October 4, 2013

שנקר המכללה לעיצוב והנדסה –מחלקת הנדסת תוכנה.

פומייז-פאזל הזזה

ספר פרוייקט הגמר

מאת : אייל וולנובסק,יוני זילברמן מנחה:חנן גזית

04/10/2013

פומייז הוא אפליקציית פאזל הזזה המתאים לכל הגילאים וקשת היכולות. המשחק משלב בתוכו מנגנוני גיימיפיקיישין ומנוע לפתרון הפאזל המבוסס על אלגוריתם מתחום האינטיליגנציה המלאכותית כל זאת בכדי להעניק למשתמש חוויה מהנה ומאתגרת

Pumiez-a sliding puzzle

Software Engineering B.Sc. Final Project Write-up

Authors: Eyal, volanovski and Yoni, zilberman | Project Advisor: Dr. Hanan, Gazit

04/10/2013

Pumiez is a sliding puzzle game application suitable for all ages and skills. This application contains gamification game mechanics and game dynamics to enhance the user experience. In addition the application implements a puzzle game solver that is based on an algorithm from the field of artificial intelligence.

Table of Contents

2 Introduction 7
2.1 Overview 7
2.1.1 Project Goal 7
2.1.2 The Problem 7
2.1.3 The Solution 7
2.1.4 Future Development 7
2.1.5 Audience 7
2.1.6 Terminology 8
3 Architecture 15
3.1 Description 15
3.2 Incentive 15
3.3 System Users and Roles 15
3.3.1 User Types 15
3.4 Application Weaknesses 16
3.4.1 Code Errors 16
3.4.2 Security vulnerabilities 16
3.4.3 User Interface 16
3.4.4 Third-Party dependencies 16
3.4.5 Cross-Platform Portability 16
4 Design 17
4.1 System Structure 17
4.2 System Design 18
4.2.1 Java calculation engine 18
4.2.2 PHP Java Bridge 22
4.2.3 The GUI 22
4.2.4 MySQL Database 25
4.3 Data Flow diagram of the GUI 26
4.4 A complete class diagram of the Java calculation engine 27
5 Development 28
5.1 Paradigm 28
5.2 Programming Language 28
5.2.1 Using Java language 28
5.2.2 Using MySQL 28

5.2.3 Using HTML/CSS/JavaScript/PHP 28
5.3 Tools 28
5.3.1 IDE 28
5.3.2 Testing 28
6 User Guide 29
6.1 Registration 29
6.2 Choosing a picture 30
6.3 Playing the game 32
6.4 Points and badges 33
7 Summary 34
7.1 Main Focal Points 34
7.1.1 Creating a cross platform puzzle game 34
7.1.2 Using gamification 34
7.1.3 Implementing a working 8-puzzle and 15- puzzle solver 34
7.2 Conclusions 34
7.2.1 Our Key for Success is a dynamic goal 34
7.2.2 Tools can help us 34
7.3 Future Work 35
7.3.1 More gamification features 35
7.3.2 Support a solution for a puzzles greater then 4X4 35
7.3.3 Better designed GUI 35
7.3.4 Improving the Database 35
8 Literature 36
8.1 Research 36
8.1.1 Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA* by Alexander Reinefeld, Paderborn Center for Parallel Computing 36
8.1.2 gamification.org 36
8.1.3 aaai.org 36
8.1.4 Wikipedia, the free encyclopedia 36
8.1.5 Psychologytoday.com 36
8.2 Technical References 36
8.2.1 Shenkar College of Engineering and Design 36
8.2.2 Stackoverflow.com 36
8.2.3 W3Schools 36

8.2.4 PHP / Java Bridge 36
8.2.5 Oracle 36
8.2.6 MySQL 36
8.2.7 phpMyAdmin 36
9 נספח:תקציר מנהלים 37

1 Abstract.

Pumiez is not a new sliding puzzle game application it is based in its core on the classic sliding puzzle that is familiar to all. In this implementation we hope that the game will quickly lead the player into hopeless addiction and by so breaking the common stigma that sliding puzzles are boring. We drew our inspiration from the gamification mechanisms. The gamification mechanisms in many cases have succeeded in keeping the player interested in a fixed content application.

The fundamental goal of any sliding puzzle is to challenge the player to slide pieces along certain routes to establish a certain end-configuration. By using the gamification mechanisms we hope to create additional challenges. On the one hand the challenges from the gamification world like accumulation of points, time scores and motivating messaging between players will keep the player motivation high. on the other hand the unique application ability, to give the player a hint of how to solve the puzzle whatever is the current configuration of puzzle is, will insure the player won't sink into despair.

In this game application we offer the player the challenge of solving 8-puzzle and 15-puzzle games with a picture to his choice. Solving these puzzles sounds simple and innocent but in fact for over two decades those puzzles have been used in computer laboratories for performance testing of search methods. Amazingly enough there is no known algorithm that finds a shortest solution for these problems efficiently. Solving an NxN puzzle problem is considered NP-Hard. Based on performance testing that have been done by others in the past on different kinds of solution algorithms we found the IDA* algorithm is the most suitable for our purposes.

Pumiez has an HTML/PHP/CSS/JavaScript GUI so it can be played in almost any platform that has a web browser installed and contain a relaxing background music that will set the right mood for a thinking game.

2 Introduction

2.1 Overview

2.1.1 Project Goal

The project's goal is to be a fun and challenging puzzle game application for all ages and personal skills. The application will allow users to solve 4X4 and 5X5 sliding puzzles and to be motivated by the gamification mechanics.
Novice users can choose less complicated photos to be scrambled into a puzzle that contains fewer parts. Experienced users will be able to choose more complicated pictures and puzzles that contained more parts.

2.1.2 The Problem

People always seek a fun and challenging experience to do in their spare time. While puzzle game application provide a short term solution to this problem in the long term users tend to get bored from it and stop using it.

2.1.3 The Solution

In order to create a sliding puzzle game application that maintains the fun and challenging experience of the user we will implement some of the gamification mechanics that addresses to the different types of users.
The Killer user type will be motivated by the game score tables.
The Achiever will be motivated by game points and badges.
The Socializer will be motivated by the between user messaging and the ability to upload and share photos between the users.
The Explorer will be motivated by new photos that being uploaded by other users by thus discovering new challenges.
The ability to play in a cross platform environment
finally the option to get a hint for solving any puzzle created by the application will keep all users types away from being frustrated from the game.

2.1.4 Future Development

Our game application, as currently implemented, contains only part of the gamification mechanics thus has the potential to give even more fun and challenging experience to the user.

In addition, the puzzle solving component only support 4x4 and 5x5 sliding puzzle solution and can be expended to suite any puzzle size (NxN).

Finally, the GUI is not final. Although the GUI is written in HTML, PHP, JavaScript
it is not yet a true cross platform application and further adjustment need to be done.
The design of GUI can be made more appealing to the eyes.

2.1.5 Audience

The sliding game application is designed to be suited for all ages and types of users.
Every user that wishes to have a fun and challenging experience will find it suitable for him.

2.1.6 Terminology

2.1.6.1 Sliding puzzle

A **sliding puzzle** is a puzzle that challenges a player to slide a piece of the puzzle along certain routes in a contained environment to establish a certain end-configuration.
The puzzle starts when all pieces are in the contained environment and there is exactly one place in the environment that does not contain a piece. Any piece that is next to the empty space can be moved to it and by thus creating an empty space in the previous location of the piece. Unlike other puzzles, a sliding puzzle prohibits temporary removing any piece from the environment and creating more than one empty space.

2.1.6.2 NxN puzzle (N-puzzles)

NxN puzzle is a sliding puzzle that contain N rows and N columns

2.1.6.3 Cross platform application

In computing, **cross-platform application** is an attribute conferred to computer software or computing methods and concepts that are implemented and inter-operate on multiple computer platforms.

2.1.6.4 Jslint

Jslint is a JavaScript Code Quality Tool.

2.1.6.5 GUI

In computing, **graphical user interface** (**GUI**) is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation

2.1.6.6 Gamification

Gamification is the concept of applying game-design thinking to non-game applications to make them more fun and engaging.

2.1.6.7 User types

In the gamification concept field **user types** are the psychological gaming profile of the users. It divides the users to 4 groups: Killer, Achiever, Socializer and Explorer.

- **Killers**: interfere with the functioning of the game world or the play experience of other players
- **Achievers**: accumulate status tokens by beating the rules-based challenges of the game world
- **Explorers**: discover the systems governing the operation of the game world
- **Socializers**: form relationships with other players by telling stories within the game world

2.1.6.8 BFS

breadth-first search (BFS) is a strategy for searching in a graph when search is limited to essentially two operations: (a) visit and inspect a node of a graph; (b) gain access to visit the nodes that neighbor the currently visited node. The BFS begins at a root node and inspects all the neighboring nodes. Then for each of those neighbor nodes in turn, it inspects their neighbor nodes which were unvisited, and so on. Compare BFS with the equivalent, but more memory-efficient Iterative deepening depth-first search and contrast with depth-first search.

Input: A graph *G* and a root *v* of G

```
procedure BFS(G,v):
    create a queue Q
    create a set V
    enqueue v onto Q
    add v to V
    while Q is not empty:
        t ← Q.dequeue()
        if t is what we are looking for:
            return t
        for all edges e in G.adjacentEdges(t) do
            u ← G.adjacentVertex(t,e)
            if u is not in V:
                add u to V
                enqueue u onto Q
    return none
```

2.1.6.9 DFS

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Input: A graph G and a vertex v of G
Output: A labeling of the edges in the connected component of v as discovery edges and back edges

```
procedure DFS(G,v):
    label v as discovered
    for all edges e in G.adjacentEdges(v) do
        if edge e is unexplored then
            w ← G.adjacentVertex(v,e)
            if vertex w is unexplored then
                label e as a discovered edge
                recursively call DFS(G,w)
            else
                label e as a back edge
     label v as explored
```

2.1.6.10 NP-hard

NP-hard (Non-deterministic Polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP". A problem H is NP-hard if and only if there is an NP-complete problem L that is polynomial time Turing-reducible to H (i.e., L ≤ TH). In other words, L can be solved in polynomial time by an oracle machine with an oracle for H. Informally, we can think of an algorithm that can call such an oracle machine as a subroutine for solving H, and solves L in polynomial time, if the subroutine call takes only one step to compute. NP-hard problems may be of any type: decision problems, search problems, or optimization problems.

2.1.6.11 A*

In computer science A* is a computer algorithm that is widely used in path finding and graph traversal, the process of plotting an efficiently traversable path between points, called nodes. Noted for its performance and accuracy, it enjoys widespread use.

Like all informed search algorithms, it first searches the routes that appear to be most likely to lead towards the goal. What sets A* apart from a greedy best-first search is that it also takes the distance already traveled into account; the g(x) part of the heuristic is the cost from the starting point, not simply the local cost from the previously expanded node.

Starting with the initial node, it maintains a priority queue of nodes to be traversed, known as the open set. The lower f(x) for a given node x, the higher its priority. At each step of the algorithm, the node with the lowest f(x) value is removed from the queue, the f and g values of its neighbors are updated accordingly, and these neighbors are added to the queue. The algorithm continues until a goal node has a lower f value than any node in the queue (or until the queue is empty). (Goal nodes may be passed over multiple times if there remain other nodes with lower f values, as they may lead to a shorter path to a goal.) The f value of the goal is then the length of the shortest path, since h at the goal is zero in an admissible heuristic.

The algorithm described so far gives us only the length of the shortest path. To find the actual sequence of steps, the algorithm can be easily revised so that each node on the path keeps track of its predecessor. After this algorithm is run, the ending node will point to its predecessor, and so on, until some node's predecessor is the start node.

Additionally, if the heuristic is monotonic (or consistent, see below), a closed set of nodes already traversed may be used to make the search more efficient.

The following pseudo code describes the algorithm:

```
function reconstruct_path(came_from, current_node)
    if current_node in came_from
        p := reconstruct_path(came_from,
came_from[current_node])
        return (p + current_node)
    else
        return current_node
```

```
function A*(start,goal)
    closedset := the empty set  // The set of nodes already evaluated.
    openset := {start}    // The set of tentative nodes to be evaluated,
initially containing the start node
    came_from := the empty map    // The map of navigated nodes.

    g_score[start] := 0    // Cost from start along best known path.
    // Estimated total cost from start to goal through y.
    f_score[start] := g_score[start] + heuristic_cost_estimate(start,
goal)

    while openset is not empty
        current := the node in openset having the lowest f_score[] value
        if current = goal
            return reconstruct_path(came_from, goal)

        remove current from openset
        add current to closedset
        for each neighbor in neighbor_nodes(current)
            tentative_g_score := g_score[current] +
dist_between(current,neighbor)
            tentative_f_score := tentative_g_score +
heuristic_cost_estimate(neighbor, goal)
            if neighbor in closedset and tentative_f_score >=
f_score[neighbor]
                    continue

            if neighbor not in openset or tentative_f_score <
f_score[neighbor]
                came_from[neighbor] := current
                g_score[neighbor] := tentative_g_score
                f_score[neighbor] := tentative_f_score
                if neighbor not in openset
                    add neighbor to openset

    return failure
```

2.1.6.12 IDA *

A search algorithm that is a combination of the A* algorithm and the DFS algorithm.it was Invented by Korf in 1985.
The idea is that successive iterations correspond not to increasing depth of search, but rather to increasing values of the total cost of a path.

Algorithm: At each iteration, perform a depth-first search, cutting off a branch when its total cost (g + h) exceeds a given threshold.
This threshold starts at the estimate of the cost of the initial state, and increases for each iteration of the algorithm. At each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold.

The following pseudo code describes the algorithm:

```
Threshold =Eval(s) ;
done =false;
while(not done) {
   done=IDA8(s,0,threshold);
    if (done ==false) threshold ++;
]
```

```
IDA* (state s, int g,threshold t) {
 h=Eval(s);
 If (h==0) return (true);
 f=g+h;
 if (f>threshold) return (false);
 for (i=1; i<=numchildren; i++){
   done= IDA*(s.child[i],g+cost(child[i],t);
   if (done ==true) return (true);
}
Return (false);
}
```

IDA* properties:

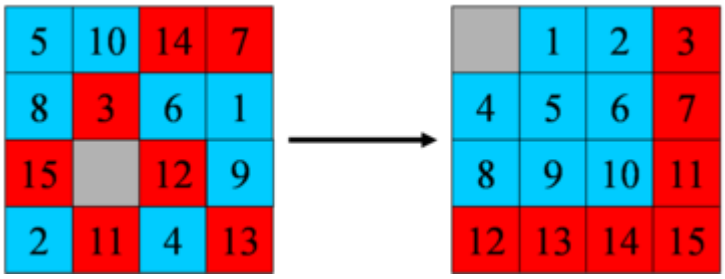
- Iterative deepening A*
- The cost of a node is (using A* terms)
- f = g + h
- g = cost incurred to get to this node
- h = heuristic estimate of getting to goal
- Iterative deepening iterates on a threshold
- Search a node as long as f <= threshold
- Either find a solution (done), or fail, in which case
- the threshold is increased and a new search started

2.1.6.13 Pattern Database

A Pattern Database stores a collection of solutions to sub-problems that must be achieved to solve the problem.
While we normally think of a heuristic as a function computed by an algorithm, any function can also be computed by a table lookup, given sufficient memory. In fact, for reasons of efficiency, heuristic functions are commonly precomputed and stored in memory.

Example: The *Fringe* database uses the Fringe target pattern (picture a). For a given arbitrary 15-Puzzle state (picture b), we map the current locations of tiles 3, 7, 11, 12, 13, 14, 15 and the empty tile into an index into the fringe database. The database will tell us the minimum number of moves required to correctly locate these 8 tiles. This is a lower bound on the remaining search effort. Once the fringe pattern has been achieved, all that is left to solve is an 8-Puzzle. Given more memory we can compute additional pattern databases from the remaining tiles.

(b) ---------- (a)



Picture 1

31 moves needed to solve red tiles. 22 moves needed to solve blue tiles
Overall heuristic is maximum of 31 moves.
Note that the red pattern and the blue pattern are **disjoint patterns**, i.e. have no tiles in common. The maximum of the heuristic values of the disjoint patterns remains admissible and close to the actual optimal cost.

2.1.6.14 Disjoint pattern Databases

Disjoint pattern databases are a special case of multiple pattern databases in which the set of pattern variables for each pattern database is disjoint from the pattern variables of all other pattern databases. In disjoint pattern databases, one must choose the set of duplicate-detection variables carefully so that it overlaps with the set of pattern variables for each external- memory pattern database. The reason for this is that the abstract pattern variables used in a pattern-space projection function must be duplicate-detection variables as well. If the two sets of variables do not overlap, then the pattern-space projection function trivially maps all patterns to a single abstract pattern, the p block for this abstract pat- tern is the entire pattern database, and there is no way to use external memory to store part of the pattern database.

On 15-puzzle, IDA* with a heuristic based on the disjoint pattern databases can optimally solve random 15 puzzle instances in less than 29 milliseconds on average. This is about 1700 times faster than with Manhattan Distance on the same machine.

3 Architecture

3.1 Description

The application is a puzzle game that allows users to solve 4X4 and 5X5 sliding puzzle suitable for a wide range of user types. The game includes gamification mechanics such as wining point, score table, custom photos uploaded by a user, badges and user messaging that is only available to registered users. Unregisters users can only solve puzzles and have no accesses to any other feature of the game.
The application contains a database for storing all user related information and a standalone component that computes the puzzle solution and connected to the application.

3.2 Incentive

The puzzle games was invented hundreds of years ago and the fact that people still solving them is a strong case in favor of their timeless relevant.
It is enough to look of the dazzling and rapid successes of a game like "candy crush" to see the users need for something to do in their spare time.
In the current days puzzle games are often preserved as boarding and thus there is the need to bring this types of games into the modern world in a proper way.

3.3 System Users and Roles

The system has three theoretical users: registered players, unregistered players and an administrator.

3.3.1 User Types

3.3.1.1 Unregistered Player

unregistered players can solve all types of puzzles available in the game but they cannot use any other features of the application. They will not be able to upload new photos, to participate in the score table and to message registered users.

3.3.1.2 Registered Player

The main user of the application is the registered player. Any person that has accesses to the Internet can use the sign up process in the application and become a registered player. As such the player can use all the application features.

3.3.1.3 Administrator

The application does not contain an actual administrator user it has an administrator entity. Since the application contains a Database and a calculation engine that both involves a server configuration and by its nature a server user name and password, he who holds all the servers user names and password is an administrator entity .
The user names and password can be hold by one or more persons and currently are in the possession of the application developing team.
The administrator job is to maintain the databases and to apply, if necessary, updates to the servers.

3.4 Application Weaknesses

The system, like any other, has some weaknesses. Some of them due to the way the system was designed and built, and some are inherent from the software domain.

3.4.1 Code Errors

Like any other software application the Pumiez application might contain errors in the code. We as developers tried to do testing and use tools like "jslint" and others to find and correct them.

3.4.2 Security vulnerabilities

Security was not a main flag in our development. There are some known ways that a user can hack and cheat in our application if he will put some effort to it. We chose to focus on other subjects and will leave the implementation of a more secure release to the future development of the application.

3.4.3 User Interface

At the current time, the user interface supports all of the system's functionality that has been implemented, but it is still lacking a high quality visual design.

3.4.4 Third-Party dependencies

Parts of the system were implemented on remote servers that us as developers does not fully control or own. The application relay on configuration and storage on those serves and are specific to those servers' capabilities. Any failure of those servers will damage the proper work of the application.

3.4.5 Cross-Platform Portability

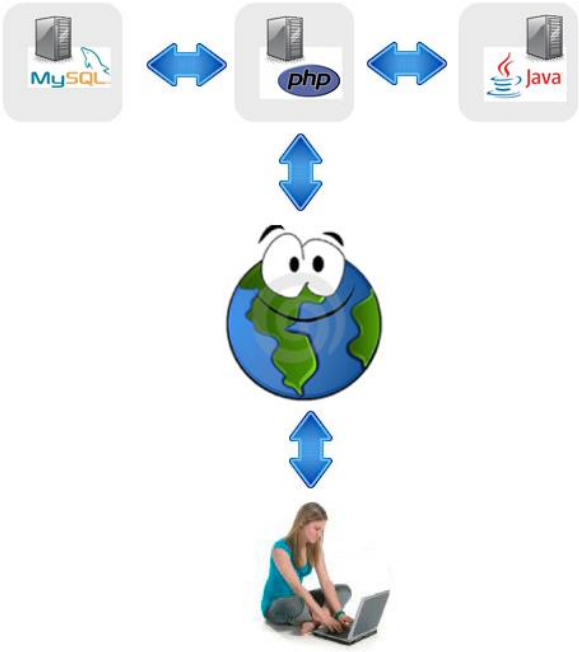
Designed for broad use the application cannot be bound to a specific platform. The application GUI was designed and implemented using PHP/HTML5/ JavaScript languages, which are known for its good cross-platform capabilities when implemented correctly. Although using those languages allow us theoretically to implement a true cross platform application it requires a lot of adaptions in the code and we were not able to address all platform in our development time frame.

4 Design

4.1 System Structure

As illustrated in picture 2 below the application is built from three main components that is stored on separated servers:
1. MySQL Database
2.PHP/HTML/JavaScript GUI
3.Java calculation engine and PHP/Java Bridge

The user access the application GUI through the Internet with any device.
The GUI written in PHP/HTML/JavaScript is connected to MySQL Database natively and to a Java calculation engine with the aid of PHP/Java Bridge which allow remote executing of Java functions from the PHP language.



Picture 2

4.2 System Design

This is a more detailed technical description of the application design. A high level overview can be found in the application structure section.

4.2.1 Java calculation engine

The purpose of the calculation engine is to revise as input a given 3x3 or a 4x4 puzzle parts configuration and to give as output a set of legal moves for the player to do to reach the wanted configuration of the (sorting the picture parts so that the picture will be as before it was scrabbled).

4.2.1.1 PuzzleSolver.java

In our development we have experienced with several algorithms and database types. In picture 4 below you can see the algorithms that we have consider in our development. Among them there is the IDA* algorithm that we saw as best fit to our cause.

Algorithm	Complete	Optimal	Time	Space
Greedy BFS	YES if finite	NO	exp	exp
A*	YES	YES if admis.	exp	exp
IDA*	YES	YES if admis.	exp	O(d) O(1)*
DF B&B	YES if finite	YES	exp	O(d)
Global Beam	NO	NO	exp	O(kd)
S.A. Hillclimb	NO	NO	O(bd)	O(d) O(1)+
Sim. Annealing	NO	NO	O(bd)	O(d) O(1)+
Local Beam	NO	NO	O(kbd)	O(k)
Genetic	NO	NO	O(k x numiter)	O(k)

* IDA uses O(d) for DFS, but only keeps f-limit between iterations.
\+ O(1) if we don't save the path to the solution.

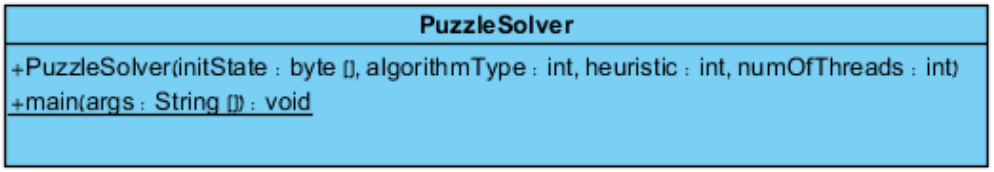
Picture 4

The declaration of the class can be seen in the following code :

```
public final class PuzzleSolver {

    public PuzzleSolver(final byte[] initState, final int algorithmType,
                        final int heuristic, final int numOfThreads) {
```

The main solver class that is described in picture 5 below is designed to support different type of puzzles sizes, algorithms types and database types those the first three input argument are so.



Picture 5

The input (15, I, p…) says that we wish to solve a 4x4 sliding puzzle using IDA* search algorithm in a disjoint pattern database. It is also possible to enter the following input - (8, I, p…) that means we wish to solve a 3x3 sliding puzzle. Following those arguments as input a series of numbers is needed to be inserted. It represents the location of the puzzle parts in the puzzle matrix.
The output of a legal input will be brief and basic information on the puzzle solving process and a list of legal moves to solve the puzzle. Every "move" contains a number that represent a piece in the puzzle and the direction that it needs to move in order to advance to a complete solution of the puzzle.

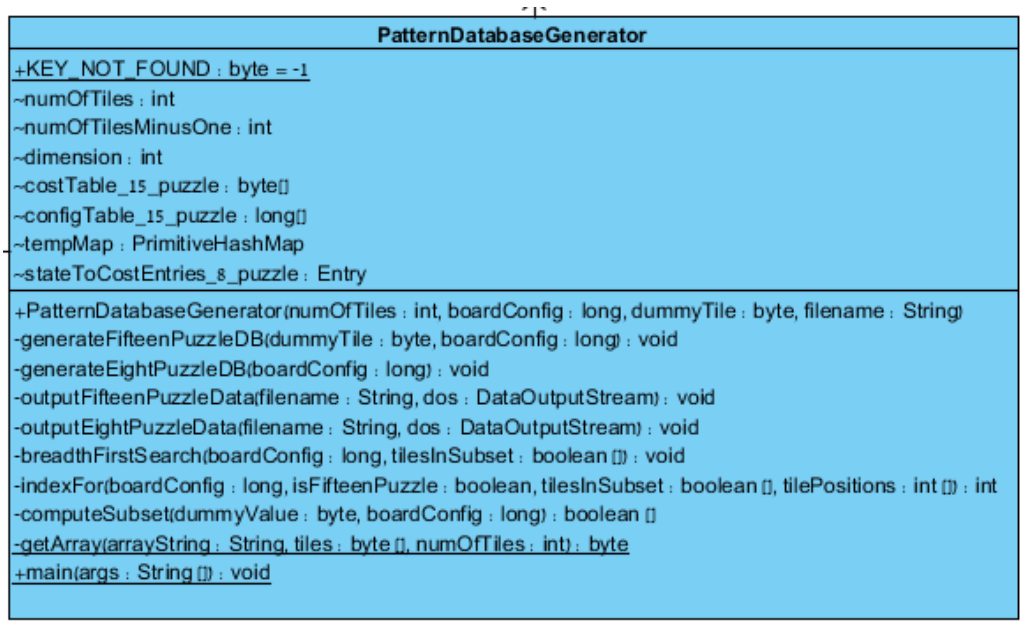
For example , the following **input**: (15,I,p,0,4,2,3,13,8,7,6,5,10,11,1,9,12,15,14)
will result in the **output** that can be seen in picture 6 below.

```
Puzzle type:          15-puzzle
Initial permutation:  0,4,2,3,13,8,7,6,5,10,11,1,9,12,15,14
Goal state:           1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0
Elapsed time:         0.422 s
Paths visited:        682,719
States explored:      1,324,100
Shortest path:        52 moves
 1.  4 - left
 2.  8 - up
 3. 13 - right
 4.  5 - up
 5. 10 - left
 6. 13 - down
 7.  7 - left
 8. 11 - up
 9. 15 - up
10. 12 - right
11. 13 - down
12.  7 - down
13. 11 - left
14.  6 - left
```

Picture 6

4.2.1.2 PatternDatabaseGenerator.java

The IDA* algorithm searches a solution for a given puzzle state in a database that contains all possible paths. The class "PatternDatabaseGenerator" that is described in picture 5 below is responsible of generating the disjoint pattern database. After the database will be generated it will be saved in different files. The files contain a standalone databases corresponding to the supported sliding puzzle sizes (3x3 , 4x4). This process is automatically preformed when calling the "puzzleSolver" class and it will be done only once. Once generated The IDA* automatically preform a search on those files for finding a solution to a given puzzle state.



Picture 7

4.2.1.3 Algorithm.java

The class described to the left in picture 8 is responsible of containing the running of the IDA* algorithm and saving the run time information of the algorithm such as the time that took the algorithm to find a solution to a given sliding puzzle state.

Picture 8

4.2.1.4 PuzzleConfiguration.java

The "PuzzleConfiguration" class that is described in picture 9 below is responsible of identifying and verifying all the needed variables before the e IDA* algorithm starts its search in the disjoin pattern database that corresponds to the puzzle size.

```
PuzzleConfiguration
+PUZZLE_8 : int = 0
+PUZZLE_15 : int = 1
+ALGORITHM_ASTAR : int = 0
+ALGORITHM_IDASTAR : int = 1
+HEURISTIC_PD : int = 0x1
+HEURISTIC_MD : int = 0x2
+HEURISTIC_LC : int = 0x4
+costTable_15_puzzle_0 : byte[] = new byte[4096]
+costTable_15_puzzle_1 : byte[] = new byte[16777216]
+costTable_15_puzzle_2 : byte[] = new byte[16777216]
<<Property>> -numOfTiles : int
<<Property>> -dimension : int
-algorithmType : int
-heuristicType : int
<<Property>> -numOfThreads : int
<<Property>> -goalState : long
<<Property>> -goalStatePositions : long
-isVerbose : boolean = false
-PuzzleConfiguration()
+initialize(puzzleType : int, algorithmType : int, heuristicType : int, numOfThreads : int) : void
+setVerbose(isVerbose : boolean) : void
+isVerbose() : boolean
+getHeuristic() : int
+stringRepresentation() : String
+initializeGoalState(numOfTiles : int) : void
-loadStreamPatternDatabase(filename : String, patternDatabase : PrimitiveHashMap) : void
-loadStreamCostTable(filename : String, costTable : byte[]) : void
```

Picture 9

4.2.1.5 IDASTAR.java

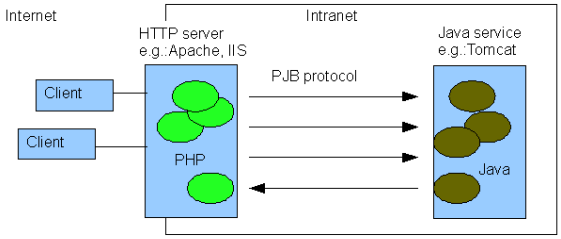
The "IDAStar" class described in picture 10 below is the full implementation of the IDA* algorithm.

```
IDAStar
-queue : BFSNode
~solvePuzzle(currentState : long, numOfThreads : int) : void
-solveMultiThreaded(currentState : long, numOfThreads : int) : void
-solveSingleThreaded(currentState : long) : void
-completeBFS(node : BFSNode) : void
-findStartingPositions(currentState : long, howMany : int) : void
+cleanup() : void
```

Picture 10

4.2.2 PHP Java Bridge

The PHP/Java Bridge is an implementation of a streaming, XML-based network protocol, which can be used to connect a native script engine in our case PHP with a Java virtual machine. It faster and requires fewer resources on the web-server side then other solutions. The scheme of its operation can be seen below at picture 11.



Picture 11

We use the bridge to invoke the PuzzleSolver function which has been discussed in section 3.2.1.1 from the PHP server.

4.2.3 The GUI

A total explanation of the GUI can be seen at the user guide at section 5.
In this section we will review some of GUI components in a deeper manner.

4.2.3.1 Connecting to a remote servers throw PHP

As said before The GUI communicates with other remote serves.
As shown in picture 12 below this is how we connected a PHP GUI component with MySQL Server:

```
1  <?php
2
3  if(!empty($_GET)){
4      $username = ''.$_GET['uname'];//
5      $str = "";
6
7      $link = mysql_connect('localhost','root','') or die('Could not connect to MySQL: ' . mysql_error());
8      mysql_select_db('gamification');
9      $scores = mysql_query("SELECT besttime FROM users WHERE uname='$username'");
10     $score = mysql_fetch_row($scores);
11
12     $query = "
```

Picture 12

And as shown in picture 13 below this is how we invoke from a PHP GUI component a remote Java procedure:

```
require_once("http://localhost:8080/JavaBridge/java/Java.inc");
$yourObj = Javav("your java class");
$yourObj->yourMethod();
$yourObj->setProperty("xxx");
```

Picture 13

4.2.3.2 The clock

For the game application needs we implemented a custom clock which counts the time it takes for the user to solve the puzzle. The time and time based points are calculated and are saved in the Database for the use of timetables and total score points.
The clock starts counting from zero and continues counting upwards until the puzzle is finished. The clock contains the following functions that can be found at the clock.js file:

- Function stopwatch() - counting the seconds, minutes and hours
- Function resetIt() – reset the clock
- Function stopIt () – stop the clock

4.2.3.3 Sound

The GUI includes a background soft music. We have used the native HTML5 capabilities to play/stop music files. For a cross browser experience more than one type of music file is available for the same melody.

4.2.3.4 The dynamic gallery

For our needs we have implemented a photo gallery from scratch since we didn't found a free and suitable solution. Our implementation of the photo gallery in JavaScript allows us to control some important parameters of the gallery. The parameters can be seen in picture 14 below

```
// Begin User Configurable Variables:

var imgsPerPg = 9; // number of img elements in the html
var imgsMax = 250;  // total number of images
var slideTimeout = 7; // seconds before loading the next slide

var gPath = 'images/gallery_images/';  // gallery files (thumbnails) path, include trailing slash
var gPrefix = 'tnshell';
var gSuffix = '';
var gExt = '.jpg';
var gZeros = true; // filename uses leading zeros?
var gDigits = 4    // total digits in filename, including leading zeros

var sPath = 'images/gallery_images/'; // slideshow files (big imgs) path, include trailing slash
var sPrefix = 'shell';
var sSuffix = '';
var sExt = '.jpg';
var sZeros = true; // filename uses leading zeros?
var sDigits = 4    // total digits in filename, including leading zeros

var captions = new Array();
// There must be (imgsMax + 1) captions.
// captions[0] is currently not used.
```

Picture 14

The final result of the gallery can be seen in picture 15 below: (the arrows are used for the purpose of scrolling the gallery)

Picture 15

4.2.3.5 The Puzzle scrambler

The part that is responsible for taking a complete picture and creating a sliding puzzle from it is found in the "puzzle.js" file. This script can cut any size of picture into a grid of pieces that is stored in the variable "intUsersize" (can be seen in picture 16 below). The value that the variable holds is the size of one piece of the puzzle in pixels unit.

The script is also responsible of scrambling the puzzle parts into a solvable puzzle problem. The scrambling of the puzzle pieces is done by emulating a legal player move. A collection of legal moves inflicted on a puzzle necessarily creates a solvable puzzle since repeating those steps backward will bring the puzzle to its original state. The number of emulated legal "moves" that will be inflicted on the original picture is saved in the variable "compleability".

```
var defaultUser="";
var intUserSize=120;
var complexability=80;
var START="Start"; //clock start
var STOP="Stop";   //clock stop
var notWin=true;   //clock stop
var POITTS_FOR_SEC=16;
var MAX_SCORE=10000;
var MIN_SCORE=400;
var COMPLEMENT_JUMP=2000;
var complement = { 0: "Intelligent", 1: "Smart", 2: "Genius",3: "Supreme", 4: "Divine", 5: "God"};
```

Picture 16

The script contains the following functions:

- function PieceClickHandler()- randomly emulate one legal "move" in the game
- function checkIfPuzzleCompleted() –check if the puzzle is in the original picture state
- gameScore() – calculate the number of points that the user gain based on the game timer.

4.2.4 MySQL Database

The database roll is to store user achievements and login information.

4.2.4.1 Users table

The table shown in picture 17 bellow stores all user information and achievements.
We as developers are well aware that passwords should not be stored in the database and a hashing verification mechanism need to be used instead.
Farther more the achievements the user gained from the games that are stored as badges need to be separated into a different table. Since simplicity in our point of view was our guiding light we chose to leave this table in form shown below.

users

	ID	password	uname	fname	lname	besttime	wongame	level	token	badges
Edit Copy Delete	1	123456	Cool Yoni	Yoni	Zilberman	14	53	1	19744	0,1,2,3,4,7,11

Picture 17

4.2.4.2 Achievements table

The table shown in picture 18 bellow stores all game achievements descriptions.

achievements

	ID	description
Edit Copy Delete	1	Won 1 Game

Picture 18

4.2.4.3 Messages table

The table shown in picture 19 bellow stores all the fixed messages that users can send between each other. We decided to use fix messages to insure that the content of the messages won't contain inappropriate content.

messeges

	Id	Messege
Edit Copy Delete	1	Hello mate, Can you do better than me ?

Picture 19

4.2.4.4 Msgtrafic table

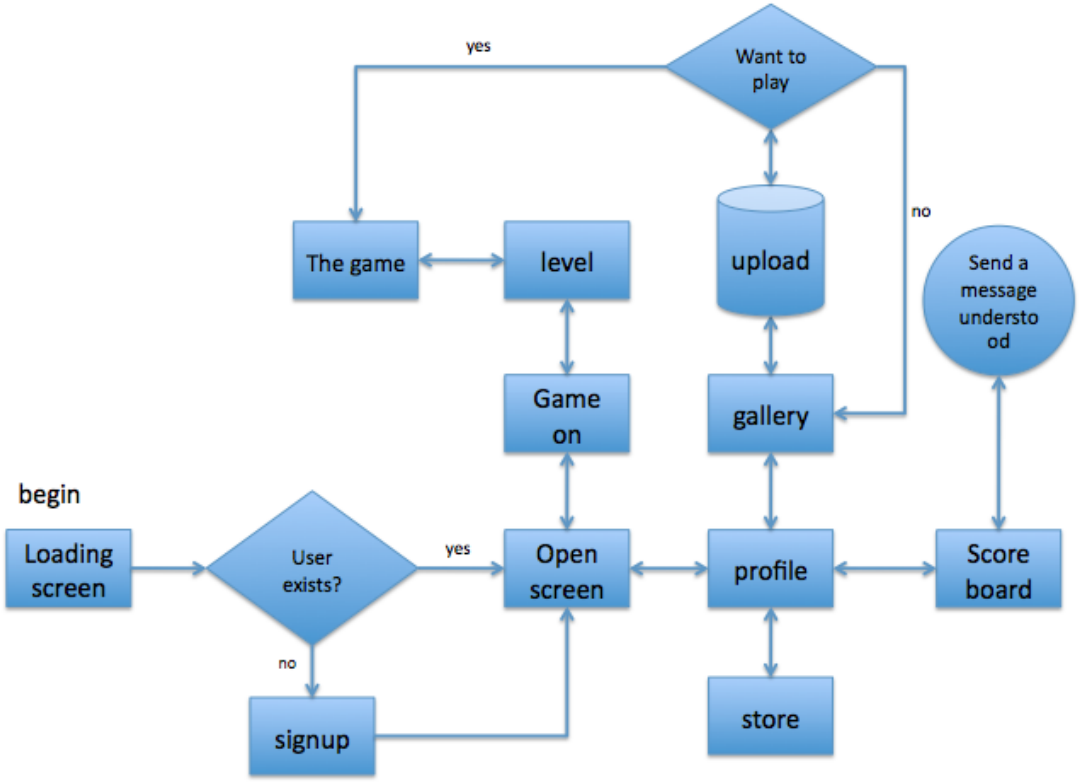
The table shown in picture 20 bellow stores the messages that was actually sent between the users and a way to know if the message have been read or not by the player.

msgtrafic

	sender	recipient	id	beenRead
Edit Copy Delete	eyal	yoni	3	0

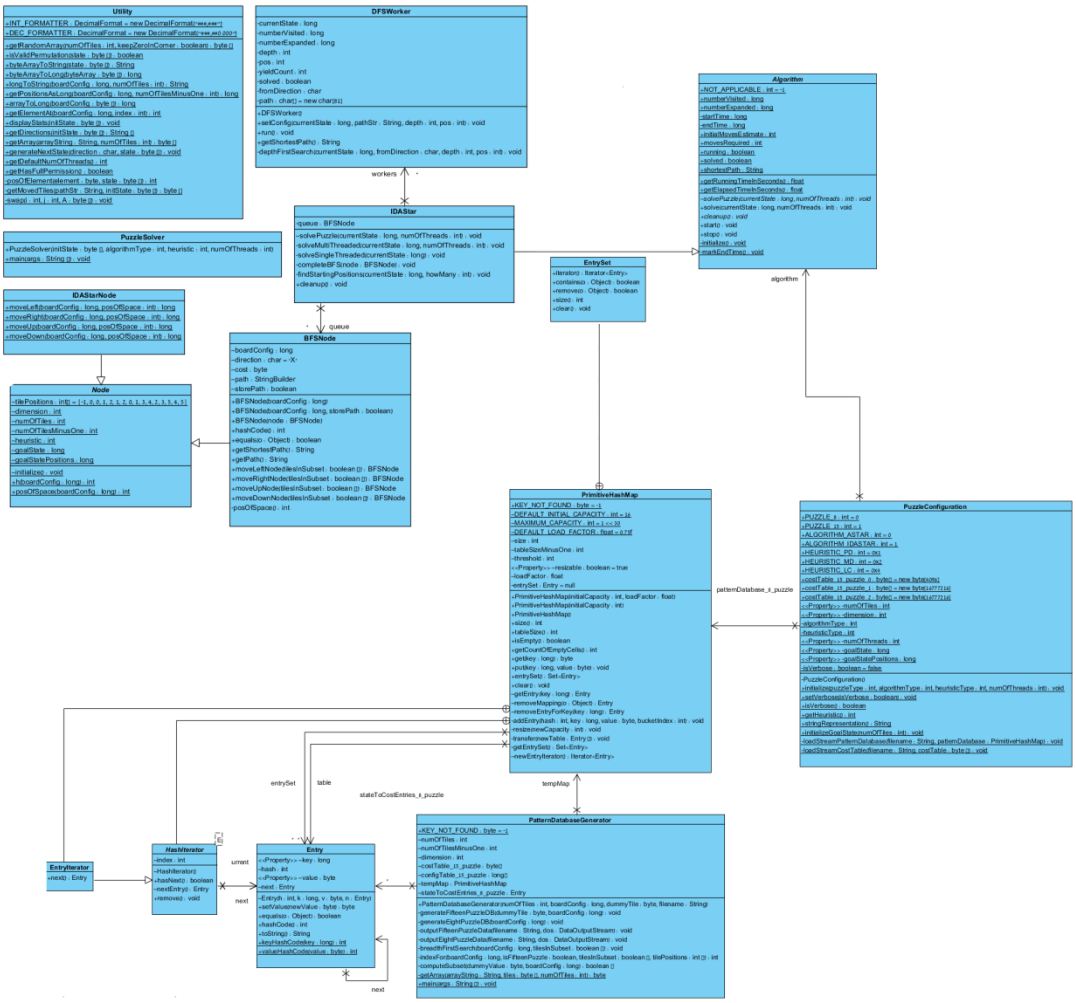
Picture 20

4.3 Data Flow diagram of the GUI



Picture 21

4.4 A complete class diagram of the Java calculation engine



Picture 22

5 Development

5.1 Paradigm

The game application was built in a finite time frame using **the incremental build** developing model since it was hard to estimate the true time frame needed for implementing all of the features that was originally planned.

The GUI, the Java calculation engine and the database are standalone parts. The separation between the different parts allows for changes to be done relatively easily. The GUI, for example, can be completely redesigned and reattached to the system.

5.2 Programming Language

The game application was built using several programming languages because of the following reasons:

5.2.1 Using Java language

The calculation engine is the most complicated part in the developing of the application. We needed it to be implemented in such a way that it will have high performance and the best portability possible. Portability was the main consideration since we didn't know where the engine will be finally deployed therefor we found that the Java language fit us best.

5.2.2 Using MySQL

MySQL was chosen as our preferred language for building the application database because it provides:

- Scalability and Flexibility
- High Performance
- High Availability
- Web and Data Warehouse Strengths
- Strong Data Protection
- Ease Of Management
- Open Source Freedom and 24 x 7 Support

5.2.3 Using HTML/CSS/JavaScript/PHP

The GUI was developed in HTML/CSS/JavaScript/PHP because of two main reasons .The first is that they provide us with the easiest and the most familiar connectivity to Java and MySQL. Secondly they provide us the best way to insure that the GUI will be cross platform in his nature.

5.3 Tools

5.3.1 IDE

The java calculation engine was developed in Eclipse Indigo Java EE IDE for Web Developers.
The MySQL administration was handled by using the server native phpMyAdmin.
The GUI was developed under IntelliJ IDEA.

5.3.2 Testing

All testing was performed in the ides and by trial and error method.

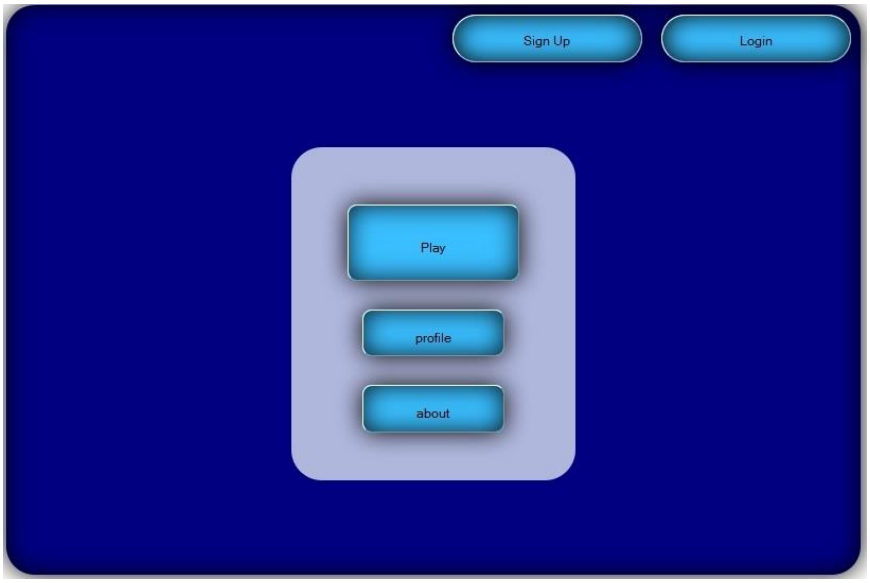
6 User Guide

6.1 Registration

In picture 23 below you can see the opening page. A user can play the game as a registered user or unregistered user as discussed in section 2.3.1. If you wish to play as an unregistered user just press The "Play" button as shown below. To enter as a registered user just press the "Login" button. After that, a box will be open as shown in picture 6. Then you can enter your details that you have entered in the registration form.

To register as a new user please press the "Sign up" button and following that enter your details at the registration form shown in picture 24 below.

The "about" button shown in picture 25 below will open an explanation about the development of the game. To return to the opening page just press the "back" button on that page.



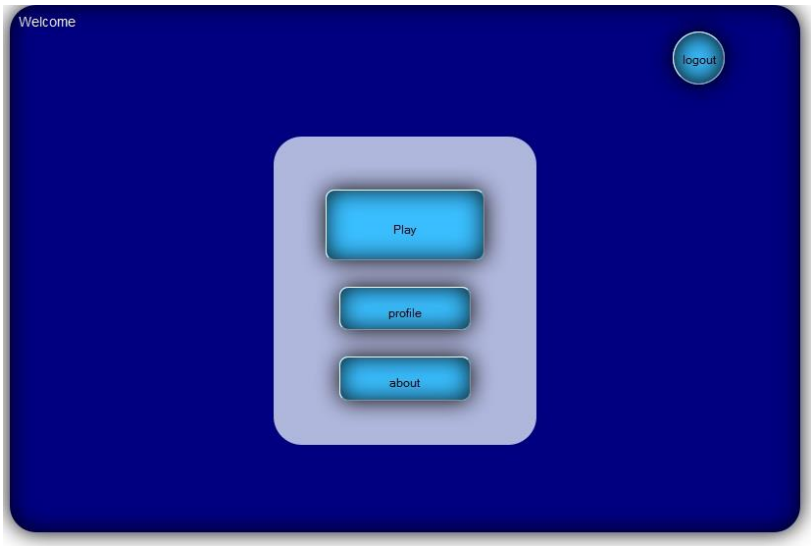
Picture 23

Picture 24

Picture 25

6.2 Choosing a picture

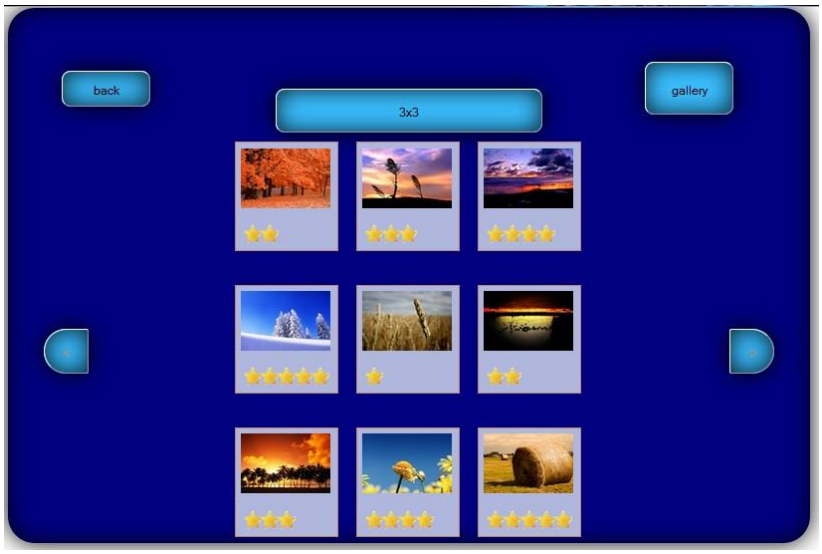
After you have logged in to the game you can logout from the game by navigating back to the opening window and pressing the "logout" button in the upper right corner of the screen as shown in picture 26 below.



Picture 26

After pressing the "Play" button a gallery of photos will be opened. You need to select one picture for the gallery by pressing on one of the photos. The photo that you have selected will be scrambled and you will need to rebuild it as explained later in this section. Pressing on the arrows in the right and left of middle of the screen will enable you to see more photos. The stars on the photos represent the difficulty to reassemble the photos.

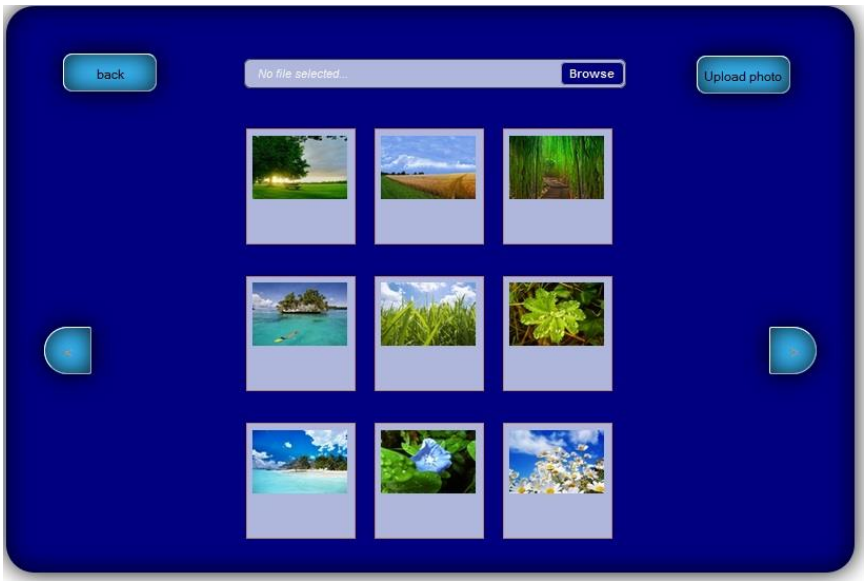
Above the gallery there is the size of the puzzle button .currently you can choose between 3x3 and 4x4 sizes only by pressing the size button. The size that is written on the button is the size the puzzle will be later on.



Picture 27

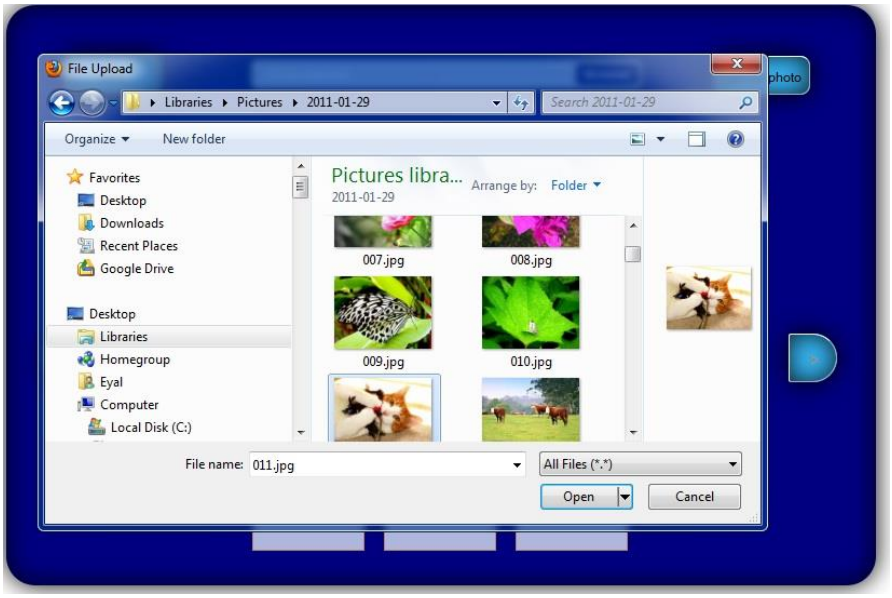
Pressing on the "gallery" button as shown in picture 27 above will lead you into the gallery of photos that users have uploaded to the game shown in picture 28 below. This option is available only to registered users. You can browse between the photos as explained above and you can upload your own photo. Take note that once you have uploaded a photo it will be available to all users.

To upload new photo just press the "browse" button.



Picture 28

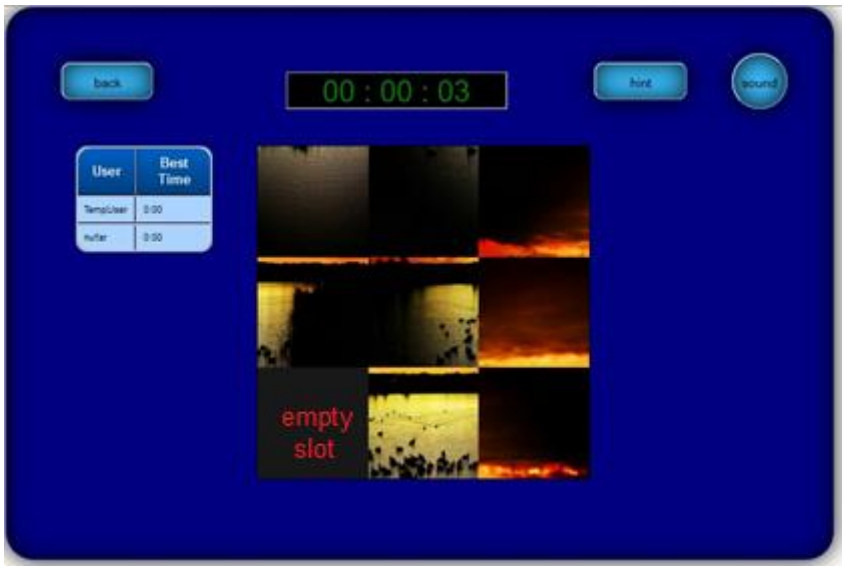
After that a new window will be open as shown in picture 29 below. Navigate to the place that you saved the picture that you wish to upload. Select it and press open. After a short while the photo will be uploaded to the user gallery and you can choose to scrabble it by pressing on it in the gallery.



Picture 29

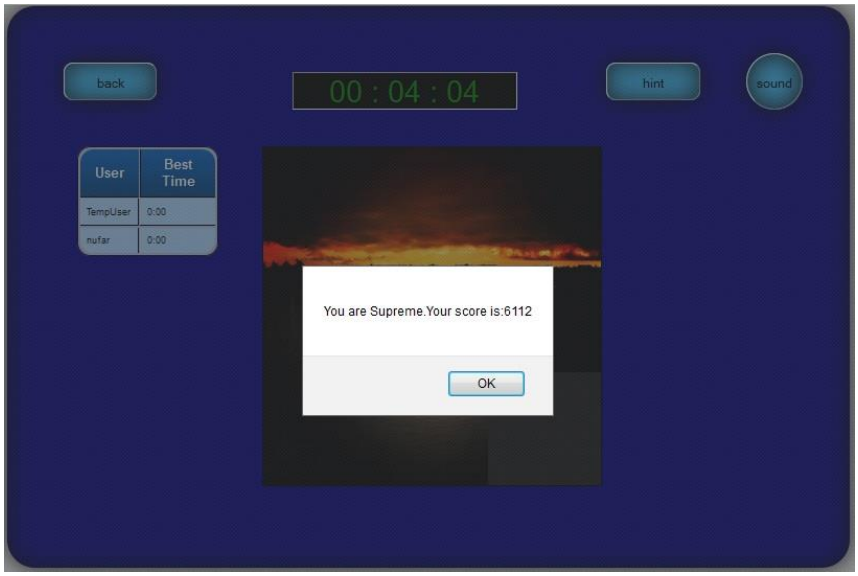
6.3 Playing the game

After choosing a picture from one of the galleries you will be automatically forward to the game screen shown below in picture 30. A Timer will be immediately opened to count the time it takes you to rearrange the puzzle parts to the original picture state. The game is played by pressing on one of the squares. If the square is next to an empty slot it will move there. if the square is in a raw that has an empty slot in it the entire row from the pressed square until the empty slot will move one location toward the empty slot.
Pressing on the "hint" button will give a player the best next move.
Pressing on the "sound" will allow the user to turn on/off the background music.



Picture 30

When all the parts of the puzzle are in the correct position a message will popup announcing victory and displaying the number of points that have been rewarded as shown in picture 31.



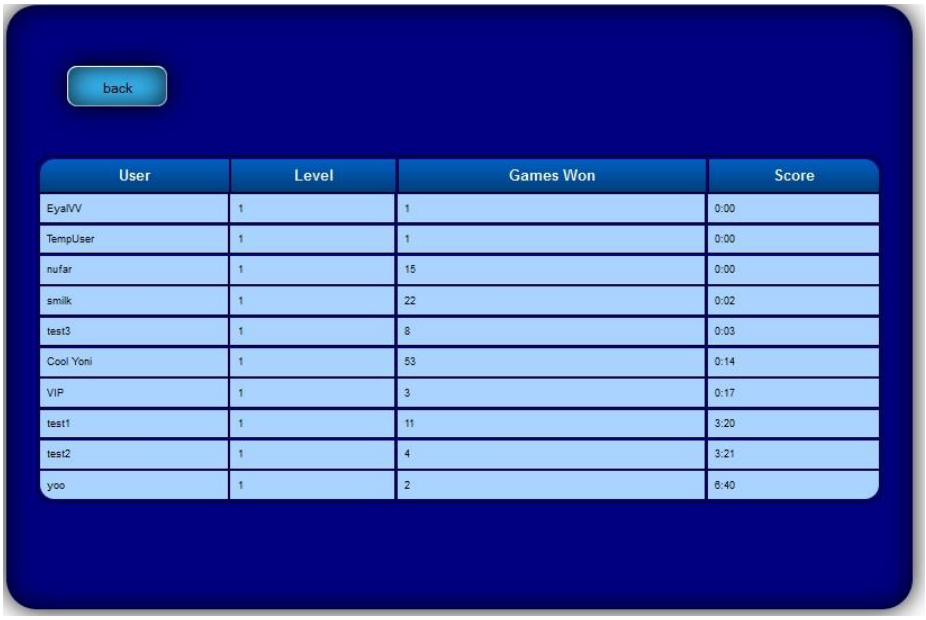
Picture 31

6.4 Points and badges

After pressing the "profile" button, as shown in picture 32, the **registered** user will reach to his personal page. In it he will be able to see all the points and badges that he have accumulated and read messages from other users. Not all features in this page were implemented.

Picture 32

Pressing on the "board" button will show the player the leading board that contains other user's best solving time relative to the personal current user best time as shown in picture 33 below. Pressing on one of the names will allow the user to send a fixed in content message.



User	Level	Games Won	Score
EyalVV	1	1	0:00
TempUser	1	1	0:00
nufar	1	15	0:00
smilk	1	22	0:02
test3	1	8	0:03
Cool Yoni	1	53	0:14
VIP	1	3	0:17
test1	1	11	3:20
test2	1	4	3:21
yoo	1	2	6:40

Picture 33

7 Summary

7.1 Main Focal Points

The project focused on the following goals:

7.1.1 Creating a cross platform puzzle game

One of the project's main objectives is to create a cross platform puzzle game application. To accomplish this goal we used programing languages that are a cross platform in their nature. We have chosen to build a GUI that has a simple design to minimize the risk of failing this goal. In our point of view the more simple the GUI is the better the chance we won't encounter a compatibility problem. It was not in our power to check all the possible platforms and it is unrealistically in our given time frame to implement this part of the application to support older versions of web browsers.

7.1.2 Using gamification

Understanding gamification and its effectiveness beyond anecdotal evidence and hype is evidently a pertinent practical issue as well as, increasingly, a scholarly pursuit. Regardless of the increasing amount of both industry chatter and scholarly articles, there still is a dearth of coherent understanding whether gamification works and under which circumstances.

In our empirical development we found the gamification game thinking ideas and its game mechanics very contributing to our goal to create a fun and challenging game that can keep the player interested in the game in the long run.

7.1.3 Implementing a working 8-puzzle and 15- puzzle solver

Over two decades, solving sliding puzzles have been used in computer laboratories for performance testing of search methods and as a result of that there are many ways to address the problem of solving sliding puzzle today. In the planning stage we had to do a deep and thorough research on this subject. It was hard for us to find the right way to implement a solution that will be best for our needs and fit to our technical capabilities. This is why the implementation of the 8-puzzle and 15- puzzle solver was the hardest part of the development of this application.

7.2 Conclusions

7.2.1 Our Key for Success is a dynamic goal

In the planning stage many ideas came to mind. The Understanding that it is not possible to implement all of the features that were initially thought of brought us to the conclusion we need to walk with too the light of the incremental development method candle. We developed the components under the Understanding that a repeated change will be made to them. This Understanding in the planning stage made these changes possible and easier during the development.

7.2.2 Tools can help us

While implementing the system without using phpMyAdmin, Eclipse, IntelliJ IDEA could be done, the amount of effort would increase dramatically and the quality of the final result would be considerably lower.

7.3 Future Work

7.3.1 More gamification features

In our application we have only implemented a small part of what gamification has to offer to us due to time constrains. Many game mechanics and game dynamics such as bonuses, combos, countdowns and quests have not been implemented. We think that there is a place to implement more of the mentioned above since it will make our game application even more fun, challenging and addicting.

7.3.2 Support a solution for a puzzles greater then 4X4

Solving the NxN sliding puzzle problem is proven to be NP-hard nevertheless there are ways to solve larger sizes of this type of puzzle in less than 250ms (normal human reaction time). The possible ability to technically solve larger puzzles in a period of time that is suitable for a game application opens the door for future development.

7.3.3 Better designed GUI

The current GUI is not complete since we did not finish implementing the between user messaging system and the hint option. Furthermore the GUI is lacking the professional touch of a web designer and it is clear to the eye that software engineers have built it.

7.3.4 Improving the Database

The building of the MySQL database was in the button of our priorities and was built quickly and in the purpose of "just to work" and test our GUI. The Database need to be redesigned in a way it will be more properly built and able to support future feature. Also the Database currently holds users password which is not the best way to securely support a login system. The whole application needs to be modified so it will support password verification by using hashing.

8 Literature

8.1 Research

8.1.1 Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA* by Alexander Reinefeld, Paderborn Center for Parallel Computing

8.1.2 gamification.org

8.1.3 aaai.org

8.1.4 Wikipedia, the free encyclopedia

- Gamification
- Sliding puzzle
- IDA*

8.1.5 Psychologytoday.com

8.2 Technical References

8.2.1 Shenkar College of Engineering and Design

- Java Programming Course

8.2.2 Stackoverflow.com

8.2.3 W3Schools

8.2.4 PHP / Java Bridge

8.2.5 Oracle

8.2.6 MySQL

8.2.7 phpMyAdmin

9 נספח:תקציר מנהלים

PUMIEZ היא אפליקציית פזאל הזזה המבוססת על משחק פאזל הזזה הקלאסי. בגירסה זו של המשחק אנו מקווים שנצליח לשבור את הסטיגמה הרווחת - שפאזל הזזה הוא משעמם ובנאלי. אנו מקווים שמשחק זה יפיח חיים רוח חדשה במשחק הקלאסי ויגרום לשחקנים לרצות לשחק בו שוב ושוב. בשביל להשיג מטרה זו החלטנו להטמיע חלק ממנגנוי גיימיפיקיישן באפליקציה שלנו. הסיבה שבחרנו להטמיע את מנגנוני המישחוק ש גיימיפיקיישן מציע היא מאחר ובמקרים רבים מנגנונים אלו הצליחו לעורר ולתפוס את סקרנותם של שחקנים רבים באפליקציות עם תוכן קבוע.

המטרת הבסיסית של משחק פאזל הזזה היא לאתגר את השחקן להמשיך להזיז את חלקי הפאזל במסלול מסויים בכדי להצליח להגיע לתצורה הסופית של הפאזל. השימוש במנגנוני המישחוק יאפשר לנו בתקווה להגביר את האתגר שהמשחק מקנה באופן טבעי ולמנוע את המצב שהשחקן ירגיש שהוא מיצא את המשחק.

מנגנוני המישחוק הקיימים בגיימיפיקיישן כוללים בין היתר: צבירת נקודות, דירוג בחתך על פי זמן ושליחת הודעות תחרותיות בין השחקנים. כל אלו עזרו במקרים רבים לשחקן לשמור על רמה מתחדשת וגבוהה של אתגר .מאידך אתגר מתמשך עלול לגרום לשחקן להתייאש ולכן היכולת הייחודית של האפליקציה שמקנה לשחקן אפשרות לבקש רמזים במקרה שאינו יודע איך להמשיך את המשחק יוצרת רשת ביטחון המונעת את הרמת היידים של השחקן.

באפליקצייה זו אנו מאתגרים את השחקן לנסות לפתור פאזל הבנוי מ8 או מ15 חלקים. כמו כן קיימת אפשרות לבחור תמונה ממאגר תמונות הקיים במשחק או קיימת אפשרות להעלות תמונה מהמחשב האישי ממנה יהיה מורכב הפאזל. פתירת הפאזלים מהסוג הזה נשמעת פשוטה על פני השטח אך העובדה היא שכבר במשך שני עשורים פאזלים אלו משמשים במעבדות מחשבים לצורך בחינת בינה מלאכותית.

באופן רשמי לא ידוע כיום על אלגוריתם מסויים אשר בוודאות פותר תמיד את בעיית פזאל ההזה בצורה הקצרה ביותר (פתירת בעיית פאזל ההזזה N x N נחשבת כבעייה בדרגת קושי של NP-Hard). על סמך מחקרים מדעיים אשר עשו ניסויים בתחום חקר הביצועים והאינטיליגנציה המלאכותית הגענו למסקנה שהאלגוריתם המתאים ביותר למטרה של מציאת פתרון מהיר לבעיית פאזל ההזה הינו אלגוריתם הנקרא IDA-star .

ממשק המשתמש של האפליקצייה נכתב בשפות html5/php4/css3/JavaScript ולכן קיימת האפשרות לשחק בו כמעט בכל פלטפורמה שבה מותקן דפדפן.כמו כן הממשק מכיל מוזיקת רקע מרגיעה שמשרה אווירה המתאימה למשחק חשיבה.