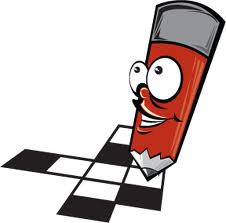
Tel-Aviv University, summer 2013

**Database Systems Project**

Crossword Mastermind

*Master your crosswords*



**Software documentation**

Introduction:

Crossword Mastermind ® *is a crosswords game that creates hot outta the oven new crosswords boards consists of questions about the user's favorites knowledge areas. It has a lot of cool and easy-to-use features that every child or adult will enjoy.   
Simply choose your topics, warm up with a little trivia question and GO!*

About YAGO's data and Crossword Mastermind:

For the purpose of creating solvable though challenging and varied crosswords boards, we began the process of planning and implementing the application by exploring the data and facts in YAGO's knowledge base. We chose the topics our crosswords will cover and then according to these topics we built a definitions table populated with 196 different definitions on which the crosswords will consist.   
The definitions are taken from ***yagoTypes.tsv*** file. Each definition is either a YAGO Wikipedia Category or a YAGO Wordnet class, e.g. "<wikicategory\_Rokdim\_Im\_Kokhavim\_participants>", "<wordnet\_programming\_language\_106898352>". Each line in yagoTypes.tsv that matches one of the definitions is extracted and the entity that appears in it is stored in entities table. Using the entities we are able to parse the answers and with a clever algorithm construct the crossword's board.   
In addition, we wanted the crosswords boards to provide not only definitions and their answers but also hints regarding these answers. Obviously, if one answer matches more than one definition, then a different definition other than that which already appears on the board may be a suitable hint. But on purpose to provide a bigger variety of hints, we used facts taken from ***yagoFacts.tsv*** and ***yagoLiteralFacts.tsv*** files. We chose predicates (YAGO relations) with relevant information about the different entities (e.g. <hasCapital>, <wasBornOnDate>) and stored them in predicates table. Each line in yagoFacts.tsv and yagoLiteralFacts.tsv files that matches one of the entities from the entities table ***and*** one of the predicates from the predicates table is extracted and stored in the database. Using these facts we construct the hints and populate the hints table.

A more detailed description of the software is in the following sections.

DB structure:

The database consists of 12 distinct tables, divided into 3 main groups:

* Eight core tables:   
  *TOPICS*, *DEFINITIONS* and *PREDICATES* describe the data we are interested in, which will be extracted from Yago files during the parsing phase.  
  *ANSWERS*, *ENTITIES* and *HINTS* hold the relevant data itself – all the answers, from which the crossword board will be constructed, their hints and the entities that bind them both. Massive import process populates these tables.  
   *DEFINITIONS\_TOPICS* and *ENTITIES\_DEFINITIONS* are connecting tables made to combine definitions with topics and entities with their definitions. Both are many-to-many relations.
* Three parsed-Yago-data tables, one for each Yago file we use:   
  *YAGO\_TYPE* holds data extracted from yagoTypes.tsv file according to the definitions in *DEFINITIONS* table, plus parsed answers and additional information.   
  *YAGO\_FACT* and *YAGO\_LITERAL\_FACT* hold data extracted from yagoFacts.tsv and yagoLiteralFacts.tsv files respectively and correlated to *PREDICATES* table. Both tables will be used for generating hints.   
  Massive import process will load data into these tables.
* One separate table, *BEST\_SCORES*, used to keep the high scores made by the users. This table won't be changed as a result of massive import.

Note: Through the following description, the term "yago\_id" does not mean the full original YagoId, rather a part of it that regards the specified entity (e.g. '1bsrlah' out of <id\_1bsrlah\_88c\_r34uav> in case the subject of that Yago fact is the specified entity or 'r34uav' if the object is the entity).   
Please notice there is an EER-diagram of the database's schema following the verbal description.

Tables, attributes and properties:

1. ***TOPICS*** *(id, name) –*   
   The topics of which the definitions/questions that will appear in the crossword relate to. The topics we chose are: Cinema & TV, General Knowledge, Geography, Israel, Music, Personalities and Sports. The topic 'User Updates' describes definitions the user added to the database.
2. **id**: int, NOT NULL, AUTO\_INCREMENT, PRIMARY KEY.
3. **name**: varchar(50), NOT NULL, UNIQUE.
4. ***DEFINITIONS*** *(id, yago\_type, definition)* –   
   The definitions that appear in the crossword. Each definition is derived from a 'yago\_type' which is a Yago entity of type <wikicategory\_xxx> or <wordnet\_xxx> taken from yagoTypes.tsv file.
5. **id**: int, NOT NULL, AUTO\_INCREMENT, PRIMARY KEY, INDEX.
6. **yago\_type**: varchar(250), NOT NUL, UNIQUE.   
   Matching yago\_type for the current definition (e.g. "<wikicategory\_1990s\_drama\_films>").
7. **definition**: varchar(500), NOT NULL.   
   The definition's string as it will appear in the crossword (e.g. "1990s drama film").
8. ***DEFINITIONS\_TOPICS*** *(definition\_id, topic\_id)* –   
   Connecting the TOPICS and DEFINITIONS tables: for each definition tells the topics it related to and for each topic the definitions it includes.
9. **definition\_id:** int, NOT NULL, FOREIGN KEY REFERENCES DEFINITIONS(id).
10. **topic\_id**: int, NOT NULL, FOREIGN KEY REFERENCES TOPICS(id).

PRIMARY KEY (definition\_id, topic\_id).

1. ***PREDICATES*** *(id, yago\_predicate, subject\_str, object\_str) –*   
   Holds the Yago predicates from yagoFacts.tsv and yagoLiteralFacts.tsv files which will be used for the hints.
2. **id**: int, NOT NULL, AUTO\_INCREMENT, PRIMARY KEY, INDEX.
3. **yago\_predicate**: varchar(50) NOT NULL, UNIQUE.  
   The predicate, taken from both yagoFacts.tsv and yagoLiteralFacts.tsv files (e.g. "<diedOnDate>", " <hasChild>").
4. **subject\_str**: varchar(250).   
   A string template from which a hint for the subject that holds this predicate will be constructed. The '?' will be replaced with the matching object.
5. **object\_str**: varchar(250).

A string template from which a hint for the object that holds this predicate will be constructed. The '?' will be replaced with the matching subject. If the object isn't related to any entity (e.g. dates, numbers) then this value is NULL.

1. ***YAGO\_TYPE*** *(subject, predicate, object, answer, additional\_informtation)* –   
   Holds data taken from yagoTypes.tsv file. The data in the table is translated to entities and answers for the ENTITIES and ANSWERS tables.
2. **subject**: varchar(100), NOT NULL.   
   Entity as it appears in Yago, concatenated by its yago\_id.
3. **predicate**: varchar(50).

Yago's predicate.

1. **object**: varchar(250), NOT NULL, FOREIGN KEY REFERENCES DEFINITIONS(yago\_type).  
   The yago\_type that holds the current predicate with the current subject (e.g. "<wikicategory\_States\_of\_the\_United\_States>"). Correlated to yago\_type from DEFINITIONS table.
2. **answer**: varchar(50).  
   The name of the entity (subject) parsed for use.
3. **additional\_information**: varchar(25).

When the answer consists of more than one word, this attribute will contain the pattern that implies the number of letters for each word.

1. ***YAGO\_FACT*** *(subject, predicate, object, is\_subject)* –   
   Holds data taken from yagoFacts.tsv file. This data will be used for constructing the hints.
2. **subject**: varchar(100).  
   Entity as it appears in Yago, concatenated by its yago\_id.
3. **predicate**: varchar(50), FOREIGN KEY REFERENCES PREDICATES(yago\_predicate).

The Yago predicate from yagoFacts.tsv . Correlated to yago\_predicate from PREDICATES table.

1. **object**: varchar(250).  
   The other entity that holds the current predicate with the subject.
2. **is\_subject**: boolean.   
   Indicates which one (subject or object) is the answer of which the hint is about.  
   is\_subject == 0 means that the subject is the hint and the object is the answer.  
   is\_subject == 1 means that the subject is the answer and the object is the hint.
3. ***YAGO\_LITERAL\_FACT*** *(subject, predicate, object)* –   
   Holds data taken from yagoLiteralsFacts.tsv file. This data will be used for constructing hints.
4. **subject**: varchar(100).   
   Entity as it appears in Yago, concatenated by its yago\_id.
5. **predicate**: varchar(50), FOREIGN KEY REFERENCES PREDICATES(yago\_predicate)  
   The Yago predicate from yagoLiteralsFacts.tsv. Correlated to yago\_predicate from PREDICATES table.
6. **object**: varchar(250).  
   The matching data for those subject and predicate. Will be used as hint.
7. ***ENTITIES*** *(id, name) –*  
   Holds the entities that match our definitions.
8. **id**: int, NOT NULL, AUTO\_INCREMENT, PRIMARY KEY, INDEX.
9. **name**: varchar(100) NOT NULL, UNIQUE.  
   The Entity's name as it appears in Yago, concatenated by its yago\_id.
10. ***ENTITIES\_DEFINITIONS*** *(entity\_id, definition\_id) –*Connecting the ENTITIES and DEFINITIONS tables: for each entity tells its matching definitions and for each definition the entities it relates.
11. **entity\_id**: int, NOT NULL, FOREIGN KEY REFERENCES ENTITIES(id), INDEX.
12. **definition\_id**: int, NOT NULL, FOREIGN KEY REFERENCES DEFINITIONS(id), INDEX.

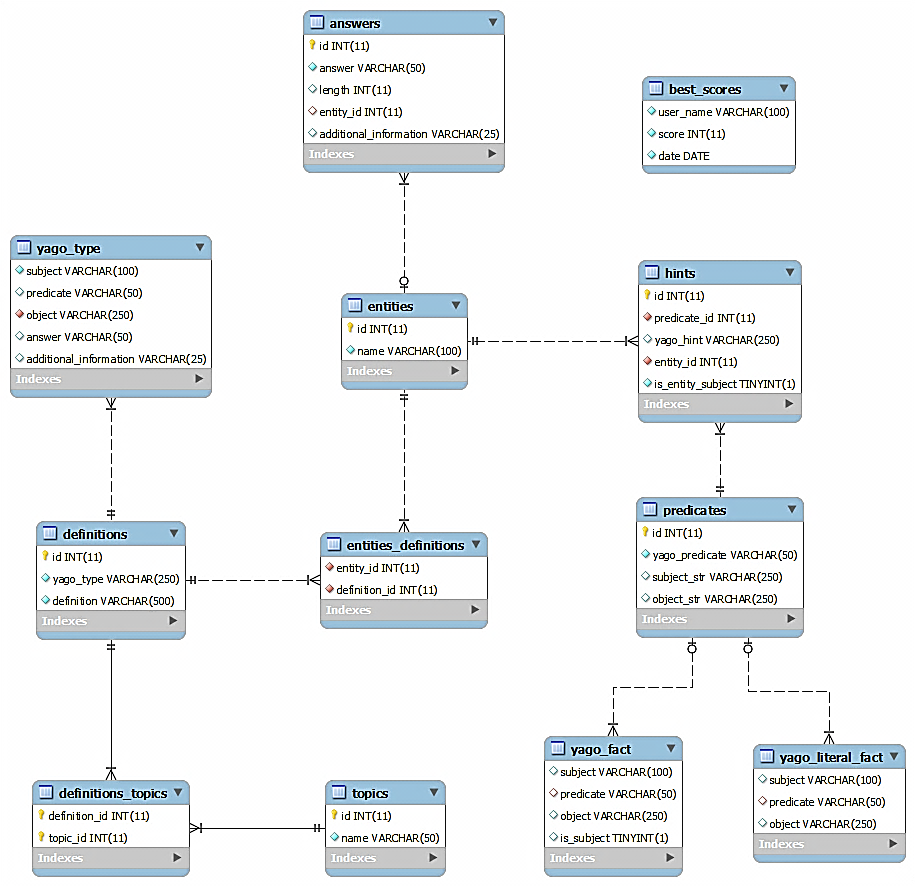
UNIQUE (entity\_id, definition\_id).

1. ***ANSWERS*** *(id, answer, length, frequency, entity\_id, additional\_information) –*Describes all possible answers in the crossword.
2. **id**: int, NOT NULL, AUTO\_INCREMENT, PRIMARY KEY.
3. **answer**: varchar(50), NOT NULL.  
   The answer as it appears on the crossword board.
4. **length**: int.  
   The answer's characters number.
5. **entity\_id**: int, FOREIGN KEY REFERENCES ENTITIES(id), INDEX.  
   Id of the entity which this answer belongs to. Refers to the id from ENTITIES table.
6. **additional\_information**: varchar(25).  
   When the answer consists of more than one word, this attribute will contain the pattern that implies the number of letters for each word. If the answer is a person's name then it will state whether it is his/her first or last name. "first name" or "last name" will appear as part of the definition on the crossword board.
7. ***HINTS*** (id, predicate\_id, yago\_hint, entity\_id, is\_entity\_subject) –

Describes the hints we collected about the different entities.

1. **id**: int, NOT NULL, AUTO\_INCREMENT, PRIMARY KEY.
2. **predicate\_id**: int, NOT NULL, FOREIGN KEY REFERENCES PREDICATES(id), INDEX.  
   Hint 'type' - the Yago predicate, taken from yagoFacts.tsv or from yagoLiteralsFacts.tsv. Correlated to yago\_predicate from PREDICATES table.
3. **yago\_hint**: varchar(250), DEFAULT '<User\_Hint>'.  
   The matching entity's name, will be parsed and replace the '?' in the hint's template string. In case the user entered this hint, this value is set to '<User\_Hint>'.
4. **entity\_id**: int NOT NULL, FOREIGN KEY REFERENCES ENTITIES(id), INDEX.  
   The id of the entity that relates to this hint.
5. **is\_entity\_subject**: boolean, NOT NULL.  
   Indicates whether this hint is about the entity above or the hint is the entity itself.
6. ***BEST\_SCORES*** *(user\_name, score, date) –*   
   Holds the information for the "HALL OF FAME" window.
7. **user\_name**: varchar(100), NOT NULL.
8. **score**: int, NOT NULL.
9. **date**: datetime, NOT NULL.

EER-diagram:



Code Structure:

The application was developed using JRE7 (thus requires java version 1.7 on the executing host) and the following external resources:

* commons-io-2.4.jar – downloading files from Yago website ([see here](http://commons.apache.org/proper/commons-io/download_io.cgi)).
* sevenzipjbinding.jar / sevenzipjbinding-AllPlatforms.jar – extracting the 7zip files from Yago website ([see here](http://sevenzipjbind.sourceforge.net/)).
* japura-1.18.2.jar – CheckComboBox widget for Management module, UI ([see here](http://sourceforge.net/projects/japura/files/Japura/Japura%20v1.18.2/)).
* glazedlists\_java15\_1.9.0.jar – Autocomplete searchbox for Management module, UI.

The main packages and general flow of the application:

1. parsing -

These two classes are used as part of the massive import procedure executed by the class *ui.MassiveImportView* and for parsing the data from the YAGO files.

*YagoFileHandler:*   
The main class that responsible for the data parsing phase.   
It provides the following functionalities:

* Downloading the required YAGO files from YAGO website and extracting them by using the *SevenZipJBindingExtractor* class.
* Retrieving from the DB the relevant definitions and predicates that need to be extracted from YAGO files.
* Parses YAGO files and writes the relevant data only into new files. Parses the YAGO's entities string and returns the corresponding answer (proper name).
* Manage the DB construction: clean the tables, import the relevant data into the DB (INSERT INTO tables *YAGO\_TYPE, YAGO\_FACT, YAGO\_LITERAL\_FACT*) , populate the core tables in DB (INSERT INTO tables *ENTITIES, ANSWERS, HINTS*).

*SevenZipJBindingExtractor:*   
Extracts the 7zip files that *YagoFileHandler* downloaded from YAGO website.

1. db -

*ConnectionPool:*   
Creates a pool of connections to the specified database using the JDBC driver, username, and password sent in the constructor. The class establishes the connection with the DB server via the createPool method. It Holds a Vector of connections. When instantiated, the pool initialized to five connections. Contains an inner class called PooledConnection, a connection with a flag indicating whether it is busy or free. It uses **synchronized** to maintain order on the connections pooling.  
The *ui.MainView* class invokes the *ConnectionPool* and keeps it till the program terminates, then the method showMessageAndClose from *ui.Utils* makes sure to close all the connections.

*DBConnection*:   
This class encapsulates the connection pool. It handles everything that has to do with getting a connection, returning it back to the connection pool when it is no longer in use, opening and closing resources (e.g. Statement, PreparedStatement, ResultSet).   
Provides all DB connectivity functionalities, from executing queries and updates through executing SQL scripts to specific updates. Uses prepared statements (in few methods the use of PreparedStatement is mostly for security considerations, when it deals with input from the user), batches and transactions wherever needed.

*DBUtils:*This is the mediator class between the core logic and UI features to the DB. It uses strongly *DBConnection*'s methods on purpose to retrieve the requested data for both the puzzle algorithm and the GUI. It implements a higher level of abstraction for communicating with the DB.   
  
*KnowledgeManagement:*  
In principle, this class is similar to *DBUtils* but for different use – connects between the GUI Knowledge Management features to the DB. It implements the data-editing abilities given to the user: deleting hints, deleting entities, adding definitions and hints.

1. core -

*PuzzleCreator:*   
The main's class.

When running the application, the user would provide through the command line the following arguments: the application working directory, password for the local DB server or "true" when working with the remote DB at Nova.   
After validating the input the main thread, *ui.MainView*, starts to run. A connection pool is created and the application is ready to go. If the JDBC driver fails to establish a connection to the DB server a message will be displayed to the user explaining that there seems to be a connectivity problem.

*AlgorithmWorker:*

Executes the crossword board creation algorithm. Returns a complete and filled board.

This Worker is run by the class *ui.PrepareGameView*with the user's choices – at least two topics that should be included in the new crossword and the game's difficulty level.

The algorithm tries to optimize the process of locating puzzle answers on the crossword board.

Since this is a NP problem, the optimization is limited, and a timer was implemented to prevent users from waiting unreasonable periods of time for the crossword to be ready.

The size of the board is set by the user, when choosing the difficulty. Each difficulty level refers to a different board size: 'easy' to 8x8, 'medium' to11x11 and 'hard' to13x13.  
The algorithm uses board templates and attempts to recursively fill a template with answers. The answers relate to the specific topics selected. For each difficulty level there are two different templates.

After 30 seconds of failed attempts, the algorithm switches to an alternative board, of the same size. After additional 30 seconds, the algorithm stops, and the GUI displays a message asking the user to select more topics. Moreover, the topic 'General Knowledge' is included in the possible answers of every crossword creation.

The classes: *Answer, BoardSolution, BoardState, PuzzleSquare* and *PuzzleDefintion* represent the data sets that maintain the board and its content during the execution of the algorithm and of the finished result board. Furthermore, the *AlgorithmWorker* thread uses their functionalities to retrieve needed data from the database.

General algorithm description:

The definitions are sorted according to the number of possible answers they hold. In each recursion phase, the algorithm chooses the definition with the lowest amount of possible answers, and tries to randomly choose an answer for it.

Each definition, has a collection of possible answers. The algorithm invariant, is that these lists remain updated at any given time. This allows the reduction of recursion levels, since selecting a certain answer for a definition, updates all affected definitions (which cross this definition's squares). If this update process causes one of the affected definitions to have no possible answers, the selected answer is removed from the possible answers list of the selected defintion. If this update process caused a definition to have only one possible answer, it will be chosen in the next recursion phase, since it will be sorted first.

The algorithm uses a stack to implement the recursion, which holds board states. Each board state, holds a board, all solved definitions, and the state of unsolved definitions. The board state also holds a reference to the last solved definition before the state was pushed to the stack. When the stack is popped, the last answer that was selected is removed from its definition's collection of possible answers, and that definition is marked as unsolved.

If the stack is empty, and no possible answers exist, the algorithm returns false. If no unsolved definitions exists, the algorithm returns true.

There is another optimizing process, which uses the board squares. This process is executed in the method optimizeBoard() in algorithmWorker. The method scans the board squares, and for each square, checks which letters can be assigned to it. The allowed letters of each squares are those that leave at least one possible answer for each definition that uses the square. After removing the problematic letters, the relevant definitions are updated to hold only those answers which include one of the square's possible letters in the correct index. This is not a recursive process, and it is executed once every recursive iteration.

1. ui -  
     
   Includes all the GUI code, consists of many classes. Following is a general description of the main different classes, their functionalities and the application's flow through them.  
   *MainView:* The main window of the application, displays all others application's windows. Catch exceptions thrown from low-level methods and handle them properly.

At any time, each one of the following windows can be opened as a response to the user's action:

* *PrepareGameView:* Starts a new game – displays a window with all the topics and fetch the user's selections. In addition, it lets the user choose the difficulty of the game: easy, medium or hard. Runs AlgorithmWorker (new thread, in the background) that calculates a new crossword board according to the selected topics. In the meantime, runs WaitView which displays a Trivia question for the user to warm up until the algorithm finish constructing the board.
* *CrosswordView:* Runs when the calculation is over and the board is ready. This display holds all the visualization and logic of the crossword. In the background runs TimerThread which manages the timer that appears on the top of the puzzle window.
* *HallOfFameView:* Displays the high scores stored in the BEST\_SCORES table.
* *ManagementView:* Displays the 'Knowledge Management' window. Allows the user to add new definitions, add / edit / delete hints and edit entities.
* *MassiveImportView*: The window from which the Massive Import is performed. It starts a new thread that runs ImportWorker which executes the import.   
  The user can choose whether to supply the application a directory containing the relevant YAGO files (yagoTypes.tsv, yagoFacts.tsv, yagoLiteralFacts.tsv) or to download the files from YAGO website. In both cases the parsing procedure and the code constructing the DB will be executed. All the user's changes will be erased.
* *HelpView*: Displays main help guiding issues.
* *AboutView*: All the important stuff there is to know about the developers staff.
* *Window*: ENUM – holds all the names of the different types of the application's windows.

The following classes are GUI utilities:

* *HintPopupMenu*: Describes a hints list for a specific answer on the board.
* *AbstractSquarePanel*: An abstract class represents a crossword's square.
* *OneDefinitionSquare*: Extends *AbstractSquarePanel*. Represents a crossword's square that belongs to one definition only.
* *TwoDefinitionSquare*: Extends *AbstractSquarePanel*. Represents a crossword's square that belongs to two definitions, a crossing square.
* InputSquare: Extends *AbstractSquarePanel*. Represents a crossword's square containing a letter.

**finally** {

The project before you represents hours of hard work (more like 'days'), many efforts and toughLifeExceptions, but also a lot of funny moments and favorites DB examples such as 'Aviv Geffen', 'Matti Caspi' and 'Noga Alon' **;->**

We truly hope that you find the application funny and enjoyable at least as we do.

Sincerely,

Crossword Mastermind team (a.k.a DbMysql02),

Yonatan, Saleet, Guy and David.

}