Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή τεχνολογικών εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Τίτλος: **Εργαλείο για την αντιμετώπιση του προβλήματος διακλάδωσης που συνδυάζεται σε χρονικές και χωρικές βάσεις δεδομένων.**

(A tool for addressing the ramification problem combined in temporal and spatial databases.)

Φοιτητής: Λυκάκης Παύλος

Α.Μ: 3847

Επιβλέπων καθηγητής: Παπαδάκης Νίκος

ΗΡΑΚΛΕΙΟ

2017

A tool for addressing the ramification problem combined in temporal and spatial databases.

Lykakis Pavlos

Department of Informatics Engineering,

Technological educational institute of Crete,

Greece, September 2017.

Likakis@hotmail.com

Table of Contents

Table of images

List of tables

A tool for addressing the ramification problem combined in temporal and spatial databases.

Abstract

In this paper, we study the ramification problem, both in the setting of geographic databases and setting of chronicle databases. The ramification problems in spatial-temporal databases are complex and multifaceted problems and no satisfactory solution has been proposed as of yet. The ramification problem is concerned with the indirect consequences of an action when integrity constraints exists. As integrity constraints we define criteria which must be satisfied in every transmission at the database. Thus, any change in database have both direct and indirect effects. If that is the case some indirect effects may generated in order to keep intact the integrity constraints. Subsequently we present a tool used in a scenario which uses integrity constraints in order to manage data and keep the database consistent. By producing the appropriate Java and SQL commands we came up to a satisfactory solution.

1. Introduction

1.1 Spatial database

A Spatial database is used to store and query data that represents objects in a geometric space. These objects can be most simple geometric objects such as points, lines and polygons. Although there are spatial databases that handle more complex shapes such as 3D objects, linear networks and topological overages. [32]

In order to be able to handle many types of geometrical shapes, the operators introduce the concept of the bounding box. The bounding box encircle any kind of shape in order to provide some comparable properties in these shapes. For example, if a polygon overlaps with another polygon, in order to avoid unnecessary time and computing power to check, we compare only the bounding box of each polygon. If there is not overlap, there is no need to check between polygons.

1.2 Temporal database

A temporal database is used to store data relating to time instances. Usually there are three temporal aspects:

a) **Valid time** is a chronically period in which a fact I true.
b) **Transaction time is** a chronically period where a fact stored in the database and was known for this period.
c) **Bitemporal data** the previous attributes can be combined and form the bitemporal data.[33]

In this paper we will examine the first one**, valid time**. We will use the valid time to check integrity constraints in a database that uses time and space.

2. The ramification problem in conventional database.

The ramification problem is a hard and infamous and ever present problems in databases. In order to explain the problem in a better way we will use an example. In a simple circuit we will present a scenario which has two conditions described as switches and one result described as a lamp. The integrity constrains that will needed are the follow:

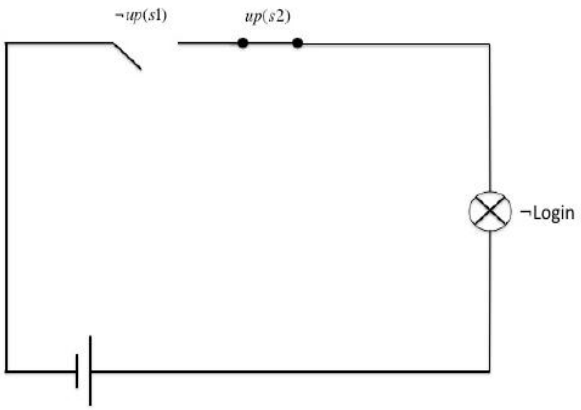
$$\text{ups (s1) AND ups(s2)} = \text{log in} \quad (1)$$

$$\neg \text{ ups (s1)} = \neg \text{ log in} \quad (2)$$

$$\neg \text{ ups (s2)} = \neg \text{ log in} \quad (3)$$

The scenario [34] is a simple scenario of log in an account. In order to login, needs both Username (s1) and password (s2) to be correct, that's described at the first (1) constraint. In the second (2) constraint it's described that if the username is wrong the login is not true. In the third (3) constraint it is described if the password is wrong then login is not true. Action toggle switch change the current state of a switch as follows:

If up(s), toggle switch(s) → ¬up(s)



If ¬up(s), toggle switch(s) → up(s)

Fig. 1

These propositions describe the direct effects may appear in a case of toggle of the switches .In order to keep the database consists we need to satisfice these integrity constraints. Assume that there is a case where S = {¬up (s1), up (s2), ¬login}, all integrity constraints are kept intact. Now in a case where toggle switch (S1) true (means username is correct), has a direct effect. Assume then that is the case S1 = {up (s1), up (s2), ¬login}, in the current case there is an inconsistency. There is a violation of the first integrity constrains and in order to maintain the circuit consistent we have two options:

S2 = {up (s1), up (s2), login}

S3 = {up (s1), ¬up (s2), ¬login}

These two situations are consistent because there is no violation of any integrity constraint. In the S2 case, we have toggle switch (s1) and the username is no longer wrong and login is true. In the S3 case, we have toggle switch (s2) and the password is incorrect. Both cases happen in order to keep the database consistent and not as a direct effects. The reasonable conclusion is that login is true. We must determine which could be the indirect effects of the action in order to infer this conclusion. We can see that because there are

integrity constraints the indirect effects exists. Briefly the main subject when we see the ramification problem is the description of the indirect effects of an action. In the past have been made many related research try to address the ramification problem based on the event calculus and [1].

2.1 Basic terminology in situation calculus

In order to understand more easily the next sections we will represent some basic terminology.

- **Fluents:** All predicates and functions when their true value changes from a state to other state.
- **Situation:** A sequence of actions that bring in a possible evolution of the world.
- **Action:** A change to values of some fluents.
- **Consistent:** When there is no violation in any integrity constrain in a specific situation.
- An action may change a situation and bring in another situation as a result.
- **Do:** With the binary do (**A, S**) we declare the situation that will outcome from an action **A** in the situation **S**.
- When **some conditions** are satisfied an action may proceed and we use for these preconditions terms described before. The **Poss** (a binary predicate) reveals whether a precondition holds. An action **A** can proceed in the situation **S** when Poss (**A, S**) is true.

Betwixt the proposed solutions the simplest ones are using a minimal change approach [2] [3]. Such solutions suggest that while an action happens in a situation S one needs to find the consistent situation S', then S' has the fewer changes from the situation S. As a sample consider the modeling of a simple circuit. Hypothesize a situation S = {¬up (s_1), up (s_2), ¬login}. The action toggle switch(s_1) changes the circuit to S' = {up (s_1), up (s_2), ¬login}, which situation is inconsistent. In the current case there are two situations which are consistent: S_1 = {up (s_1), up (s_2), login} and S_2 = {up (s_1), ¬up (s_2), ¬login}. Its logical that login is true, because toggle switch(S_2) the second proposal is not. As a result (indirect effect) of ***upping*** a switch, login is true, but it is not consistent ***downing*** one switch as an indirect effect of ***upping*** another. Thus, we prefer the situation S_1 over the S_2. Also in this case both solutions are equally close to the first S situation, so even if we approach the minimal change we can't choose one than the other. The previous problem can be solved based on the categorization of fluents [4] [5] [6]. The categorization of fluents divined into primary and secondary. In the first category a fluent can change as a direct effect of an action. Contrary, at the second category a fluent can only change as a indirect effect of an action. After an action occurs, we take the situation with the fewer changes in primary fluents. In the previous example we have as primary flutent the Fs = {login} an as secondary fluents the Fp = {up(s_1), up(s_2)}. We chose the S_1 situation because there are not any changes in its primary fluents. However, this categorization of fluents can only solve the ramification problem if the fluents can be categorized. Generally, it's not adequate this kind of solution when the same fluents are primary for some actions and secondary for other actions.

Also there is a solution that is using causal relationships [7][8][9]. Every causal relationship has two parts. The first part it's called context and declares a relationship between an action and the effect that action creates. It contains one fluent formula and either if it's true or false, we have this causal formative relationship. The second part called cause of the effect part or latter part and is the result of an indirect effect of an action. The form of a causal relationship is:

$$e \quad causes \ r \quad if \quad \Phi \quad ,$$

The meaning of each symbol, e: action, r: result of the effect, Φ: fluent formula depending on the context.

Other solutions to the ramification problem [4][5][6][10][11][12][13][14][15][16][17][18][19][20][21][22][23][24][25][26][27][28][29][30][31][32][33][34] rely on the idea that actions have effects only on the next situation. Also they rely on the persistence of fluents.

2.2 The ramification problem combined in spatial databases and temporal databases.

The ramification problem is concerned with the indirect consequences of an action when integrity constraints exists. As integrity constraints we define criteria which must be satisfied in every transmission at the database. Thus, any change in database have both direct and indirect effects. If that is the case some indirect effects may generated in order to keep intact the integrity constraints.

In this case we present the ramification problem in an example with geographical and time integrity constraints Fig.1, Fig.2. We suppose that database stores two shapes, (a square with solid outline and a shape with dashed outline) that they have two interdepended properties. First a time period and second a not overlap property. The second property is true while the current time(t) is between the times period start and end. This two properties are actually the integrity constraints. At the corners of each shape are shown the coordinates (x, y) of the points that is connected. In this two figures we can see two different cases.

Case 1: The square holds the second property (Not overlap) for a time period 1/7/2017 till 1/9/2017, which mean is true in the present 1/8/2017. At time t1, we store in database a second shape (with dashed outline) which happens to overlap the first shape. In time t2, there is a function **checkDB** which checks, if the first shape holds the not overlap property and then, moves the second shape to the nearest place where there is no overlap, nine units to the right.

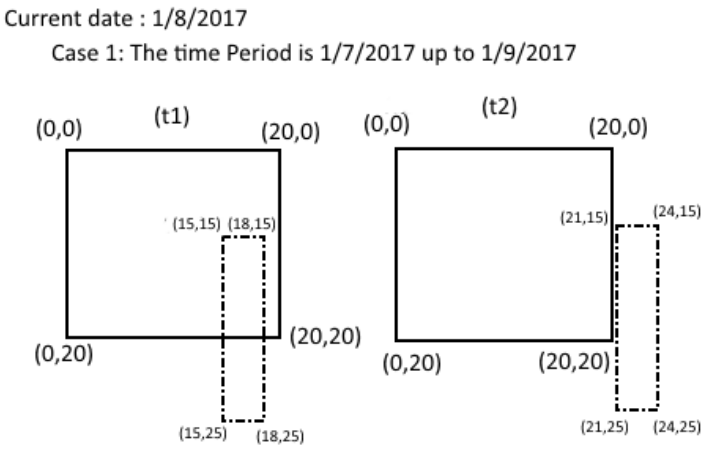


Fig. 2

Case 2: In this case, the time period of not overlap property has expired. That means after the time t1 where we store the second shape, at t2 the function **checkDB** doesn't change anything.

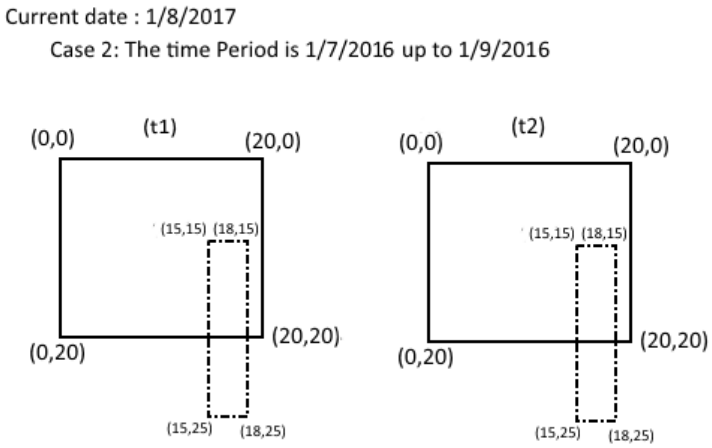


Fig. 3

In order to understand better this example we will introduce a scenario and a solution.

3. Scenario

A virtual space rental room rents space for a specific period of time. Each object in its field has two properties (Integrity constraints):

1) Not **overlapped** by another object.

2) If necessary, the client may have close to his object and **connected** objects.

When the rental period expires, the items lose the 1st property, with the result that other objects can overwhelm them, thus virtually they do not get any room.

If a client then wants to renew the rental for a further period, its objects get the NotOverLap property again, check if they overlap another object or get overlapped by it and moves it to the nearest location.

* Of course, the second property still holds before, so if it is necessary moves all the connected objects together.

3.1 Solution:

a) The objects will be imported from the center, then a check will take place a function **checkDB()**. The ID of every object that is being inserted will be shown to user.

b) The property connected can be given either after the objects have been imported or at the time they are imported.

c) There will be a function **checkDB** which will be activate after every action (move, Open Map, insert Object, extend period, connect objects). It will check for violation of integrity constraints (**Overlap** and **Expired periods)** and it will keep the database consistent.

Additional Comments:

- In order to put every object at the center we add +350 at x axis and +200 at y axis.
- The distance of every step is 4 (int by=4), it used in functions countSteps and resolveOverlap.

3.2 Oracle – Database

The tables we will need:

Polygons

ID	serial	point_X	point_Y

Table 1

A table that holds information about the objects. The data that is needed is:
ID (a unique ID of each object),
serial (a unique number for each point),
point_X (coordinate in X axis), **point_Y** (coordinate in Y axis).

ValidPeriod

ID	timeStart	timeEnd

Table 2

A table that holds information about the valid period of each object.
ID(a unique ID of each object),
timeStart (a timestamp that determines the start of valid time)
timeEnd (a timestamp that determines the end of valid time)

Connected

ID	ConnectedPol

Table 3

A table that holds information about the connected objects.
ID (the id of the current object)
ConnectPol (the id of the object that is connected to the current object)
E.g. If object No.1 is connected to object No.2 it's defined as this: 1|2 and 2|1.

NotOverLap

ID

Table 4

A table that holds information about the objects that have the 1st property is valid.
ID (the id of the objects that holds the 1st property).

3.3 Java

The following diagram shows some basic steps that our program uses.

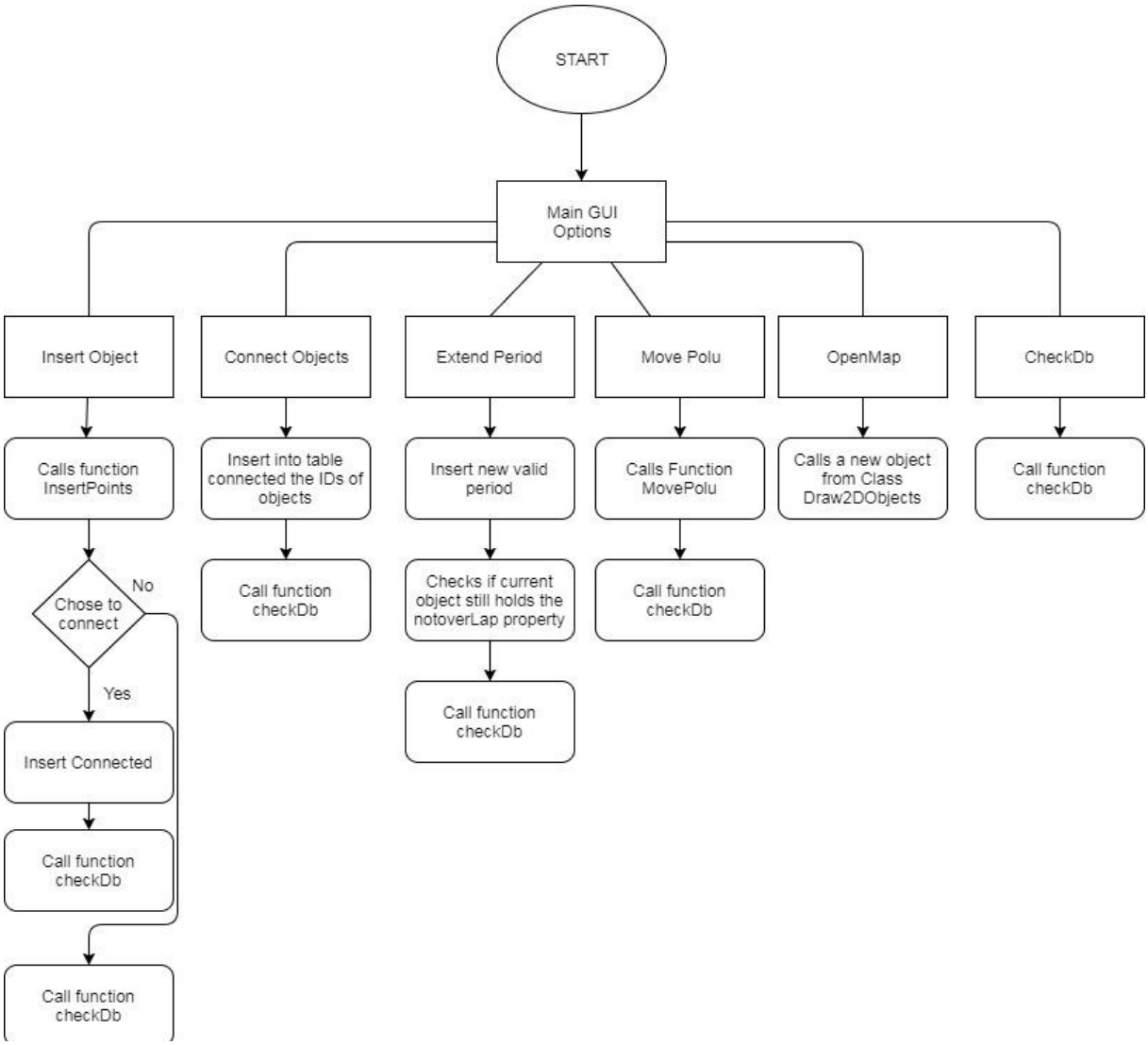


Fig. 4

3.3.1 Main GUI

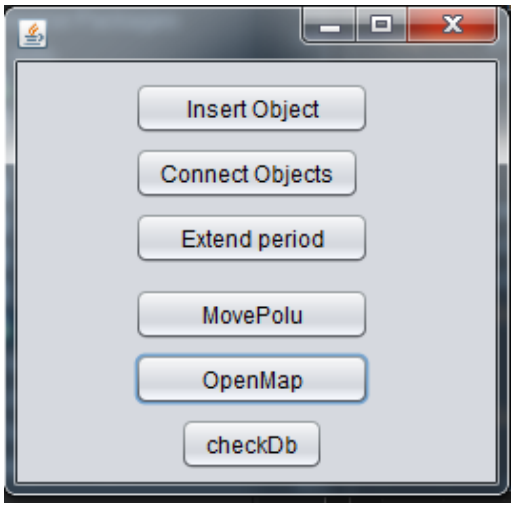
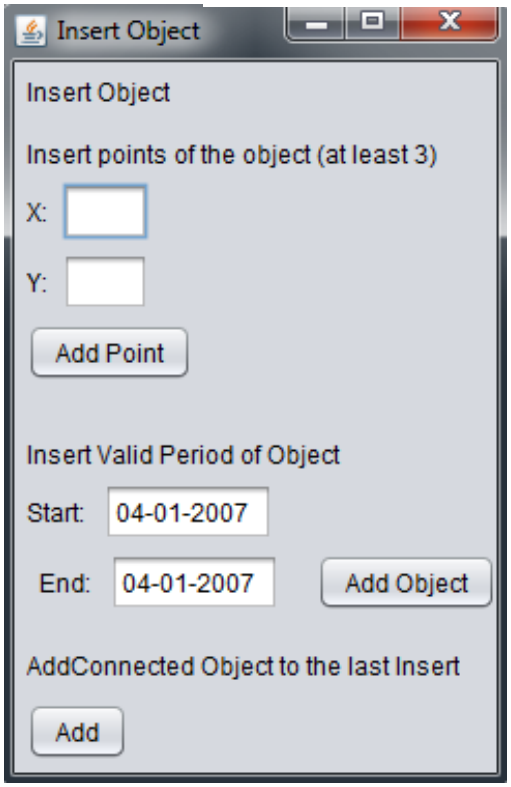


Fig. 5

This is the central panel that a user controls the database. From this panel the user can :

- Insert new objects **(Insert Object)**
- He can connect them (**Connect Objects)**
- He can extent a valid period of an object.(**Extent Period)**
- He can open a map of the objects (**Open Map**)
- He can move objects. (**Move Polu**)
- He can manually check for overlaps and expired periods and keep the database consistent, even if this function run after every action. (**checkDB**)

Fig. 6



3.3.2 Insert Object

In this frame the user can insert new objects. He imports each point of the shape he wants with the button Add point.

Then he adds the valid period of this object and press Add Object. This object now has been inserted in the database.

If the user wants can connect a object to the last he inserted with the button Add.

Every object is added to the table polygons, and its valid period at the table ValidPeriod. Also every object is added to the NotOverLap table so it holds the notOverlap constraint.

3.3.3 Connected Objects

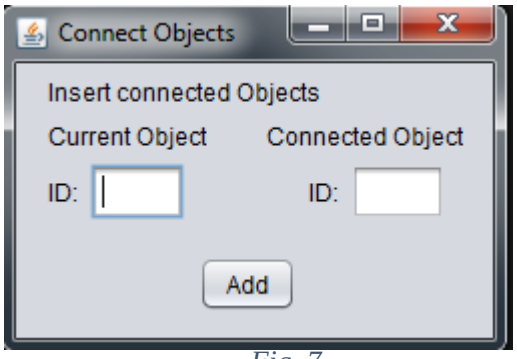


Fig. 7

In this frame the user can connect objects that are already in the database. After he presses add, the IDs of the objects are added to the connected table.

3.3.4 Extend Period

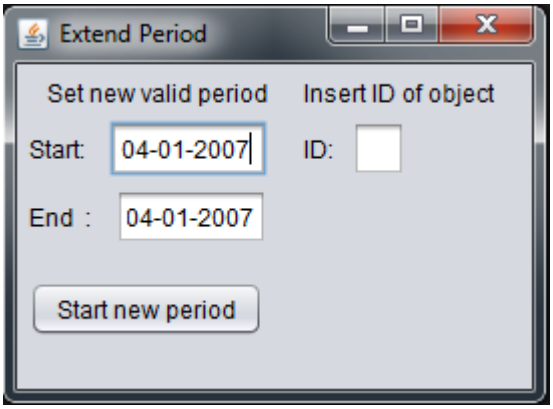


Fig. 8

In this frame the user can extend a valid period of an object. After he changes the period, a check takes place that sees if this object holds the property notOverLap (means its id exists at NotOverLap table) if not, it adds the id to the NotOverLap table.

3.3.5 Move Object

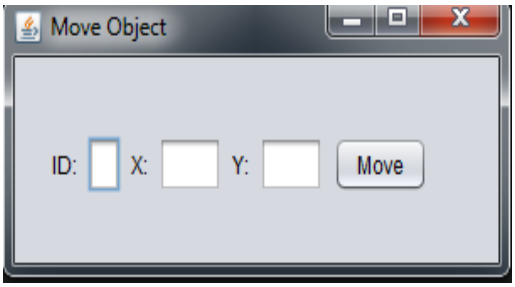


Fig 9

In this frame an object can be moved manually by the user.

3.3.6 CheckDb

Is a function that is called after an action or manually from the central panel (see more at **functions**).

- It checks from the ValidsPeriod table if there are objects with expired periods, if that's true it removes them from the NotOverLap table.
- It checks if there is any overlap, if that's true it calls the function ResolveOverlap (see more at **functions**).

3.3.7 Open Map

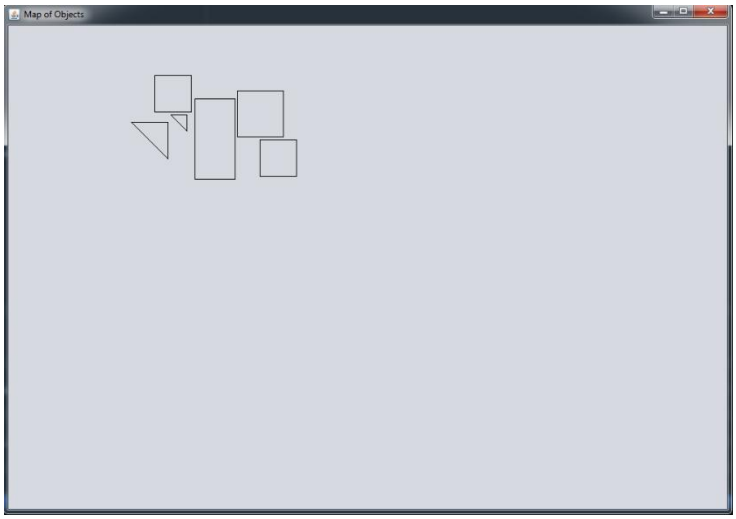


Fig. 10

Size (1000,700) – Java provides a class Canvas, and a function paint which we used to draw the shapes. More for this functions you can see below at **functions**.

4. Functions

4.1) Bounding Box Contains Point

This algorithm is used to check whether there is a point of another polygon in the bounding box of the current polygon. In the case this is true, we can proceed to check if there is an overlap, if this case is not true, there is no point to check if there is an overlap.

```
public int BoundingBoxContainsPoint(int poly1, int poly2) {
      conn = JavaConnectDb.ConnectDb();
      Statement stmt = null;
      Statement stmt2 = null;
      int minX = 0, maxX = 0, minY = 0, maxY = 0, polX = 0, polY = 0;
      String query = "select * from polygons where ID=" + poly1 + " ";
      String query2 = "select * from polygons where ID=" + poly2 + " ";
      int j = 0;
  1.) try {
        stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query);
    1.1) while (rs.next()) {
           if (j == 0) {
             maxX = minX = rs.getInt(3);
             maxY = minY = rs.getInt(4);
             j = 1;
           } else {
             polX = rs.getInt(3);
             polY = rs.getInt(4);
             if (minX > polX) {minX = polX;}
             if (maxX < polX) {maxX = polX; }
             if (minY > polY) {minY = polY; }
             if (maxY < polY) { maxY = polY;}
           }
        }
      } catch (Exception e) {JOptionPane.showMessageDialog(null, e); }
  2.) try {
        int pointX = 0, pointY = 0;
        stmt2 = conn.createStatement();
        ResultSet rs2 = stmt.executeQuery(query2);
        while (rs2.next()) {
          pointX = rs2.getInt(3);
          pointY = rs2.getInt(4);
   2.1) if (pointX >= minX && pointX <= maxX && pointY >= minY && pointY <= maxY) {
                return 1; } }
        stmt2.close();
        Thread.sleep(1);
      } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
      }
      return 0;
  }
```

Parameters:

Poly1: the id of the current polygon

Poly2: the id of the polygon that is inspected if it has any of its point in the current's bounding box.

Point: the serial number of the poly2 points.

Return:

1: if there is a point of poly2 in the bounding box of the current poly1.

0: if the 1 is false.

Steps:

1) In a try-catch we obtain from the database the points of poly1 in order to find the bounding box.

1.1) First we determinate the minimum-maximum points of the poly1, in that way we know the "corners" of x and y coordinate of its bounding box.

2) In a try-catch we obtain from the database the points of poly2 in order to find if there is any of them, inside the bounding box of poly1.

2.1) In this "if" statement we check if each of the points of poly2 is inside of the

bounding box by compare it with the minimum and maximum values of poly1. If its true returns 1, else returns 0.

4.2) Polygon Contains Point

This function is used to check whether there is a point of a polygon, inside another polygon. It uses the algorithm BoundingBoxContainsPoint first to see if there is any point inside the bounding box, if not there is no reason to use this function.

Parameters:

Poly1: The id of the current polygon

Poly2: The id of the second polygon that is inspected if it has its point in the current poly1.

Point: The serial number of the poly2 points.

Return:

1: if poly1 contains the current point.

0: if the 1 is false.

public int PolygonContainsPoint(int poly1, int poly2, int point) {

```
   1.) if (BoundingBoxContainsPoint(poly1, poly2) == 0) {
         return 0; }
      conn = JavaConnectDb.ConnectDb();
      Statement stmt = null;
      Statement stmt2 = null;
      Statement stmt3 = null;
      String query = "select point_x,point_y from polygons where ID=" + poly1 + " ";
      String query2 = "select point_x,point_y from polygons where ID=" + poly2 + " AND serial=" + point + " ";
       int test1 = 0, test2 = 0, pX = 0, pY = 0, hits = 0, lastX = 0, lastY = 0, curX = 0, curY = 0, leftX = 0;
  2.)  try {
         stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(query);
         while (rs.next()) {
            lastX = rs.getInt(1);
            lastY = rs.getInt(2);
         }
         stmt.close();
         Thread.sleep(1);
      } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
}
3.)  try {
         stmt2 = conn.createStatement();
         ResultSet rs2 = stmt2.executeQuery(query2);
         while (rs2.next()) {
            pX = rs2.getInt(1);
            pY = rs2.getInt(2);
         }
         stmt2.close();
         Thread.sleep(1);
      } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
}
         4.) try {
         stmt3 = conn.createStatement();
         ResultSet rs3 = stmt3.executeQuery(query);
         while (rs3.next()) {
            curX = rs3.getInt(1);
            curY = rs3.getInt(2);
            if (curY == lastY) {
               lastX = curX;
               lastY = curY;
            }
          5.) if (curX < lastX) {
               if (pX >= lastX) {
                  lastX = curX;
                  lastY = curY;
               }
               leftX = curX;
            } else {
               if (pX >= curX) {
                  lastX = curX;
                  lastY = curY;
               }
               leftX = lastX;
            }
            if (curY < lastY) {
               if (pY < curY || pY >= lastY)
{
                  lastX = curX;
                  lastY = curY;
               }
```

```
                if (pX < leftX) {
                    hits++;
                    lastX = curX;
                    lastY = curY;
                }
                test1 = pX - curX;
                test2 = pY - curY;
            } else {
                if (pY < lastY || pY >= curY)
{
                    lastX = curX;
                    lastY = curY;
                }
                if (pX < leftX) {
                    hits++;
                    lastX = curX;
                    lastY = curY;
                }
                test1 = pX - lastX;
                test2 = pY - lastY;
            }
            if (((lastY - curY)) != 0) {
                if ((lastX - curX) != 0) {
                    if (test1 < (test2 / (lastY -
curY) * (lastX - curX))) {
                        hits++;
                    }
                }
            }
        }
        stmt3.close();
        Thread.sleep(1);
      } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
      }
    6.) if (hits != 0) {return 1;
     } else {return 0;
         }
  }
```

Steps:

1) Runs the BoundingBoxContainsPoint function to see if there is any point in bounding box, if not returns 0.
2) Gets the last point of poly1 from database.
3) Gets the given point of poly2 from database.
4) Get each point of poly1
5) Checks each coordinate point of poly1 to see if pol1, contains the given point, if that's true, hits increased by 1.
6) If hit!=0 returns 1, else returns 0.

4.3) Polygon Contains Polygon

This function uses the previous function to determinate whether a given polygon contains or it's contained by another polygon. To do that is enough to check only the first point of each polygon.

Parameters:

Poly1: the id of the polygon one.

Poly2: the id of the polygon two.

Returns:

1: if poly1 contains or is contained by poly2

0:if the 1 is false.

```
public int PolygonContainsPolygon(int
poly1, int poly2) {
 1.)  if (PolygonContainsPoint(poly1,
poly2, 0) ==1) {
          return 1;
      }
 2.) if (PolygonContainsPoint(poly2,
poly1, 0) == 1) {
          return 1;
      }
  3.)    return 0;
   }
```

Steps:

1) The function uses the previous function to check if poly1 contains poly2, if that's true returns 1.
2) The function uses the previous function to check if poly2 contains poly1, if that's true returns 1.
3) Otherwise returns 0.

4.4) Edges intersect

This function checks whether two given edges are intersecting or not. The edges are determinate by four points: first edge = point_1, point_2, second second edge = point_3, point_4.

Parameters:

Poly1: the id of the first polygon.

Poly2: the id of the second polygon.

Point1: the first point of poly1 edge's.

Point2: the second point of poly1 edge's.

Point3: the first point of poly2 edge's .

Point4: the second point of poly2 edge's.

Returns:

1: if the two given edges intersect

0: if 1 is false.

```
public int edgesIntersect(int poly1, int point1, int point2, int poly2, int point3, int point4) {
    conn = JavaConnectDb.ConnectDb();
    int changed = 0;
    int x1 = 0, y1 = 0, x2 = 0, y2 = 0, x3 = 0, y3 = 0, x4 = 0, y4 = 0;
    int left1X = 0, left1Y = 0, right1X = 0, right1Y = 0;
    int left2X = 0, left2Y = 0, right2X = 0, right2Y = 0;
    int tmpY = 0, tmpX = 0;

    Statement stmt = null;
    Statement stmt2 = null;
    Statement stmt3 = null;
    Statement stmt4 = null;
    String query = "select point_x,point_y from polygons where ID=" + poly1 + " AND serial=" + point1 + " ";
    String query2 = "select point_x,point_y from polygons where ID=" + poly1 + " AND serial=" + point2 + " ";
    String query3 = "select point_x,point_y from polygons where ID=" + poly2 + "  AND serial=" + point3 + " ";
    String query4 = "select point_x,point_y from polygons where ID=" + poly2 + " AND serial=" + point4 + " ";
1.)     try {
        stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            x1 = rs.getInt(1);
            y1 = rs.getInt(2);
        }
        stmt.close();
    } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
    }
    try {
        stmt2 = conn.createStatement();
        ResultSet rs2 = stmt2.executeQuery(query2);
        while (rs2.next()) {
            x2 = rs2.getInt(1);
            y2 = rs2.getInt(2);
        }
        stmt2.close();
    } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
    }
    try {
        stmt3 = conn.createStatement();
        ResultSet rs3 = stmt3.executeQuery(query3);
        while (rs3.next()) {
            x3 = rs3.getInt(1);
            y3 = rs3.getInt(2);
        }
        stmt3.close();
    } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
    }
    try {
        stmt4 = conn.createStatement();
        ResultSet rs4 = stmt4.executeQuery(query4);
        while (rs4.next()) {
            x4 = rs4.getInt(1);
            y4 = rs4.getInt(2);
        }
        stmt4.close();
    } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
    }
2.)     if (x1 <= x2) {
        left1X = x1;
        left1Y = y1;
        right1X = x2;
        right1Y = y2;
    } else {
```

```
            left1X = x2;
            left1Y = y2;
            right1X = x1;
            right1Y = y1;
        }
3.)     if (x3 <= x4) {
            left2X = x3;
            left2Y = y3;
            right2X = x4;
            right2Y = y4;
        } else {
            left2X = x4;
            left2Y = y4;
            right2X = x3;
            right2Y = y3;
        }
4.)     if (left1X > left2X) {
            tmpX = left1X;
            left1X = left2X;
            left2X = tmpX;
            tmpX = right1X;
            right1X = right2X;
            right2X = tmpX;
            tmpY = left1Y;
            left1Y = left2Y;
            left2Y = tmpY;
            tmpY = right1Y;
            right1Y = right2Y;
            right2Y = tmpY;
        }
    5.) if (left1X < left2X && right1X >
left2X) {
        if (left1Y <= left2Y && right1Y >
left2Y) {
            return 1;
        }
        if (left1Y >= left2Y && right1Y <
left2Y) {
            return 1;
        }
    }
    return 0;
  }
```

Steps:

1) In the first step the function gets the coordinates of the points from the database.
2) Finds the left point of the first edge
3) Finds the left point of the second edge
4) Checks which of the points is most left, if it's the second, then the variables swaps.
5) Checks if the two edges cross each other, in that case returns 1, else returns 0.

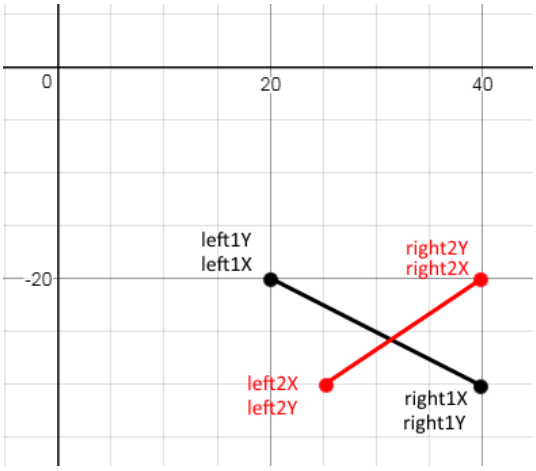


Fig. 11

In this image we can understand better how the fifth step works. First we check if the **left2X** is between the points of first edge (black), then we check if the **left1Y** <= **left2Y && right1Y** > **left2Y**. Then checks if the **left1Y** >= **left2Y && right1Y** < **left2Y**, which in this case is true.

4.5) Polygon Intersect Polygon

This function decide if two given polygons have edges that intersects. It uses the previews function for each edge of the first polygon with each edge of the second polygon.

Parameters:

Poly1: The id of the first polygon

Poly2: The id of the second polygon

Return:

1: if there is two edge of each polygon that intersect.

0: if 1 is false.

```
public int PolygonIntersectsPolygon(int
poly1, int poly2) {
    conn = JavaConnectDb.ConnectDb();
    Statement stmt = null;
    Statement stmt2 = null;
    Statement stmt3 = null;
    Statement stmt4 = null;
```

```
      String query = "select serial from
polygons where ID=" + poly1 + " ";
      String query3 = "select serial from
polygons where ID=" + poly2 + " ";

   int point1 = 0, point2 = 0, point3 = 0,
point4 =0;
1.)
      try {
         stmt = conn.createStatement();
         ResultSet rs =
stmt.executeQuery(query);
         while (rs.next()) {
            point1 = rs.getInt(1);
         }
         stmt.close();
         Thread.sleep(1);
      } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
      }
      try {
         stmt2 = conn.createStatement();
         ResultSet rs2 =
stmt2.executeQuery(query);
         while (rs2.next()) {
            point2 = rs2.getInt(1);
            try {
               stmt3 =
conn.createStatement();
               ResultSet rs3 =
stmt3.executeQuery(query3);
               while (rs3.next()) {
                  point3 = rs3.getInt(1);
               }
               stmt3.close();
               Thread.sleep(1);
            } catch (Exception e) {

JOptionPane.showMessageDialog(null,e);
            }
            try {
               stmt4 =
conn.createStatement();
               ResultSet rs4 =
stmt4.executeQuery(query3);
               while (rs4.next()) {
                  point4 = rs4.getInt(1);
         2.)   if (edgesIntersect(poly1,
point1, point2, poly2, point3, point4) ==
1) {
                     return 1;
                  }
            3.)      point3 = point4;
               }
               stmt4.close();
               Thread.sleep(1);
            } catch (Exception e) {

JOptionPane.showMessageDialog(null,e);
               }
         4.)   point1 = point2;
         }
         stmt2.close();
         Thread.sleep(1);
      } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
      }
     5.) return 0;
   }
```

Steps:

1) In the first step we take from the database each point.
2) In the second step, while we take the last point we use the function "edges intersect", to see if any of given edges (from points) are intersecting.
3) Then we change points to the next edge
4) Again change points to check the next edge
5) End of function, if there wasn't any intersect returns 0.

4.6) Polygon Overlaps polygon

This function uses the previews function and the function: "polygon Contains polygon" to check two given polygons if there is an overlap.

Parameters:

Poly1: The id of the first polygon.

Poly2: The id of the second polygon.

Returns:

1: if the given polygons overlap.

0: if 1 is false.

```
public int PolygonsOverlapsPolygon(int
poly1, int poly2) {
1.) if (PolygonIntersectsPolygon(poly1,
poly2)==1) {
      return 1;
```

```
        }
    2.) if (PolygonContainsPolygon(poly1,
    poly2) == 1) {
            return 1;
        }
    3.)     return 0;
      }
```

Steps:

1) In the first step it uses the function "polygon intersect polygon" to check if any edges are intersect. If that's true returns 1.
2) In the second it uses the function "polygon contains polygon" to check if the second polygon contains the first. If that's true returns 1.
3) If neither 1 or 2 are true, the function returns 0.

4.7) Move Polygon

This function moves a polygon. It takes the id, the distance of X coordinate and Y coordinate and move each point of the current polygon to this direction. It also, moves a connected polygon to the current polygon, to the same direction. It is also use the Function: "Move only".

Parameters:

Id: The id of the given polygon.
moveX: The distance in the X axis
moveY: The diastance in the Y axis

```
public void MovePolygon(int id, int
moveX, int moveY) {
        conn = JavaConnectDb.ConnectDb();
        Statement stmt = null;
        String query = "select connectpol
from connected where ID=" + id + " ";
        int curPol = 0;
        MoveOnly(id, moveX, moveY);
        try {
            stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(query);
    1.)     while (rs.next()) {
              curPol = rs.getInt(1);
    2.)       MoveOnly(curPol, moveX,
moveY);
            }
            stmt.close();
        } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
        }
    }
```

Steps:

1) In the first step moves the current polygon to the given direction.
2) In the second step it takes from the table connected each polygon that's is connected to the given.
3) For each of connected polygon it uses the "move only" to move it to the same direction.

4.8) Move only

It's a simple assistant the previews function that updates the given polygon's points in the database.

Parameters:

Id: The id of the given polygon.
moveX: The distance in the X axis
moveY: The diastance in the Y axis

```
public void MoveOnly(int id, int moveX,
int moveY) {
        conn = JavaConnectDb.ConnectDb();
        try {
    1.)     String sql = "update polygons set
POINT_X
=POINT_X+?,POINT_Y=POINT_Y+?
where  id=?";
            pst = (OraclePreparedStatement)
conn.prepareStatement(sql);
            pst.setInt(1, moveX);
            pst.setInt(2, moveY);
            pst.setInt(3, id);
            pst.executeUpdate();
            pst.close();
            Thread.sleep(1);
        } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
        }
    }
```

Steps:

1) In the first and the only step, the function uses an sql statement that updates each point X and Y. It takes the previews values and adds the given moveX and moveY.

4.9) Count steps to move

This function is used to count the distance (in Steps, each step=4) that's is needed to move a polygon to avoid overlap with another polygon. It uses the id of the current polygon and the direction and moves the polygon until there is no overlap. It may be the most important function in this project.

Parameters:

Id: The id of the given polygon.

moveX: the direction in X axis (-1,0,1)

moveY: The direction in Y axis(-1,0,1)

```
public int countStepsToMove(int id, int moveX, int moveY) {
    int steps = 0, curPol = 0;
    int size = 0, changed = 0;
    int by = 4;
    boolean sec = false;
    conn = JavaConnectDb.ConnectDb();
    Statement stmt = null;
    Statement stmt2 = null;
    String query = "select id from notOverlap";

    String query2 = "select connectpol from connected where ID=" + id + "";
    ArrayList<Integer> ConnectedPols = new ArrayList();
    try {
      stmt2 = conn.createStatement();
      ResultSet rs2 = stmt2.executeQuery(query2);
      while (rs2.next()) {
 1) ConnectedPols.add(rs2.getInt(1));
      }
      stmt2.close();
    } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
    }
    steps++;
    MovePolygon(id, moveX * steps, moveY * steps);
    try {
      stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery(query);
2)while (rs.next()) {
        curPol = rs.getInt(1);
2.1) if (id != curPol) {
2.2) if (ConnectedPols.size() != 0) {
          for (int i = 0; i<ConnectedPols.size(); i++) {
            if (ConnectedPols.get(i) != curPol) {
2.3) while ((PolygonsOverlapsPolygon(id, curPol) != 0) || (PolygonsOverlapsPolygon(ConnectedPols.get(i), curPol) != 0)) {
    if (moveX == 0) {
      if (moveY < 0) {
        MovePolygon(id, 0, -by);}
      if (moveY > 0) {
        MovePolygon(id, 0, by);}
    }
    if (moveY == 0) {
      if (moveX < 0) {
        MovePolygon(id, -by, 0);
      }
      if (moveX > 0) {
        MovePolygon(id, by, 0); }
    }
if (moveY != 0 && moveX != 0) {
      if (moveX < 0 && moveY > 0) {
        MovePolygon(id, -by, by);
      }
      if (moveX < 0 && moveY < 0) {
        MovePolygon(id, -by, -by);
      }
  if (moveX > 0 && moveY < 0) {
    MovePolygon(id, by, -by);
      }
  if (moveX > 0 && moveY > 0) {
      MovePolygon(id, by, by);
      }
    }
    steps++;
  }
 }
}
} else {
2.4) while ((PolygonsOverlapsPolygon(id, curPol) != 0)) {
    if (moveX == 0) {
```

```
            if (moveY < 0) {
                MovePolygon(id, 0, -by);
                    }
                 if (moveY > 0) {
             MovePolygon(id, 0, by);
                    }
                  }
                  if (moveY == 0) {
                     if (moveX < 0) {
                MovePolygon(id, -by, 0);
                    }
                     if (moveX > 0) {
                MovePolygon(id, by, 0);
                    }
                  }
        if (moveY != 0 && moveX != 0) {
        if (moveX < 0 && moveY > 0) {
           MovePolygon(id, -by, by);
                    }
           if (moveX < 0 && moveY < 0)
{
            MovePolygon(id, -by, -by);
                    }
         if (moveX > 0 && moveY < 0) {
           MovePolygon(id, by, -by);
                    }
         if (moveX > 0 && moveY > 0) {
                MovePolygon(id, by, by);
                    }
                  }
                  steps++;
                }
              }
           }
         }
3)    if (moveX == 0 && moveY == -1) {
           MovePolygon(id, 0, -3);
         }
        if (moveX == 1 && moveY == -
1){
           MovePolygon(id, 3, -3);
         }
        if (moveX == 1 && moveY == 0) {
           MovePolygon(id, 3, 0);
         }
        if (moveX == 1 && moveY == 1) {
           MovePolygon(id, 3, 3);
         }
         if (moveX == 0 && moveY == 1)
{
           MovePolygon(id, 0, 3);
         }
        if (moveX == -1 && moveY == 1)
{
           MovePolygon(id, -3, 3);
        }
        if (moveX == -1 && moveY == 0)
{
           MovePolygon(id, -3, 0);
        }
        if (moveX == -1 && moveY == -
1) {
           MovePolygon(id, -3, -3);
        }
        stmt.close();
      } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
      }
4) for (int i = 0; i < steps; i++) {
        if (moveX == 0) {
           if (moveY < 0) {
              MovePolygon(id, 0, by);
           }
           if (moveY > 0) {
              MovePolygon(id, 0, -by);
           }
        }
        if (moveY == 0) {
           if (moveX < 0) {
              MovePolygon(id, by, 0);
           }
           if (moveX > 0) {
              MovePolygon(id, -by, 0);
           }
        }
        if (moveY != 0 && moveX != 0) {
           if (moveX < 0 && moveY > 0)
{
              MovePolygon(id, by, -by);
           }
           if (moveX < 0 && moveY < 0)
{
              MovePolygon(id, by, by);
           }
           if (moveX > 0 && moveY < 0)
{
              MovePolygon(id, -by, by);
           }
           if (moveX > 0 && moveY > 0)
{
              MovePolygon(id, -by, -by);
```

```
            }
          }
        }
        return steps;
    }
```

Steps:

1) First we make an array list (ConnectedPols) with the IDs of the polygons that are connected to the current polygon. While we move it and check for over lap, we also check the connected polygon.

2) Then we check each polygon if over laps with the current and it's connected polygons.

2.1 Pass if current polygon id = id (from table polygons).

2.2 If there are not connected polygons (ConnectedPol=0) then function only checks the current polygon for overlap.

2.3 If there are connected polygons, we check each of them for over lap with any other polygon.

2.4 Function only checks the current polygon for overlap.

3) Because each time polygon moves by 4, when it needs to move back, there is a gap of 3 distance, so we need to cover it up that, or the "count to steps" will be not accurate. This problem is explained below with an example of a point.

Direction	X	Y
X=0 Y= 0	200	111
X=0 Y=-1	200	114
X=1 Y=-1	197	117
X=1 Y=0	194	117
X=1 Y=1	191	114
X=0 Y=1	191	111
X=-1 Y=1	194	108
X=-1 Y=0	197	108
X=-1 Y=-1	200	111

Table 5

4) After the correction the function moves back the polygon the returns the steps.

4.10) Resolve Over lap

This function uses the "count steps to move" function and find's the smallest distance to move an polygon so that doesn't over laps another polygon.

Parameters:

Id: The id of the polygon that over laps another polygon

```
public void ResolveOverLap(int id)
throws InterruptedException {
    int steps = 0, minSteps, bestX, bestY;
    int by = 4;
  1) minSteps = countStepsToMove(id,
0, -1);
    bestX = 0;
    bestY = -1;
    System.out.println("0 -1 : " +
minSteps);
    steps = countStepsToMove(id, 1, -1);
  2) if (steps < minSteps) {
       minSteps = steps;
       bestX = 1;
       bestY = -1;
    }
    steps = countStepsToMove(id, 1, 0);
    if (steps < minSteps) {
       minSteps = steps;
       bestX = 1;
       bestY = 0;
    }
    steps = countStepsToMove(id, 1, 1);
    if (steps < minSteps) {
       minSteps = steps;
       bestX = 1;
       bestY = 1;
    }
    steps = countStepsToMove(id, 0, 1);
    if (steps < minSteps) {
       minSteps = steps;
       bestX = 0;
       bestY = 1;
    }
    steps = countStepsToMove(id, -1, 1);
    if (steps < minSteps) {
       minSteps = steps;
       bestX = -1;
       bestY = 1;
    }
```

```
steps = countStepsToMove(id, -1, 0);
if (steps < minSteps) {
   minSteps = steps;
   bestX = -1;
   bestY = 0;
}
steps = countStepsToMove(id, -1, -1);
if (steps < minSteps) {
   minSteps = steps;
   bestX = -1;
   bestY = -1;
}
3) for (int i = 0; i < minSteps ; i++) {
   if (bestX == 0) {
      if (bestY < 0) {
         MovePolygon(id, 0, -by);
      }
      if (bestY > 0) {
         MovePolygon(id, 0, by);
      }

   }
   if (bestY == 0) {
      if (bestX < 0) {
         MovePolygon(id, -by, 0);
      }
      if (bestX > 0) {
         MovePolygon(id, by, 0);
      }
   }
   if (bestY != 0 && bestX != 0) {
      if (bestX < 0 && bestY > 0) {
         MovePolygon(id, -by, by);
      }
      if (bestX < 0 && bestY < 0) {
         MovePolygon(id, -by, -by);
      }
      if (bestX > 0 && bestY < 0) {
         MovePolygon(id, by, -by);
      }
      if (bestX > 0 && bestY > 0) {
         MovePolygon(id, by, by);
      }
   }
}
```

Steps:

1) At the first step the function defines as the shortest path the direction x=0, y=-1 and a distance that takes from "count steps to move".

2) After that, the function try all the other directons (1,-1)(1,0)(1,1)(0,1)(-1,1)(-1,0)(-1,-1) to find the shortest distance that is needed to resolve the overlap, with the help of "count steps to move" function.

3) At the and, it moves the polygon in the shortest path. Step by step, the polygon moves by 4(int by = 4) each time for "minSteps" times.

4.11) Check DB

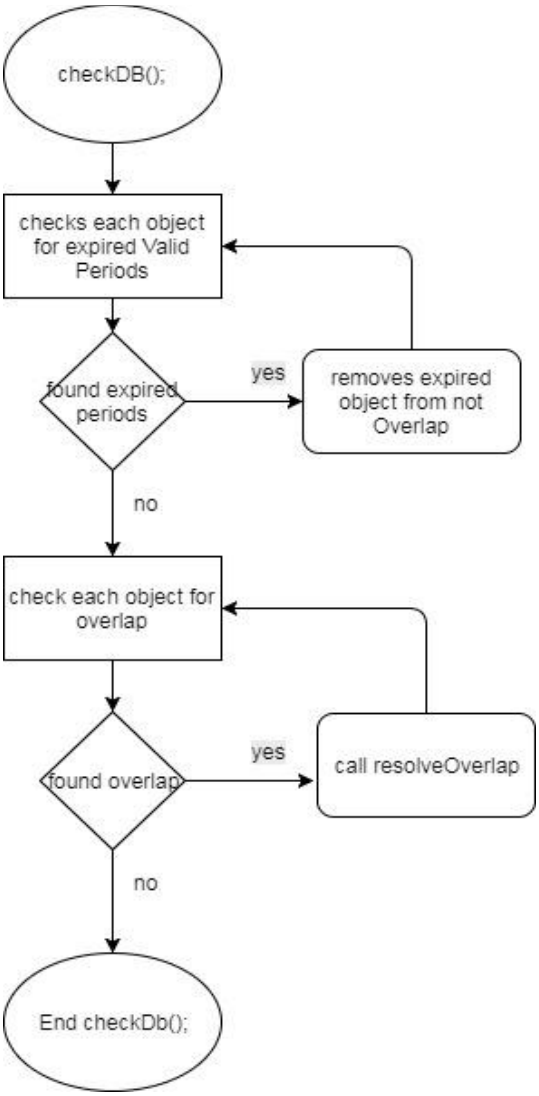


Fig. 12

This function is used after every action to check every integrity constraint and keep the database intact. In order to do that, first check if the periods are valid, then check for overlaps, if there is overlap, it uses the previous function resolve overlap. It doesn't takes parameters because it uses

all the data from the database. In the end shows a window with the number of resolves that took place.

```
public void checkDB() throws InterruptedException {
1)  ArrayList<Integer> NotOverLap = new ArrayList();
        ArrayList<Integer> polugons = new ArrayList();
        conn = JavaConnectDb.ConnectDb();
        Statement stmt = null;
        Statement stmt2 = null;
        Statement stmt3 = null;
2)  //-------Checks Valids Periods ----------
            String query3 = "select timestart,timeend,id from validperiod";
        //Get id of all NotOverLap
        try {
            stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(query3);
            while (rs.next()) {

                String str_date = rs.getString(1).substring(0, 10);
                String str_end = rs.getString(2).substring(0, 10);
                //From string to date
                DateFormat formatter;
                formatter = new SimpleDateFormat("yyyy-MM-dd");
                java.util.Date date = formatter.parse(str_date);
                java.sql.Date sqlDateStart = new java.sql.Date(date.getTime());
                //From string to date
                DateFormat formatter2;
                formatter2 = new SimpleDateFormat("yyyy-MM-dd");
                java.util.Date date2 = formatter2.parse(str_end);
                java.sql.Date sqlDateEnd = new java.sql.Date(date2.getTime());
                //Create LoacalDay
                LocalDate todayDate = LocalDate.now();
                LocalDate startDate = date.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
                LocalDate endDate = date2.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();

                if (todayDate.isAfter(startDate) && todayDate.isBefore(endDate)) {
                                } else {
                    removeOverLap(rs.getInt(3));
                }
            }
            stmt.close();

        } catch (Exception e) {

            JOptionPane.showMessageDialog(null, e);
        }

3)  //-------checks for overlap---
        String query = "select ID from notoverlap";
        String query2 = "select ID from polygons";
        //Get id of all NotOverLap
        try {
            stmt3 = conn.createStatement();
            ResultSet rs3 = stmt3.executeQuery(query);
            while (rs3.next()) {
                NotOverLap.add(rs3.getInt(1));
            }
            stmt3.close();

        } catch (Exception e) {

            JOptionPane.showMessageDialog(null, e);
        }
        //Get id of all polugons
        try {
            stmt2 = conn.createStatement();
            ResultSet rs2 = stmt2.executeQuery(query2);
            while (rs2.next()) {
                polugons.add(rs2.getInt(1));
            }
            stmt2.close();

        } catch (Exception e) {

            JOptionPane.showMessageDialog(null, e);
        }
        //Get 1 ID of each polu
```

```
        Set<Integer> hs = new HashSet<>();
        hs.addAll(polugons);
        polugons.clear();
        polugons.addAll(hs);

        //Get 1 ID of NotOverlap polu
        Set<Integer> hs2 = new HashSet<>();
        hs2.addAll(NotOverLap);
        NotOverLap.clear();
        NotOverLap.addAll(hs2);

        int resolves = 0;
        Draw2DObjects app2 = new
Draw2DObjects();
        //Ckecks
        for (int i = 0; i < NotOverLap.size();
i++) {
            for (int j = 0; j < polugons.size();
j++) {
                if (polugons.get(j) !=
NotOverLap.get(i) &&
app2.BoundingBoxContainsPoint(polugon
s.get(j), NotOverLap.get(i)) == 1) {
                    if (polugons.get(j) !=
NotOverLap.get(i) &&
app2.PolygonsOverlapsPolygon(polugons.
get(j), NotOverLap.get(i)) == 1) {

app2.ResolveOverLap(polugons.get(j));
                        resolves++;
                    }
                }
            }
        }

JOptionPane.showMessageDialog(this,
"Finish with :" + resolves + " resolves",
"Message",
JOptionPane.INFORMATION_MESSAG
E);
    }
```

Steps:

1) First we define two arraylists, one with the polygons that holds the not overlap property, and one with all polygons. Also we establish a connection with database.
2) Then we extract from the table **Valid Periods** the periods that is true the not overlap property. If we found a period that has expire we remove with the function **RemovOverLap** the current polygon from the **NotOverLap** table. It's important to do this check first, and then check for overlaps, because we need to keep the **Valid Periods** table updated.
3) After that we can now check for overlaps. We extract from the database, all the polygons, and the polygons that holds the not overlap property and check each of them for overlap. At the begging there is a check so the polygons we compare are not the same, then we use the BoundingBoxContainsPoint function, if it is true, we also use the PolygonsOverlapsPolygon if this is also true, means there is overlap between the current two polygons and needs the resolveOverlap function.

4.12) RemoveOverLap

A simple function used by the previous one to remove from the NotOverLap table, polygons that have expired in the period.

Parameters:

Id : the id of the polygon that has expired and needs to be removed.

```
void removeOverLap(int id) throws
SQLException {
        conn = JavaConnectDb.ConnectDb();
1)    String update2 = "Delete  from
NOTOVERLAP where ID= ?";
        try {
            pst = (OraclePreparedStatement)
conn.prepareStatement(update2);
            pst.setInt(1, id);
            pst.execute();
            pst.close();
        } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
        }
    }
```

Steps:

1) We use a query that deletes from the notOverlap table the given ID

4.13) getPoint

A simple function that extracts from the database the points of the objects. It gets as a parameter a char which specify if it is an X or a Y point. It returns a table with

the coordinates of all X or Y. We use this function to draw the shapes.

Parameters:
Id: The id of the current polygon.

Point: A char that defines if it's a X or Y point

```
public int[] getPoint(int id, char point) {
    conn = JavaConnectDb.ConnectDb();
    int[] x = null;
    Statement stmt = null;
    Statement stmt2 = null;
    String query = "select point_" + point
+ " from polygons where ID=" + id + " ";
    int i = 0;
    int count = 0;
    try {
      stmt2 = conn.createStatement();
      ResultSet rs2 =
stmt2.executeQuery(query);
  1)  while (rs2.next()) {
        count++;
      }
      stmt2.close();
      Thread.sleep(1);
    } catch (Exception e) {

JOptionPane.showMessageDialog(null, e);
    }
    x = new int[count];
    for (int j = 0; j < count; j++) {
      String query2 = "select point_" +
point + " from polygons where ID=" + id +
" AND serial =" + j + " ";
      try {
        stmt = conn.createStatement();
        ResultSet rs =
stmt.executeQuery(query2);
        while (rs.next()) {
2)            x[i] = rs.getInt(1);
          i++;
        }
        stmt.close();
        Thread.sleep(1);
      } catch (Exception e) {
JOptionPane.showMessageDialog(null, e);
      }
    }
3)  return x;
  }
```

Steps:

1) In the first step we count the points of the current polygon.
2) In the second step we add each X or Y coordinate in a table
3) In the last step we return the table.

4.14) countPolygons

A simple function that finds all the objects that exist in the database. It uses a simple find max algorithm because the id's of the objects are auto increment.

```
public int countPolygons() {
    conn = JavaConnectDb.ConnectDb();
    Statement stmt = null;
    int curr = 0;
    int max = 0;
    String query = "select ID from
polygons";
    try {
      stmt = conn.createStatement();
      ResultSet rs =
stmt.executeQuery(query);
      while (rs.next()) {
        curr = rs.getInt(1);
  1)      if (curr > max) {
          max = curr;
        }
      }
      stmt.close();
      Thread.sleep(1);
    } catch (Exception e) {
JOptionPane.showMessageDialog(null, e);
    }
    return max;
  }
```

Steps:

1) We check each id from the table polygons and we find the biggest one. That's the number of the polygons.

4.15) Class MyCanvas and Paint

In order to draw the polygons, we used a function that is provide from java for 2D graphics. This function takes a shape (polygon) and connects each point this shape have.

First we use an array list to add all the polygons:

```
for (int i = 1; i <= countPolygons(); i++) {

 k.add(new Polygon(getPoint(i, 'X'),
getPoint(i, 'Y'), getPoint(i, 'Y').length));

      }
```

As we see the class shape provides a constructor for the shape polygon. It gets as parameters: a table with all X coordinates, a table with all Y coordinates and the number of the coordinates.

Then we use the function paint from the class MyCanvas that extends **Canvas**.

```
class MyCanvas extends Canvas {
public void paint(Graphics graphics) {
Graphics2D g = (Graphics2D) graphics;
      for (int i = 0; i < k.size(); ++i) {
g.draw(k.get(i));
                }
            }
         }
```

As we can see it takes each shape from the k arraylist and draws it.

5. Complexity and Evaluation results

5.1) Complexity

Functions	**Complexity**
BoundingBoxContains	O(n)
polygonContainsPoint	O(n)
polygonContainsPolygon	O(n)
edgesIntersect	O(n)
polygonIntersectsPolygon	$2n^2+n = O(n^2)$
polygonOverlapsPolygon	$2n^2+n+n = O(n^2)$
MovePolygon	O(n)
moveOnly	O(1)
countStepsToMove	$O(n^4)$
resolveOverlap	$O(n^4)$
checkDb	$O(n^6)$

Table 6

Above we chose to describe the complexity of most important functions

5.1.1) The complexity of countSteps:

16 while()
24 Move polygon(n)
28 while(){
32 for(){
34 while(PolygonsOverlapsPolygon($2n^2+n$)+ PolygonsOverlapsPolygon($2n^2+n$)){
Move polygon(n)
}
}
67 while(PolygonsOverlapsPolygon){
70 Move polygon(n)
}
131 for()

More specific, at row 16 and 24 we have $n*n=n^2= O(n^2)$
At row 28-70 we have $n*(n*(2n^2+n)+(2n^2+n)) = 2n^4+n^3+2n^3+n^2= n^4+ n^3 =$ **$O(n^4)$**
At row 131 we have O(n). So the complexity of countsteps is **$O(n^4)$.**

5.1.2) The complexity of ResolveOverlap:

7 * CountSteps(n^4)
50 for(){
movePolygon
The function countSteps() is called seven times, so we have $7* n^4+n^2 =$ **$O(n^4)$.**

5.1.3) The complexity of CheckDb:

14 while()
51 while()
63 while()
86 for(){
87 for(){
88 BoundingBoxContainsPoint (n)
89 PolygonsOverlapsPolygon $(2n^2+n)$
90 ResolveOverLap (n^4)

More specific, we have $n+n+n*(n*(n+2n^2+n+n^4)) = 4*n+n*(2n^2+2n^3+n^5)$
$= 4*n+ 2n^3+2n^4+ n^6 = n^6 =$ **$O(n^6)$**

5.2) Evaluation results

In order to have a general view of the tool which we presented we will present some tests and results we found. These tests have been made in a PC with AMD Phenom II X4 965 (3.4 GHz) CPU and 4 GB DD3 RAM. These test examines the different effects of each parameter and even the different effects when combined parameters are changed. Parameters as number of points, range between points, connected objects etc.

5.2.1) Simple tests

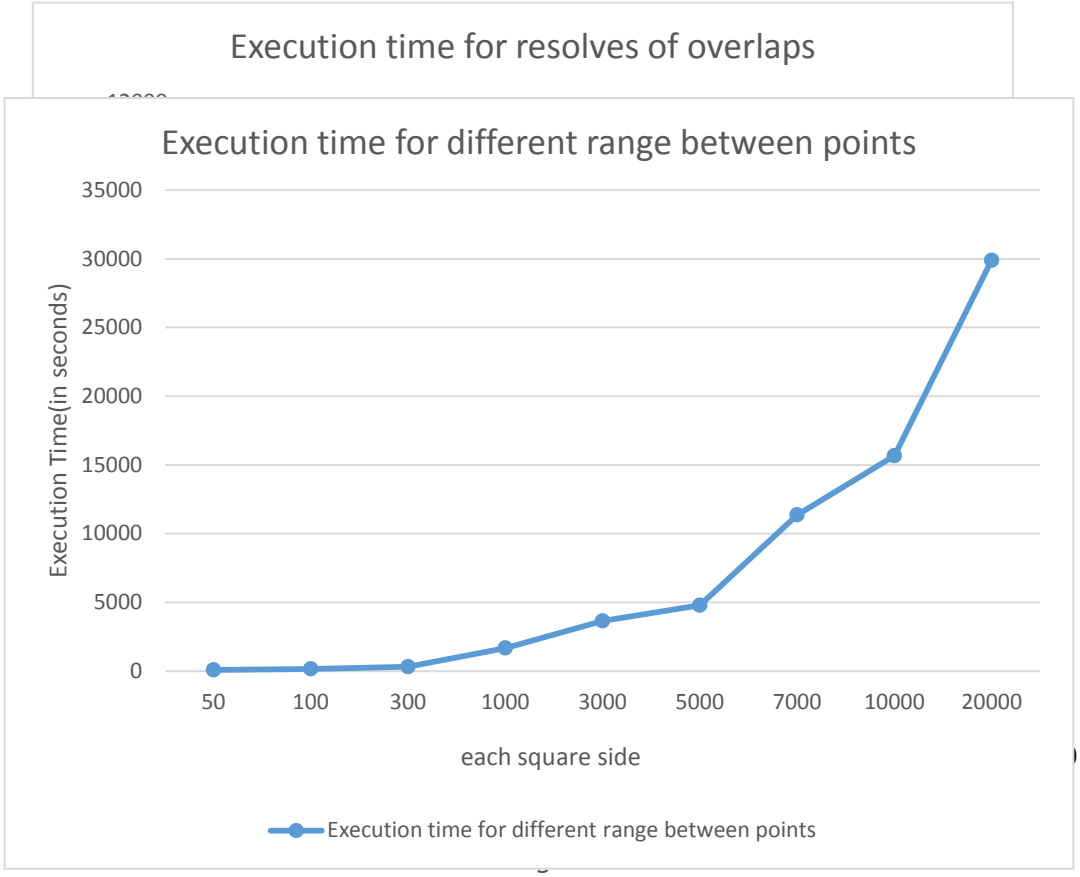
Indicatively we run some simple tests where we increase each time resolves of overlaps, range of polygons, connected polygons. More specific:

Test1

We made a 50*50 square, and each time we add another one on top of that. At the beginning there was one overlap, so one resolve was needed, the second one on the first. Each time we add another one on top, so the second time was two objects on top of the first so two resolves was needed.

Test2

In the second test we will examine a paradigm there is one resolve for overlap each



time, but different range between the points. This paradigm also uses squares as test objects.

Test 3

In the third test we will examine a paradigm where there is one resolve overlap, squares 100*100, and each time we increase the connected objects to the object we move.

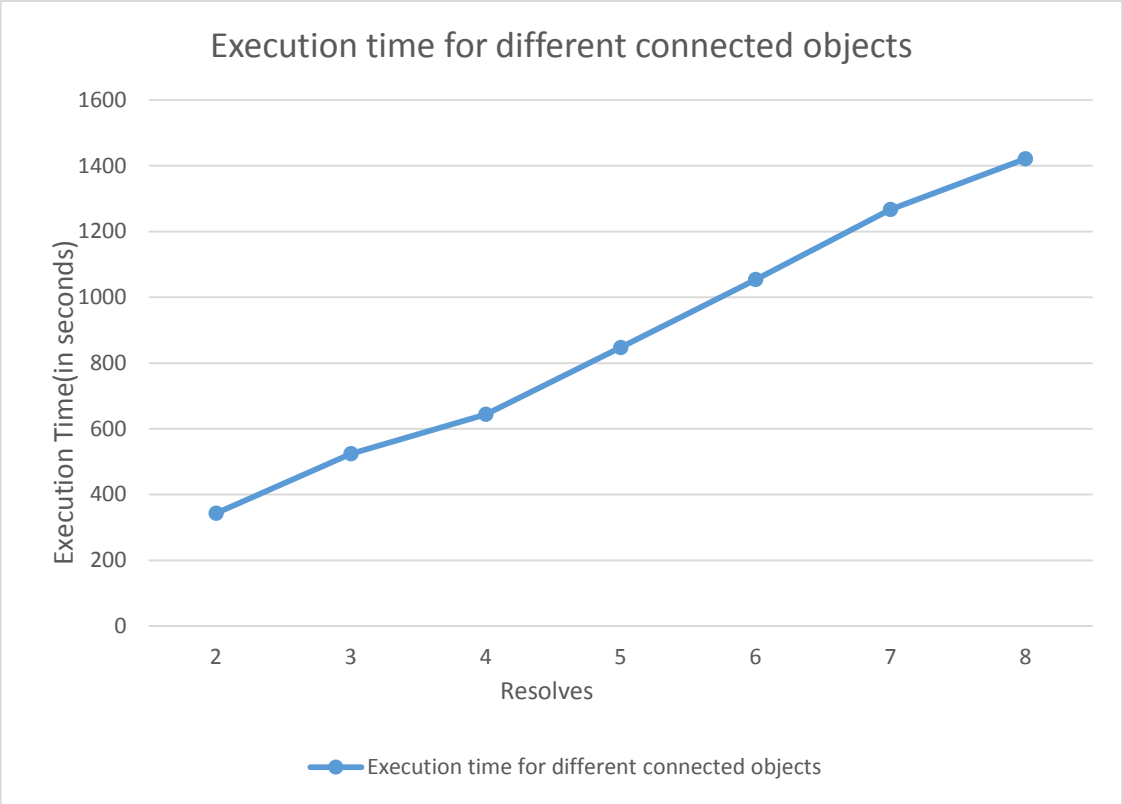


Diagram 3

5.2.2) Complex tests

In order to make better conclusions, we run some test that are more complex, where each time we increase two parameters, as **: range between points and connected objects**(1), **number of points and connected objects**(2), **range between points and number of points**(3). We also made one test with three parameters increased each time: **with number of points, range of points and connected objects** (4)**.**

Test1

Above we see a chart with three lines, each one describes:

Line	Resolves	Number of Points	Range between points	Connected polygons
(1)	1	4	+50 each time	+1 each time
(2)	1	+4 each time	+50 each time	0
(3)	1	+4 each time	100	+1 each time

Table 7

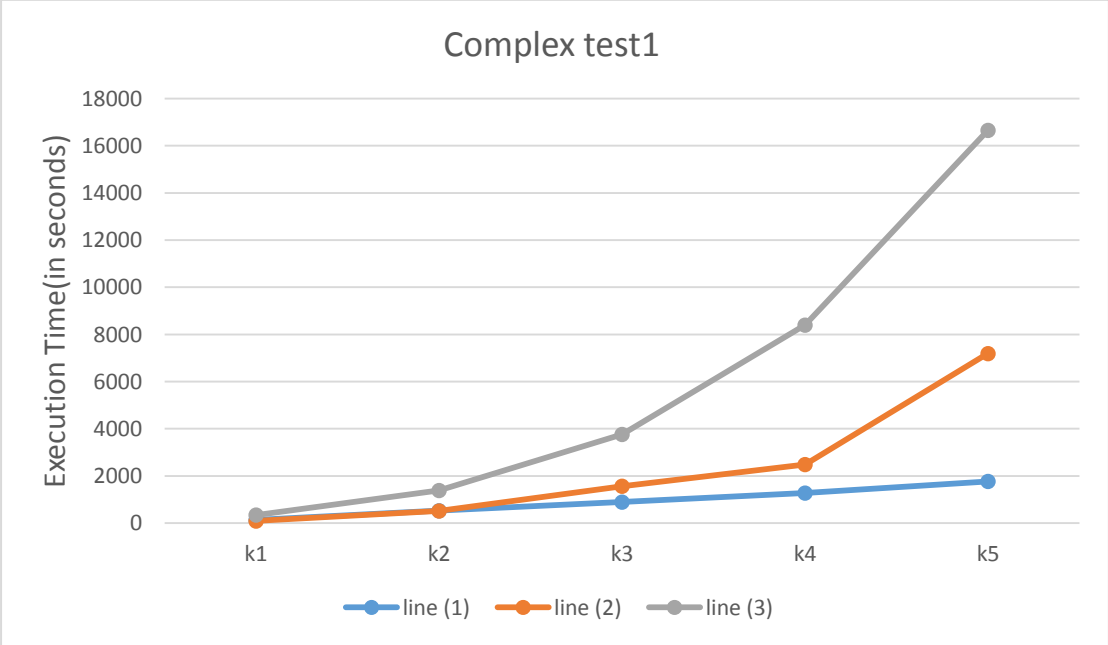


Diagram 4

Line	K1	K2	K3	K4	K5
(1)	Range: **50** Points: **4** Connected: **2**	Range: **100** Points: **4** Connected: **3**	Range: **150** Points: **4** Connected: **4**	Range: **200** Points: **4** Connected: **5**	Range: **250** Points: **4** Connected: **6**
(2)	Range: **50** Points: **4** Connected: **0**	Range: **100** Points: **8** Connected: **0**	Range: **150** Points: **12** Connected: **0**	Range: **200** Points: **16** Connected: **0**	Range: **250** Points: **20** Connected: **0**
(3)	Range: **100** Points: **4** Connected: **2**	Range: **100** Points: **8** Connected: **3**	Range: **100** Points: **12** Connected: **4**	Range: **100** Points: **16** Connected: **5**	Range: **100** Points: **20** Connected: **6**

Table 8

Test2

As we see from the test1, the more complex the parameters we increased, the more time the program need to do the resolve. So in this test2, we will increase all three parameters to see the results. In order to see the difference, we also show the line (3).

Line	Resolves	Number of Points	Range between points	Connected polygons
(4)	1	+4 each time	+50 each time	+1 each time

Table 9

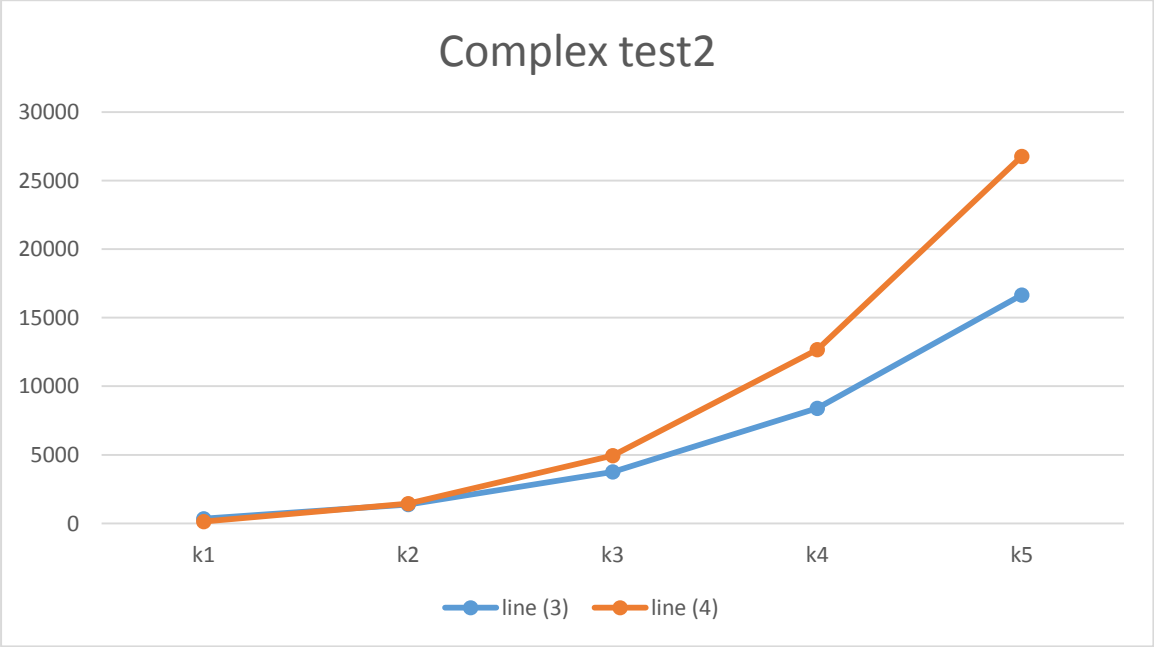


Diagram 5

Line	K1	K2	K3	K4	K5
(3)	Range: **100** Points: **4** Connected: **2**	Range: **100** Points: **8** Connected: **3**	Range: **100** Points: **12** Connected: **4**	Range: **100** Points: **16** Connected: **5**	Range: **100** Points: **20** Connected: **6**
(4)	Range: **50** Points: **4** Connected: **2**	Range: **100** Points: **8** Connected: **3**	Range: **150** Points:**12** Connected: **4**	Range: **200** Points: **16** Connected: **5**	Range: **250** Points: **20** Connected: **6**

Table 10

6. Conclusions

As main sources in this paper we used two papers [35] [36], we also used the main idea of the algorithms that are provided at [35] and we adjusted them to our needs. In this paper we tried to combine integrity constraints of a database that are refer to space and time. As we seen from the results, the more integrity constraints we have to satisfy, the more complex and time consuming the program gets.

The ramification problems in spatial-temporal databases are complex and multifaceted problems and no satisfactory solution has been proposed as of yet .First of all, the ramification problems can only be addressed in a specific and logical row as we saw in the paradigm. That means we can't use parallel algorithms and the solutions can't be fast. Even so the solution we came up isn't fully satisfactory, because in each case we should predict all the changes that can or cannot happen, either in the future or in the past. Also we can face the problem in each kind of database only separately.

In this paper we studied the ramification problem in a spatial-temporal database. More specifically we proposed a scenario and a solution with a tool, where in order to keep consistent a database we placed integrity constraints. The key ideas of our approach are:

- To analyze the problem in relational spatial-temporal database.
- To create algorithms and functions in order to locate and delete inconsistencies.
- To use dynamic rules in order to restrict the direct effects of an action, and static rules to restrict the indirect effects of actions.
- To develop a system which implement the solution in Java and SQL for relational databases.

Bibliography:

[1] J.McCarthy and P.J. hayes. Some philosophical problem and the standpoint of artificial intelligence, 1969.

[2] M. Winslett, Reasoning about action using a possible models approach. Proceedings of AAAAI-88, pp, 89-93, Saint Paul ,MN ,August 1988.

[3] M. Ginsberg and D. Smith. Reasoning about action I: A possible worlds approach. Artificial Intelligence, 35:165-195, 1988.

[4] V. Lifshitz. Frames in the space of situations, Artificial Intelligence, 46:365-376, Cambrige, MA, 1991.

[5] V. Lifshitz. Towards a metatheory of action. In J.F Allen, R. Fikes, and E. Sandewall, editors, Proceedings of the International Conference on Principles of Knowledge Representation and reasoning, pages 376-386, Cambridge, MA, 1991.

[6] V. Lifshitz. Towards a metatheory of action. Proceedings of KR'91, pp. 376-386, Cambridge, MA, 1991.

[7] N. McCain and H. Turner. A causal theory of ramifications and qualifications. Proceedings of IJCAI-95, pp.1978-1984, Montreal, Canada, August 1995.

[8] E. Marakakis, C. Kounali, K. Vassilakis, A Method for Removing Unused Argument from Logic Programs, Proceeding of the 10th IASTED International Conference on Artificial Intelligence and Soft Computing, ASC 2006, pages 197-202, August 28-30, 2006, Palma de Mallorca, Spain, ISBN 0-88986-4, ISSSN 1482-7913.

[9] Antonis Kakas and Rob Miller, A simple Declarative Language for Describing Narratives with Actions, The Journal of Logic Programming, Vol 31(1-3) (Special Issue on Reasoning abot Action and Change), pages 157-200, Elsevier, 1997.

[10]Marc Deneckerand Eugenia Ternovska. Inductive situation calculus, Artificial Intelligence archive Volume 171, Issue 5-6, pages: 332-360, April 2007.

[11] Edmund Clarke, Jeannete Wing, Formal Mehtods: state of the Art and Future Directions, ACM Computing Surveys, Vol. 28, No. 4, December 1996, pp.626-643.

[12] Dimitris Plexousakis, John Mylopoulos. Accommodating Integrity Constraints During Database Design. Proceedings of EDBT 1996, pp.497-513, Avignon, France, 1996.

[13] C. Elkan. Reasoning about action in first order logic. Proceedings of the conference of the Canadian Society of Computational Studies in Intelligence (CSCSI), pp. 221-227, Vancouver, May1992.

[14] J. McCarthy and P.J. Hayes. Some philophical problem from the standpoint of artificial intelligence. In B. Meltzer and D. Mitchie , editors, Machine Intelligence 5, pp.463-502. American Elsevier, 1969.

[15] A. Fusaoka. Situation Calculus on a Dense Flow of Time, Proceedings of AAAI-96, pp. 663-638, 1996.

[16] A.C. Kakas, R.S. Miller and F. Toni, E-RES: Reasoning about actions, Events and observations, in Proceedings of LPNMR2001, pp. 254-266, Springer Verlag, 2001.

[17] R.A. Kowalski. Database updates in the event calculus. Journal of Logic Programming, 1992.

[18] Nikos Papadakis, Dimitris Plexousakis. Action Theories in Temporal Databases. Proceeding of the 8th Panhellenic Conference of Informatics, pp.254-264, Cyprus, Nov. 2001.

[19] Rob Miller and Murray Shanahan. The Event Calculus in Classical Logic-Alternative Axiomatisations. Linkping Electronic Articles in Computer and Information Science, 5(16), 1999.

[20] Nikos Papadakis, Dimitris Plexousakis, "The Ramification and Qualification Problem in Temporal Databases", 2th PanHellenic Conference on Artificial Intelligent, p. 18-30 LNAI 2308 April 2002, Thessalonica, Greece.

[21] Nikos Papadakis, Dimitris Plexousakis. Action with Duration and Constraints: The ramification problem in Temporal Databases. 14th IEEE ICTAI, 2002, Washington D.C.

[22]J. Pinto. Temporal Reasoning in the Situation Calculus. Ph.D. thesis, Dept. of Computer Science, Univ. of Toronto, Jan. 1994.

[23] Nikos Papadakis, Dimitris Plexousakis. Action with Duration and Constraints: The ramification problem in Temporal Databases. International Journal of Artificial Intelligent Tools(IJAIT) pp. 315-353, volume 12, Number 3, September 2004.

[24] R. Reiter. Knowledge in Action Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge 2001.

[25] R. Reiter. Natural Actions, Concurrency and Continuous Time in the Situation Calculus, KR 96, pages 2-13, 1996.

[26] M. Thielscher. Ramification and causality. Artifical Intelligence, 89(1-2), pp: 317-364, 1997.

[27] M. Thielscher. Reasoning about actions: Steady versus stabilizing state constraints. Artifical Intelligence, 104:339-355, 1988.

[28] Chen, J. and P. Gong, practical GIS: Building and Maintaining a Successful GIS, Science Press, Beijing, p.186, 1998.

[29] J. Pinto and R. Reiter. Temporal Reasoning in Logic Programming: A case for the Situation Calculus, Proceeding of the 10th Int. Conf. on Logic Programming, Budapest, Hungary, June 21-24, 1993.

[30] Peter Lindsay, A Survey of Mechanical Support for Formal Reasoning, Software Engineering Journal, Vol. 3, No. 1, January 1988, pp.3-27.

[31] Koubarakis M. Skiadopoulos S., "Querying temporal and spatial constraint networks in PTIME", Department of Electronic and Computer Engineering, Technical University of Crete, Artificial Intelligence Volume 123, Issues 1-2, October 2000, Pages 223-263.

[32]Spatial database Available: https://en.wikipedia.org/wiki/Spatial_database

[33]Temporal database Available: https://en.wikipedia.org/wiki/Temporal_database

[34]Paul Kefalas, "Spatio-temporal Reasoning in XML Databases Including integrity constraints", Aristotel University of Thessaloniki computer science department, ,Greece,January 2011, Pages 11-13.

[35]Nikos Papadakis, Yiannis Christodoulou, “A tool for addressing the ramification problem in spatial databases: A solution implemented in SQL”, Department of sciences, Technological Educational Institute of Crete, Greece, 2009.

[36] Nikos Papadakis, Dimitris Plexousakis, Grigoris Antoniou, Myron papadakis, Katerina Boutsika, “A tool for addressing the ramification problem in temporal databases”, Technological Educational Institute of Crete, Greece, 2007.