

Capítulo 1

Introducción: Problema y Solución

Capítulo 2

Comprensión de Programas

Capítulo 3

Análisis de Identificadores: Estado del Arte

Capítulo 4

Identifier Analyzer (IDA)

4.1. Introducción

En el capítulo anterior, se explicó la importancia de analizar ids ubicados en los códigos. Los mismos ocultan información que es propia del Dominio del Problema (ver capítulo 2). Desafortunadamente, las personas ajenas al código, no comprenden a primera vista la información que esconden los ids e invierten tiempo en entender su presencia en el código. Es por esto, que las herramientas automáticas de análisis de ids son bienvenidas en el ámbito de la Comprensión de Programas (CP). Con estas se logra achicar los tiempos de comprensión de ids y revelar la información poco visible que contienen.

Dada la importancia que tienen las herramientas de análisis de ids, se tomó como iniciativa desarrollar una llamada Identifier Analyzer (IDA). Esta herramienta le permite al usuario ingresar un archivo JAVA y luego IDA analiza los ids que están en el archivo, mostrando como resultado una tabla del análisis realizado.

Para que IDA pueda realizar su tarea, contiene implementados 2 algoritmos de división de ids (Greedy y Samurai) y 1 algoritmo de expansión de abreviaturas (Expansión Básica), los mismos fueron explicados en el capítulo precedente.

A lo largo de este capítulo, se explicarán los distintos módulos que IDA posee y que proceso de ejecución debe realizar el usuario para analizar los ids.

Al final del capítulo, se describen algunos casos de estudio que demuestran la importancia de haber construido IDA. Todas estas explicaciones, están acompañadas con capturas de pantalla y tablas que facilitarán al lector entender el funcionamiento de la herramienta IDA. Para comenzar con la descripción de IDA, en la siguiente sección se explica la arquitectura y cuales son sus componentes principales.

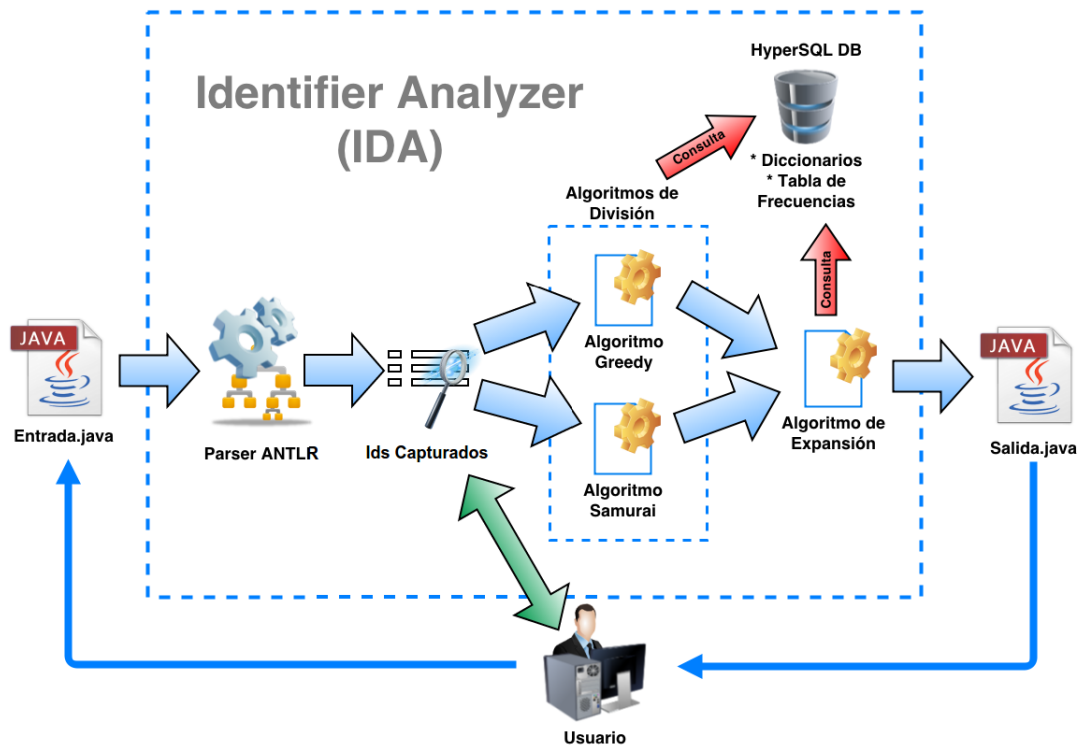


Figura 4.1: Arquitectura de IDA.

4.2. Arquitectura

En la figura 4.1 se puede apreciar la arquitectura de IDA. Esta arquitectura describe tres partes principales, la primera consiste en la *extracción de datos*, la segunda trata sobre la *división de ids* y la tercera sobre *expansión de ids*. A continuación se detallan cada una de ellas.

Extracción de Datos: En el primer paso, la herramienta IDA recibe como entrada un archivo JAVA que ingresa el usuario (ver Figura 4.1 Entrada.java), luego este archivo se procesa por un Analizador Sintáctico (AS) (ver Figura 4.1 Parser ANTLR). El AS, extrae y almacena en estructuras internas la información estática perteneciente al código del archivo ingresado. Esta información extraída, está relacionada con ids, literales y comentarios (ver próxima sección para más detalles). El usuario a través de la interfaz de IDA, puede visualizar esta información capturada del código por medio de

tablas claramente definidas (ver figura 4.1 - Flecha Verde).

División de Ids: Una vez completada la extracción de información, se da comienzo a la fase de división de ids. La misma tiene implementada dos algoritmos de división; uno es el Algoritmo Greedy y el otro es el Algoritmo Samurai ambos explicados en el capítulo anterior. Estos algoritmos reciben como entrada la información capturada en la parte de extracción de datos, luego estos algoritmos se encargan de dividir los ids del archivo JAVA (ver figura 4.1 Algoritmos de División). Los resultados de las divisiones se almacenan en estructuras internas que serán consultadas en la próxima parte. Cabe recordar que estos algoritmos de división necesitan datos externos para funcionar, uno es el diccionario de palabras (en caso de Greedy) y el otro es lista de frecuencias globales de aparición de palabras (en caso de Samurai). Estos datos externos se encuentran almacenados en una base de datos embebida (esta base se explica con mas detalles en la), y son consultados por ambos algoritmos de división (parte superior de la figura 4.1 - Flechas Rojas).

Fase de Expansión de Ids: La tercera y última etapa tiene implementado el Algoritmo de Expansión Básico de abreviaturas (ver sección ??). Este lgoritmo toma como entrada los ids divididos en la etapa anterior, tanto de Greedy como de Samurai. Luego, el Algoritmo de Expansión procede a expandir los ids obtenidos por estos 2 algoritmos, dando como resultado ids expandidos desde Greedy y desde Samurai (ver figura 4.1 Algoritmo de Expansión). En este punto, el usuario podrá elegir que expansión es la más adecuada y reemplazar los ids originales en el archivo de entrada, generando de esta manera un nuevo archivo de salida (ver figura 4.1 Salida.java). El Algoritmo de Expansión también necesita de un diccionario de palabras, por eso se realizan consultas a la base de datos embebida (parte superior de la figura 4.1 - Flechas Rojas).

4.3. Analizador Sintáctico (Parser)

Como se explicó en la sección previa, cuando el usuario ingresa un archivo JAVA, IDA utiliza un Analizador Sintáctico (Parser) que examina y extrae información estática presente en el archivo ingresado. Esta información está compuesta por identificadores, comentarios y literales.

La construcción de este Parser se llevó a cabo, primero investigando herramientas encargadas de construir Analizadores Sintácticos. Se dio preferencia a aquellas que emplean la teoría asociada a las gramáticas de atributos [1]. De la investigación previamente descrita, se determinó que la herramienta *ANTLR*¹ era la que mejor se ajustaba a las necesidades antes planteadas. La herramienta *ANTLR* genera un Analizador Sintáctico JAVA en función de la gramática JAVA² que es pasada como entrada. En esta gramática se agregaron acciones semánticas que capturan información relacionada a ids, comentarios y literales. Estas acciones no alteran la estructura original de la gramática. Una vez insertadas estas acciones y si están correctamente colocadas, *ANTLR* se encarga de leer la gramática y generar el Parser adicionando las acciones que fueron insertadas. De esta manera, se obtiene un Parser que examina código JAVA y recolecta datos necesarios (ids, comentarios, literales) que serán utilizados en próximas etapas de la herramienta IDA.

4.4. Base de Datos Embebida

Dado que IDA necesita diccionarios y listas de palabras para poder funcionar, se utilizan casi los mismos que emplean los autores de las técnicas de análisis de ids explicadas en la sección ???. Estos diccionarios/listas están almacenadas dentro de IDA en una base de datos embebida. Esta base de datos utiliza una tecnología llamada HSQLDB³ y al estar desarrollada en JAVA permite una correcta integración con IDA. Otra ventaja que tiene esta tecnología es que responde rápidamente las consultas.

¹ANother Tool for Language Recognition. <http://wwwantlr.org>

²Especificaciones de la gramática JAVA en <http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf>

³Hyper SQL Data Base. <http://www.hsqldb.org>

A continuación, se describen los diccionarios/listas de palabras que están alojadas en la base de datos explicada en el párrafo anterior. También se nombran los algoritmos implementados en IDA que consultan cada lista.

Diccionario en Inglés (ispell): Contiene palabras en Inglés que pertenecen a la lista de Palabras de *Ispell*⁴. Se utiliza en el Algoritmo de Greedy (ver sección ??) y en el Algoritmo de Expansión (ver sección ??).

Lista de Palabras Excluyentes (stop-list): Esta compuesta con palabras que son poco importantes o irrelevantes en el análisis de ids. Se utiliza en el Algoritmo de Greedy (ver sección ??) y en el Algoritmo de Expansión (ver sección ??).

Lista de Abreviaturas y Acrónimos Conocidas: Contiene abreviaturas comunes del idioma Inglés y Acrónimos conocidos de programación (gif, jpg, txt). Se emplea en el Algoritmo Greedy (ver sección ??).

Lista de Prefijos y Sufijos Conocidos: Posee Sufijos y Prefijos conocidos en Inglés, esta lista fue confeccionada por el autor del Algoritmo Samurai (ver sección ??). Se consulta solo en dicho algoritmo.

Frecuencias Globales de Palabras: Lista de palabras, junto con su frecuencia de aparición. Esta lista fue armada por el autor del Algoritmo Samurai. Se emplea solo en dicho algoritmo, más precisamente en la función de *Scoring* (ver sección ??).

Cabe destacar que las listas y diccionarios que fueron descriptos poseen palabras que pertenecen al idioma Inglés, dado que los autores así lo determinaron. Por lo tanto, para que la herramienta IDA analice correctamente los ids, se recomienda ingresar en IDA archivos JAVA con comentarios, literales e ids acordes a la lengua Inglesa.

Habiendo descripto a grandes rasgos los principales módulos de la herramienta, en la próxima sección se explicará el proceso que debe seguir el usuario para analizar ids a través de IDA.

⁴Comando de Linux. <http://wordlist.aspell.net>



Figura 4.2: Barra de Menú de IDA

4.5. Proceso de Análisis de Identificadores

En esta sección, se describe el flujo de ejecución que debe seguir el usuario con la herramienta IDA para llevar a cabo el análisis de los ids. Se explicarán en detalles la función de cada elemento de IDA y de como cada uno de estos ayuda al usuario a analizar los ids presentes en los archivo JAVA.

4.5.1. Barra de Menús

Al ejecutar la herramienta IDA, el primer componente de interacción es una simple barra de menús ubicada en el tope de la pantalla, los botones de esta barra son *Archivo*, *Diccionarios* y *Ayuda* (ver figura 4.2).

Al pulsar¹ en *Archivo* de la barra antedicha, se despliega un menú con los siguientes ítems (ver figura 4.2 - Flecha 1):

Abrir archivo(s) JAVA: Abre una ventana que permite elegir uno o varios archivos con extensión JAVA (ver figura 4.3). Los archivos seleccionados serán analizados por IDA (más detalles en la próxima sección).

Cerrar Todo: Cierra todos los archivos JAVA abiertos actualmente en la aplicación.

Salir: Cierra la Aplicación.

Cuando se pulsa en *Diccionarios* de la barra de menús, se despliega otro menú con con los siguientes ítems (ver figura 4.2 - Flecha 2):

¹El término ‘pulsar’ o ‘presionar’ se utilizará a lo largo del capítulo, significa hacer click con el puntero del ratón.

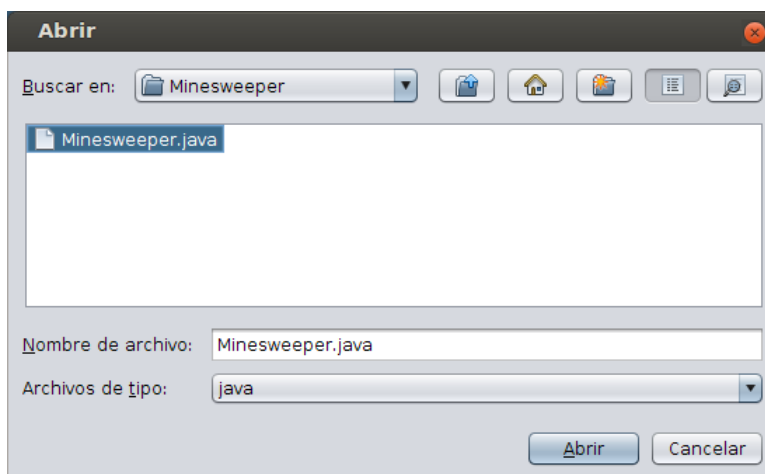


Figura 4.3: Ventana para seleccionar Archivos JAVA.

Ver Diccionarios: Abre una ventana, que muestra un listado de palabras en Inglés que pertenece al comando de Linux *ispell*. La ventana antedicha, también muestra un listado de palabras irrelevantes o stoplist (ver sección ??). Esta ventana, se explica con más detalles en la sección 4.5.5.

Restablecer B.D.(Base de Datos): Regenera la base de datos HSQldb. En caso de haber problemas con la base de datos, es útil regenerarla.

Finalmente al presionar *Ayuda* de la barra de menús, se despliega un solo botón (ver figura 4.2 - Flecha 3):

Acerca de: Brinda información sobre el autor y directores involucrados en la construcción de la herramienta IDA.

4.5.2. Lectura de Archivos JAVA

Cuando se pulsa en el botón abrir *Abrir archivo(s) JAVA* explicado en la sección anterior (ver figura 4.2 - Flecha 1), se abre una ventana para que el usuario elija uno o varios archivos JAVA (ver figura 4.3).

Una vez que el usuario elige el/los archivo/s, IDA utiliza un programa externo llamado JACOB¹. Este programa JACOB, recibe como entrada

¹<http://www.tiobe.com/jacobe>

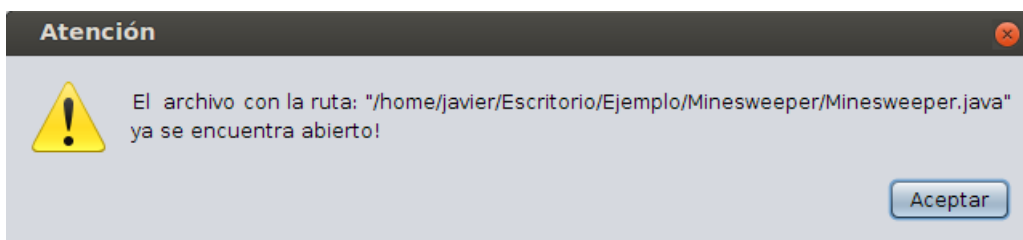


Figura 4.4: Aviso sobre Archivo JAVA ya abierto en IDA.

un archivo JAVA y embellece el código fuente contenido en el archivo. Este embellecimiento se realiza para facilitar la lectura del código al usuario. En la próxima sección se describe el panel que IDA tiene para visualizar el código leído del archivo.

La herramienta IDA, además realiza un control de los archivos abiertos, impidiendo que se abra el mismo archivo más de una vez. En caso de que esto suceda, se muestra un cartel informando al usuario (ver figura 4.4). Este control se realiza por cuestiones de coherencia al momento de analizar los archivos.

4.5.3. Panel de Elementos Capturados (Parte 1)

Una vez que el usuario selecciona el/los archivo/s JAVA, y JACOBE embellece el código de cada uno, los mismos son procesados por el parser construido con ANTLR. Al finalizar el procesamiento del parser, el *Panel de Elementos Capturados* aparece (ver figura 4.5). Este panel en la parte superior posee pestañas, cada pestaña esta rotulada con el nombre del archivo abierto correspondiente (ver figura 4.5 - Flecha 1). En caso de querer cerrar un archivo particular, esto se logra pulsando en la cruz ubicada al lado del rótulo de cada pestaña (ver figura 4.5 - Flecha 1). Es posible abrir varios archivos mediante la ventana de selección de archivos (ver figura 4.3), o también se puede ir eligiendo de a un archivo por apertura de esta ventana.

Cada pestaña en su interior posee el mismo subpanel que se divide en dos partes principales. La parte superior contiene el código leído y embellecido (por JACOBE) del archivo JAVA (ver figura 4.5 - Flecha 2). La parte inferior, muestra toda la información extraída por el parser elaborado con ANTLR

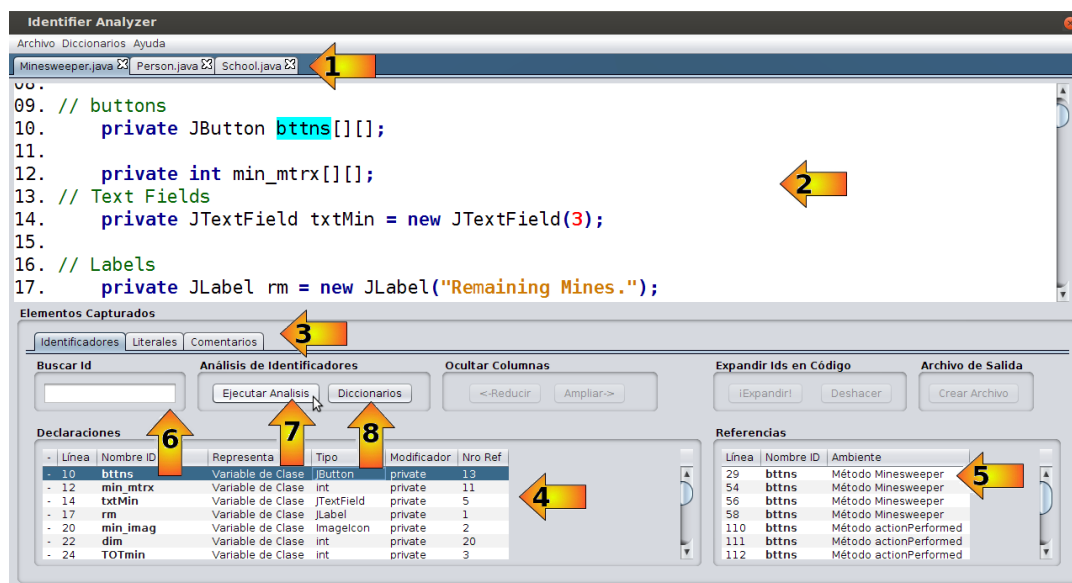


Figura 4.5: Panel de Elementos Capturados

referente a ids, literales y comentarios. Estos últimos tres poseen una pestaña que clasifican a cada uno (ver figura 4.5 - Flecha 3).

Al pulsar la *Pestaña de Literales* (ver figura 4.5 - Flecha 3), se puede apreciar que contiene una tabla y un práctico buscador para agilizar la búsqueda de literales en la tabla (ver figura 4.6 - Flecha 1). Esta tabla contiene 3 columnas, número de línea del literal, el literal propiamente dicho y la eventual asociación que pueda tener el literal con algún id (ver figura 4.6 - Flecha 2). Al pulsar sobre alguna fila de la tabla antedicha, automáticamente el literal correspondiente se resalta en el código (del archivo JAVA) ubicado en la parte superior del *Panel de Análisis* (ver figura 4.6 - Flecha 3).

Al presionar la *Pestaña de Comentarios* (ver figura 4.5 - Flecha 3), se visualiza una tabla listando los comentarios encontrados en el archivo y un buscador (ver figura 4.7 - Flecha 1) de comentarios en la tabla. Esta tabla (ver figura 4.7 - Flecha 2) contiene 2 columnas que corresponden, por un lado al comentario y por el otro al número de línea donde se encuentra el comentario dentro del código (ver figura 4.7 - Flecha 2). Al igual que se describió en el párrafo anterior, al presionar en una de las filas de la tabla inmediatamente se resalta en el código mostrando la ubicación del comentario seleccionado

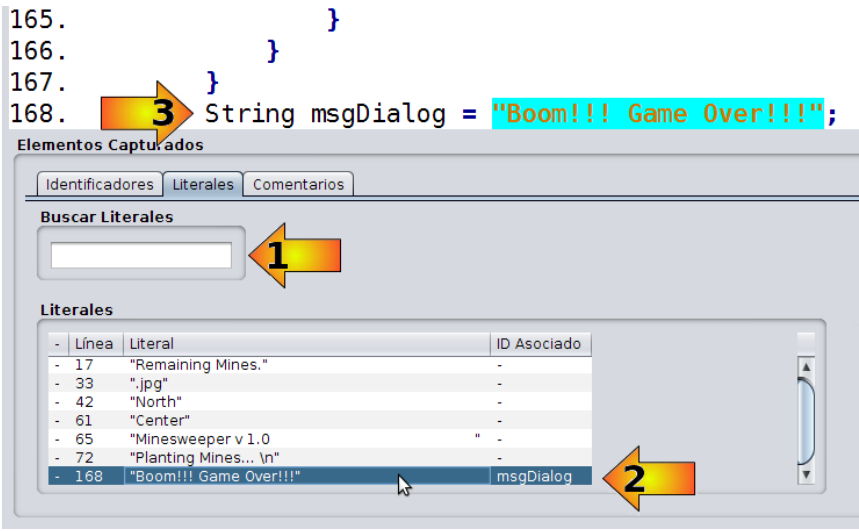


Figura 4.6: Literales Capturados

(ver figura 4.7 - Flecha 3).

Al pulsar la *Pestaña de Identificadores* (ver figura 4.5 - Flecha 3), se muestra la *Tabla de Declaraciones* (ver figura 4.5 - Flecha 4), aquí no solo se detalla el nombre del id, sino que también el número de línea donde esta declarado, el tipo (int, char, etc.), el modificador (publico, privado, protegido), lo que el id representa (variable de clase, constructor, método de clase, etc.) y cuantas referencias tiene (cantidad de lugares desde donde se utiliza como referencia, sin contar la declaración). A su vez, estas referencias son listadas en una tabla contigua llamada *Tabla de Referencias* (ver figura 4.5 - Flecha 5), la misma indica el número de línea y ambiente (ubicación de cada referencia).

Tanto la *Tabla de Declaraciones*, como la *Tabla de Referencias*, al presionar en una fila, inmediatamente se resalta con color en el código del panel superior la declaración del id o la referencia, según corresponda (ver figura 4.5 - Flecha 2). Es el mismo comportamiento que tienen las tablas de literales y comentarios explicados anteriormente. Cabe mencionar, que la *Tabla de Declaraciones* también posee un buscador, que realiza búsquedas por el nombre del id (ver figura 4.5 - Flecha 6).

Hasta aquí, solo se ha descripto como IDA exhibe la información cap-

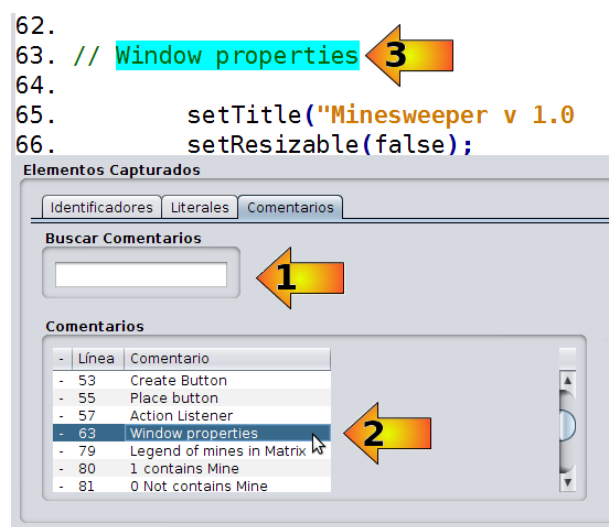


Figura 4.7: Comentarios Capturados

turada del código. A continuación, se explicará como se emplea IDA para analizar los ids. Para comenzar, se debe pulsar en el botón *Ejecutar Análisis* ubicado dentro del cuadro *Análisis de Identificadores* (ver figura 4.5 - Flecha 7), al hacerlo se abrirá el *Panel de Análisis* que será explicado en la próxima sección.

4.5.4. Panel de Análisis

El *Panel de Análisis* (ver figura 4.8) contiene 3 partes principales, (de izquierda a derecha) la primera consiste en los ids capturados, los mismos están listados en el cuadro inferior izquierdo (ver figura 4.8 - Flecha 1), arriba de estos se encuentran dos botones *Palabras Capturadas* y *Diccionarios* (ver figura 4.8 - Flecha 2), estos brindan información al usuario sobre los datos que se utilizan para ejecutar los algoritmos de análisis. Los mismos serán explicados en la próxima sección.

Siguiendo con el *Panel de Análisis*, en el cuadro central superior (ver figura 4.8 - Flecha 3) se pueden seleccionar los dos algoritmos de división de ids (Greedy y Samurai), el botón *Divir* del mismo cuadro ejecuta la técnica seleccionada mostrando en el cuadro inferior la tabla con los resultados (ver figura 4.8 - Flecha 4). De la misma manera, en los 2 paneles subsiguientes

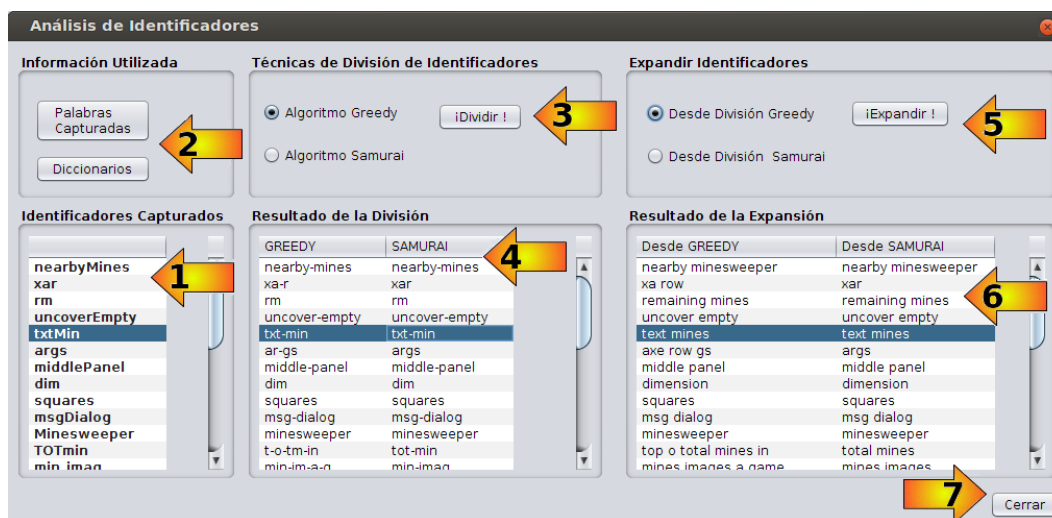


Figura 4.8: Panel de Análisis

de la derecha, permiten expandir las palabras y mostrar los resultados en el cuadro de abajo. Al presionar el botón *Expandir*, situado en el cuadro superior derecho (ver figura 4.8 - Flecha 5) ejecuta el algoritmo de expansión básico, tomando como entrada los ids divididos desde Greedy o desde Samurai, según haya seleccionado el usuario en este mismo cuadro (ver figura 4.8 - Flecha 5). Los resultados de la expansión se muestran en la tabla ubicada en el cuadro inferior derecho (ver figura 4.8 - Flecha 6).

4.5.5. Palabras Capturadas y Dicionarios

En la sección anterior, se describieron dos botones *Palabras Capturadas* y *Diccionarios*, ubicados en el *Panel de Análisis* (ver figura 4.8 - Flecha 2). Al pulsar el primero, abre una ventana que posee dos cuadros (ver figura 4.9), el cuadro de la izquierda contiene una tabla que muestra las frecuencias correspondiente al Algoritmo Samurai (ver figura 4.9 - Flecha 1). Esta tabla tiene 3 columnas, en la primera posee tokens¹, los mismos fueron capturados por el Parser ANTLR. La segunda columna, contiene la frecuencia local de cada token, cabe recordar que la frecuencia local se construye en función de la frecuencia absoluta de aparición de los tokens en el código del archivo

¹Genérico utilizado por el autor, para denotar ids, palabras de comentarios y literales.

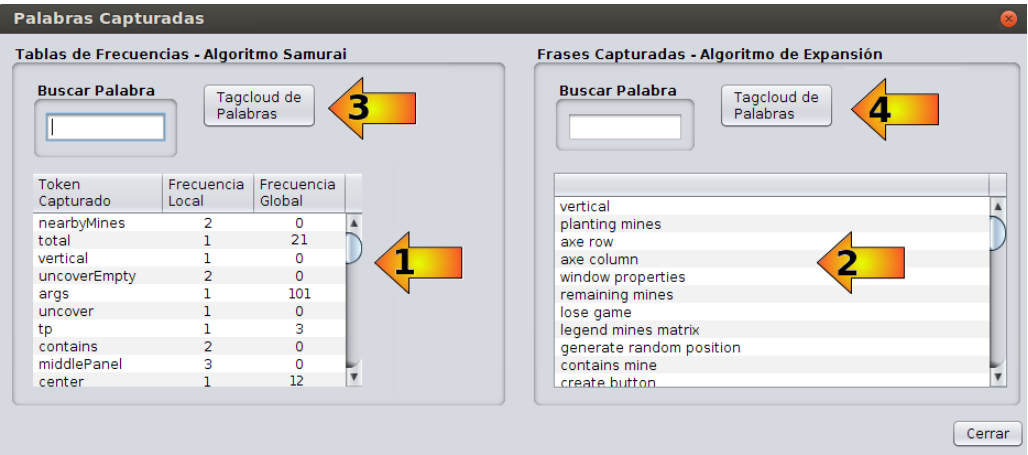


Figura 4.9: Información Utilizada para el Análisis de Ids



Figura 4.10: TagCloud: Nube de Etiquetas

actual (ver sección ??). La tercera y última columna de la tabla denota la frecuencia global de cada token, la misma esta predefinida en la base de datos HSQLDB (ver sección 4.4) y se armó mediante el análisis hecho por los autores del Algoritmo Samurai (ver sección ??).

El cuadro de la derecha, lista en una tabla las frases capturadas (ver figura 4.9 - Flecha 2), estas frases se obtienen de los comentarios y los literales strings, el Algoritmo de Expansión es el encargado de utilizarlas (ver sección ??).

Cada uno de los botones con el nombre *TagCloud de Palabras* (ver figura 4.9 - Flechas 3 y 4) abre una ventana conteniendo una *Nube de Etiquetas*

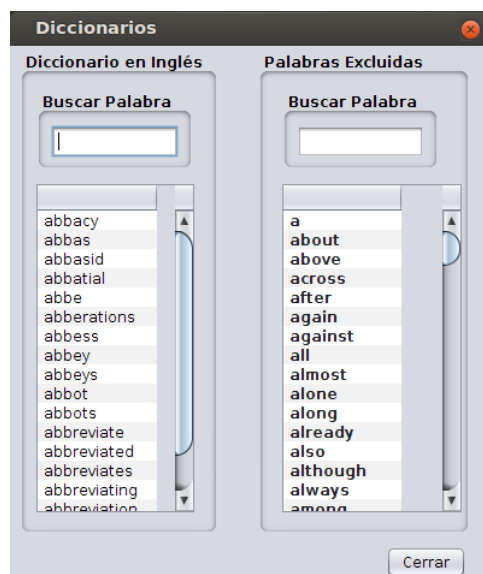


Figura 4.11: Panel de Diccionarios

(ver figura 4.10). Esta nube de palabras resalta en tamaño más grande las palabras que más frecuencia de aparición tienen. Para generar esta nube se emplea una librería de JAVA llamada OpenCloud¹. Esta nube de palabras ayuda a ver con mas claridad que palabras son más frecuentes en el código. Para el caso de la nube de frecuencias de Samurai (ver figura 4.9 - Flecha 3), el tamaño de cada palabra depende de la Frecuencia Local de cada token (ver figura 4.9 - Flecha 1). En el caso de la nube de las frases capturadas (ver figura 4.9 - Flecha 4), el tamaño de las palabras esta dado por el numero de apariciones dentro de esta tabla de frases (ver figura 4.9 - Flecha 2).

A modo de agilizar la búsqueda de alguna palabra y/o token, la misma puede ser hecha en los correspondientes cuadros de texto con rótulo *Buscar Palabra* situados al lado de cada botón *TagCloud de Palabras* (ver figura 4.9 - Flechas 3 y 4).

Volviendo al *Panel de Análisis* si se pulsa el botón *Diccionarios* (ver figura 4.8 - Flecha 2), se abre el *Panel de Diccionarios* (ver figura 4.11). Este panel posee dos tablas, la tabla de la izquierda lista todas las palabras en ingles que tiene el diccionario del comando de Linux *ispell*, esta lista de palabras es

¹<http://opencloud.mcavallo.org>

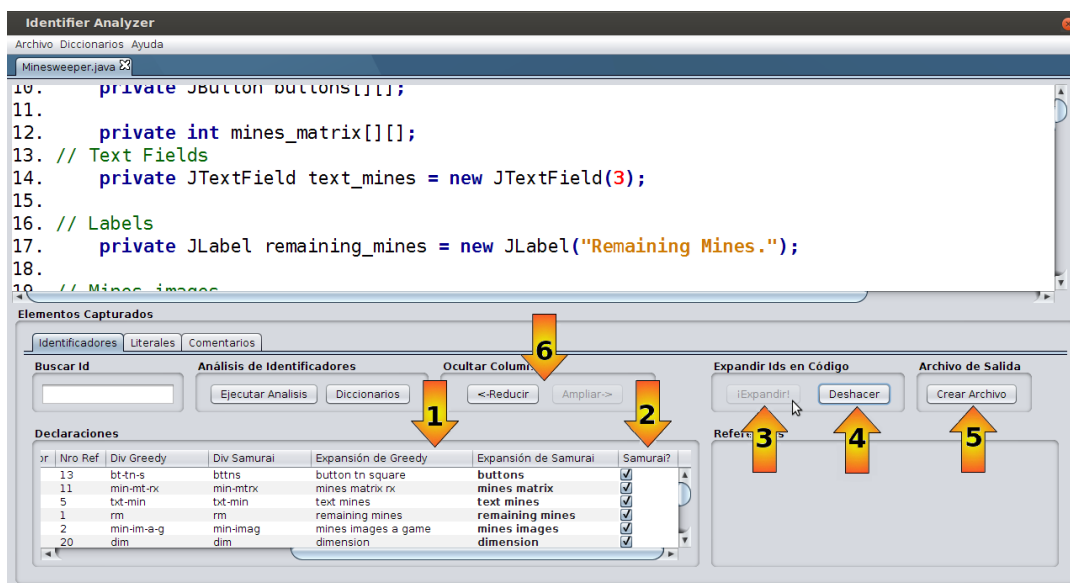
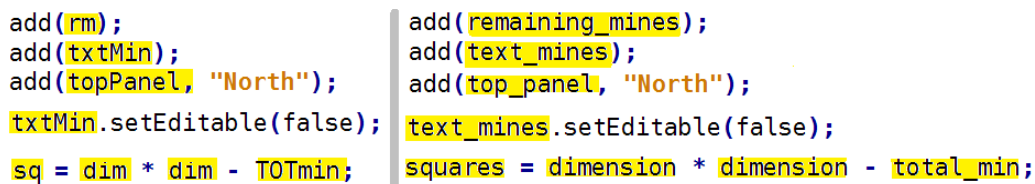


Figura 4.12: Panel de Elementos Capturados

utilizada por el Algoritmo Greedy (ver sección ??) y el Algoritmo Expansión Básica (ver sección ??). La segunda tabla de la derecha enumera las palabras que pertenecen a la stoplist o lista de palabras irrelevantes, también utilizada por los dos algoritmos antedichos. Convenientemente, ambas tablas poseen un buscador por palabras dado que el contenido de cada una es amplio (ver figura 4.11). Este *Panel de Diccionarios* puede ser invocado desde otros lugares de la herramienta IDA. Uno de ellos es desde la barra de menú (ver Figura 4.2 - Flecha 2). Otro sitio donde puede abrirse, es desde el *Panel de Elementos Capturados*, pulsando el botón que está al lado de *Ejecutar Análisis* (ver figura 4.5 - Flecha 8).

4.5.6. Panel de Elementos Capturados (Parte 2)

Una vez que los ids fueron analizados (Divididos y Expandidos) mediante el *Panel de Análisis*, el mismo debe ser cerrado presionando el botón *Cerrar* (ver figura 4.8 - Flecha 7). Esta acción, retorna al *Panel de Elementos Capturados* nuevamente (ver figura 4.12). Como se puede observar, en la tabla *Declaraciones* que detalla los ids extraídos e información asociada a estos, se



```
add(rm);
add(txtMin);
add(topPanel, "North");
txtMin.setEditable(false);
sq = dim * dim - TOTmin;

add(remaining_mines);
add(text_mines);
add(top_panel, "North");
text_mines.setEditable(false);
squares = dimension * dimension - total_min;
```

Figura 4.13: Comparación entre dos trozos de código

le suman nuevas columnas (ver figura 4.12 - Flecha 1). Estas nuevas columnas contienen los resultados obtenidos de los algoritmos de división (Greedy, Samurai) y el algoritmo de expansión ejecutados en el *Panel de Análisis*. En estas nuevas columnas, también se muestran al final casillas de selección (ver figura 4.12 - Flecha 2). Estas casillas permiten al usuario elegir, la mejor expansión realizada desde Greedy o Samurai o ninguna de las 2 dejando al id en su formato original.

Al agregar las columnas nuevas se habilita el cuadro *Ocultar Columnas* (ver figura 4.12 - Flecha 6). En el se encuentran dos botones *Reducir* y *Ampliar*. El primero de ellos a modo de facilitar la visualización, oculta las columnas que hay entre los ids y las columnas que contienen el análisis de ids (las columnas que se ocultan son: tipo, modificador, número de referencias y Representa), de esta manera el usuario puede comparar más claramente las distintas divisiones y expansiones de ids. Mientras que el botón *Ampliar* restablece las columnas originales (ver figura 4.12 - Flecha 6).

Cuando el usuario termina de seleccionar la mejor expansión de cada id, se procede al cuadro *Expandir Ids en Código*. Este cuadro contiene dos botones, al presionar *Expandir* (ver figura 4.12 - Flecha 3), la herramienta IDA reemplaza los ids del código de acuerdo a lo seleccionado en las casillas de selección en la tabla de *Declaraciones* que fue explicado al principio de esta sección. Una vez realizado esto, el código es más comprensivo para el usuario, en la figura 4.13 se puede observar dos trozos de códigos, el de la izquierda se observa el código normal, mientras que el de la derecha posee los ids expandidos. En caso de querer retrotraer la acción de reemplazo de ids, se puede pulsar en el botón *Deshacer* ubicado también en el cuadro *Expandir Ids en Código* (ver figura 4.12 - Flecha 4) y restablecer el código original.

Luego si el usuario lo decide, al presionar el botón *Crear Archivo* en el cuadro *Archivo de Salida* (ver figura 4.12 - Flecha 5), crea un nuevo archivo igual que el pasado como entrada, nada más que con los nuevos ids expandidos reemplazando los originales. Una vez realizado esto, se abre una ventana informando al usuario la creación del nuevo archivo.

4.6. Casos de Estudio

En esta sección se presentarán 3 casos de estudios realizados con la herramienta IDA. El primero será sencillo, mientras que los demás contendrán más ids para analizar. En cada uno de estos casos, se examinan los ids de un único archivo JAVA. A través de tablas, se irán mostrando los resultados parciales que se van obteniendo durante el proceso de análisis de los ids. Con estos casos, se pretende ostentar la utilidad de la herramienta IDA en lo que respecta al análisis de ids y mostrar que es un aporte al área de la CP.

4.6.1. Buscaminas (Minesweeper)

El archivo JAVA denominado Minesweeper.java, al ejecutarlo posee el clásico y conocido juego llamado Buscaminas (Minesweeper en Inglés - ver figura 4.14). Este archivo contiene 223 líneas que serán analizadas por IDA.

Al ingresar el archivo Minesweeper.java a la herramienta IDA, comienza la fase de extracción de datos y el analizador sintáctico captura información referente a los ids. Esta información la exhibe IDA al usuario en el *Panel de Elementos Capturados*, en la tabla *Declaraciones* (ver figura 4.5 - flecha 4). En la tabla 4.1 se aprecia esta información: la línea donde está declarado el id, que representa en el código analizado, el tipo, el modificador y la cantidad de referencias que tiene la declaración del id en el código.

Para hacer una descripción más detallada sobre los ids capturados, en la tabla 4.1 se exhibe que el archivo Minesweeper.java tiene ids del tipo *hardwords* y *softwords* (ver sección ??). Algunos *hardwords* que se pueden observar son `min_mtx`, `TOTmin`, `topPanel` (entre otros), ya que estos poseen una marca de separación que destacan las palabras que lo componen. Por

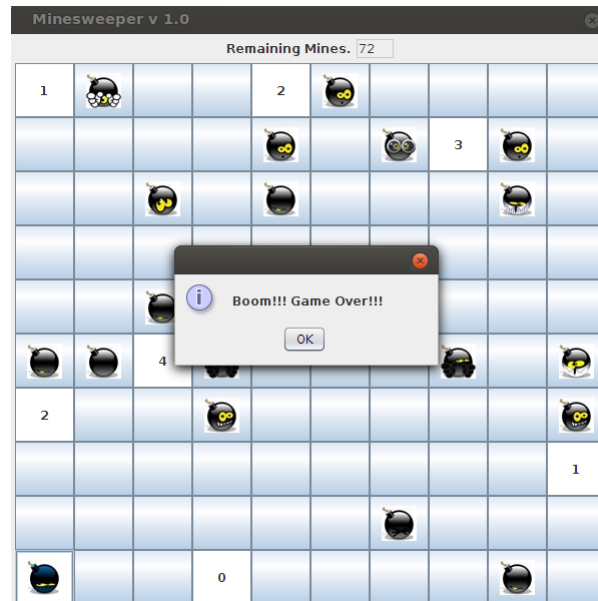


Figura 4.14: Captura del Juego Buscaminas programado en JAVA

otro lado, algunos de los *softwords* que se capturaron son `ae`, `plantmines`, `bttns` (entre otros).

A su vez, se capturaron los comentarios junto a la línea del código donde se ubican cada uno, los mismos se muestran en tabla 4.2. De la misma forma, los literales Strings que se extrajeron se exhiben en la tabla 4.3. En esta tabla, se puede observar que además del número de línea donde se ubica el literal, también muestra un eventual asociación que puede tener un literal con un id (Ejemplo: `msgDialog` = "Message"). Tanto los literales y los comentarios se visualizan en IDA a través del *Panel de Elementos Capturados*, eligiendo la pestaña correspondiente (ver figura 4.5 - Flecha 3).

Es importante recordar, que los comentarios y literales de las tablas 4.2 y 4.3 son usados para armar la lista de frases que se muestra en la ventana de *Palabras Capturadas* en la figura 4.9 - flecha 4. Esta información es útil para el Algoritmo de Expansión.

A continuación, se procede a analizar los ids, el usuario realiza esto con el *Panel de Análisis* (ver figura 4.8) aplicando las técnicas de división y después la técnica de expansión de abreviaturas. Los resultados más destacados se pueden apreciar en la tabla 4.4.

Análisis de Resultados

La información mostrada en la tabla 4.4, las columnas de *Greedy* y *Samurai* muestran los resultados de división de dichos Algoritmos. En las columnas *Expansión desde Greedy* y *Expansión desde Samurai* se enumeran los resultados de haber expandido las distintas partes del id, que resultaron desde los Algoritmos Greedy y Samurai respectivamente.

Los ids analizados por la herramienta IDA (ver tabla 4.4) en lo que respecta a *hardwords*, se pueden encontrar con guión bajo *min_imag*, *min_mtrx* para el tipo camel-case *uncoverEmpty*, *msgDialog* y para el caso especial *TOTmin*, *MINSnum* (variante camel-case), entre otros. El algoritmo Greedy manifiesta irregularidades a la hora de dividir ya que siempre considera que la mayor cantidad de divisiones es la mejor opción, esto puede observarse en casos como *min-im-ag*, *bt-tn-s*, *min-sn-um* (ver tabla 4.4 - columna Greedy). Para el caso especial *MINSnum* es interesante ver como Samurai se da cuenta de ello y lo separa correctamente (ver sección ??), mientras que Greedy supone que es del tipo camel-case haciendo la separación incorrecta *min-sn-um*.

En lo que respecta a *softwords* se aprecia la presencia de acrónimos como *rm* y *ae* (entre otros). El Algoritmo de Expansión consulta la lista de frases (ver figura 4.9 - flecha 4) conformada por comentarios y los literales capturados (ver tablas 4.2 y 4.1), al encontrar coincidencia con el literal “*remaining mines*” y el comentario “*action event*”, selecciona ambos como la expansión correspondiente de *rm* y *ae* (ver tabla 4.4 - Columnas de Expansión). El resto de los *softwords* se puede considerar a *mins*, *bttns*, *dim*, aquí Greedy también acusa inconvenientes separando los ids, mientras que Samurai no los divide (ver tabla 4.4).

Para finalizar, los ids *i*, *j* son comunes en la mayoría de los códigos, y son difíciles de traducir. Por ende, el Algoritmo de Expansión busca en las tablas de Literales y Comentarios (ver tablas 4.2 y 4.3), palabras que estén dentro del *Dominio del Problema* y de esta manera tratar de darle una traducción válida en este contexto.

Línea	Nombre ID	Representa	Tipo	Modificador	Nro. de Ref.
7	Minesweeper	Clase	–	public	–
10	bbtns	Variable de Clase	JButton	private	13
12	min_mtrx	Variable de Clase	int	private	11
14	txtMin	Variable de Clase	JTextField	private	5
17	rm	Variable de Clase	JLabel	private	1
20	min_imag	Variable de Clase	ImageIcon	private	2
22	dim	Variable de Clase	int	private	2
24	TOTmin	Variable de Clase	int	private	3
25	sq	Variable de Clase	int	private	6
37	topPanel	Variable Local	JPanel	–	3
48	middlePanel	Variable Local	JPanel	–	2
71	plantmines	Método de Clase	void	private	2
71	mins	Parámetro	int	–	1
102	main	Método de Clase	void	public	–
102	args	Parámetro	String[]	–	–
107	actionPerformed	Método de Clase	void	public	–
107	ae	Parámetro	ActionEvent	–	1
124	uncoverEmpty	Método de Clase	void	private	1
124	j	Parámetro	int	–	1
124	i	Parámetro	int	–	1
150	win	Método de Clase	void	private	1
159	boom	Método de Clase	void	private	1
172	msgDialog	Variable Local	String	–	1
179	nearbyMines	Método de Clase	int	private	1
188	MINSnum	Variable Local	int	–	5
179	xar	Parámetro	int	–	2
179	yac	Parámetro	int	–	2

Tabla 4.1: Identificadores extraídos por el Parser ANTLR

Línea	Comentario	Línea	Comentario
9	buttons	85	Generate random position
13	Text Fields	91	Place mine
16	Labels	93	Display mines panel
19	Mines images	107	Action Event
21	Dimension	126	Uncover an empty square
23	total mines	129	Nearby Mines
27	Time tp	136	restart game
31	load Images	152	Win the game
36	Top Panel	161	lose the game
47	Button panel	165	Mines Random Images
50	Create and place button	183	x axe row
53	Create Button	184	y axe column
55	Place button	186	return the number of mines
57	Action Listener	192	horizontal
63	Window properties	199	vertical
80	Legend of mines in Matrix	207	diagonal
81	1 contains Mine	208	Top left corner
82	0 Not contains Mine	209	copy of axes
83	Place random mine	224	top right corner

Tabla 4.2: Comentarios extraídos por el Parser ANTLR

Línea	Literal	Id Asociado
17	“Remaining Mines.”	—
33	“.jpg”	—
42	“North”	—
61	“Center”	—
65	“Minesweeper v 1.0 ”	—
73	“Planting Mines...”	—
153	“You Win!!! Game Over!!!”	msgDialog
155	“Message”	msgDialog
173	“Boom!!! Game Over!!!”	msgDialog
175	“Message”	msgDialog

Tabla 4.3: Literales extraídos por el Parser ANTLR

Id	Greedy	Samurai	Exp. desde Greedy	Exp. desde Samurai
Minesweeper	minesweeper	minesweeper	minesweeper	minesweeper
bttns	bt-tn-s	bttns	button tn square	buttons
min_mtrx	min-mt-rx	min-mtrx	mines matrix rx	mines matrix
txtMin	txt-min	txt-min	text mines	text mines
rm	rm	rm	remaining mines	remaining mines
min_imag	min-im-ag	min-imag	mines images ag	mines images
dim	dim	dim	dimension	dimension
TOTmin	tot-min	tot-min	total mines	total mines
sq	sq	sq	square	square
topPanel	top-panel	top-panel	top panel	top panel
middlePanel	middle-panel	middle-panel	middle panel	middle panel
plantmines	plant-mines	plant-mines	planting minesweeper	planting minesweeper
mins	min-s	mins	mines square	mines
main	main	main	main	main
args	args	args	args	args
actionPerformed	action-performed	action-performed	action performed	action performed
ae	ae	ae	action event	action event
uncoverEmpty	uncover-empty	uncover-empty	uncover empty	uncover empty
j	j	j	jpg	jpg
i	i	i	images	images
win	win	win	window	window
boom	boom	boom	boom	boom
msgDialog	msg-dialog	msg-dialog	message dialog	message dialog
nearbyMines	nearby-mines	nearby-mines	nearby minesweeper	nearby minesweeper
MINSnum	min-sn-um	mins-num	mines sn um	mines number
xar	xa-r	xar	xa row	x axe row
yac	y-ac	yac	yellow action	y axe column

Tabla 4.4: Análisis Realizado a los Ids extraídos de Minesweeper.java

4.6.2. Ejemplo 2

....

Bibliografía

- [1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, August 2006.
- [2] Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft. *Data structures and algorithms / Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman*. Addison-Wesley Reading, Mass, 1983.
- [3] Thomas Ball. The concept of dynamic analysis. In Oscar Nierstrasz and Michel Lemoine, editors, *ESEC / SIGSOFT FSE*, volume 1687 of *Lecture Notes in Computer Science*, page 216–234. Springer, 1999.
- [4] K. Bennett and V. Rajlich. Software maintenance and evolution: a roadmap. In *ICSE - Future of SE Track*, page 73–87, 2000.
- [5] M. Beron, P. Henriques, and R. Uzal. *Inspección de Programas para Interconectar las Vistas Comportamentaly Operacional para la Comprensión de Programas*. PhD thesis, Universidade do Minho, Braga, Portugal, 2010.
- [6] Mario Berón, Pedro Henriques, Maria João Pereira, and Roberto Uzal. Program inspection to interconnect behavioral and operational view for program comprehension. University of York, 2007.
- [7] Mario Berón, Daniel Eduardo Riesco, Germán Antonio Montejano, Pedro Rangel Henriques, and Maria J Pereira. Estrategias para facilitar la comprensión de programas. In *XII Workshop de Investigadores en Ciencias de la Computación*, 2010.

-
- [8] Dave Binkley, Marcia Davis, Dawn Lawrie, Jonathan Maletic, Christopher Morrell, and Bonita Sharif. The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, 18(2):219–276, 2013. (early access).
 - [9] R. Brook. A theoretical analysis of the role of documentation in the comprehension of computer programs. *Proceedings of the 1982 conference on Human factors in computing systems.*, pages 125–129, 1982.
 - [10] Bruno Caprile and Paolo Tonella. Nomen est omen: analyzing the language of function identifiers. In *Proc. Sixth Working Conf. on Reverse Engineering*, page 112–122. IEEE, October 1999.
 - [11] Bruno Caprile and Paolo Tonella. Restructuring program identifier names. In *Proc. Int’l Conf. on Software Maintenance*, page 97–107. IEEE, 2000.
 - [12] Florian DeiBenbock and Markus Pizka. Concise and consistent naming. In *IWPC*, page 97–106. IEEE Computer Society, 2005.
 - [13] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Aiding program comprehension by static and dynamic feature analysis. In *ICSM*, pages 602–611, 2001.
 - [14] Ramez A. Elmasri and Shankrant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.
 - [15] David W Embley. Toward semantic understanding: an approach based on information extraction ontologies. In *Proceedings of the 15th Australasian database conference-Volume 27*, pages 3–12. Australian Computer Society, Inc., 2004.
 - [16] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *6th IEEE International Working Conference on Mining Software Repositories.*, page 71–80. IEEE, may. 2009.

-
- [17] Henry Feild, David Binkley, and Dawn Lawrie. An empirical comparison of techniques for extracting concept abbreviations from identifiers. In *Proceedings of IASTED International Conference on Software Engineering and Applications.*, 2006.
 - [18] Henry Feild, David Binkley, and Dawn Lawrie. Identifier splitting: A study of two techniques. In *Proceedings of the Mid-Atlantic Student Workshop on Programming Languages and Systems.*, pages 154–160, 2006.
 - [19] Fangfang Feng and W. Bruce Croft. Probabilistic techniques for phrase extraction. *Inf. Process. Manage.*, 37(2):199–220, 2001.
 - [20] José Luís Freitas, Daniela da Cruz, and Pedro Rangel Henriques. The role of comments on program comprehension. In *INForum*, 2008.
 - [21] Wilhelm Hasselbring, Andreas Fuhr, and Volker Riediger. First international workshop on model-driven software migration (mdsm 2011). In *CSMR '11 Proceedings of the 2011 15th European Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*, pages 299–300, Washington, DC, USA, März 2011. IEEE Computer Society.
 - [22] Emily Hill, Zachary P. Fry, Haley Boyd, Giriprasad Sridhara, Yana Novikova, Lori Pollock, and K. Vijay-Shanker. Amap: Automatically mining abbreviation expansions in programs to enhance software maintenance tools. In *Proceedings of the 5th Int'l Working Conf. on Mining Software Repositories.*, page 79–88. ACM, 2008.
 - [23] IEEE. Ieee standard for software maintenance. *IEEE Std 1219-1998*, page i–, 1998.
 - [24] IEEE. *Standard Glossary of Software Engineering Terminology 610.12-1990.*, volume 1. IEEE Press, 1999.
 - [25] D. Lawrie, H. Feild, and D. Binkley. Extracting meaning from abbreviated identifiers. In *Source Code Analysis and Manipulation*, 2007.

- SCAM 2007. Seventh IEEE International Working Conference on*, page 213–222, Sept 2007.
- [26] D. Lawrie, C. Morrell, H. Feild, and D. Binkley. What’s in a name? a study of identifiers. In *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on.*, page 3–12, June 2006.
- [27] Dawn Lawrie, Henry Feild, and David Binkley. Syntactic identifier conciseness and consistency. In *SCAM*, page 139–148. IEEE Computer Society, 2006.
- [28] Dawn Lawrie, Henry Feild, and David Binkley. Quantifying identifier quality: an analysis of trends. *Empirical Software Engineering*, 12(4):359–388, 2007.
- [29] Michael P O’Brien. Software comprehension—a review & research direction. *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report.*, 2003.
- [30] Roger S. Pressman. *Software Engineering - A Practitioner’s Approach*. McGraw-Hill, 5 edition, 2001.
- [31] T.A. Standish. *Data structure techniques*. Computer Sciences. Addison-Wesley, 1980.
- [32] M. Storey. Theories, methods and tools in program comprehension: past, present and future. In *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on.*, page 181–191, May 2005.
- [33] M. A. Storey, F. D. Fracchia, and H. A. Müller. Cognitive design elements to support the construction of a mental model during software exploration. *The Journal of Systems & Software.*, 44(3):171–185, 1999.
- [34] P.F. Tiako. Maintenance in joint software development. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International.*, page 1077–1080, 2002.
- [35] Tim Tiemens. Cognitive models of program comprehension, 1989.

-
- [36] Marco Torchiano, Massimiliano Di Penta, Filippo Ricca, Andrea De Lucia, and Filippo Lanubile. Software migration projects in italian industry: Preliminary results from a state of the practice survey. In *ASE Workshops.*, page 35–42. IEEE, 2008.
 - [37] A von Mayrhauser and AM. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, Aug 1995.