ANNALS of Faculty Engineering Hunedoara
– International Journal of Engineering
Tome XV [2017] – Fascicule 3 [August]

ISSN: 1584-2665 [print; online]
ISSN: 1584-2673 [CD-Rom; online]
a free-access multidisciplinary publication
of the Faculty of Engineering Hunedoara

[1.]Anca-Elena IORDAN

DESIGN AND IMPLEMENTATION OF A PUZZLE GAME USING JAVA SE

[1.] University Politehnica Timisoara, Faculty Engineering Hunedoara, ROMANIA

Abstract: In this article are rendered the requisite stages for the accomplishment of a puzzle game. The puzzle consists in moving the polyhedra of different types and colours, having five levels of difficulty. It can choose from three types of polyhedra: prisms, pyramids or pyramidal frustums. The game is ended when on every line the polyhedra have the same colour and on every column have the same polygon as base in an increasing number of sides of the base. The design of the puzzle is accomplished by the next UML diagrams: use-case diagram, class diagram and component diagram. By achieving these types of diagrams, the puzzle is described in an obvious and concrete approach, without ambiguousness. The implementation of the game was realised through the Java programming language with NetBeans IDE. The game also allows the determination of the optimal solution by implementing of A* informed search algorithm using Manhattan admissible heuristic function.

Keywords: Puzzle, UML, ArgoUML, Java, NetBeans IDE, A* algorithm

1. INTRODUCTION

The puzzle game presented consists in a number of polyhedra of different types and colours that can be moved using the free available spaces. The problem requires to be found the movements in order the game starts from an initial configuration and reaches to a determined configuration. The puzzle has the option for choosing from three types of polyhedra: prisms, pyramids or pyramidal frustums. The difficulty of puzzle depends by the number of the used colours. The game is ended when on every line the polyhedra have the same colour and on every column have the same polygon as base in an increasing number of sides of the base. The game also allows the determination of the optimal solution by implementing of A* heuristic algorithm [1]. Because the problem is enough complex, the principal difficulty in solving it is given by dimension of search space, that leads to necessity of a heuristic search [2].

2. THE ANALYSIS OF THE PUZZLE

From the perspective of unified modelling language, the analysis of puzzle application includes the representation of the use-case diagram. The use-case diagram [3, 4] offers simplified and graphical representation of what the puzzle game must really do. Use-case diagram is based upon functionality and thus will focus on the "what" offers the game and not the "how" will be realized.

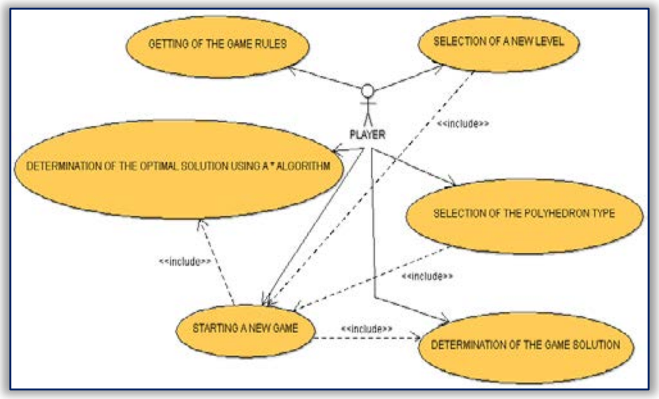

GETTING OF THE GAME RULES
SELECTION OF A NEW LEVEL
PLAYER
<<include>>
DETERMINATION OF THE OPTIMAL SOLUTION USING A * ALGORITHM
SELECTION OF THE POLYHEDRON TYPE
<<include>>
<<include>>
STARTING A NEW GAME
<<include>>
DETERMINATION OF THE GAME SOLUTION

Figure 1. Use-case diagram

The use case diagram corresponding to this puzzle software, presented in figure 1, includes:

- ❖ One actor - the user who is external entity with that the puzzle game interacts.
- ❖ Seven use-cases which describe the performance of the game.
- ❖ Relationships between user and use-cases (association relationships), and relationships between use-cases (dependency and generalization relationships).

3. THE DESIGN OF THE PUZZLE

Conceptual modelling allows identifying the very important elements for the interactive software [5, 6]. Class diagram is represented in figure 2 in order to be observed the connection mode between the classes and the interfaces that are used and also the composition and aggregate relationships between instances.

After identifying the specific elements of the puzzle, they were been implemented 7 interfaces and 16 classes grouped into three packages. Package "geometrie" brings together concepts that correspond to the polyhedra used by puzzle, consisting of 8 classes and one interface. Of the eight classes, three are abstract classes.

Package "algoritm" brings together concepts that correspond to the A* heuristic algorithm, consisting of 6 classes and 5 interfaces.

For memorizing a state configuration, there was implemented "Stare3D" class which implements "Stare" interface. For memorizing a node of search tree, it was implemented "Nod_Puzzle" class which realizes "Nod" interface and for memorizing an expanded node together with its successors, it was implemented "Nod_Expandat_Puzzle" class which realizes "Nod_Expandat" interface.

A* heuristic search algorithm is implemented by using "Algoritm_A_Stea_Puzzle" class. An instance of this class is composed by two instances of "Nod_Puzzle" class, according to composition relationship which exists in diagram, but can also contain instances of "Nod_Expandat_Puzzle" class, as it can be observed from aggregate relationship presented in diagram.

Package "gui" brings together concepts that correspond to the graphical user interface, consisting of 2 classes and one interface. For designing the graphical interface of the game, it was implemented the „Puzzle" class. It can observe that this class implements the „Runnable" interface. An instance of this class is composed by one instance of „Suprafata3D" class, but can also contain instances of "Algoritm_A_Stea_Puzzle" and "Nod_Puzzle" classes, as it can be observed from aggregate relationship presented in diagram.

4. THE IMPLEMENTATION OF THE PUZZLE

The component diagram [7, 8] enables the visualization modules that compose the software and the dependencies between them. The diagram that is shown in figure 2 describes the collection of components that all together ensure the functionality of the puzzle game.

Central component of the diagram is *Puzzle.class,* a component obtained by transforming by the Java compiler into executable code of the *Puzzle.java* component. As can be seen that component interacts directly with components *Suprafata3D.class.*

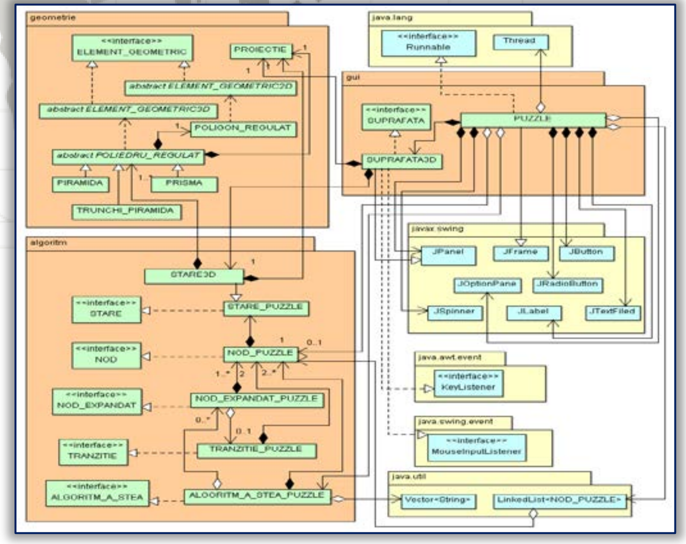


Figure 2. Class diagram

5. GRAPHICAL USER INTERFACE

The puzzle game is accomplished using the Java object-oriented programming language [9, 10, 11].Given that specified requisites in uses cases diagram (figure 1) it was designed graphical user interface of the puzzle. In figures 4 and 5 are presented the five levels of difficulty. In figure 6 is presented the corresponding final state of the third level of difficulty. The structure of the class corresponding to the graphical interface is presented in the following:

```
public class PUZZLE extends java.awt.Frame implements Runnable {
SUPRAFATA3D s3;
Thread fir;
LinkedList<NOD_PUZZLE> solutie_optima;
public PUZZLE() { ... }
public void actualizareNrMutari(int nr) { ... }
private void initComponents(){ ... }
private void exitForm(WindowEvent evt){ ... }
private void nouActionPerformed(ActionEvent evt){ ... }
private void nivelStateChanged(ChangeEvent evt){ … }
private void ajutorActionPerformed(ActionEvent evt) { … }
private void prismaActionPerformed(ActionEvent evt) { … }
private void piramidaActionPerformed(ActionEvent evt){ … }
private void trunchiActionPerformed(ActionEvent evt) { … }
private void optimActionPerformed(ActionEvent evt) { … }
private void nouSimpluActionPerformed(ActionEvent evt) { … }
public static void main(String args[]) { … }
private javax.swing.JButton ajutor;
private javax.swing.JLabel l1;
private javax.swing.JLabel l3;
private javax.swing.JLabel l4;
private javax.swing.JTextField mutare;
private javax.swing.JSpinner nivel;
private javax.swing.JButton nou;
private javax.swing.JButton nouSimplu;
private javax.swing.JButton optim;
private javax.swing.JRadioButton piramida;
private javax.swing.ButtonGroup poliedru;
private javax.swing.JPanel principal;
private javax.swing.JRadioButton prisma;
private javax.swing.JRadioButton trunchi;
}
```

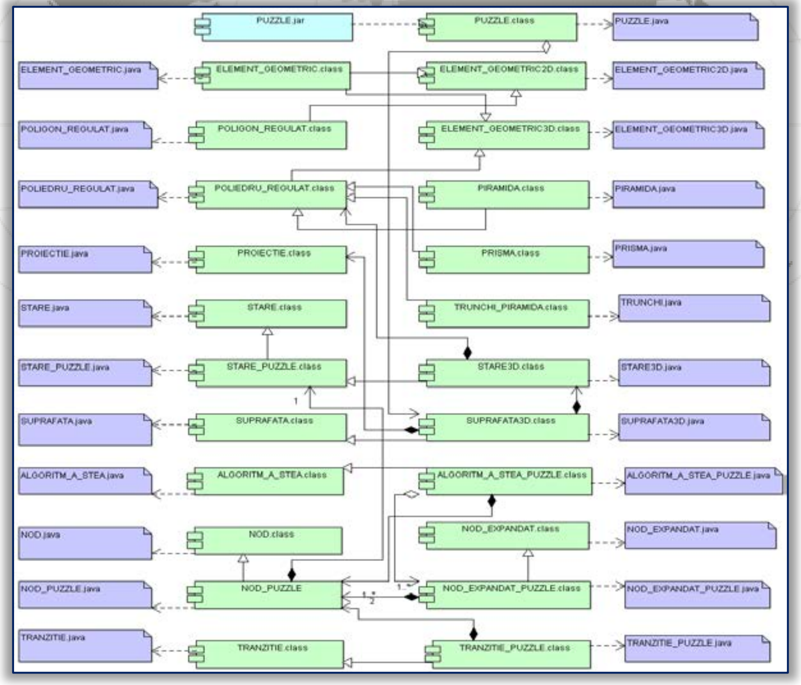


Figure 3. Component diagram

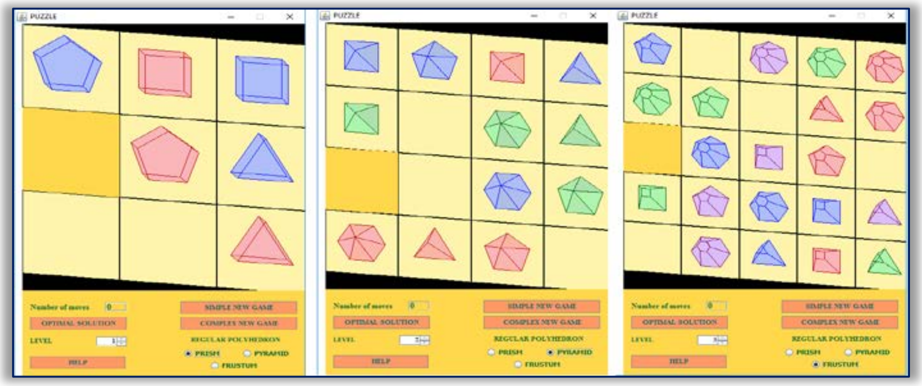


Figure 4. First three levels of difficulty

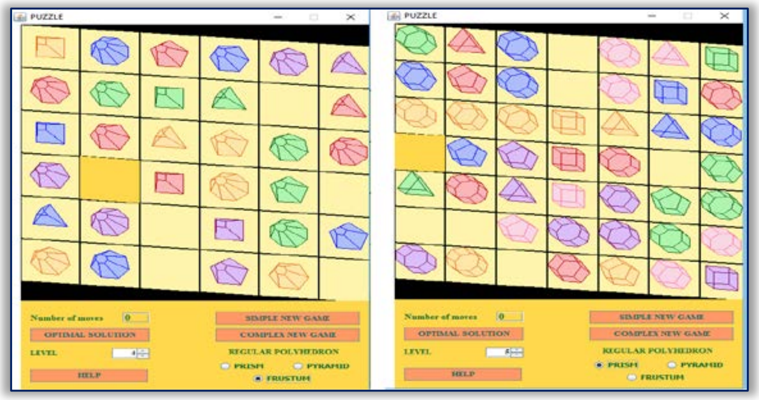


Figure 5. Last two levels of difficulty

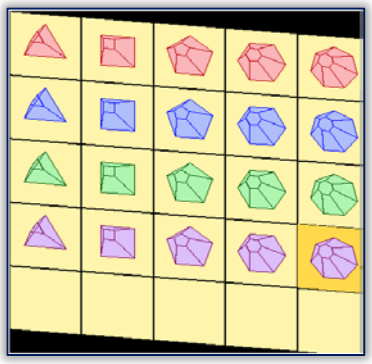
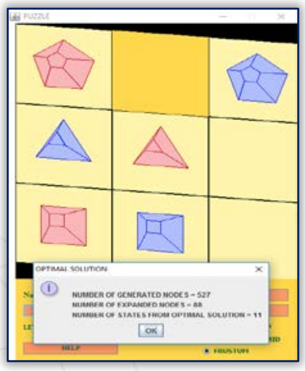


Figure 6.Corresponding final state to the third level of difficulty

By selecting the button "OPTIMAL SOLUTION" is instancing an object of "Algoritm_A_Stea_Puzzle" class which permits the determination of the optimal solution by running of the A* heuristic algorithm. In figure 7 is presented the result returned by this algorithm for an example.



```
OPTIMAL SOLUTION
NUMBER OF GENERATED NODES = 527
NUMBER OF EXPANDED NODES = 88
NUMBER OF STATES FROM OPTIMAL SOLUTION = 11
OK
```

Figure 7. Optimal solution

The structure of the class corresponding to the A* heuristic algorithm is presented in the following:

```
public class ALGORITM_A_STEA_PUZZLE implements ALGORITM_A_STEA {
private NOD_PUZZLE si;
private final NOD_PUZZLE sf;
private LinkedList<NOD_PUZZLE> frontiera;
private LinkedList<NOD_EXPANDAT_PUZZLE> teritoriu;
private LinkedList<NOD_PUZZLE> solutie;
private long nr;
private Vector<String> concluzie;
private LinkedList<NOD_PUZZLE> succesori;
public ALGORITM_A_STEA_PUZZLE(STARE3D s1) { ... }
public LinkedList<NOD_PUZZLE> getFrontiera() { ... }
public LinkedList<NOD_EXPANDAT_PUZZLE> getTeritoriu() { ... }
public LinkedList<NOD_PUZZLE> getSolutie() { ... }
public LinkedList<NOD_PUZZLE> getSuccesori() { ... }
public Vector<String> getConcluzie() { ... }
public void setFrontiera(LinkedList<NOD_PUZZLE> L) { ... }
public void setTeritoriu(LinkedList<NOD_EXPANDAT_PUZZLE> L) { ... }
public void initializare() { ... }
public LinkedList<NOD_PUZZLE> generareSuccesori(NOD_PUZZLE opt) { ... }
public void algoritm() { ... }
public NOD_PUZZLE optim() { ... }
public int apartine(NOD_PUZZLE s, LinkedList<NOD_PUZZLE> L) { ... }
public void inserare(NOD_PUZZLE s) { ... }
public void solutie(NOD_PUZZLE O) { ... }
public void setaresuccesori(int p, NOD_PUZZLE np) { ... }
}
```

The structure of the "NOD_PUZZLE" class is presented in the following:

```
public class NOD_PUZZLE implements NOD {
private STARE_PUZZLE stare;
private NOD_PUZZLE predecesor;
private int f;
private int g;
private int h;
public NOD_PUZZLE() { ... }
public NOD_PUZZLE(STARE3D s) { ... }
public NOD_PUZZLE(STARE3D s, NOD_PUZZLE p) { ... }
public NOD_PUZZLE(NOD_PUZZLE S) { ... }
public void setare(STARE3D s,NOD_PUZZLE p,int f1,int g1,int h1) { ... }
public void setStare(STARE3D s) { ... }
public void setPredecesor(NOD_PUZZLE p) { ... }
public void setF(int f1) { ... }
public void setG(int g1) { ... }
public void setH(int h1) { ... }
public int getF() { ... }
public int getG() { ... }
public int getH() { ... }
public STARE_PUZZLE getStare() { ... }
public NOD_PUZZLE getPredecesor() { ... }
public void F() { ... }
public void G() { ... }
public void H() { ... }
public boolean egal(Object S1) { ... }
public String toString() { ... }
}
```

The structure of the “NOD_EXPANDAT_PUZZLE” class is presented in the following:

```
public class NOD_EXPANDAT_PUZZLE implements NOD_EXPANDAT {
private NOD_PUZZLE expandat;
private LinkedList<NOD_PUZZLE> succesori;
public NOD_EXPANDAT_PUZZLE() { ... }
public NOD_EXPANDAT_PUZZLE(NOD_PUZZLE n) { ... }
public NOD_EXPANDAT_PUZZLE(NOD_EXPANDAT_PUZZLE n) { ... }
public void setNodExpandat(NOD_PUZZLE n) { ... }
public void setSuccesori(LinkedList<NOD_PUZZLE> L) { ... }
public void adaugSuccesor(NOD_PUZZLE n) { ... }
public int getNrSuccesori() { ... }
public NOD_PUZZLE getSuccesor(int i) { ... }
public LinkedList<NOD_PUZZLE> getSuccesori() { ... }
public NOD_PUZZLE getNodExpandat() { ... }
public void generareSuccesori() { ... }
}
```

6. CONCLUSIONS

Because the representation of the diagrams corresponding to all three phases: analysis, design and implementation, the puzzle game has been described in an obvious and objective manner, without ambiguity. The use of the unified modelling language for the achievement of the diagrams is characterized by rigorous syntactic, rich semantic and visual modelling support.

References

[1.] W. Ertel, Introduction to Artificial Intelligence, Springer-Verlag, Berlin, 2011
[2.] Z. Shi, Advanced Artificial Intelligence, Word Scientific Publishing, 2011
[3.] P. Jalote, A Concise Introduction to Software Engineering, Springer-Verlag, 2008
[4.] M. Fowler, K. Scott, UML Distilled: A Brief Guide to the Standard Object Modelling Language, Pearson Education, 2003
[5.] J. Hunt, Guide to the Unified Process featuring UML, Java and Design Patterns, Springer London Ltd, 2014
[6.] K. Lunn, J. Skelton, S. Bennett, Schaum’s Outline of UML, McGraw-Hill Education, 2004
[7.] B. Bruegge, A. Dutoit, Object Oriented Software Engineering Using UML, Patterns, and Java, Pearson Education, 2013
[8.] A. Dennis, B. H. Wixom, D. Tegarden, Systems Analysis and Design with UML, John Wiley & Sons Ltd, 2012
[9.] R. Martin, Java Application Architecture, Pearson Education, 2012
[10.] J. Bryant, Java 7 for Absolute Beginners, Springer, 2012
[11.] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, The Java Language Specification, Oracle, 2013

ANNALS of Faculty Engineering Hunedoara
– International Journal of Engineering