Documentación

# Requerimientos

* El programa debe poder manejar por lo menos 200,000 jugadores
* El programa debe ser capaz de buscar el dato que el usuario quiera
* El programa debe ser capaz de leer un archivo csv
* El programa debe poder añadir o eliminar jugadores desde la interfaz grafica
* El programa debe poder modificar las estadísticas de los jugadores
* Al menos 4 estadísticas deben tener una búsqueda de complejidad de tiempo de O (log n)
* Todo jugador debe tener nombre, edad, equipo y 5 estadísticas

# Requerimientos no funcionales

* La interfaz gráfica debe ser fácil de entender
* El programa debe estar organizado
* El uso de un árbol rojo, negro
* El uso serialización de los arboles binarios

# Diagrama de clases

# Pruebas unitarias

# Diagrama de pruebas unitarias

# TAD

|  |
| --- |
| **TAD** BinaryTree |
| |  | | --- | | Node<T,E> root | |
| { inv:  } |
| Operaciones primitivas:  BinaryTree  ·         getRoot    null -> Node<T,E>  ·         setRoot    Node<T,E> -> null  ·         insert T,E -> null  ·         search T,E -> Node<T,E>  ·         getSameValueNode T,E -> ArrayList<Node<T,E>>  treeSuccessor    Node<T,E> -> Node<T,E>  delete   T,E -> Node<T,E>  deleteSpecificPlayer    T,E -> Node<T,E>  printTree    Node<T,E> -> void |

|  |
| --- |
| BinaryTree()  “Constructor de la clase, crea un objeto con un Node<T,E> vacio”  { pre: un valor para T y E }    { post:  BinaryTree = {Node<T,E>:null }} |

|  |
| --- |
| getRoot()  “Devuelve el valor del root, es decir, el Node”  { pre: la existencia de root}    { post:  element = {root }} |

|  |
| --- |
| search (T el, E Player)  “Le da un valor a root”  { pre: La existencia del root }  { post:  root = {root }} |

|  |
| --- |
| getSameValueNode(T el, E player)  “Devuelve todos los nodos que tengan el mismo valor T que el de los parametros”  { pre: la existencia de root y que este tenga un valor}  { post: Se retorna un arreglo de nodos con el mismo valor T} |

|  |
| --- |
| insert (T el, E player)  “Añade un nodo al árbol binario”  { pre: la existencia de root}  { post:  element = {root }} |

|  |
| --- |
| treeSuccessor(Node<T,E> node)  “Busca el nodo que este más a la izquierda del nodo que se entregó por parámetros”  { pre: node no puede ser nulo }  { post:  retorna el nodo más a la izquierda de node} |

|  |
| --- |
| delete(T el, E Player)  “Se borra el primer nodo que se encuentre con el valor: el”  { pre: root no puede ser nulo}  { post:  retorna el nodo que se borra} |

|  |
| --- |
| deleteSpecificPlayer (T el, E Player)  “Se borra el primer nodo que se encuentre con el valor: el y Player”  { pre: root no puede ser nulo}  { post:  retorna el nodo que se borra} |

|  |
| --- |
| printTree (Node<T,E> root)  “Imprime todos los valores de todos los nodos de root”  { pre: root no puede ser nulo}  { post:  } |