



# **System for Area Planning**

Donatas Daubaras (110336260)

BSc Computing Science, May 2014

Supervisor: Dr Victor Khomenko

# **Abstract**

Nowadays area planning is one of the key components in creating living area, cities or any other type of area. Therefore, there is a need to create a tool, which would allow architectures to design possible solutions and even provide them with the optimal solution.

The aim of this project is to create a tool, which would visualise one's predefined map and would allow placing various objects into it. Moreover, this tool will allow one to perform layout optimisation. After using this feature user will be provided with graphical representation for this optimisation problem.

## **Declaration**

I declare that this dissertation represents my own work except where otherwise stated.

## **Acknowledgments**

I would like to thank my project supervisor Dr Victor Khomenko for his valuable support and guidance throughout this project. Finally, I want to thank my parents that gave me an opportunity to study at Newcastle University and be involved in this project. To them I dedicate this dissertation.

# Table of Contents

<b>Abstract.....</b>	<b>2</b>
<b>Declaration .....</b>	<b>3</b>
<b>Acknowledgments .....</b>	<b>4</b>
<b>Table of Content .....</b>	<b>5</b>
<b>Chapter 1: Introduction.....</b>	<b>8</b>
1.1 Introduction.....	8
1.2 Purpose.....	8
1.3 Aim and objectives .....	9
1.3.1 Aim .....	9
1.3.2 Objectives.....	9
1.4 System Requirements .....	10
1.5 Project Schedule.....	11
<b>Chapter 2: Background Research.....</b>	<b>12</b>
2.1 Introduction.....	12
2.2 2D Packing Problem .....	12
2.2.1 2D-Strip-Packing Problem (SPP) .....	13
Next-Fit Decreasing Height (NFDH).....	14
First-Fit Decreasing Height (FFDH).....	15
Best-Fit Decreasing Height (BFDH) .....	16
2.2.2 2D-Bin-Packing Problem (2BP).....	16
2.3 Existing 2D Packing Solvers .....	18
2.4 2.5D Visualization.....	19
2.5 Implementation Technology .....	20
2.5.1 C/C++.....	20
2.5.2 C# .....	21

2.5.3	<u>Java .....</u>	<u>21</u>
2.5.4	<u>Decision.....</u>	<u>22</u>
2.6	<i>Existing Tools for Area Planning.....</i>	<i>23</i>
<b>Chapter 3:</b>	<b>System Design and Implementation.....</b>	<b>25</b>
3.1	<i>Introduction.....</i>	<i>25</i>
3.2	<i>Software Development Model.....</i>	<i>25</i>
3.3	<i>Software Development Phases .....</i>	<i>25</i>
3.4	<i>GUI Design and Implementation.....</i>	<i>26</i>
	<u>Menu Bar .....</u>	<u>27</u>
	<u>Map View .....</u>	<u>27</u>
	<u>Tool Bar.....</u>	<u>28</u>
	<u>Data Table .....</u>	<u>29</u>
	<u>GUI .....</u>	<u>30</u>
3.5	<i>Map Visualization and Control Features .....</i>	<i>31</i>
3.6	<i>Object Placing .....</i>	<i>34</i>
3.7	<i>Optimisation Algorithm Development and Implementation.....</i>	<i>38</i>
<b>Chapter 4:</b>	<b>Results .....</b>	<b>40</b>
4.1	<i>Introduction.....</i>	<i>40</i>
4.2	<i>Actual Results .....</i>	<i>41</i>
4.3	<i>Conclusion.....</i>	<i>49</i>
<b>Chapter 5:</b>	<b>Conclusion.....</b>	<b>50</b>
5.1	<i>Introduction.....</i>	<i>50</i>
5.2	<i>Satisfaction of the Aim and Objectives .....</i>	<i>50</i>
	<u>Objective One.....</u>	<u>50</u>
	<u>Objective Two.....</u>	<u>50</u>
	<u>Objective Three .....</u>	<u>51</u>

<u>Objective Four .....</u>	<u>51</u>
<u>Aim .....</u>	<u>51</u>
<i>5.3 Overall Conclusion .....</i>	<i>52</i>
<i>5.4 Future Work.....</i>	<i>52</i>
<b>References.....</b>	<b>54</b>
<b>Code .....</b>	<b>58</b>

# **Chapter 1: Introduction**

## **1.1 Introduction**

This section is aimed to introduce the purpose of this project, to name the aim and objectives of the project. System requirements are introduced in order for reader to understand how objectives will be achieved. Finally, the project schedule is showed in order to display the main tasks and their time phases.

## **1.2 Purpose**

As the population level is raising rapidly on Earth, the number of cities increment or current cities expand respectively. Moreover, as urban areas expand the problem of lack of enough space to cultivate grows, which can lead to low production of agriculture products that contribute to the economy [1]. To design a new city efficiently is a crucial and hard task. Therefore, creating a tool that would help architects is of great importance. The tool that would allow manually plan urban area would be very useful, however it would be even better if actual planning could be made automatically and would provide an optimal solution.

Moreover, this project is relevant to the two – dimensional packing problem. The optimisation of area seeks to provide an optimal solution by placing user defined objects. Each object itself contains the measurements and the value that will be obtained by placing it. Accordingly, this dissertation will be trying to tackle two – dimensional bin packing problem. This problem is considered NP-Hard, which mean it is at least as hard as the hardest problems in NP [2]. Advancing knowledge of two-dimensional packing is of high importance as well, because it involves numerous other packing problems like two-dimensional strip packing, 2D knapsack problem, and two – dimensional cutting – stock problem. Improving current solutions to these problems might have an enormous impact on the industry.



## **1.3 Aim and objectives**

This dissertation is analysing how the optimal house packing by the value can be achieved, while researching two-dimensional packing problem. Moreover, graphical representation should be provided to user via graphical user interface. Furthermore, one should as well be able to manually edit the predefined area and all this is as well will be displayed via graphical user interface.

### **1.3.1 Aim**

The aim of this project is to develop a system for urban area planning that would be easy usable by architects. This application will allow user to specify area using measurements via GUI. Users will be able to drag-and-drop various objects with predefined sizes into the area. This piece of software will be able to perform the layout optimisation.

### **1.3.2 Objectives**

**Objective 1 :** Design a graphical user interface for the system that will be used for area planning.

**Objective 2 :** Implement the basic functionality for this tool, such that user could create a map and interact with it by placing various objects.

**Objective 3 :** Implement basic graphical functionality that would allow user to adjust how the map and its objects are displayed.

**Objective 4 :** Implement a feature to this tool, which would allow to perform optimization, which would compute optimal or highly optimal solution.

## 1.4 System Requirements

In order to achieve the objectives named above, the following requirements will specify the deliveries of this project. They are split into different sections, relating to pieces of the system.

### Map Creation

- The system must allow user to predefine map size.
- The application must display map then user inputs its size.
- The map can be removed by the user.
- Map will be displayed using 2.5D visualization technique.
- Map should be created in scrollable view, so user could be able to see all parts of it.

### Additional features

- The system must have a tool panel, which contains various objects and modification that user can make to map.
- Application has to allow user to zoom in or zoom out the current displayed view.
- The user can choose whether or not he wants to see gridded map.

### Object placing

- User should be allowed to define house object size.
- Only objects that fit inside the map should be placed in.
- User should be able to see where the object is going to be placed and whether or not it fits.
- Roads placed one to each other should adjust their positions to display properly build road.

### Optimization

- User should be able to input the data for every house type he is trying to place inside the map.
- Each house should have the option whether they can be rotated or not.
- After user enters the data and decides to optimise the predefined map, application should find optimal or high optimal solution and display in the map.

## 1.5 Project Schedule

This project schedule for this academic year with details of the main tasks is displayed in Figure 1.5.1 Project Schedule below.

Task	Start	End
Background Research	07/10/2013	09/12/2013
Design	20/11/2013	20/12/2013
Implementation	21/12/2013	25/04/2014
System Testing	25/04/2014	02/05/2014
Evaluation	01/05/2014	04/05/2014
Write-Up	15/12/2013	10/05/2014

*Figure 1.5.1 Project Schedule*

## **Chapter 2: Background Research**

### **2.1 Introduction**

This chapter introduces the fundamental concepts relating to this project. Some closely related problems are analysed as well. Moreover, current issues regarding 2D packing problem is discussed and some currently available “solutions” to solve it. Finally, existing tools for area planning are discussed.

### **2.2 2D Packing Problem**

The packing problems are a class of optimization problems in mathematics that involves an attempt to pack objects into container. The goals of this problem is to either pack objects wasting as less space as possible or pack all the projects so that the number of container used would be as few as possible. Most of these problems are related to real life packaging, storage and transportation dilemma.

Two dimensional packing problems is a type of optimization problem in mathematics that involves an attempt to pack 2D objects into 2D container. The goals of this problem is to either pack objects wasting a less space as possible or pack all of the objects using as few container as possible. This type of packing problem therefore has numerous other problems related, in example:

- **2D-Strip-Packing**
- **2D-Bin-Packing**

However, even simple cases of these problems are known to be NP-hard, and hence, it is very likely that no efficient algorithm for them exists [3]. This means that if we wanted to find the optimal solutions for this problem all the time this would require an exponential computation, which would be very slow and the calculation might not be completed at all due to the time required for it. So it is reasonable to look for alternative algorithm solutions, which would provide fast run time and would produce near optimal solutions.

### **2.2.1 2D-Strip-Packing Problem (SPP)**

The two-dimensional strip-packing problem is to pack a given set of rectangles into a strip with given width and infinite height so as to minimize the required height of the packing. This problem from computational point of view is considered to be NP-hard problem. Due to this reason many researchers concentrated on finding approximation algorithms for this problem. "The class of algorithms studied solving 2D strip packing problem is called level algorithms which are primarily used when the entire information about the items to be packed is available. In this class, the strip is divided into horizontal levels whose height corresponds to the height of the tallest rectangle on a previous level. The first level is the bottom of the strip" [4]. In a paper written by N. Ntene and J.H. van Vuuren "A survey and comparison of level heuristics for 2D oriented strip packing problem" [5] heuristics from literature were tested on a wide range of benchmark data sets and were explained in detail view. Moreover, some modifications were suggested for some of heuristics. Most of algorithms implemented provide a very quick run time and produces highly optimal solutions.

## Next-Fit Decreasing Height (NFDH)

Next-Fit Decreasing Height heuristic for 2D strip bin packing is a greedy approach to provide not the optimal, but highly optimal solution, while not doing a big amount of calculations. The first step for this algorithm is to sort all the rectangles that need to be placed into the bin by having largest height object at the start and smallest at the end of the list. Afterwards each rectangle is placed left-justified on a current level, by beginning for the start of the list, until the next rectangle will not fit. If there is no room on the current level, a new level is started at the height of the tallest (first) rectangle on the previous level.

This heuristic allows quite fast run time, because the most expensive computation for this algorithm is sorting the objects and previous levels of the bin is not revisited. Nevertheless, this is a greedy way of solving 2D-strip-packing problem and therefore usually waste quite big amount of space in a bin. The demonstration for this algorithm is provided in the Figure 2.2.1 NFDH demonstration.

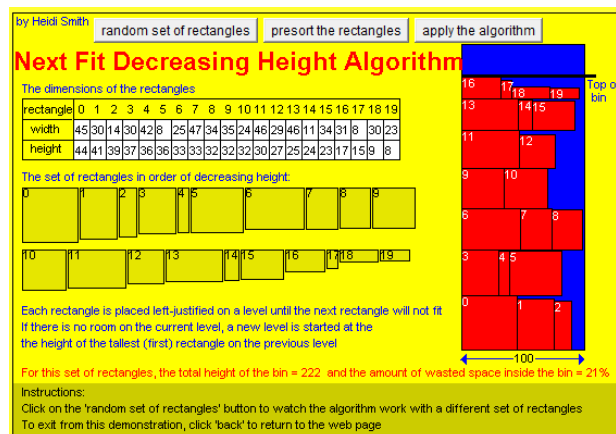


Figure 2.2.1 NFDH demonstration [6]

## First-Fit Decreasing Height (FFDH)

First-fit-decreasing height algorithm for 2D strip packing problem, is similar to NFDH heuristic. Nevertheless, this method would most of the time give better solution, but there are more computations done, while taking it. This algorithm in a sense is still a greedy approach, because it tries to pack objects that have biggest height first. Therefore, this algorithm will not provide an optimal solution all the time, but could give a highly optimal one.

FFDH algorithm starts with sorting the objects that needed to be packed in a decreasing height order (same as NFDH). Afterwards it tries to packs every object from the start of the list till the end. Items are packed on the lowest level where it fits. A new level is created on top of the top level if an item does not fit in any of the existing levels.

This algorithm allows to produce a highly optimal solution in quite reasonable time, because the most expensive computational calculation is sorting. Nevertheless, if a bigger number of levels would be created in the bin the run time could be delayed, because this algorithm has to revisit every level if object does not fit in any current levels. The demonstration for this algorithm is provided below in the Figure 2.2.2 FFDH Demonstration .

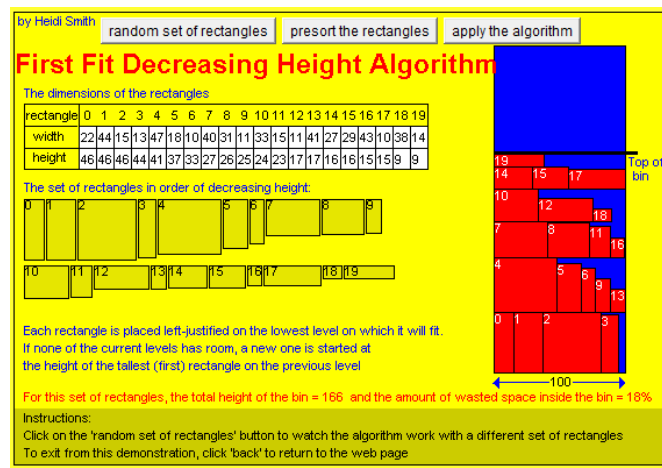


Figure 2.2.2 FFDH Demonstration [7]

## **Best-Fit Decreasing Height (BFDH)**

Best-fit decreasing height heuristic for strip packing problem is analogous to the FFDH algorithm except that in this algorithm objects are placed from the most top level, while going to most bottom level. If the object cannot be put into currently available levels a new level is created on the most top level and the object is placed inside it.

### **2.2.2 2D-Bin-Packing Problem (2BP)**

The two-dimensional bin-packing problem is for objects of different area must be packed into a finite number of bins or container each of same area in a way that minimizes the number of bins used. In computational complexity theory, it is a NP-hard problem.

Moreover, this problem has several variations such as packing by cost, fixing the number of bins available, packing by area, and so on. This problem can also be seen as a special case of the 2D cutting stock problem. If we applied the constraints that the number of bins is one and we are trying to pack by cost we would as well get the case that this project is trying to solve.

The heuristics for 2BP mostly are of greedy type, and can be classified into two families: one phase algorithms and two phase algorithms.

#### **One Phase Algorithms**

This class of algorithms for 2D-bin-packing problems directly pack the items into finite bins.

##### **Finite Next-Fit (FNF)**

Finite next fit heuristic for 2D bin-packing directly packs the items into finite bins similarly as NFDH algorithm. The first step as in NFDH is to sort the objects by decreasing height. After doing so objects are starting to be placed in finite bin containers. As in NFDH heuristics then the first object is placed the level in a bin is made. If the item cannot fit in any of the bin levels an item is then placed in a new bin.

This algorithm provides similar efficiency and time complexity as a NFDH. The nature of this heuristic is as well greedy and it will rarely provide an optimal result. However, this algorithm can provide a highly optimal solution quite fast.



### **Finite First-Fit (FFF)**

The strategy for this algorithm is adopted from FFDH, which is a heuristic for strip packing problem. This algorithm tries to pack items on the lowest level of the first bin where it fits; if no level can accommodate it, a new level is created in the first bin having sufficient vertical space, otherwise, the new level is created in a new bin.

### **Two phase algorithms**

This class of algorithms first start by packing the items into a single strip, where a bin is having width  $W$  and infinite height. In the second phase, the strip solution is used to construct a packing into finite bins.

#### **Hybrid First-Fit (HFF)**

The first phase of this algorithm is obtained by the FFDH algorithm. After completing this phase the levels from strip gained in step one are packed from bottom to top. If the level does not fit into the bin it is then placed in a new bin.

#### **Hybrid Next-Fit (HNF)**

The first phase is for this heuristic is completed using NFDH algorithm. After completing this step the levels from the retrieved strip are packed in a way that the level is either placed in a current bin or either in a new bin.

## 2.3 Existing 2D Packing Solvers

Currently there are no perfect 2D packing solvers, because of this problem's hardness. Even if such a program that would give an optimal solution would exist its run-time for providing such solution would be unreasonable. This is, because finding the optimal solution requires an exponential computational calculation, which would require a long time to proceed with large data.

Nevertheless, there are few solvers that implements heuristics and tries to provide highly optimal results under reasonable time. One of the examples would be 2D load packer [8]. This tool may be used to optimize multi-product, multi-container load plans for any size rectangular containers, trucks, trailers, railcars, crates, boxes, as well as be applied to free type 2D cutting or any other 2D sheet layout planning. The details provided by the creators of this application as well state that an average of 95% cutting yield is achieved. The calculation time given is as very reasonable and solution usually is provided in seconds.

Another solver that author managed to find was designed by Kevin Binkley. The program designed by him is called 2D packing solver, which was created as a useful that can help you graphically explore and illustrate how a Genetic Algorithm (GA) can solve packing problems [9]. However, this software could not be tested, because it seems no longer available.

The last tool that is currently available is Linpacker [10]. This is a scientific tool for studying rectangle packing. This piece of software provides a graphical user interface and some algorithms for this purpose. The problem this program is trying to solve is 2D bin packing, where a number of small rectangles are put into a rectangular container optimally and the container has a fixed width, but no fixed height. Although, this software was released nearly ten years ago (2006) it is still a good way to see how certain algorithms work.

## 2.4 2.5D Visualization

2.5D visualization (sometimes referred  $\frac{3}{4}$  perspective and pseudo-3D) is a hybrid of 2D and 3D visualization. This technique is common in video game that is restricted to a two-dimensional plane. This graphical representation was as well added to the Google Maps application [11]. This map visualization has a much better look than a 2D one. To achieve this graphical representation the most common technique is to have a 2:1 pixel ratio for a single square unit in a map. The Figure 2.4.1 below demonstrates how such perspective can be achieved by doing so.

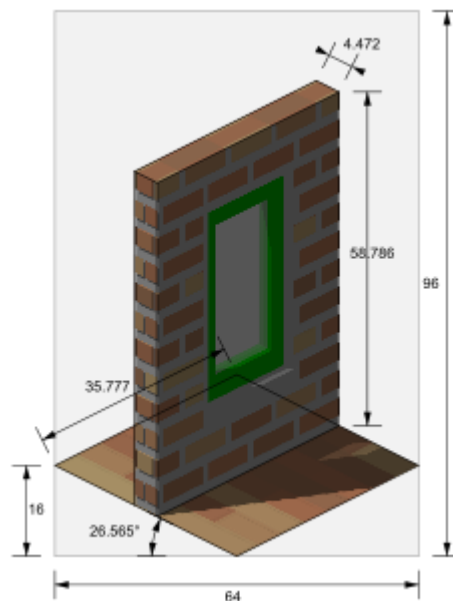


Figure 2.4.1 [12]

What is more the objects that are placed inside the map have to have the same ratio as a square unit of the map. What is more, the height of object can as well be adjusted to give the visual view for the user, how tall they are. Nevertheless, as this project is based on 2 dimensional packing there is no need not to use fixed height.

## **2.5 Implementation Technology**

An important decision that needs to be made before the application's development is the decision of a programming language for implementing the tool. The number of programming languages available is enormous. Therefore, the focus will be made only on the most popular ones that can be used to create area planning with a user friendly GUI. The choice of programming language will be as well considered by the factors of their visualization features, portability, extendibility and the performance they offer.

### **2.5.1 C/C++**

C is a powerful functional programming language developed in the early 1970's by the Ken Thompson and Dennis Ritchie at the Bell Telephone Laboratories [13]. This programming language is a general-purpose programming language and is closely associated with UNIX system where it was developed, since both the system and most of the programs that run on in are written in C. Nevertheless, the language is not tied to any operating system and is used in many computing environments. What is more, C provides a direct access to memory making it ideal for applications that were previously coded in Assembly language.

In 1983, the name of the language was changed from C with Classes to C++ [14]. During this change many new features were added and the most notable of which are virtual functions, function overloading, references with the & symbol, the const keyword and single-line comments using "//". C++ is one of the most popular programming languages and is widely used in the software industry. Moreover, due to big support for graphics and animation features this programming language is very popular in game industry. C++ is as well one of the fastest programming languages during run-time that exploits the usage of memory as much as possible without slowing down the system. Nevertheless, the control of the memory has to be taken into account by the developer, because he is responsible for freeing the allocated memory, which makes coding harder and more complicated. Even though, fast run-time is achieved with this programming language the portability for it is very minimal due to its compilation into machine code, which can only be executed in particular platform.

### **2.5.2 C#**

C# was released in late-2005 as part of Microsoft's development suite, Visual Studio 2005 [15]. This programming language is derived from the C programming language. Nevertheless, it has the features such as garbage collection that allows developer not to worry about the control of the memory. This programming language is object-oriented and is very similar to Java. Due to the rich library C# makes many functions easy to be implemented. Nevertheless, the applications written in this language requires .NET framework to be installed in the machine. Due to the reason that this framework is mostly available on Windows the applications portability becomes the issue. For this reason, programs written in C# would have problems to be run on other platforms such as Mac and UNIX.

### **2.5.3 Java**

Java is a computer programming language that is object oriented, concurrent and was specifically designed to have as few implementation dependencies as possible. Originally Java compilers virtual machines and class libraries were developed by Sun from 1991 and first released in 1995. The main advantage of Java is that it is platform-independent and can be moved easily from one computer system to another. Moreover, Java has a huge number of libraries available, which allows easier program implementation. Furthermore, this programming language allows developers not to worry about the memory allocation, because it has its own garbage collector. In addition, Java provides a comprehensive Application Programming Interface (API) that allows developers to reuse or even extend existing classes of the API. What is more, Java has decent graphical support that allows to create user friendly GUI.

#### **2.5.4 Decision**

C/C++ provides one of the fastest language run-time if it is written properly and provides a decent support for graphics. However, this is really complicated language due to manual memory allocation and might be even slower than Java or C# if written inappropriately. What is more, it has some portability issues that are not the case with Java. C# and Java are quite similar programming languages and most of the features can be found in both of them. They also provide rich graphical environment and good standard libraries. Nevertheless, C# has some portability issues regarding other platforms than Windows. In conclusion, Java meets most of considerable factors stated above. For this reason, the best programming language to be used for this system would be Java, due to the reasons listed before.

## 2.6 Existing Tools for Area Planning

Currently there are quite many area planning tools that allows user manually plan urban area or any other type of area. Such applications usually vary from 2D and 3D graphical representation of the area. Most of the advanced programs although offer 3D visualization, which allow user to construct a very realistic city model. Nevertheless, there is no urban area planning tool that would allow user to automatically optimise the predefined map with provided predefined houses.

Lumion3D [16] is architectural visualization software for urban planning. It is an area planning system, which allow user to represent their map and objects placed inside the map in 3D graphical representation. This tool allow user to manually create city model. What is more, it has a feature of traffic simulation, which allows user to have an interactive city model. Moreover, this software lets user to animate various objects in a way he wants. Furthermore, this tool allows indoor architecture simulation and interior 3D design. Although, this software is a great tool for manually creating city models and simulating city traffic it has no optimization features that would allow user to improve their urban planning.

CAD [17] is another good example of the architectural tools. This software is probably the most popular tools used by architectures and is used widely by the University students. This software is probably the most advanced tool there is currently available, because of its numerous available features. What is more, this tool allows programming UI for specialised applications. This means that the user could program this software to apply optimizations tasks if needed. Nevertheless, this tool is more useful for small objects or single building design, but not the entire city planning.

Citilabs [18] is a powerful tool for transportation modelling, traffic engineering, GIS and urban planning. This tool provides 2D graphical representation and supports GIS data format. Moreover, this software allows analysing various scenarios for transport network, while providing reports that help the analyst to create high quality charts and tables of single or multiple scenarios. Nevertheless, this software concentrates more on transportation planning and traffic engineering and is not so useful for urban area planning.

UrbanSim [19] is a software-based simulation system for supporting planning and analysis of urban development, incorporating the interactions between land use, transportation, the economy, and the environment. It is intended for use by Metropolitan Planning

Organizations (MPOs), cities, counties, non-governmental organizations, researchers and students interested in exploring the effects of infrastructure and policy choices on community outcomes such as motorized and non-motorized accessibility, housing affordability, greenhouse gas emissions, and the protection of open space and environmentally sensitive habitats.



## **Chapter 3: System Design and Implementation**

### **3.1 Introduction**

This chapter provides a detailed description of the system design process that was done in order to develop this application. Moreover, the design decisions will be justified taking into account their pros and cons. Furthermore, the requirements in chapter one will be identified, whether or not they have been met. What is more, the tools and the technologies used will be described. Finally, the actual implementation will be provided in detail.

### **3.2 Software Development Model**

Designing what software model for the system one is trying to create is very important choice to make. Due to the reason that the time given to create this application is considerably short and the problem we are trying to tackle is known to be difficult the best software development model to follow would be prototyping model. This model will be a great way to stick throughout the implementation, because the author will first be trying to create a prototype system. Moreover, this model allows consistent improvement to the latest prototype that is created.

### **3.3 Software Development Phases**

Before starting doing starting the implementation it is very important to understand what sort of steps you need to complete in order to successfully build your system. The system for area planning will consist of 5 phases that are:

- GUI design.
- Map visualization.
- Map control features.
- Object placing.
- Optimization algorithm development.

### 3.4 GUI Design and Implementation

This step involves designing a user friendly graphical user interface. Before starting designing it we need to understand what kind of components this interface will consist of and what purpose containers will have. After doing so it is crucial to design positioning for these components.

The main component for this system GUI will be the one in which the map will be visualized. What is more, this container should be scrollable in case the map that needs to be visualized does not fit fully. The obvious position for this component is to be in the centre of main container. The next component to consider for GUI is menu component. This component will have the option that will allow user to generate the map or any other graphical adjustment for it. The best place to position for this component will be the north position of the application. Furthermore, the GUI should as well contain a container for the objects that user can manually place into the map. This container could be named as a toolbar container, which would contain all the possible option that user can do to modify the content of the map. It should be placed on the right side of the application. The final component to consider is the one where user can define the house objects he wants to place inside the map using optimization. This component should look as a table and the best position to place it would be on the left side.

After deciding the GUI components positioning it is much easier to draw a sketch how actually it will look like. The Figure 3.4.1 GUI sketch displays the sketch.

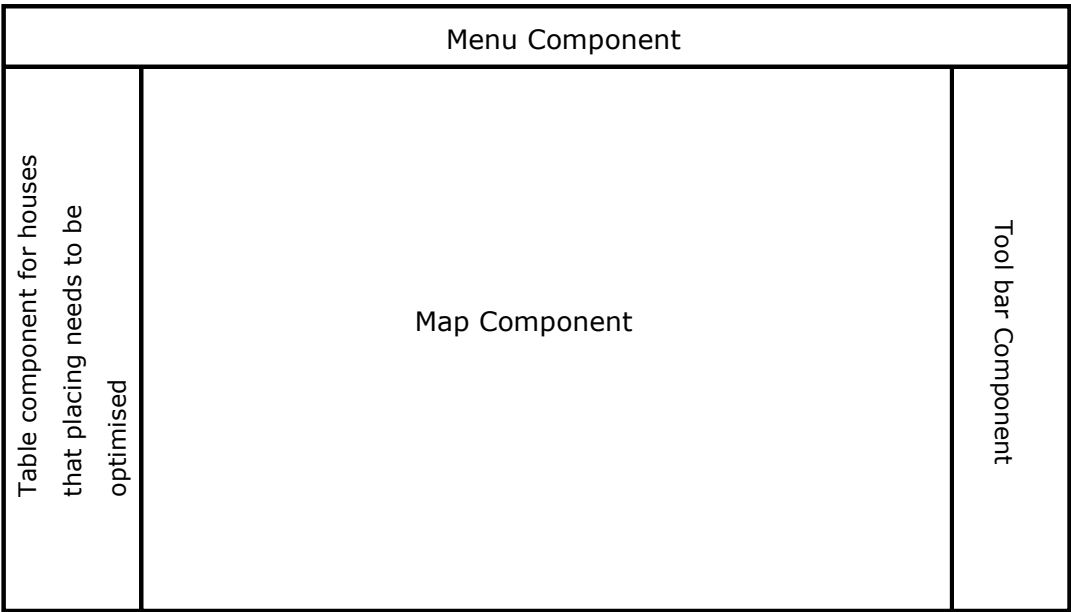


Figure 3.4.1 GUI sketch

## Menu Bar

After designing the GUI it is now a time to implement it. First of all, to implement Menu component JMenuBar [20] will be extended from the swing library. By doing this it is now very easy to add menu items that will contain various options for changing the look of the map. After completing menu component implementation it looks at is shown in Figure 3.4.2 Menu Bar below.



*Figure 3.4.2 Menu Bar*

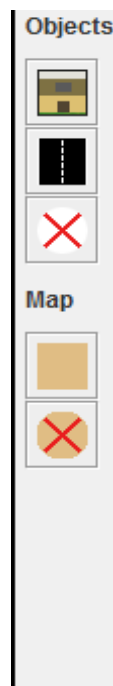
The Map item consist of the options to create a new map or remove currently displayed map. Options consist of either enabling or disabling the gridding of the map. The view tab has an option either to zoom in the view or zoom out. The implementations for these visual features are discussed at 0.

## Map View

The second component of GUI to consider is the map component in which the map will be displayed. Due to the fact that this component will have to visualize different size maps it important to make it scrollable in order for user to be able to access larger map that could be displayed in this container. For this reason, this container will extend JScrollPane class [21], which is a part of swing library. The actual displaying of the map in this container will be discussed at 0.

## Tool Bar

Furthermore, the implementation of tool bar component follows. The tools that this component will contain can be divided into two groups. The first one is objects. This group of tools will allow user to add house units and road into the map or either remove them. By pressing mouse right click button on the house tool user will be able to define the width and the height of the house he wants to place inside the map. The second group of the tools available will deal with the actual map grid. The tools in this group will allow user to remove or recreate any square of the map, if it does not contain an object placed in it. All the tools available to user will have an icon to make it easier for one to understand what they do. To make toolbar implementation easier this component will extend JToolBar class [22], which is a part of swing library. After completing this component it looks as it shown in the Figure 3.4.3 below.

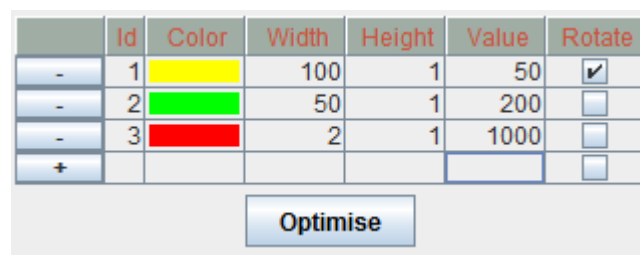


*Figure 3.4.3 Tool bar Component*

The implementation of this tool's actions will be discussed in 0.

## Data Table

The last component to consider is the table of the houses that will be involved in optimisation. The data available to be input for certain house unit will consist of the colour house unit will be displayed. Moreover, user will be able to define their width and height that will have a constraint to be higher than zero or less and equal to hundred. Furthermore, the value that will obtained by placing this object, which will can be in the interval from one to thousand including these numbers. Finally, user will be able to choose whether or not this house unit can be rotated by 90 degrees. After inputting the data for the houses that needs to be packed user will have an option to perform optimisation, which can be done by pressing the button "Optimise". Due to the reason, swing library provides an existing interface and some functionality for this sort of the table this component will therefore be extended from JTable class [23]. After completing the implementation of this component it looks like showed in the Figure 3.4.4 Data table.



	Id	Color	Width	Height	Value	Rotate
-	1	Yellow	100	1	50	<input checked="" type="checkbox"/>
-	2	Green	50	1	200	<input type="checkbox"/>
-	3	Red	2	1	1000	<input type="checkbox"/>
+						<input type="checkbox"/>

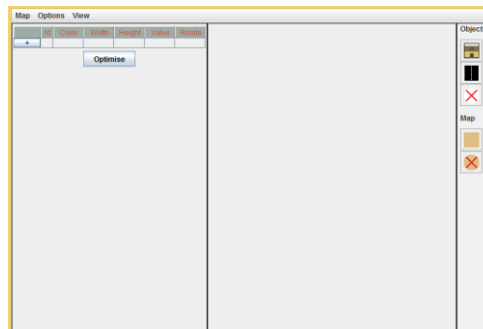
Optimise

*Figure 3.4.4 Data table*

The implementation and development of Optimization algorithm can be found in 3.7.

## GUI

After implementing these individual components, the GUI was finally created. The GUI for this system is displayed below in Figure 3.4.5.



*Figure 3.4.5 GUI*

### 3.5 Map Visualization and Control Features

Due to the reason it was chosen to visualise the map in 2.5D graphical representation the actual map will be painted as a rhombus. The ratio of width and height for it will be 2:1, due to the reasons mentioned in 2.4. The boundary therefore will be visualized as a polygon with four points. The point's coordinates can be expressed by the Formula 3.5.1 below.

$$\begin{array}{l} \text{LeftPoint}\left(0, \frac{\text{height}}{2}\right) \text{ BottomPoint}\left(\frac{\text{width}}{2}, \text{height}\right) \\ \text{RightPoint}\left(\text{width}, \frac{\text{height}}{2}\right) \text{ TopPoint}\left(\frac{\text{width}}{2}, 0\right) \end{array}$$

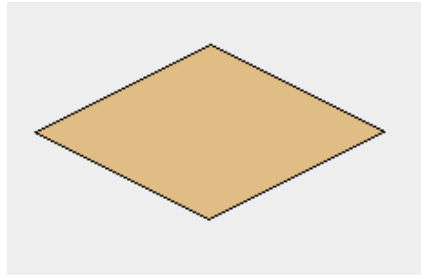
*Formula 3.5.1*

The height in this formula represents the map boundary distance from bottom most and top most point and width is the distance from left most to right most point. To calculate these distances user input is needed, where he defines map W (width) and H (height). This means that map area can be calculated by multiplying W and H. The area of the map then can be represented as smaller rhombus of the same ration 2:1 and the standard constant is defined in program as  $TILE\_SIZE = (2x, x)$ . The height and width can then be found by the Formula 3.5.2 showed below.

$$\text{width} = 2x * \frac{H + W}{2} ; \text{height} = x * \frac{H + W}{2}$$

*Formula 3.5.2*

Implementing these formulas user is now able to create and see a map that is showed in Figure 3.5.1 below.



*Figure 3.5.1 Map Boundary*

In order to provide a functionality that user could grid the map we need to split the map boundary into the number ( $W * H$ ) smaller rhombus. In order, to do as least calculations as possible we are going to parallel lines to the lines that are parallel in the map boundary. To calculate how many parallel lines for width and height lines to draw will be needed can be calculated using Formula 3.5.3.

$$W_{lines} = W - 1 ; H_{lines} = H - 1$$

*Formula 3.5.3*

After calculating the number of lines that needs to be draw we can calculate the points of those lines by Formula 3.5.4 below.

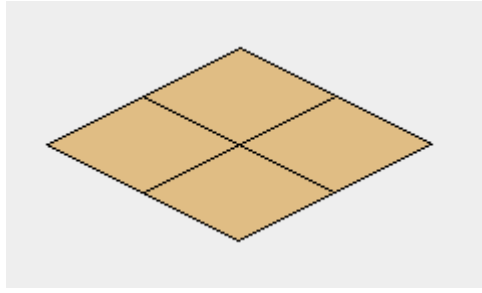
$$WidthLine(i) = \left( \begin{array}{l} \left( TopPoint.x - (x * i), TopPoint.y + \frac{x * i}{2} \right) , \\ \left( RightPoint.x - (x * i), RightPoint.y + \frac{x * i}{2} \right) \end{array} \right), \text{ where } i > 0 \text{ \& } i \leq W_{lines}$$

$$HeightLine(i) = \left( \begin{array}{l} \left( TopPoint.x + (x * i), TopPoint.y + \frac{x * i}{2} \right) , \\ \left( LeftPoint.x + (x * i), LeftPoint.y + \frac{x * i}{2} \right) \end{array} \right), \text{ where } i > 0 \text{ \& } i \leq H_{lines}$$

*Formula 3.5.4*

Implementing this formula allows user to grid the map. This feature can be found in the Menu Bar (3.4) in the option tab. After applying this feature the visualization of the map in Figure 3.5.1 Map Boundary will look as shown in Figure 3.5.2 Map with grid below.



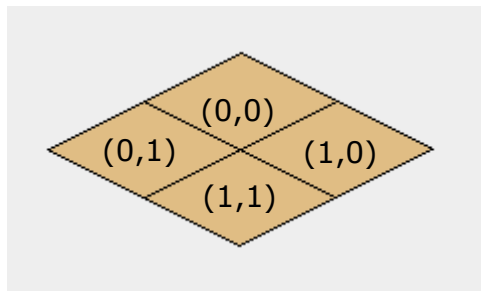


*Figure 3.5.2 Map with grid*

The last visual manipulation feature for the map is zoom in and zoom out is simply made by changing the constant `TILE_SIZE` value. Due to the fact it has to have 2:1 ratio the Values of this constant will either is going to be multiplied by 2 (if user wants to zoom in) or divided by 2 (if user wants to zoom out). Moreover, these values cannot exceed certain minimum or maximum value. The implementation for zoom feature to work with placed objects in the map will be discussed in 0.

### 3.6 Object Placing

To allow user to place various objects it is crucial to decide what main parameters they should have. First of all, each object has to have a starting position and ending position data. Moreover, having these positions we can then identify object size. Due to the fact that map can be represented as a two dimensional array the starting point and end point could contain the indexes of such array in order to know where the object was placed. In the Figure 3.6.1 Array Representation it is showed how two dimensional array representation would look like.



*Figure 3.6.1 Array Representation*

The key point in placing the object into the map is to know the starting position and the size of the object that is being placed by having this we can then calculate the ending position with Equation 3.6.1.

$$Pos_{end} = (Pos_{start}.x + (size.x - 1), Pos_{start}.y + (size.y - 1))$$

*Equation 3.6.1*

The next thing to have in mind is to remember that due to the fact 2.5D graphical representation is chosen each object will be represented as a polygon as well. By having this in mind we need to calculate the polygon for the objects that will be the base for it. To do so it needed to calculate four points of the polygon and this done by this Equation 3.6.2.

$$\begin{aligned}
TopP_{obj} &= \left( TopPoint.x - x * (Pos_{start}.y - Pos_{start}.x), TopPoint.y + \frac{x}{2} * (Pos_{start}.y + Pos_{start}.x) \right) \\
LeftP_{obj} &= \left( (TopP_{obj}.x - x * size.x), (TopP_{obj}.y + \frac{x}{2} * size.y) \right) \\
BottomP_{obj} &= \left( (LeftP_{obj}.x + x * size.x), (LeftP_{obj}.y + \frac{x}{2} * size.y) \right) \\
RightP_{obj} &= \left( (BottomP_{obj}.x + x * size.x), (BottomP_{obj}.y - \frac{x}{2} * size.y) \right)
\end{aligned}$$

Equation 3.6.2

By doing this we know can as well show the user where the object he is trying to place will be placed. Nevertheless, for this we as well need to do some collision testing, because objects cannot be placed outside the map. The main checks for is to be sure that the object is trying to be placed do not intersect with the grid boundary meaning they not collide. Moreover, to check that the object is being placed inside the map. This is easy to achieve, because of the implementation we just need to check that the cursor is inside the map boundary. The last task is to check whether or not by placing this object it might collide with other objects. This can be done by checking if the tiles that the object is trying to occupy are not linked to any other objects. By completing this task we can now display user where his object is going to be placed at and whether or not it a valid position. If position is valid for chosen object it the area will be highlighted in green if not that area will be highlighted in red. The example is given in Figure 3.6.2 and code can be found in Code 1.

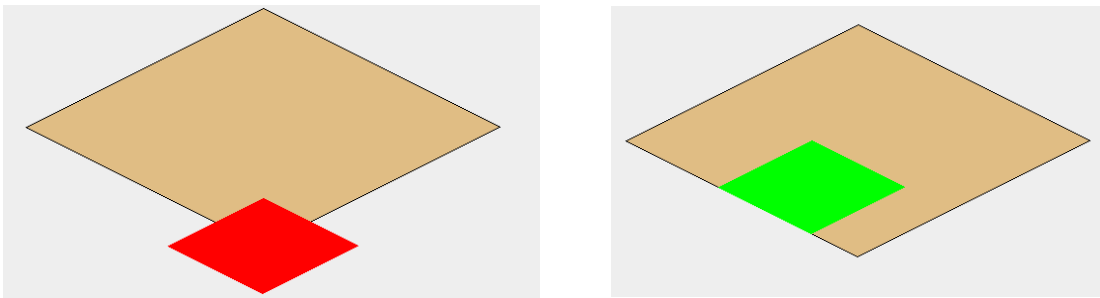
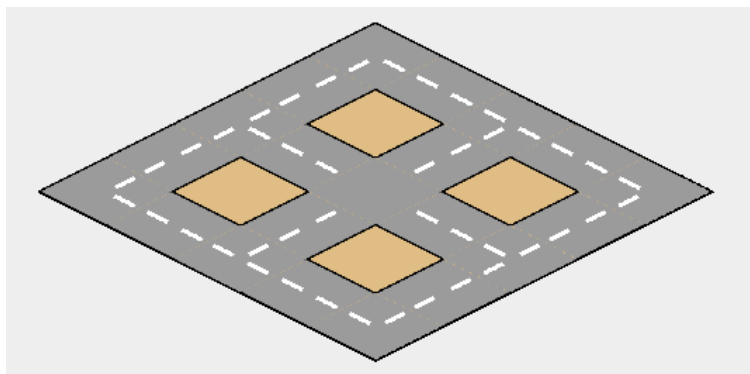


Figure 3.6.2 Collision Detection

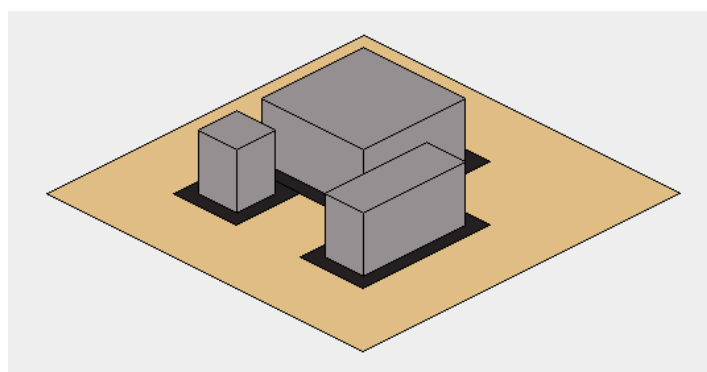
The road visualization in this system will be displayed from the image file, which will be adjusted that it would fit the required area. Moreover, road objects will adjust by each other in order to make the visualization more realistic. To complete this each road object has to check whether or not there are other road objects next to it. After doing this the texture for current object is chosen. Moreover, the neighbours of new road object that was place has to do the same check in case they need to be visualized using different image. The code for

this implementation is provided in Code 2. The visualization of such road is provided below in Figure 3.6.3.



*Figure 3.6.3 Road Visualization*

The next step to follow is to visualize the building that are going to be placed by the user. Due to the fact that building size can vary by the user input these objects cannot be painted as an image and has to be generated. Due to the fact that we know the base of the object that is represented as a polygon and was calculated with Equation 3.6.2. Having this in mind we need then to create the altitude for the building. This is done by creating lines from all the polygon points. The length of each line will be stored as a constant. Furthermore, then we just need to calculate the ending position of each line and connect the points in order to get a polygon, which would be a roof of the object. By implementing this the representation of objects will look as given in Figure 3.6.4.



*Figure 3.6.4 Building Visualization*

Moreover, due the fact that map is represented in 2.5D graphics the order in which objects will be drawn is of great importance. This is, because some objects will have to cover other objects that are in the visual perspective further away. If the objects would be painted in a single block at the time the solution could be just to draw the objects starting from the top

then going left bottom and after completing the row moving to the next one on the right. Nevertheless, the objects might not be a single block size, which means there is a need to write an algorithm in, which would order the drawing of the objects. In addition, before starting to draw the objects they should have an instance, which would indicate whether or not they have been painted already. The code for this implementation is provided Code 3 Map Object Painting.

### **3.7 Optimisation Algorithm Development and Implementation**

The development and implementation of optimisation algorithm for this tool is the hardest problem due to the problem origin. The development of the optimal algorithm began with trying to find the optimal solution in brute force. The idea of such implementation is to try and place house objects every way possible and look for the best solution until all the variation are tested. This sort of problem approach was actually very successful for finding the optimal solutions, because all possibilities are tried. Nevertheless, it is important to have in mind the time complexity of such approach. Even though, optimal solution can be obtained every time the run time increases exponentially by either increasing map size or the number of house types that can be used. So even though the computation will always provide an optimal solution the actual result might not arrive for a long time. Having this in mind it is reasonable to say that this sort of problem approach is not acceptable.

After not succeeding in a first attempt to design reasonable algorithm for optimizing the map it is important to understand why this attempt failed. Even though, in theory it provided optimal solutions the run time for computing it was too long. Having this in mind it is important to lower the time complexity in the next attempt that is going to take place. The simplest way to do it would be try the greedy approach optimization. To develop a greedy algorithm we need to find a way to compare building types in a way that certain type would be better than other. This could be done by either comparing the area, the value or the value for the single square of the building. As it was discussed before the optimization should find the solution that has as highest value as possible. Having this in mind, comparing buildings by area is rejected. Moreover, comparing by value or by single square value are both appropriate ways to compare. Nevertheless, the single square value comparison is in a sense a bit better, because it as well takes into the consideration house area. For this reason, this building evaluation approach is chosen. What is more, it is important how the objects will be packed. The good solution for this greedy approach would be start looking for possible position for current type of object from top to left then moving right to next row. After placing this way as many one type houses as possible the algorithm would move to the next type building. The run time for finding the solution is very reasonable and the solution will be found very fast. Nevertheless, due to the fact we made run time very good the solutions were a bit less optimal as it was wanted. Due to this reason, there was conducted another attempt, which would combine first and second approach.

From the previous approaches it is possible to see that run time is kind of linked to the foundation of optimal solution. In other words the longer the run time the more optimal solution can be found. Having this in mind it is reasonable to try and make middle approach for the previous ones. The basic idea of this approach is to try finding the solution that is the most optimal by considering only the area of the map and houses. After finding such solution the greedy packing algorithm then would try to see if it actually possible to place that number of objects listed in founded solution. Even though, greedy packing does not guarantee optimal solution it might find it in some cases. The finding of the most optimal possible solution was implemented in a brute force way, there all the possible solutions are calculated. This is achieved by calculating the maximum number of times that each type of object can fit in the map by area and then conducting checking all possible variations and leaving only the ones that do not exceed map area. In a way it is an exponential calculation, but it is much faster than the one implemented in approach number one. The algorithm for such calculation would be as shown in Figure 3.7.1.

1. Create a list what contains the maximum number of types the building can be placed inside the map by area.
2. Calculate all possible solutions between maximum number that do not exceed map area.
3. Sort the solutions by the profit it would give.

*Figure 3.7.1 Algorithm for calculating all solutions.*

After doing this we then start trying to pack any of these solutions from the most valuable to the least. The first solution that can be packed will be the result given by this approach. After implementing this approach the results provided by the tool seems highly optimal and the run time is quite reasonable as well. The actual results can be found in Chapter 4:

## **Chapter 4: Results**

### **4.1 Introduction**

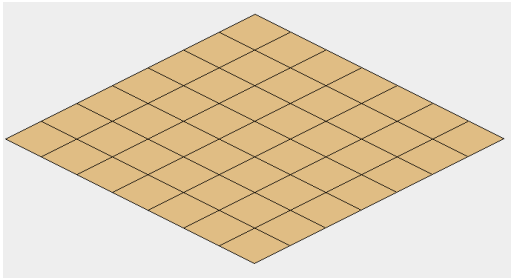
This chapter will show the 2D packing results that were retrieved from the implemented system. Moreover, these results will be analysed by the run time taken and by the optimal solution that should have been produced. Finally, the results will be showed with the graphical representation that the system provided.



## 4.2 Actual Results

### Test 1

The map size for this test is chosen to be 7 by 7 as showed in *Map 1*.

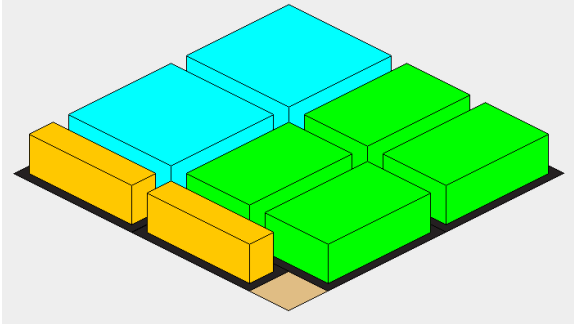


Map 1

Id	Color	Width	Height	Value	Rotate
1	<span style="color: red;">■</span>	1	2	1	<input checked="" type="checkbox"/>
2	<span style="color: orange;">■</span>	1	3	1	<input checked="" type="checkbox"/>
3	<span style="color: yellow;">■</span>	2	2	3	<input type="checkbox"/>
4	<span style="color: green;">■</span>	2	3	5	<input checked="" type="checkbox"/>
5	<span style="color: cyan;">■</span>	3	3	8	<input type="checkbox"/>

Buildings Data 1

Optimal Value	Found Value	Difference	Calculation time
40	40	0%	114 sec.



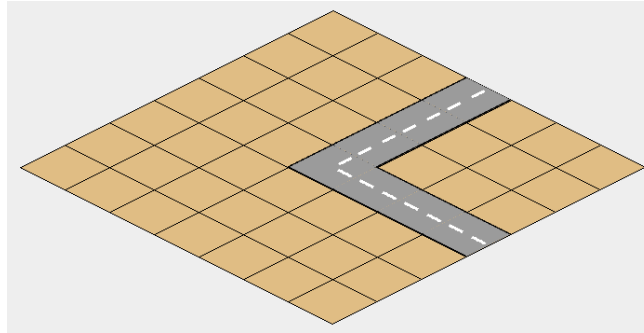
Results 1

Graphical Representation 1

During this test the optimal solution was found, which shows that the implemented algorithm is capable to find best solution. Moreover, the run time for calculating this solution was considerable short as well. Due to the facts named above we can state that *Test 1* was successful.

## Test 2

The *Test 2* was conducted using same map as it was used in *Test 1*, nevertheless the map contained some road objects before the optimization took place. The graphical representation for such map is showed below *Map 2*.

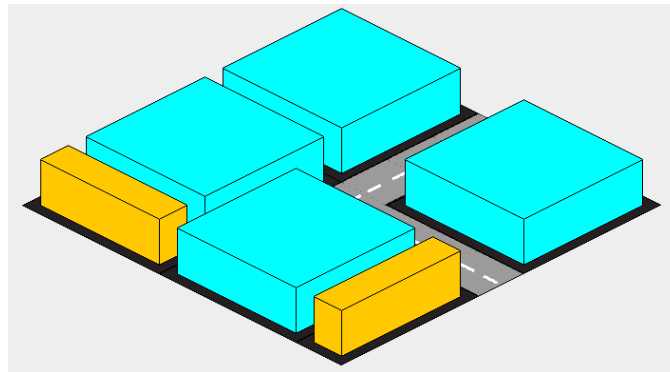


*Map 2*

Furthermore, the building data was chosen to be same as it was in *Test 1*, which can be found in *Buildings Data 1*.

Optimal Value	Found Value	Difference	Calculation time
36	36	0%	18 seconds

*Results 2*

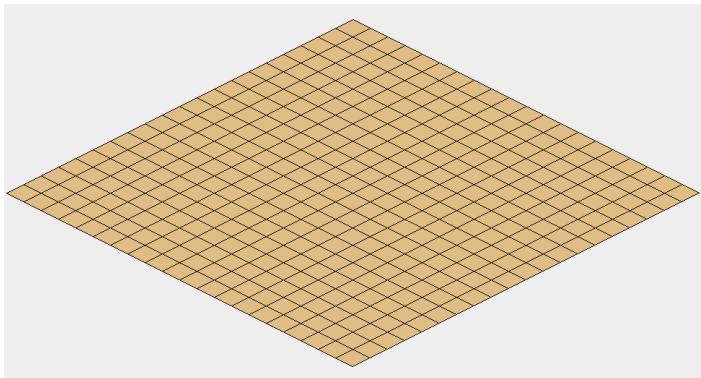


*Graphical Representation 2*

The optimal solution, while conducting *Test 2* was found. The run time for this test was short and the solution was found in seconds. What is more, it is reasonable to state that the tool works well when the map already has some objects placed inside it before the optimization.

**Test 3**

The map size for this test was chosen considerably much larger than for previous ones. As showed in *Map 1* which is 20 by 20.



*Map 3*

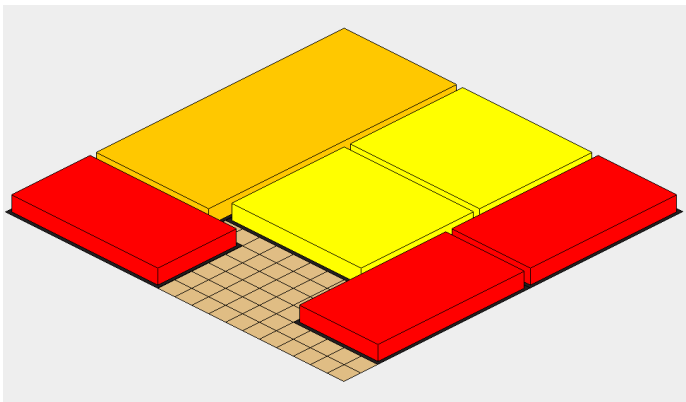
Furthermore, the buildings for this test are chosen to be larger than they were in previous tests as well. The house data is showed below in *Buildings Data 2*.

Id	Color	Width	Height	Value	Rotate
1	<div></div>	9	5	50	<input checked="" type="checkbox"/>
2	<div></div>	15	7	135	<input checked="" type="checkbox"/>
3	<div></div>	7	8	70	<input checked="" type="checkbox"/>

*Buildings Data 2*

Optimal Value	Found Value	Difference	Calculation time
425	425	0%	1 second

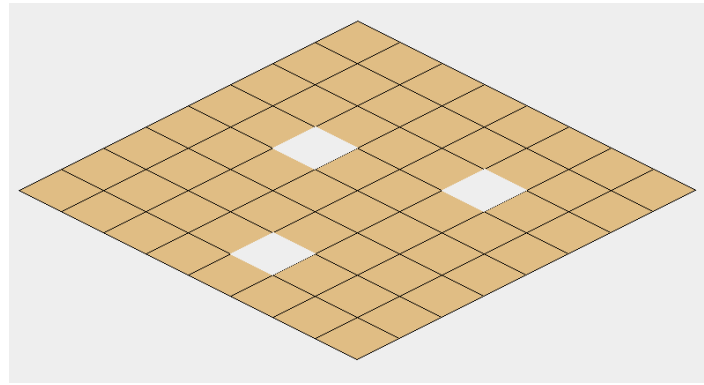
*Results 3*



During this test the tool managed to provide an optimal solution in just a second. This sort of short computation was achieved due to the fact that the number of possible solutions was small. Nevertheless, it is reasonable to say that the algorithm was well then the map and houses are defined to be large size.

#### **Test 4**

The map size for this test was chosen to be 8 by 8. What is more some of the map area was deleted. The map is showed below in *Map 4*.

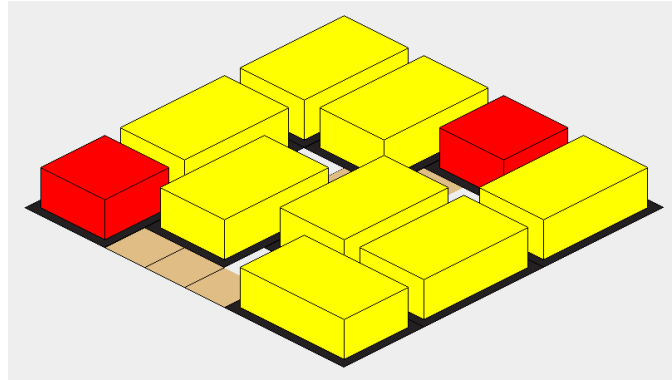


*Map 4*

The available houses for this optimization were chosen to be similar. The data table for this is showed below.

Id	Color	Width	Height	Value	Rotate
1	<div></div>	2	2	6	<input type="checkbox"/>
2	<div></div>	2	3	10	<input checked="" type="checkbox"/>

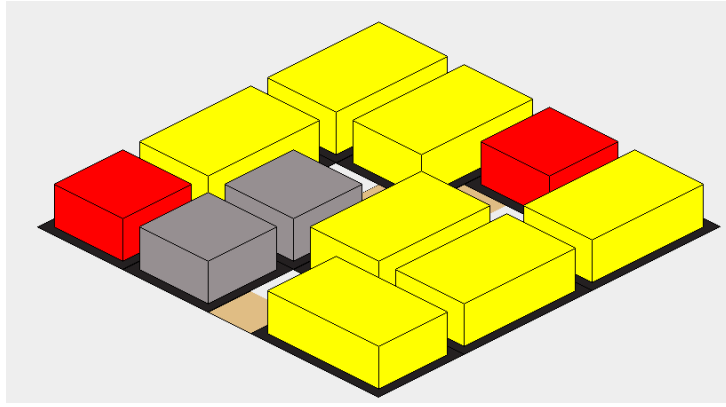
*Buildings Data 3*



*Graphical Representation 4*

Optimal Value	Found Value	Difference	Calculation time
94	92	2%	1 second

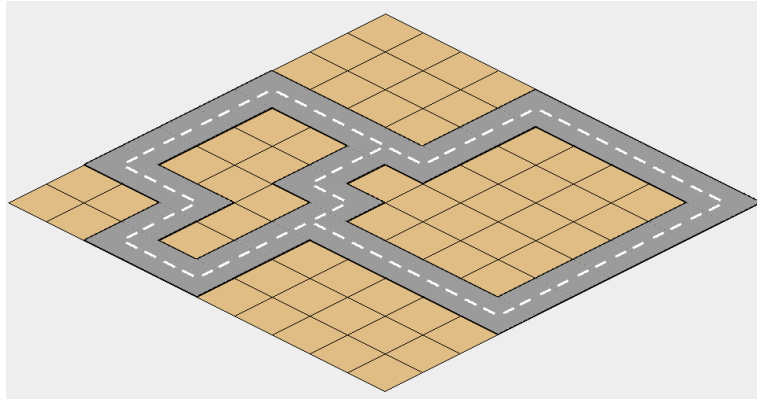
The results retrieved shows that the *Results 4* algorithm did not managed to produce the most optimal solution. Nevertheless, it did produce a solution, which is highly optimal and the value difference from the optimal solution is only 2%. What is more, the time taken to compute this solution was under a second. Even though, the optimal solution was not provided during this test the results of it were very good in a sense that the solution was provided in a second and the solution that was provided was only 2% lower. The optimal solution is displayed below.



*Representation of Optimal Solution 1*




### Test 5

The map for this was chosen to be 10 by 10. What is more, there was placed some road objects in it to simulate the possible map of the city that user might be having as a starting point. The visual representation of this map is displayed below in *Map 5*.



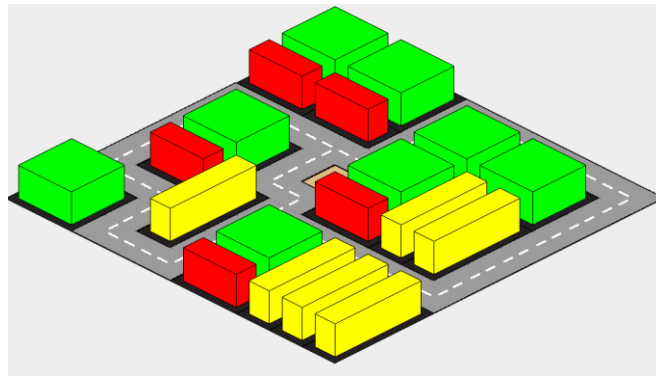
*Map 5*

The building types for optimization were chosen to be considerably small by the size. The data of the buildings is showed below.

Id	Color	Width	Height	Value	Rotate
1		1	2	3	<input checked="" type="checkbox"/>
2		1	3	5	<input checked="" type="checkbox"/>
3		2	2	7	<input type="checkbox"/>

*Buildings Data 4*

Due to the fact that map size chosen is considerably large and the house objects are chosen to be relatively small it is hard to know what is the optimal solution is. For this reason, we cannot compare the value difference between optimal and computed solution. Even though, the optimization solution is complicated for this test the result was computed by the tool in just 20 seconds and the value obtained was 101. The graphical representation for this solution is provided below in *Graphical Representation 5*.



*Graphical Representation 5*

The solution retrieved from this test seems to be highly optimal if not optimal. Moreover, the run time is quite short and is reasonable to the complexity of the test.



### **4.3 Conclusion**

The tests results provided in 4.2 were of high quality considering that most of them were able to provide optional or high optional solutions. Moreover, the run time that was spent computing these solutions were reasonable and did not required waiting for a long time. Nevertheless, there are some limitations for this tool, because calculation of all possible solutions is exponential. This means that the run time would increase rapidly if the map was enlarged. However, due to the complexity of the problem there is a need to make some sacrifices either making the run time longer or making the solution less optimal. For the reasons mentioned above it is reasonable to say that using this tool highly optimal solutions for two-dimensional packing are achieved in a reasonable run time.

# **Chapter 5: Conclusion**

## **5.1 Introduction**

This chapter presents a conclusion to the project aim and objectives outlined in Chapter 1:. The conclusion will be justified based on the implementation of the system described in Chapter 3:, and the analysis of testing data presented in Chapter 4:.

## **5.2 Satisfaction of the Aim and Objectives**

The aim of the project was to develop system for area planning. This was split into several objectives, which are going to be discussed.

### **Objective One**

To design a graphical user interface for the system that will be used for area planning.

The implementation and development for graphical user interface was discussed in 3.4. This objective was successfully achieved and the evidence of completing it successfully was presented in 3.4.

### **Objective Two**

To implement the basic functionality for this tool, such that user could create a map and interact with it by placing various objects.

This functionality was as well split in in smaller requirements in the 1.4 and the implementation for these functionalities were discussed in 0. This objective was fully completed due to the fact that the tool allowed user to place various objects inside the map. Moreover, user is as well informed by graphical representation whether or not the object can be placed in selected area. The evidence of this feature is shown in Figure 3.6.2 Collision Detection.

### **Objective Three**

To implement basic graphical functionality that would allow user to adjust how the map and its objects are displayed.

This objective was completed successfully and the implementation of this functionality was discussed in 0. Moreover, the view where the map is displayed as well can be scrolled. For this reason, user can traverse through the map if the currently defined map does not fit in the view.

### **Objective Four**

To implement a feature to this tool, which would allow to perform optimization, which would compute optimal or highly optimal solution.

The implementation and design of optimization algorithm, which required to complete this objective is discussed in 3.7. Moreover, after algorithm was implemented to the system the testing was done in order to check if the system provides optimal or highly optimal solution. The testing data and results are provided in 4.2.

Even though, the tests that been carried showed very good results, this algorithm is still limited. In theory, highly optimal result will be calculated with any types of input. Nevertheless, practically certain types of data input might take very long time and sometimes it is even impossible to tell how long it might take. Having this in mind, this objective was only completed partially.

### **Aim**

Even though first three objectives were completed fully the last one and the most difficult one was completed partially. Therefore, it reasonable to state that the aim of the project was satisfied partially.

### **5.3 Overall Conclusion**

The system for this project was created successfully and by providing some promising results. Moreover, most of the objectives were achieved without rising any problems. Nevertheless, the objective that required implementing the feature for area optimization was only satisfied partially. However, due to the reason this is NP-hard problem succeeding to complete this objective fully might not be even possible. In conclusion, the work done during this project was not completed fully, due to the difficulty of the problem and it will lead into further research of the problem.

### **5.4 Future Work**

The future work for this project will be aimed on improving the currently implemented tool. Even though, current system is capable of calculating highly optimal solutions it still has some limitations that are due to the reason that calculating all possible solutions is exponential computation. In theory, to avoid such computation we could take into account dynamic programming. Using dynamic programming it is possible to use branch and bounces algorithm, which would be trying to calculate the optimal solution in dynamic way and if the problem would fail it would try the next optimal solution that is calculated dynamically. Even thought, in a worse case this approach would give same computation as it is implemented currently; other cases would calculate solution much faster. What is more, currently implemented 2D packing heuristic might be studied even more in order to improve it or suggest other heuristic in order to improve the optimization.

Furthermore, the current system could as well be improved by making the visualization in 3D. By doing this the user would be able to receive better looking graphical representation. Moreover, tool could be made to allow user define its own objects in graphical level. For example, allow user to create building models that would be more realistic to the ones that he is planning to build.

Moreover, the tool could be adapted not only to be used for area planning purposes, but as well for other reasons that would involve 2D packing problem. For example, the tool could be adjusted so that the user could be allowed to solve material cutting or transportation problems.

Finally, this project could be as well extended to 3D area planning, which would as well take into consideration the building altitude. Nevertheless, this would involve studying the 3D

packing problem, which is even harder than 2D one. For this reason, such project extension would require a long time till reasonable results would be visible.

# References

- [1] Unknown, "OverBlog," 2014. [Online]. Available: <http://elouanmcguir1127.over-blog.com/pages/importance-of-urban-planning-7753398.html>. [Accessed 2014].
- [2] "Stackoverflow," 09 12 2013. [Online]. Available: <http://stackoverflow.com/questions/1857244/np-vs-np-complete-vs-np-hard-what-does-it-all-mean>. [Accessed 2014].
- [3] British German Academic Research, "British German Academic Research Collaboration Programme," 11 04 2014. [Online]. Available: <http://cgi.csc.liv.ac.uk/~epa/index.html>. [Accessed 2014].
- [4] Nthabiseng Ntene and Jan van Vuuren, "2D Strip Packing Problem," 2008. [Online]. Available: <http://dip.sun.ac.za/~vuuren/repositories/levelpaper/spp%5B1%5D.htm>. [Accessed 2014].
- [5] J. v. V. N. Ntene, A survey and comparison of level heuristics, Republic of South Africa, 2006.
- [6] H. Smith, "Bin Packing," 2001. [Online]. Available: <http://users.cs.cf.ac.uk/C.L.Mumford/heidi/FFDHquarters.html>. [Accessed 2014].
- [7] H. Smith, "Bin Packing," 2001. [Online]. Available: <http://users.cs.cf.ac.uk/C.L.Mumford/heidi/NFDHquarters.html>. [Accessed 20014].
- [8] Astrokettle Algorithms , "Astrokettle Algorithms Software Products for Engineering Optimization," Astrokettle Algorithms , [Online]. Available: <http://www.astrokettle.com/prod.html>. [Accessed 2014].
- [9] K. Binkley, "Softpedia," 2012. [Online]. Available: <http://www.softpedia.com/get/Science-CAD/2D-Packing-Solver.shtml>. [Accessed 2014].
- [10] T. Nagy, "Linpacker," 31 10 2006. [Online]. Available: <http://www.freehackers.org/~tnagy/linpacker.html>. [Accessed 2014].
- [11] M. Pegg, "Google Maps Mania," 2007. [Online]. Available: <http://googlemapsmania.blogspot.co.uk/2007/04/google-maps-adds-25d-buildings-and.html>. [Accessed 2014].
- [12] Unknown, "Wikipedia," [Online]. Available: [http://upload.wikimedia.org/wikipedia/commons/e/ea/Sprite\\_anatomy\\_2d.svg](http://upload.wikimedia.org/wikipedia/commons/e/ea/Sprite_anatomy_2d.svg). [Accessed 2014].
- [13] K. &. Ritchie, The C Programming Language (2nd edition), Prentice Hall, 1988.

- [14] A. S., "Cplusplus," 2013. [Online]. Available: <http://www.cplusplus.com/info/history/>. [Accessed 2014].
- [15] Wikibooks, C Sharp Programming, 2013.
- [16] Act-3D, "Lumion3d," Act-3D, 2013. [Online]. Available: <http://lumion3d.com/urban-planning-software/>. [Accessed 2014].
- [17] CAD International, "CAD town planning," 1998. [Online]. Available: <http://www.cad.com.au/by-category/outside/town-planning.htm>. [Accessed 2014].
- [18] Citilabs, "Citilabs transportation planning, modelling traffic engineering, GIS and urban planning," 2013. [Online]. Available: <http://www.citilabs.com/>. [Accessed 2014].
- [19] Urban Analytics, "UrbanSim planning and analysis of urban development," 2013. [Online]. Available: <http://www.urbansim.org/Main/WebHome>. [Accessed 2014].
- [20] Oracle, "Java SE Documentation," 1993-2014. [Online]. Available: <http://docs.oracle.com/javase/7/docs/api/javax/swing/JMenuBar.html>. [Accessed 2014].
- [21] Oracle, "Java SE Documentation," 1993-2014. [Online]. Available: <http://docs.oracle.com/javase/7/docs/api/javax/swing/JScrollPane.html>. [Accessed 2014].
- [22] Oracle, "Java SE Documentation," 1993-2014. [Online]. Available: <http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.html>. [Accessed 2014].
- [23] Oracle, "Java SE Documentation," 1993-2014. [Online]. Available: <http://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html>. [Accessed 2014].





# Code

## Code 1 Collision detection

```
public static boolean intersects(Polygon obj, Polygon gridBoundary)
{
    Line2D[] objLines = new Line2D[obj.npoints];
    Line2D[] gridLines = new Line2D[gridBoundary.npoints];

    objLines[0] = new Line2D.Double(obj.xpoints[0], obj.ypoints[0] + 1,
    obj.xpoints[1] + 1, obj.ypoints[1]);
    objLines[1] = new Line2D.Double(obj.xpoints[1] + 1, obj.ypoints[1],
    obj.xpoints[2], obj.ypoints[2] - 1);
    objLines[2] = new Line2D.Double(obj.xpoints[2], obj.ypoints[2] - 1,
    obj.xpoints[3] - 1, obj.ypoints[3]);
    objLines[3] = new Line2D.Double(obj.xpoints[3] - 1, obj.ypoints[3],
    obj.xpoints[0], obj.ypoints[0] + 1);

    for (int i = 0; i < gridLines.length; i++)
    {
        gridLines[i] = new Line2D.Double(gridBoundary.xpoints[i],
        gridBoundary.ypoints[i], gridBoundary.xpoints[(i + 1)
        % gridBoundary.npoints],
        gridBoundary.ypoints[(i + 1) % gridBoundary.npoints]);
    }
    for (int i = 0; i < objLines.length; i++)
    {
        for (int j = 0; j < gridLines.length; j++)
        {
            if (objLines[i].intersectsLine(gridLines[j]))
            {
                return true;
            }
        }
    }
    return false;
}

public static boolean insideBounds(Tile[][] grid, Point start, Point end)
{
    return (start.x < grid.length && start.y < grid[start.x].length
    && end.x < grid.length && end.y < grid[end.x].length);
}

public static boolean noCollision(Tile[][] grid, Point start, Point end)
{
    for (int i = start.x; i <= end.x; i++)
    {
        for (int j = start.y; j <= end.y; j++)
        {
            if (grid.length <= i || grid[i].length <= j)
            {
                return false;
            }
            else if (grid[i][j].getObject() != null)
            {
                return false;
            }
            else if (!grid[i][j].isAvailable())
            {
                return false;
            }
        }
    }
    return true;
}
```

## Code 2 Finding road texture

```
private void findTexture(HapObject obj, int x, int y, String text, boolean iterate)
{
    if (obj instanceof Road)
    {
        String result = text;
        // 0 - Bottom, 1 - Top, 2 - Right, 3 - Left
        HapObject[] nearObjects = getNearObjects(x, y);

        result = nearObjects[0] != null && (nearObjects[0] instanceof Road) ? result.replace("B", "") : result;
        result = nearObjects[1] != null && (nearObjects[1] instanceof Road) ? result.replace("T", "") : result;
        result = nearObjects[2] != null && (nearObjects[2] instanceof Road) ? result.replace("R", "") : result;
        result = nearObjects[3] != null && (nearObjects[3] instanceof Road) ? result.replace("L", "") : result;

        switch (result)
        {
            case "":
                result = "C";
                break;
            case "LRTB":
                result = "Basic";
                break;
            case "RTB":
                result = "TB";
                break;
            case "LTB":
                result = "TB";
                break;
            case "LRT":
                result = "LR";
                break;
            case "LRB":
                result = "LR";
                break;
        }
    }
    Road temp = (Road) obj;
    temp.setTexture(result);
    if (iterate)
    {
        for (int i = 0; i < nearObjects.length; i++)
        {
            if (nearObjects[i] != null)
            {
                if (nearObjects[i] instanceof Road)
                {
                    Point cord = nearObjects[i].getStartPos();
                    String tempText = text;
                    tempText = i == 0 ? tempText.replace("T", "") : tempText;
                    tempText = i == 1 ? tempText.replace("B", "") : tempText;
                    tempText = i == 2 ? tempText.replace("L", "") : tempText;
                    tempText = i == 3 ? tempText.replace("R", "") : tempText;
                    findTexture(nearObjects[i], cord.x, cord.y, tempText, false);
                }
            }
        }
    }
}

private HapObject[] getNearObjects(int x, int y)
{
    HapObject[] nearObjects = new HapObject[4];

    nearObjects[0] = x < grid.length-1 ? grid[x+1][y].getObject() : null;
    nearObjects[1] = x > 0 ? grid[x-1][y].getObject() : null;
    nearObjects[2] = y > 0 ? grid[x][y-1].getObject() : null;
    nearObjects[3] = y < grid[x].length-1 ? grid[x][y+1].getObject() : null;

    return nearObjects;
}
```

Code 3 Map Object Painting

```
for (int i = 0; i < grid.length; i++)
{
    for (int j = 0; j < grid[i].length; j++)
    {
        MapObject temp = grid[i][j].getObject();

        if (temp != null)
        {
            preparePaintObject(g, temp);
        }
    }
}

private void preparePaintObject(Graphics g, MapObject mapObject)
{
    if (!mapObject.isPainted())
    {
        if ((mapObject.getTileSize().height > 1) && mapObject.getStartPos().y != 0)
        {
            for (int i = 0; i < mapObject.getStartPos().y; i++)
            {
                for (int j = mapObject.getStartPos().x + 1; j <= mapObject.getEndPos().x; j++)
                {
                    if (grid[j][i].getObject() != null)
                    {
                        preparePaintObject(g, grid[j][i].getObject());
                    }
                }
            }
        }
        if ((mapObject.getTileSize().width > 1) && mapObject.getStartPos().x != 0)
        {
            for (int i = 0; i < mapObject.getStartPos().x; i++)
            {
                for (int j = mapObject.getStartPos().y + 1; j <= mapObject.getEndPos().y; j++)
                {
                    if (grid[i][j].getObject() != null)
                    {
                        preparePaintObject(g, grid[i][j].getObject());
                    }
                }
            }
        }
        paintObject(g, mapObject);
    }
}
```