique su funcion de costo.

**Parameters**

<i>abeja</i>	- Es la abeja que va a "bailar"
--------------	---------------------------------

**Returns**

El valor de la funcion que contiene el destino.

Manda a llamar a dentro de el archivo para ajustar sus datos.

**5.4.3.3 free\_abeja()**

```
void free_abeja (
    ABEJA * abeja )
```

Libera de memoria una abeja.

Realiza la liberacion de memoria de la estructura abeja.

**Parameters**

<i>abeja</i>	- La abeja a liberar.
--------------	-----------------------

Liberamos primero al tablero que contiene a la abeja. despues ya liberamos a la abeja de la memoria.

#### 5.4.3.4 init\_abeja()

```
ABEJA* init_abeja (
    TABLERO * tablero )
```

Inicia una nueva abeja.

Inicializa los valores dentro de una estructura abeja.

##### Parameters

<i>tablero</i>	- Es la ruta de la cual la abeja parte.
----------------	---

##### Returns

Un apuntador a una estructura inicializada abeja.

Creamos en memoria una abeja nueva y iniciamos tu tablero y la funcion con el valor del .

#### 5.4.3.5 set\_tablero\_abeja()

```
void set_tablero_abeja (
    TABLERO * tablero,
    ABEJA * abeja )
```

Cambia el origen de la solucion de la abeja.

Se le cambia el apuntador del tablero que la abeja tomara para partir a realizar una nueva ruta a una nueva solucion.

##### Parameters

<i>tablero</i>	- Es el nuevo tablero de la abeja.
<i>abeja</i>	- Es la abeja que deseamos modificar.

Al cambiar el de la , tambien hay que realizar la nueva modificacion a la de la abeja.

## 5.5 lib/dump.c File Reference

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

```
#include "dump.h"
```

### Functions

- void `agrega_basura` (`TABLERO *tablero`)  
*Agregamos basura a la "pila".*
- void `limpia` (void)  
*Limpia la "pila" de basura.*

### 5.5.1 Detailed Description

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

13 Jun 2017 Aqui se implementa todo lo que queremos vaciar.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/AceptacionUmbral>

### 5.5.2 Function Documentation

#### 5.5.2.1 agrega\_basura()

```
void agrega_basura (
    TABLERO * tablero )
```

Agregamos basura a la "pila".

Con esta funcion agregamos basura a una lista para que eventualmente sea liberado el tablero que se guardo. @param tablero - Es el tablero que queremos eliminar.

#### 5.5.2.2 limpia()

```
void limpia (
    void )
```

Limpia la "pila" de basura.

Se ejecuta un para todos los elementos actuales de la lista .

## 5.6 lib/dump.h File Reference

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

```
#include <unistd.h>
#include <glib.h>
#include "tablero.h"
```

## Functions

- void `agrega_basura` (`TABLERO *tablero`)  
*Agregamos basura a la "pila".*
- void `limpia` (void)  
*Limpia la "pila" de basura.*

## Variables

- `GList *` `basurero`  
*Basurero.*

### 5.6.1 Detailed Description

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

13 Jun 2017 Aqui se guardara todo lo que queremos vaciar.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/AceptacionUmbral>

### 5.6.2 Function Documentation

#### 5.6.2.1 `agrega_basura()`

```
void agrega_basura (  
    TABLERO * tablero )
```

Agregamos basura a la "pila".

Con esta funcion agregamos basura a una lista para que eventualmente sea liberado el tablero que se guardo. @  
param tablero - Es el tablero que queremos eliminar.

### 5.6.2.2 limpia()

```
void limpia (
    void )
```

Limpia la "pila" de basura.

Se ejecuta un para todos los elementos actuales de la lista .

## 5.6.3 Variable Documentation

### 5.6.3.1 basurero

```
GList * basurero
```

Basurero.

Es un objeto lista que evita acumular basura Con el proposito de liberar memoria, hacemos uso de este objeto para no tener Segmentation fault.

## 5.7 lib/funcion.c File Reference

File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class.

```
#include "funcion.h"
#include "math.h"
#include <stdlib.h>
```

### Macros

- #define [INFINITO](#) -1\*INFINITY
- #define [degreesToRadians](#)(angleDegrees) (angleDegrees \* M\_PI / 180.0)
- #define [radiansToDegrees](#)(angleRadians) (angleRadians \* 180.0 / M\_PI)
- #define [pitagoras](#)(x, y) (sqrt(pow(x,2)+pow(y,2)))
- #define [valor\\_abs](#)(x, y) (x > y ? (x-y) : (y-x))
- #define [mi\\_max](#)(x, y) (x > y ? x : y)

## Functions

- double [revisa\\_horizontal](#) (TABLERO \*tablero, int columna)  
*Entrega un numero que representa la horizontalidad.*
- double [horizontalidad](#) (TABLERO \*tablero)  
*Regresa un numero representativo para un dato del tablero.*
- double [atrapado](#) (TABLERO \*tablero, int x, int y)  
*Regresa 1 si el cuadrado se encuentra atrapado.*
- double [cuenta\\_atrapados](#) (TABLERO \*tablero)  
*Revisa cuantos cuadrados atrapados hay en el tablero.*
- int [cuenta\\_cubierta](#) (TABLERO \*tablero, int x, int y)  
*Nos dice si existe un bloque ocupado encima.*
- double [cuenta\\_cubiertas\\_total](#) (TABLERO \*tablero)  
*Cuenta los bloques tapados.*
- double [probabilidad\\_tetris](#) (TABLERO \*tablero, int j)  
*Un estimado de que pronto se haga un tetris.*
- double [waggle\\_dance](#) (TABLERO \*tablero)  
*Waggle dance es es baile de las abejas.*

### 5.7.1 Detailed Description

File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 En este archivo se implementan los criterios que deben tener las buenas rutas que una tenga. El baile que realiza o tambien llamado nos devuelve un valor que entre mayor sea, mas atractiva es el polen.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/abejas-tetris>

### 5.7.2 Macro Definition Documentation

#### 5.7.2.1 degreesToRadians

```
#define degreesToRadians(  
    angleDegrees ) (angleDegrees * M_PI / 180.0)
```

Convierte de grados a radiales.

### 5.7.2.2 INFINITO

```
#define INFINITO -1*INFINITY
```

Redefine el infinito para usarlo negativo.

### 5.7.2.3 mi\_max

```
#define mi_max(  
    x,  
    y ) (x > y ? x : y)
```

Nos regresa el maximo de dos numeros.

### 5.7.2.4 pitagoras

```
#define pitagoras(  
    x,  
    y ) (sqrt(pow(x,2)+pow(y,2)))
```

Obtenemos la raiz del cuadrado de dos numeros.

### 5.7.2.5 radiansToDegrees

```
#define radiansToDegrees(  
    angleRadians ) (angleRadians * 180.0 / M_PI)
```

Convierte de radianes a grados.

### 5.7.2.6 valor\_abs

```
#define valor_abs(  
    x,  
    y ) (x > y ? (x-y) : (y-x))
```

Nos regresa la diferencia entre dos numeros en positivo.

## 5.7.3 Function Documentation

### 5.7.3.1 atrapado()

```
double atrapado (  
    TABLERO * tablero,  
    int x,  
    int y )
```

Regresa 1 si el cuadrado se encuentra atrapado.

Revisa que un cuadro no se encuentre atrapado en el tablero, en otras palabras, revisa que exista un vecino diferente a un bloque ocupado.



**Parameters**

<i>tablero</i>	- Es el tablero que queremos observar.
<i>x</i>	- Es la posicion x del cuadrito.
<i>y</i>	- Es la posicion y del cuadrito.

**Returns**

1 si se encuentra atrapado y 0 en caso contrario.

**5.7.3.2 cuenta\_atrapados()**

```
double cuenta_atrapados (
    TABLERO * tablero )
```

Revisa cuantos cuadrados atrapados hay en el tablero.

Cuenta la cantidad de cuadritos que poseen a todos sus vecinos ocupados de bloques pero el no esta ocupado.

**Parameters**

<i>tablero</i>	- Es el tablero que queremos evaluar.
----------------	---------------------------------------

**Returns**

El numero de cuadritos de 1x1 atrapados.

**5.7.3.3 cuenta\_cubierta()**

```
int cuenta_cubierta (
    TABLERO * tablero,
    int x,
    int y )
```

Nos dice si existe un bloque ocupado encima.

Con el proposito de conocer si es posible acceder a bloque, revisamos todos los bloques a partir de .

**Parameters**

<i>tablero</i>	- Es el tablero en que trabajamos.
<i>x</i>	- Es la posicion del ancho del bloque.
<i>y</i>	- Es la altura del bloque.

**Returns**

0 si esta libre y 1 en caso contrario.

**5.7.3.4 cuenta\_cubiertas\_total()**

```
double cuenta_cubiertas_total (
    TABLERO * tablero )
```

Cuenta los bloques tapados.

Cuenta cuantos bloques es imposible acceder ya que tienen algo encima.

**Parameters**

<i>tablero</i>	- Es el tablero que observamos.
----------------	---------------------------------

**Returns**

El numero de bloques tapados o cubiertos.

**5.7.3.5 horizontalidad()**

```
double horizontalidad (
    TABLERO * tablero )
```

Regresa un numero representativo para un dato del tablero.

Regresa un entero mayor igual a cero para representar todos los posibles caminos que puede tomar la linea skyline.

**Parameters**

<i>tablero</i>	- Es el tablero a revisar.
----------------	----------------------------

**Returns**

Regresa un numero mayor igual a cero. Entre mas horizontal este la linea superior menor sera el numero.

**5.7.3.6 probabilidad\_tetris()**

```
double probabilidad_tetris (
    TABLERO * tablero,
    int j )
```

Un estimado de que pronto se haga un tetrís.

Revisa cuantos cuadrados hay ocupados por bloques en este nivel. El nivel debe no tener cuadros atrapados y ser el mas alto posible.

#### Parameters

<i>tablero</i>	- Es el tablero que queremos evaluar.
<i>j</i>	- Es el nivel actual que estamos analizando.

#### Returns

Un numero mayor igual a cero. Entre mas grande mejor.

#### 5.7.3.7 `revisa_horizontal()`

```
double revisa_horizontal (
    TABLERO * tablero,
    int columna )
```

Entrega un numero que representa la horizontalidad.

Revisa para cada columna cuanta distancia hay de entre el ultimo nivel horizontal hasta el primer bloque o tetrominoide que encuentre.

#### Parameters

<i>tablero</i>	- Es el tablero a revisar.
<i>columna</i>	- Es la columna particular a revisar.

#### Returns

Regresa la diferencia con la columna y el techo.

#### 5.7.3.8 `waggle_dance()`

```
double waggle_dance (
    TABLERO * tablero )
```

Waggle dance es es baile de las abejas.

Regresa un numero que entre mayor sea mejor. Utiliza las funciones con las siguientes condiciones: Le resta los por cierta constante -50. Le resta los por cierta constante -10 Le suma los numero de tetrís realizado por constante 1000 Le suma la que pronto se haga.

## 5.8 lib/funcion.h File Reference

File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class.

```
#include "tablero.h"
```

### Functions

- double [waggle\\_dance](#) ([TABLERO](#) \*[tablero](#))  
*Waggle dance es es baile de las abejas.*

#### 5.8.1 Detailed Description

File containing the Artificial bee colony algorithm for a final project for the "Combinatorial Optimization Heuristics" class.

##### Author

Jose Ricardo Rodriguez Abreu

##### Date

14 May 2017 En este archivo se define los criterios que deben tener las buenas rutas que una tenga. El baile que realiza o tambien llamado nos devuelve un valor que entre mayor sea, mas atractiva es el polen.

El programa usa el estandar de documentacion que define el uso de doxygen.

##### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/abejas-tetris>

#### 5.8.2 Function Documentation

##### 5.8.2.1 waggle\_dance()

```
double waggle_dance (  
    TABLERO * tablero )
```

Waggle dance es es baile de las abejas.

Esta es la funcion que nos regresa un numero que representa la abundancia de polen que contiene la ruta de una abeja en particular.

## Parameters

<code>tablero</code>	- Es el tablero que la abeja debe tener.
----------------------	--

## Returns

Un numero que entre mayor sea, mas abundancia habra.

Regresa un numero que entre mayor sea mejor. Utiliza las funciones con las siguientes condiciones: Le resta los por cierta constante -50. Le resta los por cierta constante -10 Le suma los numero de tetris realizado por constante 1000 Le suma la que pronto se haga.

## 5.9 lib/lock.h File Reference

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

```
#include <pthread.h>
```

## Variables

- pthread\_mutex\_t [lock](#)  
*Lock.*
- pthread\_mutex\_t [lock\\_pieza](#)  
*Lock de las piezas.*
- pthread\_mutex\_t [lock\\_basura](#)  
*Lock de la lista basura.*

### 5.9.1 Detailed Description

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

## Author

Jose Ricardo Rodriguez Abreu

## Date

13 Jun 2017 Definimos un objeto mutex para evitar modificar un objeto NULL.

El programa usa el estandar de documentacion que define el uso de doxygen.

## See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/AceptacionUmbral>

## 5.9.2 Variable Documentation

### 5.9.2.1 lock

```
pthread_mutex_t lock
```

Lock.

Es un objeto mutex que evita continuar la ejecucion. Con el proposito de liberar memoria y dibujar la interfaz, hacemos uso de este objeto para no tener Segmentation fault.

### 5.9.2.2 lock\_basura

```
pthread_mutex_t lock_basura
```

Lock de la lista basura.

Es un objeto mutex que evita continuar la ejecucion. Con el proposito de liberar memoria y dibujar la interfaz, hacemos uso de este objeto para no tener Segmentation fault.

### 5.9.2.3 lock\_pieza

```
pthread_mutex_t lock_pieza
```

Lock de las piezas.

Es un objeto mutex que evita continuar la ejecucion. Con el proposito de liberar memoria y dibujar la interfaz, hacemos uso de este objeto para no tener Segmentation fault.

## 5.10 lib/pieza.c File Reference

File containing the struct and funtions to simulate Tetris pieces for the "Combinatorial Optimization Heuristics" class.

```
#include "pieza.h"
```

## Functions

- `PIEZA * init_pieza` (int id, FORMA tipo)
- `PIEZA * copy_pieza` (PIEZA \*pieza)
- `void free_pieza` (PIEZA \*pieza)
- `void set_x_pieza` (PIEZA \*pieza, int x)
- `void set_y_pieza` (PIEZA \*pieza, int y)
- `void actualiza_cuadrado` (PIEZA \*pieza)
- `void actualiza_left_gun` (PIEZA \*pieza)
- `void actualiza_right_gun` (PIEZA \*pieza)
- `void actualiza_left_snake` (PIEZA \*pieza)
- `void actualiza_right_snake` (PIEZA \*pieza)
- `void actualiza_i` (PIEZA \*pieza)
- `void actualiza_t` (PIEZA \*pieza)
- `void actualiza_bloques` (PIEZA \*pieza)
- `void set_punto_pieza` (PIEZA \*pieza, int x, int y)
- `void rotar_pieza` (PIEZA \*pieza)
- `void rota_pieza` (PIEZA \*pieza, int grado)
- `void actualiza_posicion_pieza` (PIEZA \*pieza, int x, int y)
- `void deja_caer_pieza` (PIEZA \*pieza)
- `void levanta_pieza` (PIEZA \*pieza)
- `bool borra_bloque_pieza` (PIEZA \*pieza, int x, int y)
- `void bajar_bloque_pieza` (PIEZA \*pieza, int x, int y)

### 5.10.1 Detailed Description

File containing the struct and funtions to simulate Tetris pieces for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 En este archivo implementamos las funciones pieza y junto a la firma de funciones que se hacen uso de las estructuras y manejo de las estructuras definidas como PIEZA, su manejo en memoria y la de los atributos que constituye su estructura.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/AceptacionUmbral>

### 5.10.2 Function Documentation

#### 5.10.2.1 actualiza\_bloques()

```
void actualiza_bloques (  
    PIEZA * pieza )
```

**Parameters**

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

**5.10.2.2 actualiza\_cuadrado()**

```
void actualiza_cuadrado (  
    PIEZA * pieza )
```

**Parameters**

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

**5.10.2.3 actualiza\_i()**

```
void actualiza_i (  
    PIEZA * pieza )
```

**Parameters**

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

**5.10.2.4 actualiza\_left\_gun()**

```
void actualiza_left_gun (  
    PIEZA * pieza )
```

**Parameters**

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

**5.10.2.5 actualiza\_left\_snake()**

```
void actualiza_left_snake (  
    PIEZA * pieza )
```



## Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

5.10.2.6 `actualiza_posicion_pieza()`

```
void actualiza_posicion_pieza (
    PIEZA * pieza,
    int x,
    int y )
```

## Parameters

<i>pieza</i>	-
<i>x</i>	-
<i>y</i>	-

5.10.2.7 `actualiza_right_gun()`

```
void actualiza_right_gun (
    PIEZA * pieza )
```

## Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

5.10.2.8 `actualiza_right_snake()`

```
void actualiza_right_snake (
    PIEZA * pieza )
```

## Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

5.10.2.9 `actualiza_t()`

```
void actualiza_t (
    PIEZA * pieza )
```

**Parameters**

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

**5.10.2.10 bajar\_bloque\_pieza()**

```
void bajar_bloque_pieza (  
    PIEZA * pieza,  
    int x,  
    int y )
```

**Parameters**

<i>pieza</i>	-
<i>x</i>	-
<i>y</i>	-

**5.10.2.11 borra\_bloque\_pieza()**

```
bool borra_bloque_pieza (  
    PIEZA * pieza,  
    int x,  
    int y )
```

**Parameters**

<i>pieza</i>	-
<i>x</i>	-
<i>y</i>	-

**5.10.2.12 copy\_pieza()**

```
PIEZA* copy_pieza (  
    PIEZA * pieza )
```

**Parameters**

<i>pieza</i>	-
--------------	---

## Returns

5.10.2.13 `deja_caer_pieza()`

```
void deja_caer_pieza (
    PIEZA * pieza )
```

## Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

5.10.2.14 `free_pieza()`

```
void free_pieza (
    PIEZA * pieza )
```

## Parameters

<i>pieza</i>	-
--------------	---

5.10.2.15 `init_pieza()`

```
PIEZA* init_pieza (
    int id,
    FORMA tipo )
```

## Parameters

<i>id</i>	-
<i>tipo</i>	-

5.10.2.16 `levanta_pieza()`

```
void levanta_pieza (
    PIEZA * pieza )
```

**Parameters**

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

**5.10.2.17 rota\_pieza()**

```
void rota_pieza (
    PIEZA * pieza,
    int grado )
```

**Parameters**

<i>pieza</i>	-
<i>grado</i>	-

**5.10.2.18 rotar\_pieza()**

```
void rotar_pieza (
    PIEZA * pieza )
```

**Parameters**

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

**5.10.2.19 set\_punto\_pieza()**

```
void set_punto_pieza (
    PIEZA * pieza,
    int x,
    int y )
```

**Parameters**

<i>pieza</i>	-
<i>x</i>	-
<i>y</i>	-

## 5.10.2.20 set\_x\_pieza()

```
void set_x_pieza (
    PIEZA * pieza,
    int x )
```

## Parameters

<i>pieza</i>	-
<i>x</i>	-

## 5.10.2.21 set\_y\_pieza()

```
void set_y_pieza (
    PIEZA * pieza,
    int y )
```

## Parameters

<i>pieza</i>	-
<i>y</i>	-

## 5.11 lib/pieza.h File Reference

File containing the struct and funtions to simulate Tetris pieces for the "Combinatorial Optimization Heuristics" class.

```
#include <stdbool.h>
#include <stdlib.h>
#include "lock.h"
```

### Classes

- struct [punto](#)  
*La estructura punto.*
- struct [pieza](#)  
*La estructura PIEZA contiene datos basicos de ella.*

### Typedefs

- typedef enum [FORMA](#) [FORMA](#)  
*La estructura pieza.*
- typedef struct [punto](#) [PUNTO](#)  
*La estructura punto.*
- typedef struct [pieza](#) [PIEZA](#)  
*La estructura PIEZA contiene datos basicos de ella.*

## Enumerations

- enum `FORMA` {  
`Sq`, `LG`, `RG`, `LS`,  
`RS`, `I`, `T` }

*La estructura pieza.*

## Functions

- `PIEZA * init_pieza` (int id, `FORMA` tipo)
- `PIEZA * copy_pieza` (`PIEZA *pieza`)
- void `free_pieza` (`PIEZA *pieza`)
- void `set_x_pieza` (`PIEZA *pieza`, int x)
- void `set_y_pieza` (`PIEZA *pieza`, int y)
- void `actualiza_cuadrado` (`PIEZA *pieza`)
- void `actualiza_left_gun` (`PIEZA *pieza`)
- void `actualiza_right_gun` (`PIEZA *pieza`)
- void `actualiza_left_snake` (`PIEZA *pieza`)
- void `actualiza_right_snake` (`PIEZA *pieza`)
- void `actualiza_i` (`PIEZA *pieza`)
- void `actualiza_t` (`PIEZA *pieza`)
- void `actualiza_bloques` (`PIEZA *pieza`)
- void `set_punto_pieza` (`PIEZA *pieza`, int x, int y)
- void `rotar_pieza` (`PIEZA *pieza`)
- void `rota_pieza` (`PIEZA *pieza`, int grado)
- void `actualiza_posicion_pieza` (`PIEZA *pieza`, int x, int y)
- void `deja_caer_pieza` (`PIEZA *pieza`)
- void `levanta_pieza` (`PIEZA *pieza`)
- bool `borra_bloque_pieza` (`PIEZA *pieza`, int x, int y)
- void `bajar_bloque_pieza` (`PIEZA *pieza`, int x, int y)

### 5.11.1 Detailed Description

File containing the struct and funtions to simulate Tetris pieces for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 En este archivo definimos la estructura pieza y junto a la firma de funciones que se hacen uso de las estructuras y manejo de las estructuras definidas como `PIEZA`, su manejo en memoria y la de los atributos que constituye su estructura.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/AceptacionUmbral>

## 5.11.2 Typedef Documentation

### 5.11.2.1 FORMA

```
typedef enum FORMA FORMA
```

La estructura pieza.

Su funcion es mantener una pieza unida y con la informacion que necesitemos en su vida util dentro de un tablero.

### 5.11.2.2 PIEZA

```
typedef struct pieza PIEZA
```

La estructura PIEZA contiene datos basicos de ella.

Usamos esta estructura para mantener unidos los datos que contiene una pieza como tipo, orientacion, ..., y le asignamos un id para identificarla de manera univocua.

### 5.11.2.3 PUNTO

```
typedef struct punto PUNTO
```

La estructura punto.

Su funcion es mantener una pieza ubicada por cada uno de sus bloques que contienen una posicion.

## 5.11.3 Enumeration Type Documentation

### 5.11.3.1 FORMA

```
enum FORMA
```

La estructura pieza.

Su funcion es mantener una pieza unida y con la informacion que necesitemos en su vida util dentro de un tablero.

#### Enumerator

Sq	Cuadrado .
LG	Left Gun: .
RG	Right Gun: L .
LS	Left snake.
RS	Right snake.
I	Forma de  .
T	Forma de T.

## 5.11.4 Function Documentation

### 5.11.4.1 actualiza\_bloques()

```
void actualiza_bloques (  
    PIEZA * pieza )
```

#### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

### 5.11.4.2 actualiza\_cuadrado()

```
void actualiza_cuadrado (  
    PIEZA * pieza )
```

#### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

### 5.11.4.3 actualiza\_i()

```
void actualiza_i (  
    PIEZA * pieza )
```

#### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

### 5.11.4.4 actualiza\_left\_gun()

```
void actualiza_left_gun (  
    PIEZA * pieza )
```

#### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------



#### 5.11.4.5 actualiza\_left\_snake()

```
void actualiza_left_snake (
    PIEZA * pieza )
```

##### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

#### 5.11.4.6 actualiza\_posicion\_pieza()

```
void actualiza_posicion_pieza (
    PIEZA * pieza,
    int x,
    int y )
```

##### Parameters

<i>pieza</i>	-
<i>x</i>	-
<i>y</i>	-

#### 5.11.4.7 actualiza\_right\_gun()

```
void actualiza_right_gun (
    PIEZA * pieza )
```

##### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

#### 5.11.4.8 actualiza\_right\_snake()

```
void actualiza_right_snake (
    PIEZA * pieza )
```

##### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

#### 5.11.4.9 actualiza\_t()

```
void actualiza_t (
    PIEZA * pieza )
```

##### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

#### 5.11.4.10 bajar\_bloque\_pieza()

```
void bajar_bloque_pieza (
    PIEZA * pieza,
    int x,
    int y )
```

##### Parameters

<i>pieza</i>	-
<i>x</i>	-
<i>y</i>	-

#### 5.11.4.11 borra\_bloque\_pieza()

```
bool borra_bloque_pieza (
    PIEZA * pieza,
    int x,
    int y )
```

##### Parameters

<i>pieza</i>	-
<i>x</i>	-
<i>y</i>	-

#### 5.11.4.12 copy\_pieza()

```
PIEZA* copy_pieza (
    PIEZA * pieza )
```

## Parameters

<i>pieza</i>	-
--------------	---

## Returns

5.11.4.13 `deja_caer_pieza()`

```
void deja_caer_pieza (  
    PIEZA * pieza )
```

## Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

5.11.4.14 `free_pieza()`

```
void free_pieza (  
    PIEZA * pieza )
```

## Parameters

<i>pieza</i>	-
--------------	---

5.11.4.15 `init_pieza()`

```
PIEZA* init_pieza (  
    int id,  
    FORMA tipo )
```

## Parameters

<i>id</i>	-
<i>tipo</i>	-

#### 5.11.4.16 levanta\_pieza()

```
void levanta_pieza (
    PIEZA * pieza )
```

##### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

#### 5.11.4.17 rota\_pieza()

```
void rota_pieza (
    PIEZA * pieza,
    int grado )
```

##### Parameters

<i>pieza</i>	-
<i>grado</i>	-

#### 5.11.4.18 rotar\_pieza()

```
void rotar_pieza (
    PIEZA * pieza )
```

##### Parameters

<i>pieza</i>	- Es la pieza que deseamos modificar.
--------------	---------------------------------------

#### 5.11.4.19 set\_punto\_pieza()

```
void set_punto_pieza (
    PIEZA * pieza,
    int x,
    int y )
```

##### Parameters

<i>pieza</i>	-
<i>x</i>	-
<i>y</i>	-

## 5.11.4.20 set\_x\_pieza()

```
void set_x_pieza (
    PIEZA * pieza,
    int x )
```

## Parameters

<i>pieza</i>	-
<i>x</i>	-

## 5.11.4.21 set\_y\_pieza()

```
void set_y_pieza (
    PIEZA * pieza,
    int y )
```

## Parameters

<i>pieza</i>	-
<i>y</i>	-

## 5.12 lib/tablero.c File Reference

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

```
#include "tablero.h"
```

## Functions

- `TABLERO * init_tablero (int ancho, int alto)`
- `TABLERO * copy_tablero (TABLERO *tablero)`
- `void free_tablero (TABLERO *tablero)`
- `void crear_pieza_tablero (TABLERO *tablero)`
- `void agrega_pieza_tablero (TABLERO *tablero, PIEZA *pieza)`
- `void borra_pieza_tablero (TABLERO *tablero, PIEZA *pieza)`
- `bool mover_pieza_tablero (TABLERO *tablero, PIEZA *pieza)`
- `bool rotar_pieza_tablero (TABLERO *tablero, PIEZA *pieza)`
- `bool mover_izquierda_tablero (TABLERO *tablero, PIEZA *pieza)`
- `bool mover_derecha_tablero (TABLERO *tablero, PIEZA *pieza)`
- `void busca_solucion_actual (TABLERO *tablero)`
- `void tetris_nivel (TABLERO *tablero, int l)`

- void `tetris` (`TABLERO *tablero`)
- void `set_pieza_nueva` (`TABLERO *tablero`, `PIEZA *pieza`)
- `PIEZA * get_pieza` (`TABLERO *tablero`, int x, int y)
- void `siguiente_turno_tablero` (`TABLERO *tablero`)
- void `imprime_tablero` (`TABLERO *tablero`)

### 5.12.1 Detailed Description

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 En este archivo implementamos la estructura tablero y junto a las funciones que se hacen uso de las estructuras y manejo de las estructuras definidas como TABLERO, su manejo en memoria y la de los atributos que constituye su estructura.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/AceptacionUmbral>

### 5.12.2 Function Documentation

#### 5.12.2.1 `agrega_pieza_tablero()`

```
void agrega_pieza_tablero (
    TABLERO * tablero,
    PIEZA * pieza )
```

##### Parameters

<i>tablero</i>	-
<i>pieza</i>	-

#### 5.12.2.2 `borra_pieza_tablero()`

```
void borra_pieza_tablero (
```

```
TABLERO * tablero,  
PIEZA * pieza )
```

**Parameters**

<i>tablero</i>	-
<i>pieza</i>	-

**5.12.2.3 busca\_solucion\_actual()**

```
void busca_solucion_actual (  
    TABLERO * tablero )
```

**Parameters**

<i>tablero</i>	-
----------------	---

**5.12.2.4 copy\_tablero()**

```
TABLERO* copy_tablero (  
    TABLERO * tablero )
```

**Parameters**

<i>tablero</i>	-
----------------	---

**Returns****5.12.2.5 crear\_pieza\_tablero()**

```
void crear_pieza_tablero (  
    TABLERO * tablero )
```

**Parameters**

<i>tablero</i>	-
----------------	---

#### 5.12.2.6 free\_tablero()

```
void free_tablero (
    TABLERO * tablero )
```

##### Parameters

<i>tablero</i>	-
----------------	---

#### 5.12.2.7 get\_pieza()

```
PIEZA* get_pieza (
    TABLERO * tablero,
    int x,
    int y )
```

##### Parameters

<i>tablero</i>	-
<i>x</i>	-
<i>y</i>	-

##### Returns

#### 5.12.2.8 imprime\_tablero()

```
void imprime_tablero (
    TABLERO * tablero )
```

##### Parameters

<i>tablero</i>	-
----------------	---

#### 5.12.2.9 init\_tablero()

```
TABLERO* init_tablero (
    int alto,
    int ancho )
```



## Parameters

<i>alto</i>	-
<i>ancho</i>	-

## Returns

## 5.12.2.10 mover\_derecha\_tablero()

```
bool mover_derecha_tablero (  
    TABLERO * tablero,  
    PIEZA * pieza )
```

## Parameters

<i>tablero</i>	-
<i>pieza</i>	-

## Returns

## 5.12.2.11 mover\_izquierda\_tablero()

```
bool mover_izquierda_tablero (  
    TABLERO * tablero,  
    PIEZA * pieza )
```

## Parameters

<i>tablero</i>	-
<i>pieza</i>	-

## Returns

#### 5.12.2.12 mover\_pieza\_tablero()

```
bool mover_pieza_tablero (
    TABLERO * tablero,
    PIEZA * pieza )
```

##### Parameters

<i>tablero</i>	-
<i>pieza</i>	-

##### Returns

#### 5.12.2.13 rotar\_pieza\_tablero()

```
bool rotar_pieza_tablero (
    TABLERO * tablero,
    PIEZA * pieza )
```

##### Parameters

<i>tablero</i>	-
<i>pieza</i>	-

##### Returns

#### 5.12.2.14 set\_pieza\_nueva()

```
void set_pieza_nueva (
    TABLERO * tablero,
    PIEZA * pieza )
```

##### Parameters

<i>tablero</i>	-
<i>pieza</i>	-

## 5.12.2.15 siguiente\_turno\_tablero()

```
void siguiente_turno_tablero (
    TABLERO * tablero )
```

## Parameters

<i>tablero</i>	-
----------------	---

## 5.12.2.16 tetris()

```
void tetris (
    TABLERO * tablero )
```

## Parameters

<i>tablero</i>	-
----------------	---

## 5.12.2.17 tetris\_nivel()

```
void tetris_nivel (
    TABLERO * tablero,
    int l )
```

## Parameters

<i>tablero</i>	-
<i>l</i>	-

## 5.13 lib/tablero.h File Reference

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "lock.h"
#include "pieza.h"
```

## Classes

- struct [tablero](#)

*La estructura TABLERO contiene datos basicos de el.*

## Typedefs

- typedef struct `tablero` `TABLERO`

*La estructura TABLERO contiene datos basicos de el.*

## Functions

- `TABLERO * init_tablero` (int alto, int ancho)
- `TABLERO * copy_tablero` (`TABLERO *tablero`)
- void `free_tablero` (`TABLERO *tablero`)
- void `crear_pieza_tablero` (`TABLERO *tablero`)
- void `agrega_pieza_tablero` (`TABLERO *tablero`, `PIEZA *pieza`)
- void `borra_pieza_tablero` (`TABLERO *tablero`, `PIEZA *pieza`)
- bool `mover_pieza_tablero` (`TABLERO *tablero`, `PIEZA *pieza`)
- bool `rotar_pieza_tablero` (`TABLERO *tablero`, `PIEZA *pieza`)
- bool `mover_izquierda_tablero` (`TABLERO *tablero`, `PIEZA *pieza`)
- bool `mover_derecha_tablero` (`TABLERO *tablero`, `PIEZA *pieza`)
- void `busca_solucion_actual` (`TABLERO *tablero`)
- void `tetris_nivel` (`TABLERO *tablero`, int l)
- void `tetris` (`TABLERO *tablero`)
- void `set_pieza_nueva` (`TABLERO *tablero`, `PIEZA *pieza`)
- `PIEZA * get_pieza` (`TABLERO *tablero`, int x, int y)
- void `siguiente_turno_tablero` (`TABLERO *tablero`)
- void `imprime_tablero` (`TABLERO *tablero`)

### 5.13.1 Detailed Description

File containing the struct and funtions to simulate Tetris board for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 En este archivo definimos la estructura tablero y junto a la firma de funciones que se hacen uso de las estructuras y manejo de las estructuras definidas como TABLERO, su manejo en memoria y la de los atributos que constituye su estructura.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/AceptacionUmbral>

### 5.13.2 Typedef Documentation

### 5.13.2.1 TABLERO

```
typedef struct tablero TABLERO
```

La estructura TABLERO contiene datos basicos de el.

Usamos esta estructura para mantener unidos los datos que contiene un juego como piezas, actual.

## 5.13.3 Function Documentation

### 5.13.3.1 agrega\_pieza\_tablero()

```
void agrega_pieza_tablero (  
    TABLERO * tablero,  
    PIEZA * pieza )
```

#### Parameters

<i>tablero</i>	-
<i>pieza</i>	-

### 5.13.3.2 borra\_pieza\_tablero()

```
void borra_pieza_tablero (  
    TABLERO * tablero,  
    PIEZA * pieza )
```

#### Parameters

<i>tablero</i>	-
<i>pieza</i>	-

### 5.13.3.3 busca\_solucion\_actual()

```
void busca_solucion_actual (  
    TABLERO * tablero )
```

#### Parameters

<i>tablero</i>	-
----------------	---

#### 5.13.3.4 copy\_tablero()

```
TABLERO* copy_tablero (  
    TABLERO * tablero )
```

##### Parameters

<i>tablero</i>	-
----------------	---

##### Returns

#### 5.13.3.5 crear\_pieza\_tablero()

```
void crear_pieza_tablero (  
    TABLERO * tablero )
```

##### Parameters

<i>tablero</i>	-
----------------	---

#### 5.13.3.6 free\_tablero()

```
void free_tablero (  
    TABLERO * tablero )
```

##### Parameters

<i>tablero</i>	-
----------------	---

#### 5.13.3.7 get\_pieza()

```
PIEZA* get_pieza (  
    TABLERO * tablero,  
    int x,  
    int y )
```

## Parameters

<i>tablero</i>	-
<i>x</i>	-
<i>y</i>	-

## Returns

## 5.13.3.8 imprime\_tablero()

```
void imprime_tablero (  
    TABLERO * tablero )
```

## Parameters

<i>tablero</i>	-
----------------	---

## 5.13.3.9 init\_tablero()

```
TABLERO* init_tablero (  
    int alto,  
    int ancho )
```

## Parameters

<i>alto</i>	-
<i>ancho</i>	-

## Returns

## 5.13.3.10 mover\_derecha\_tablero()

```
bool mover_derecha_tablero (  
    TABLERO * tablero,  
    PIEZA * pieza )
```

**Parameters**

<i>tablero</i>	-
<i>pieza</i>	-

**Returns****5.13.3.11 mover\_izquierda\_tablero()**

```
bool mover_izquierda_tablero (  
    TABLERO * tablero,  
    PIEZA * pieza )
```

**Parameters**

<i>tablero</i>	-
<i>pieza</i>	-

**Returns****5.13.3.12 mover\_pieza\_tablero()**

```
bool mover_pieza_tablero (  
    TABLERO * tablero,  
    PIEZA * pieza )
```

**Parameters**

<i>tablero</i>	-
<i>pieza</i>	-

**Returns**



#### 5.13.3.13 rotar\_pieza\_tablero()

```
bool rotar_pieza_tablero (
    TABLERO * tablero,
    PIEZA * pieza )
```

##### Parameters

<i>tablero</i>	-
<i>pieza</i>	-

##### Returns

#### 5.13.3.14 set\_pieza\_nueva()

```
void set_pieza_nueva (
    TABLERO * tablero,
    PIEZA * pieza )
```

##### Parameters

<i>tablero</i>	-
<i>pieza</i>	-

#### 5.13.3.15 siguiente\_turno\_tablero()

```
void siguiente_turno_tablero (
    TABLERO * tablero )
```

##### Parameters

<i>tablero</i>	-
----------------	---

#### 5.13.3.16 tetris()

```
void tetris (
    TABLERO * tablero )
```

## Parameters

<i>tablero</i>	-
----------------	---

## 5.13.3.17 tetris\_nivel()

```
void tetris_nivel (
    TABLERO * tablero,
    int l )
```

## Parameters

<i>tablero</i>	-
<i>l</i>	-

## 5.14 src/interfaz-grafica.c File Reference

File containing the body for OpenGL GUI functions for a final project for the "Combinatorial Optimization Heuristics" class.

```
#include "interfaz-grafica.h"
```

## Functions

- void [myInit](#) (char \*progname)  
*Inicializa la interfaz grafica.*
- void [print](#) (double x, double y, double z, char \*string)  
*Pinta texto cuando sea necesario.*
- void [dibuja\\_cuadro](#) (double h, double w)
- void [dibuja\\_pieza](#) (PIEZA \*pieza)  
*Funcion que dibuja cada pieza de tetris.*
- void [dibuja\\_figuras](#) (void)  
*Funcion que dibuja todas las piezas de tetris.*
- void [reshape](#) (int width, int height)
- void [display](#) (void)
- void [deja\\_caer](#) (void)  
*Dejamos caer a la pieza.*
- void [myKeyboard](#) (unsigned char key, int x, int y)  
*Le da comportamiento a la entrada.*
- void [specialKeyInput](#) (int key, int x, int y)  
*Le da comportamiento a la entrada (flechas).*
- void [end\\_visual\\_main](#) (void)  
*Termina la ventana.*
- void [visual\\_main](#) (int argc, char \*\*argv, TABLERO \*\*tablero, double zoom\_p)  
*Metodo visual principal.*

### 5.14.1 Detailed Description

File containing the body for openGL GUI functions for a final project for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 Este archivo implementa las funciones que necesitamos para observar las el juego de tetris que la heuristica de abejas juegs.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/abejas-tetris>

### 5.14.2 Function Documentation

#### 5.14.2.1 deja\_caer()

```
void deja_caer (  
    void )
```

Dejamos caer a la pieza.

Para acelerar el juego dejamos caer la pieza donde queramos.

#### 5.14.2.2 dibuja\_cuadro()

```
void dibuja_cuadro (  
    double h,  
    double w )
```

Esta funcion horrible dibuja la cuadrícula como Turing nos da a entender.

#### 5.14.2.3 dibuja\_figuras()

```
void dibuja_figuras (  
    void )
```

Funcion que dibuja todas las piezas de tetris.

Dado el tablero actual global, dibujara todas las piezas que el tablero posee.

#### 5.14.2.4 dibuja\_pieza()

```
void dibuja_pieza (  
    PIEZA * pieza )
```

Funcion que dibuja cada pieza de tetris.

Dependiendo del tipo de cada pieza de tetris dibuja cada pieza con sus coordenadas en el tablero actual global.

**Parameters**

<i>pieza</i>	- Es la pieza actual que se desea dibujar.
--------------	--

**5.14.2.5 display()**

```
void display (  
    void )
```

Limpiamos el buffer y cargamos la matriz identidad. Modificamos el zoom de la vista de la cuadrícula. Dibujamos las figuras y cambiamos los buffers antes de repintar.

**5.14.2.6 end\_visual\_main()**

```
void end_visual_main (  
    void )
```

Termina la ventana.

Modifica la variable de .

**5.14.2.7 myInit()**

```
void myInit (  
    char * prognome )
```

Inicializa la interfaz grafica.

Inicializa el modo de diplay mientras que le da tamaño a la ventana y su posición en la pantalla.

**5.14.2.8 myKeyboard()**

```
void myKeyboard (  
    unsigned char key,  
    int x,  
    int y )
```

Le da comportamiento a la entrada.

Revisamos los casos Exit - salimos. Espacio - Dejar caer la pieza.

#### 5.14.2.9 print()

```
void print (
    double x,
    double y,
    double z,
    char * string )
```

Pinta texto cuando sea necesario.

Codigo de para imprimir texto:

See also

[https://www.opengl.org/discussion\\_boards/showthread.php/169189-Printing-Text](https://www.opengl.org/discussion_boards/showthread.php/169189-Printing-Text)

#### 5.14.2.10 reshape()

```
void reshape (
    int width,
    int height )
```

Solo debe cambiar la perspectiva dependiendo del tamaño.

#### 5.14.2.11 specialKeyInput()

```
void specialKeyInput (
    int key,
    int x,
    int y )
```

Le da comportamiento a la entrada (flechas).

Cada que se presiona una flecha ocurre un evento.

Parameters

<i>key</i>	- Es la flecha que es presionada.
<i>x</i>	- Es la tecla x que corresponde a la posicion del evento.
<i>y</i>	- Es la tecla y que corresponde a la posicion del evento.

#### 5.14.2.12 visual\_main()

```
void visual_main (
    int argc,
```

```
char ** argv,
TABLERO ** tablero,
double zoom_p )
```

Metodo visual principal.

Instanciamos a las variables principales como o para despues entrar en un ciclo potencialmente infinito para seguir dibujando el tablero.

## 5.15 src/interfaz-grafica.h File Reference

File containing the openGL GUI for a final project for the "Combinatorial Optimization Heuristics" class.

```
#include <stdio.h>
#include <GL/glew.h>
#include <GL/gl.h>
#include <GL/freeglut.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "lock.h"
#include "tablero.h"
#include "tetromino.h"
```

### Functions

- void `myInit` (char \*progname)  
*Inicializa la interfaz grafica.*
- void `print` (double x, double y, double z, char \*string)  
*Pinta texto cuando sea necesario.*
- void `myReshape` (int width, int height)  
*Vuelve a pintar el espacio visual.*
- void `myDisplay` (void)  
*Dibuja el espacio de busqueda.*
- void `myKeyboard` (unsigned char key, int x, int y)  
*Le da comportamiento a la entrada.*
- void `end_visual_main` (void)  
*Termina la ventana.*
- void `visual_main` (int argc, char \*\*argv, TABLERO \*\*tablero, double zoom\_p)  
*Metodo visual principal.*

## Variables

- `TABlero * tablero_principal`  
*Tablero principal.*
- `TABlero ** tablero_principal_pointer`  
*El apuntador al .*
- `double ancho`  
*Ancho.*
- `double alto`  
*Alto.*
- `double zoom`  
*Zoom.*
- `bool game_over`  
*Estatus del juego.*

### 5.15.1 Detailed Description

File containing the openGL GUI for a final project for the "Combinatorial Optimization Heuristics" class.

#### Author

Jose Ricardo Rodriguez Abreu

#### Date

14 May 2017 Este archivo define las funciones que necesitamos para observar las el juego de tetris que la heuristica de abejas juegs.

El programa usa el estandar de documentacion que define el uso de doxygen.

#### See also

<http://www.stack.nl/~dimitri/doxygen/manual/index.html>  
<https://github.com/ricardorodab/abejas-tetris>

### 5.15.2 Function Documentation

#### 5.15.2.1 end\_visual\_main()

```
void end_visual_main (  
    void )
```

Termina la ventana.

Metodo para terminar la ejecucion del programa visual.

Modifica la variable de .

#### 5.15.2.2 myDisplay()

```
void myDisplay (
    void )
```

Dibuja el espacio de busqueda.

Dibuja y muestra la interfaz grafica.

#### 5.15.2.3 myInit()

```
void myInit (
    char * programe )
```

Inicializa la interfaz grafica.

Inicia los valores que debe tener un GUI en OpenGL y glut.

##### Parameters

<i>programe</i>	- Es el nombre del programa a ejecutar.
-----------------	---

Inicializa el modo de diplay mientras que le da tamanio a la ventana y su posicion en la pantalla.

#### 5.15.2.4 myKeyboard()

```
void myKeyboard (
    unsigned char key,
    int x,
    int y )
```

Le da comportamiento a la entrada.

Cada que se presiona una tecla ocurre un evento.

##### Parameters

<i>key</i>	- Es la tecla que es presionada.
<i>x</i>	- Es la tecla x que corresponde a la posicion del evento.
<i>y</i>	- Es la tecla y que corresponde a la posicion del evento.

Revisamos los casos Exit - salimos. Espacio - Dejar caer la pieza.

#### 5.15.2.5 myReshape()

```
void myReshape (
    int width,
    int height )
```



Vuelve a pintar el espacio visual.

Cada que se ejecute una actualizacion del espacio se debe llamar.

**Parameters**

<i>width</i>	- Es el ancho del espacio visual.
<i>height</i>	- Es el largo del espacio visual.

**5.15.2.6 print()**

```
void print (
    double x,
    double y,
    double z,
    char * string )
```

Pinta texto cuando sea necesario.

Dado coordenadas, pintar el texto en el espacio.

**Parameters**

<i>x</i>	- Es la coordenada x del espacio de OpenGL.
<i>y</i>	- Es la coordenada y del espacio de OpenGL.
<i>z</i>	- Es la coordenada z del espacio de OpenGL.
<i>string</i>	- Es la cadena que queremos mostrar.

Codigo de para imprimir texto:

**See also**

[https://www.opengl.org/discussion\\_boards/showthread.php/169189-Printing-Text](https://www.opengl.org/discussion_boards/showthread.php/169189-Printing-Text)

**5.15.2.7 visual\_main()**

```
void visual_main (
    int argc,
    char ** argv,
    TABLERO ** tablero,
    double zoom_p )
```

Metodo visual principal.

Este metodo se encarga de decirle al thread visual que debe hacer.

**Parameters**

<i>argc</i>	- Es el numero de parametros que el usuario paso.
<i>argv</i>	- Son los parametros del usuario.
<i>tablero</i>	- Es el apuntador a la ubicacion del tablero.
<i>zoom↵ _p</i>	- Es el zoom que el usuario desea ver su tablero.

---

Instanciamos a las variables principales como o para después entrar en un ciclo potencialmente infinito para seguir dibujando el tablero.

### 5.15.3 Variable Documentation

#### 5.15.3.1 alto

```
double alto
```

Alto.

Es la altura que nuestra cuadrícula de OpenGL tendrá.

#### 5.15.3.2 ancho

```
double ancho
```

Ancho.

Es el ancho que nuestra cuadrícula de OpenGL tendrá.

#### 5.15.3.3 game\_over

```
bool game_over
```

Estatus del juego.

Es el estado actual del algoritmo y tablero.

#### 5.15.3.4 tablero\_principal

```
TABLERO * tablero_principal
```

Tablero principal.

Contiene la información principal del tablero que debemos ver.

#### 5.15.3.5 tablero\_principal\_pointer

```
TABLERO ** tablero_principal_pointer
```

El apuntador al .

Como la información del tablero cambia, necesitamos una variable para conocer la ubicación del nuevo tablero.

#### 5.15.3.6 zoom

```
double zoom
```

Zoom.

Es el tamaño con el que cada cuadro de la cuadrícula tendrá. [english]article [T1]fontenc [latin9]inputenc fancyhdr amstext graphicx

babel



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROYECTO FINAL DE HEURÍSTICAS DE OPTIMIZACIÓN COMBINATORIA

# **Algoritmo colonia de abejas artificiales aplicado al juego de Tetris**

*José Ricardo Rodríguez Abreu*

dirigido por

Dr. Canek Peláez Valdés

June 19, 2017

---

## 1. Enunciado del problema

Muchos de los problemas *interesantes* que podemos observar en el universo, al quererlos resolver con una computadora caen en el conjunto de aquellos la catogore aquellos problemas que pueden ser resueltos en tiempo polinomial por una mina de Turing no determinista, tambionocida como la *clase NP*. ?

Al ser las minas de Turing no deterministas difl de conseguir en la tienda de la esquina, nos limitamos a tratar de dar un aproximado a la soluci problemas pertenecientes a la clase NP con algoritmos heuricos<sup>1</sup>.

Para este proyecto, nos enfocamos en estudiar un solo problema y una heurica por cuestiones de tiempos y alcances.

### 1.2 Tetris

El problema que se estuditia siguiente pregunta: Puedes hacer que una computadora juegue infinitamente el juego de Tetris? y resulta que no ase la siguiente pregunta que nos debemos de hacer como computos es: Dado un conjunto aleatorio de piezas, qun bien puedo hacer jugar a la computadora? Cu es el mayor nmero de Tetris que puedo hacer que realice antes que la computadora pierda?

Se decidilizar la versi-line sobre la versif-line debido al manejo de memoria que debe realizar la memoria off-line. Existe la demostracilo adjunto a este documento. que ambas versiones son equivalentes y son *NP-Completo*s. ?

Para poder atacar el problema se tuvo que investigar y observar que existen 7 posibles piezas con al menos cuatro posibles orientaciones<sup>2</sup>. Para cada pieza, suponiendo un tablero de anchoalto y  $nm$  cuadros, existen  $\frac{n}{4}cm$  posibilidades para la siguiente pieza. Para la siguiente pieza se tienen aproximadamente  $\frac{n}{4}cm$  donde  $c$  representa la cantidad de piezas que se encuentran en algn nivel de  $m$ . Podemos ver que una cota aproximada de un juego de tetris de  $t$  cantidad de piezas es:

$$\left(\frac{ncm}{4}\right)^t$$

que ciertamente es n nmero bastante grande.

Imagen 1.1 Las piezas originales de tetris con sus posibles combinaciones.

### 1.2 Abejas

El algoritmo que se decidilizar fue un modelo modificado de la heurica de colonia de abejas artificiales. Esta heurica consiste en lo siguiente:

Se usaron dos tipos de abejas: La abeja waggle (una especie de abeja reina) y las abejas exploradoras.

En la versiiginal se usan tres tipos de abejas: la empleada, la abeja en espera y las abejas exploradoras. En mi modelo podemos ver que una abeja exploradora en descanso (o no realizando ninguna accis automcamente transformada en una abeja en espera mientras que todas tienen que realizar el trabajo de empleada para explorar su fuente de alimento. El algoritmo original tiene la siguiente estructura: ?

---

<sup>1</sup> Con un algoritmo heurico (o simplemente heurica), nos limitaremos a hablar de aquellos algoritmos que tienen una caracterica probabilistica o aleatoria.

<sup>2</sup> Ver imagen 1.1

- 
- Se producen fuentes de alimentos iniciales por cada abeja empleada.
  - REPETIR
    - Cada abeja empleada va a una fuente de alimento en su memoria y determina una fuente vecina, entonces evalúa su cantidad de nar y danza en la colmena.
    - Cada abeja en espera observa el baile de cada abeja empleada y escoge una de sus fuentes dependiendo de las danzas, y entonces va a esa fuente. Después escoger un vecino alrededor, evalúa su cantidad de nar.
    - Son determinadas las fuentes de alimentos a abandonar y reemplazadas por las nuevas fuentes de alimentos descubiertas por las exploradoras. La mejor fuente de alimentos encontrada es registrada.
  - HASTA QUE (los requisitos se cumplan)

Imagen 1.2 Diagrama de flujo del algoritmo que sigue la heurística de colonia de abejas.

## 2. Integración heurística al problema

### 2.1 Problema de On-line Vs Off-line

Al comenzar el modelado de este problema se observó al realizar el problema de buscar solución problema on-line, tenemos un tipo de abeja que no eran necesarias ya que sus funciones realizaban al mismo tiempo otras. Las abejas exploradoras realizan el trabajo de las abejas empleadas ya que no podemos revisar “vecinos siguientes” a las porque el vecino puede ser cualquiera de las 7 piezas dentro de todas las posibilidades del tablero, lo cual también es un número bastante grande.

En el modelo también contemplamos el objetivo de las abejas empleadas era confirmar que la dirección una solución fuera mucho “polem”, que es el indicador de que tan buena es una solución se decidió al encontrar una solución jugar una cantidad predefinida de partidas (que llamaremos “distancia”) para estimar que el tablero de tetris no se hiciera “malo” repentinamente con la decisión la abeja exploradora. ?

### 2.2 Abejas empleadas

El programa comienza con un número predefinido de abejas empleadas que se dedican a explorar su tablero. Cada una de las abejas realiza una copia del tablero original y mueve la pieza actual hasta obtener una solución quiere decir que la pieza actual no pueda seguir cayendo en el espacio.

Una vez que se decide colocarla, la abeja empleada mueve un par de veces la pieza hacia la izquierda o derecha hasta que la pieza quede fija en todos los posibles sentidos.

Al tener un tablero con una posible solución una abeja va hacia las abejas “observadoras” que se traduce a una función costo que se calcula con una función este paso se llama el Waggle dance de la abeja.

Si la abeja resulta la mejor local o la mejor global, se le concede el “honor” de ser coronada como la “Waggle bee”. Un equivalente a lo que será abeja reina. ?

---

## 2.3 Waggle bee

La “Waggle bee” o la abeja Waggle o la abeja reina es aquella que despue cada iteraciopia el tablero que obtuvo la mayor cantidad de puntos “waggle”, o sea la mayor cantidad de puntos durante la evaluaci su tablero.

La Waggle Bee tambis la encargada de comunicar a sus abejas empleadas cual es el tablero que todas deben tener antes de comenzar la siguiente iteraci

## 2.4 Waggle dance

Es la funcis importante que da la relacirecta entre las abejas y el tablero. Esta funcivisa muchos factores como la cantidad de tetromin el tablero, el nmero de tetrís que se han realizado, el nmero de bloques sobre un mismo nivel y que tan horizontal es la la que dibujan todos los bloques superiores.

## 2.4 El tablero y las fichas

El tablero de tetrís es definido para cada entrada del programa sin embargo las fichas son las que siempre hemos conocido. Siete tetromine consisten en todas sus posibles combinaciones sin repeticiones.

Las reglas del tablero consisten en las mismas que en un tablero de tetrís normal: Si se llegara a realizar una línea horizontal llena de bloques de tetromin dice que se realiz“tetrís” y esa línea es eliminada.

Como nos hemos apegado a un juego completamente “on-line” podemos comparar las reglas con las que existieron para el juego original.

Imagen 2.1. La primera versi Tetris, lanzado en 1984, corriendo en un emulador de una computadora DVK-2 de la ex Univica.



---

### **3. Experimentaci**



---

Se realizó experimentación buscando la semilla que contuviera la mayor cantidad de "Tetris" dado una secuencia pseudoaleatoria de piezas y la función costo.

Durante la experimentación se necesitó llevar el conteo de la cantidad de espacios vacíos que se iban dejando en el transcurso de un juego como un factor negativo. Como el objetivo era realizar la mayor cantidad de Tetris en el juego, se consideró principal mantener una "horizontalidad" entre las piezas y una función dijera que tan probable era realizar un tetris en el siguiente turno.

De todas las semillas que se probaron, la que mayor cantidad de Tetris obtuvo fue la semilla 5117 con la función generadora de números aleatorios de MacOS y la semilla 915 con la función del sistema Linux.

Imagen 3.1 Fin de la mejor partida obtenida con la semilla 5117 en el sistema MacOS. 42 Tetris.



---

## **4. Resultados preliminares:**



---

Aunque la heurística ayude a encontrar soluciones inmediatas a ciertas secuencias de piezas, el problema de ver “hacia el futuro” de las piezas se mantiene y es básicamente imposible resolver con una lista lo suficientemente grande.

Convertir el problema a un problema “off-line” posiblemente incrementar número de tetris pero al ser un juego en el que eventualmente perder ambos problemas convergen al mismo punto por las mismas limitantes: ¿Cu conviene al futuro después de  $n$  piezas?





---

## 5. Documentación:



---

# Bibliography

Maurtua, I. et al., 2006. Ambient Intelligence in Manufacturing,

B. Akay and D. Karaboga. Artificial bee colony algorithm for large-scale problems and engineering design optimization. *Journal of Intelligent Manufacturing*, 23(4):1001–1014, 2012.

Demaine, E.D., Hohenberger, S. & Liben-Nowell, D., 2002. Tetris is Hard, Even to Approximate. , pp.1–56. Available at: <http://arxiv.org/abs/cs/0210020>.

Arora, S. & Barak, B., 2007. Computational Complexity: A Modern Approach. *Computational Complexity*, (January).