UNIVERSIDAD DE GRANADA, ETSIIT
ESTRUCTURA DE DATOS 2017-2018

Prática Final: ¿Quién es quién?

Julián Arenas Guerrero
Ana Isabel Guerrero Tejera

January 7, 2018

1 TRABAJO REALIZADO

Se han implementado todos los métodos pedidos, los de la parte obligatoria y los de la parte opcional. Se ha implementado también una función para eliminar los nodos redundantes del árbol.

Los únicos ficheros que han sido modificados son *quienesquien.h* y *quienesquien.cpp*, y no se han añadido ficheros adicionales.

1.1 IMPLEMENTACIÓN MÉTODOS

QuienEsQuien :: *crear_arbol*(): nos hemos servido de una función recursiva auxiliar. Nos hemos servido de dos tableros y vectores de personajes auxiliares, en los cuales vamos cargando la información dividida después de hacer una pregunta. Anotar que tanto en esta, como en el resto de funciones auxiliares que se han implementado, se han pasado los parámetros por referencia, lo cual es mucho más rápido. También señalar que el uso de *const* en los parámetros es importante para no modificar variables que no queremos y en las funciones recursivas dependiendo que si queremos modificar variables en función de los niveles.

QuienEsQuien :: *eliminar_nodos_redundantes*(): se va recorriendo recursivamente el árbol con una función auxiliar. Se va comprobando si hay preguntas inútiles, y en el caso de

que se encuentre una, mediante la función $assign\_subtree()$ se asigna, al nodo padre de la pregunta inútil, su hijo no nulo .

$QuienEsQuien :: iniciar\_juego()$: esta función no presenta gran complicanción, hemos decidido mostrar antes de realizar una pregunta los personajes levantados y las preguntas formuladas, de manera que sea más sencillo jugar y se pongan en uso las funciones $QuienEsQuien :: informacion\_jugada(bintree < Pregunta >:: nodejugada\_actual)$ y $QuienEsQuien :: preguntas\_formuladas(bintree < Pregunta >:: nodejugada)$.

$QuienEsQuien :: informacion\_jugada(bintree < Pregunta >:: nodejugada\_actual)$: nos hemos servido de una función auxiliar, de manera que recorremos el árbol recursivamente y al llegar a los nodos hoja guardamos los nombres en el set, que se pasa por parámetro y en el que guardamos la información final.

$QuienEsQuien :: informacion\_jugada(bintree < Pregunta >:: nodejugada\_actual)$: la implementación es muy parecida al método anterior. En este caso al llegar a las hojas, en vez de guardar los nombres, guardamos la profundidad (la cual se va incrementando por parámetro al recorrer recursivamente el árbol). La profundidad se va sumando sobre una variable que se pasa como parámetro por referencia, que después devolvemos dividida entre el número de personajes. La variable $totalProfundidad$ es de tipo $float$ para obtener los decimales en la dividión que hacemos en el return.

$QuienEsQuien :: preguntas\_formuladas(bintree < Pregunta >:: nodejugada)$: recorremos el árbol iterativamente hacia arriba. Vamos guardando la información de las preguntas en un string. Para saber si se respondió si ó no, hacemos una simple comprobación.

$QuienEsQuien :: aniade\_personaje(stringnombre, vector < bool > caracteristicas)$: primero introducimos el nombre al final del vector de personajes y las características al final del tablero. Después vamos recorriendo el árbol iterativamente según las características del personaje a añadir hasta llegar a un nodo hoja. Encontrar el índece en el tablero de este nodo hoja (que es un personaje). Después obtenemos el string de la última pregunta que se formuló para llegar a este nodo hoja. Posteriormente buscamos el índice de esta pregunta en el tablero (en la dimendión de los atributos). Después buscamos el siguente atributo (pregunta) válida, la cual tenga una respuesta distinta para el personaje del nodo hoja alcanzado y el personaje a añadir. Finalmente sustituímos el nodo hoja por esta pregunta y se le añaden como hijo los dos personajes. Ir actualizando, además, el número de personajes que quedan levantados antes de cada pregunta.

$QuienEsQuien :: elimina\_personaje(stringnombre)$: encontramos el índice del personaje en el vector de personajes. Recorrer el árbol según las características del personaje en tablero hasta alcanzarlo, e ir actualizando el número de personajes levantados antes de cada pregunta. Guardamos también si el nodo por el que se va era un hijo derecha ó izquierda, para obtener después la información de su hermano. Este hermano lo gardamos como un árbol con un sólo nodo el cual mediante la función $replace\_subtree()$ introduciremos como

subárbol a su padre. Finalmente actualizamos el tablero y el vector de personajes.