Improving The Programming Logic of Children by Creating Computer Games on Greenfoot Framework

Yunus Emre Aziz
2011-01-17
Subject: Software Technology
Level: Master
Course code: 4DV01E
Abstract

In our modern time, age of education is decreasing day by day. People have started to learn more advanced things at earlier ages. This fact is also true for children. They are learning more advanced thing now because they are facing new technology in every aspect of life.

Today, one of the most popular subject is computer programming so it is also one of the most popular technologies to investigate, inevitably. One reason to such popularity is that computer programming is very important in every field of life or business and it seems that this importance will improve with time. Certain levels of logic and synthesis are needed for programming. It is crucial to improve these skills.

This master thesis tries to improve and tame the programming abilities of children and strengthen the knowledge about the basic concepts of mathematics and physics, which are related to the computer science by creating simple computer games using Greenfoot Framework.

In this project, tutorials like approach was used. Designs of the games was studied as tutorials and at the end of the last tutorial, two tests and a survey were applied.

According to the results, purpose about improving and taming the programming abilities of children is achieved. For the other purposes, these are increases around 8% and 16%. Since they are not dramatic increases and the target number is 6 I can not say clearly that these purposes are also achieved.

Keywords: Computer Science Education, Basic Game Programming, Greenfoot, Introductory Programming.
Table of Contents

1 Introduction .................................................................................................................. 1
  1.1 Purpose .................................................................................................................... 1
  1.2 Methodology and Criteria ......................................................................................... 1
  1.3 Selection of The Target Group ................................................................................... 1
  1.4 Structure of The Report ............................................................................................ 2

2 Motivation and Tools ........................................................................................................ 3
  2.1 Playing Games, Children and Learning ...................................................................... 3
  2.2 Why Use Game Programming .................................................................................... 3
  2.3 Tools of The Trade ..................................................................................................... 4
  2.4 Tools of The Trade Java ............................................................................................ 4
  2.5 Tools of The Trade BlueJ, Karel The Robot and Greenfoot ........................................ 4

3 Introduction to Programming Language Features and How to Use Them .................. 7
  3.1 Problem of Which Features to Use ............................................................................. 7
  3.2 Programming Languages .......................................................................................... 7
  3.3 Programming Language Features ............................................................................. 7
  3.4 Selected Features from Programming Languages ..................................................... 9

4 Game Programming Requirements .................................................................................. 11
  4.1 Programming Language Requirements ..................................................................... 11
  4.2 Specialized Requirements ......................................................................................... 11

5 Game Programming Tutorials ......................................................................................... 12
  5.1 Tutorial Like Approach ............................................................................................. 12
  5.2 Tutorial Contents ....................................................................................................... 12
    5.2.1 Greenfoot Introduction Tutorial ........................................................................... 13
    5.2.2 Simple Car Game 1 Tutorial ............................................................................... 14
    5.2.3 Simple Car Game 2 Tutorial ............................................................................... 14
    5.2.4 How to Write onto The World of Greenfoot Tutorial .......................................... 15
    5.2.5 Find the Correct Traffic Sign Tutorial ................................................................. 16
    5.2.6 Collect the Objects Tutorial ................................................................................. 17
  5.3 Programming Language Features Learned in Tutorials ........................................... 18
  5.4 Results and Discussion .............................................................................................. 19

6 Conclusion and Further Studies ...................................................................................... 22

7 References .......................................................................................................................... 23

A Appendices ...................................................................................................................... 26
  A.1 Tutorials .................................................................................................................... 26
    A.1.1 Greenfoot Introduction Tutorial .......................................................................... 26
    A.1.2 Simple Car Game 1 ............................................................................................. 43
    A.1.3 Simple Car Game 2 ............................................................................................. 58
    A.1.4 Writing Tutorial .................................................................................................. 75
    A.1.5 Find The Correct Traffic Sign Tutorial .............................................................. 91
    A.1.6 Collect The Objects Tutorial ............................................................................. 108
  A.2 Test ............................................................................................................................ 139
  A.3 Survey ........................................................................................................................ 140
List of Figures

Figure 2.1: BlueJ ................................................................................... 5
Figure 2.2: Karel The Robot ................................................................. 6
Figure 2.3: Greenfoot .......................................................................... 6
Figure 5.1: Tutorial 1 - Greenfoot Introduction .................................... 13
Figure 5.2: Tutorial 2 - Greenfoot Simple Car Race 1 ....................... 14
Figure 5.3: Tutorial 3 - Greenfoot Simple Car Race 2 ....................... 15
Figure 5.4: Tutorial 4 - How to Write onto The World of Greenfoot ... 16
Figure 5.5: Tutorial 4 – Find the Correct Traffic Sign ....................... 17
Figure 5.6: Tutorial 4 – Collect the Correct Objects ......................... 18
Figure 5.7: Results of the Survey ....................................................... 19
Figure 5.8: Results of the Test ........................................................... 20
1 Introduction

In this chapter, purpose of this project will be introduced. Also information will be given about which concepts and methods will be followed and how the selection of the target group is made to reach the purpose of this project.

1.1 Purpose

Purpose of this project is to teach programming to children via game programming. By doing this it is aimed to teach fundamentals of programming and to reinforce some basic concepts of mathematics such as logic and coordinate system concept. Why teach programming? It strengthen thinking ability, problem solving, analytical thinking and logic skill. Also it is a really good assistant for teaching basics of mathematics and physics (Kurland, et al). Also, today, mathematical logic is highly related with computer science (Gurevich, Y).

Computer literacy has become part of our everyday lives. From 4-90 years old, almost everybody knows somethings about computers. We seem to be unable to live without computers. Why use game programming to teach programming? Teaching by using computer games increase motivation, create good learning atmosphere and increase success in the lectures (Lee, J, et al). According to Entertainment Software Association, big percentage of people, 30 years old or younger, frequently playing games and surprisingly, women cover 45% of all game players (ESA). Programming is to use computers to accomplish tasks. When graphical interfaces are not enough we need to use programming to accomplish tasks be it a scripting or a real time programming. For all of these reasons, it seems suitable to teach programming via game programming.

1.2 Methodology and Criteria

To reach my purpose, firstly I will choose a suitable programming language and a platform to work on it. It should be convenient for the target group and also it should satisfy the properties I will use. Then, I will decide what scenario(s) I will develop to represent the concepts I have mentioned in the purpose section. After that, I have to test my work. To do this I will collect test results, before and after applying the tutorials to the target group. Finally, I will interpret the results.

The main purpose about improving the programming logic of the children is reached if the results of the given survey is successful enough. Also, the other purposes about the improving the knowledge about the basic concepts of the science is reached if the test results of the given tests are enough.

1.3 Selection of The Target Group

When we come to the problem how to select the students to apply the tutorials, I can say that they should be selected according to the certain criteria.

Today, most-known programming languages are using English Language so target students should know English enough to understand the structures and keywords used by the selected programming language. Also they should have knowledge about the coordinate system and basic logic so that they will have no problem to understand the coordinate system of the Greenfoot World and control structures. In addition, they should have interest in computers and games and should be eager to study the tutorials. Otherwise, because of the lack of motivation, students can be easily get bored and this
can affect the results of this project.

For these reasons, 8th grade students, who are willing to participate, from private schools in which lectures are in English are selected as our target group. According to the curriculum 8th grade students already took lectures about coordinate system and the basic logic. At this point I want to mention that during this selection process random students were selected. They are random students who are 8th grades and know enough English to understand the key structures like “if”, “for”, etc. If we would not use a random selection, for example if we would select the best students in the classes, there can be some side effects in the results of this project.

1.4 Structure of The Report
After introducing the project in the first chapter, second chapter will continue with the motivation of it. In third and fourth chapters, background information about the the concepts used in this work will be introduced. In the fifth chapter, game designs will be presented as tutorials and also survey and tests and their results will be discussed. Finally, in sixth chapter conclusion about this project will be mentioned.
2 Motivation and Tools
In this chapter information about the motivations and the tools used in this project will be introduced.

2.1 Playing Games, Children and Learning
Most people especially children like to play games. This fact can be seen in statistics. It is indicated that roughly 70% of children playing computer games every week (Facer, K). Game Industry become a giant of its own in recent years reaching the Hollywood film industry in its revenue (Thomas, Adam).

Although some people, especially parents, think computer games are harmful for children, games can be used as an effective learning tool. By using computer games, users can affect normally not-changeable things and observe the results according to these changes. Users can put themselves into different roles in computer games and gain different point of view. Computer games can make the time goes faster so users can observe the changes in a system or environment normally not possible to observe in such a short time (Squire, K).

With the development of object-oriented programming it is easy to make modeling of the real world so with the examples from real world situations, it becomes easier to draw students attention.

The most critical element of proper learning is motivation. Complicated and technical subjects can be very boring for students because generally it is a bit hard to give motivation to students about these subjects. Computer games are giving really good motivation to children so it is really a good idea to include computer games in learning (Prensky, Marc).

Games teach children differently from formal and structured way. Most of the time children do not realize learning process but they learn anyway (M.H.M. Yatim et al).

Games teach puzzle, solving, team playing etc. to children without seeming to do so. Therefore playing games, especially computer games has to be in education of children (K. H. McNichols, et al).

All these good relations between children, learning and computer games, and also successful previous works encourage me to process this project and show this work can be addressed toward children.

2.2 Why Use Game Programming
As it is said in:

Making games is considered to be one of the most creative activities. Many people and especially children love to play games and naturally want to design and develop their own ones. Not only playing games, but explicitly making games can be beneficial for children in many ways. It allows them to express their creativity, implement original ideas and learn structured and logical thinking as well as they can gain media literacy and a deep understanding of computer technology (M. H.M. Yatim et al).

Most children like to play and create variation of existing games. It is for these purposes almost all games include various level of difficulties. But creating games,
especially modern games are very hard and expensive to do so. Game programming and creation is no longer one man job. For modern games this cost can be 10 million dollars and big percentage of this cost goes for salaries of these programmers and other people who is working on creating the computer game (Edwards, Ralph).

But playing and creation of simple games can be beneficial to children and can be very easy to do. Since programming language and their environments vastly improved in last decade, it has become much easier to creating simple games of the 90s.

2.3 Tools of The Trade
A lot of companies create environments and tools to teach programming to children. Some of these companies support universities for this purpose such as Sun Microsystems as in Greenfoot. Some of them directly support building such tools such as Microsoft with its Kodu system (Kodu : XBox Game Engine). Some of them like Logo uses its traditional product line with additional supplements to reach wider audience (Logo Mindstorm NXT : Logo Game Engine).

2.4 Tools of The Trade Java
I choose to use Java Programming Language since it has become one of the de-facto programming languages in business and scientific world as can be seen in job listings, programming books written. Java is the most popular programming language in the world for 5 years. Therefore teaching a subset of Java to children seems most appropriate (Tiobe).

2.5 Tools of The Trade BlueJ, Karel The Robot and Greenfoot
After choosing Java, choosing Environment to teach programming becomes more easy to do so. Although Java has a lot of IDE (Integrated Development Environments) and other tools to program in, most of these tools are geared toward more advanced audience. BlueJ is developed with only purpose of teaching Java Programming Language. Therefore different from other IDEs it has some essential features missing from its tool set for business or general purpose programming such as code completion, code formatting. But different from other IDEs BlueJ has some powerful features which does not exist in other IDEs. Such as Object Bench, Code pad.

Also, BlueJ is designed specifically for pedagogical purposes and to teach programming. It reflects the object-oriented face of java programming language in its user interface. In this interface, users can interact with the objects and classes directly (Kölling, M et al).

Michael Kölling, one of the BlueJ developers, describes the main goals of the BlueJ IDE like this:

- To make the environment truly object-oriented by representing objects and classes as first-class entities in the user interface. This visualization was intended to create a mental emphasis on class and object relationships, instead of concentrating on the positioning of semicolons and parenthesis.
- To allow interaction with individual objects to encourage small-scale experimentation. We believed that frequent interaction with single objects and methods would almost automatically lead to a better understanding of the inner workings of any program in particular and Java in general.
To simplify the user interface of the environment to a degree that presents minimal distraction from the principles of programming. While learning to deal with IDEs is a worthwhile goal for a whole curriculum, it should not be forced on students simultaneously with learning their first programming concepts. It is a question of ordering: We wanted to free the teaching from the environment overhead. (Kölling, Michael).

By looking the main goals of the BlueJ described above, and considering the purpose and the motivation of this project, It can be said that BlueJ is a good tool to be used in this project. Like described in this goals, this tool is strong in object and class relationship, interaction between these objects and methods and simple user interface.

Figure 2.1: BlueJ

Karel is a programming environment designed by Richard Pattis to teach basic concepts of programming. It has a easy-to-use editor. In this editor, there is a world composed of vertical and horizontal grids and three kind of objects; a robot called Karel, beepers and walls. Karel can move, turn left, pick beeper and put beeper. Also it can be aware of its direction and detect the walls and beepers. Karel The Robot has been used in lots of educational institutions. Strongest feature of Karel The Robot is its visualization ability (Buck, D et al).
Greenfoot is a Java-based IDE which is built upon BlueJ. Greenfoot carries features of BlueJ and has a framework which provides essentials for game programming such as collision detection, infinite game loop, necessary keyboard, mouse support, sound and graphic capabilities. It is designed to develop simulations on its two dimensional world. Greenfoot takes its object-oriented representation and object interaction from BlueJ and takes its visualization and its world, which is composed of two dimensional grid system, from Karel The Robot. So we can say that design of Greenfoot based on BlueJ and Karel The Robot. We can create objects and put them wherever we want in the world of Greenfoot (Henriksen, P et al).
3 Introduction to Programming Language Features and How to Use Them

In this chapter, I will mention about how I choose the programming language and its concepts to use for the purpose of the project.

3.1 Problem of Which Features to Use

Even though we choose Java as programming Language to teach the basic concepts of programming, a question remains about which features to teach. Java as a modern programming Language supports different paradigms, structured programming, object oriented programming and generic programming.

Java Programming Language has lots of different features. Some of them are core features, some of them are specified features and some of them are advanced features. Therefore we decided to list programming language features and choose the ones which are useful and related with our project. Another reason for this is that, since some of the programming features are hard to understand even for university students, like polymorphism, it seemed logical thing to do.

3.2 Programming Languages

According to Kenneth C. Louden:

*A programming language is a notational system for describing computation in machine-readable and human-readable form* (Louden, K).

And as it is said in the book Encyclopedia of Computer Science:

*A programming language is a set of characters, rules for combining them and rules specifying their effects when executed by a computer* (Hemmendinger, D).

Since we are exposed to computers and computer-generated artifacts more and more, programming languages are becoming more and more important. Even small baby toys have small computers in them nowadays.

We use programming languages to change behavior of machines particularly computers.

3.3 Programming Language Features

Programming Languages are classified according to their features and their programming paradigms. At that point it is good to talk about these paradigms and features and some well-known programming languages which supports them. I will start from general paradigms and programming languages which satisfy these paradigms. Then, I will mention the features of selected programming language, Java. And Finally, I will indicate the features selected to teach in this project.

In our time, object-oriented programming is the most influential programming approach. It is commonly used by every type of industry and education. This approach makes the design process more implementable and it simplifies to design solutions to the daily life technology software problems. It promotes maintenance and reuse of code especially in big projects (Kölling, M). Java, C# and Smalltalk are well-known languages which use object-oriented approach (Gosling, J et al), (Liberty, J), (King, C et al).
In functional programming approach, programs are written using functions. A function takes inputs as argument and returns an output according to this input. The main program composed of functions and each of these functions can be also composed of other functions. That is to say the main program itself is a function and like every other function it takes inputs and return outputs. This approach makes easy to control the flow of functions and gives the program design a deep modularity (Hughes, J). F# and Scala are examples which use functional programming approach (Syme, D et al), (Odersky, M et al).

Procedural programming approach is based on calling procedures ay anytime during execution. In functional programming, functions which return some results are used, whereas in procedural programming, procedures which change their environment are used (Ishida, T et al), (Hogg, J). C and Pascal can be given as examples of procedural programming languages (Wiedenbeck, S), (Friedman, D.).

Generic programming is an important understanding in which, software reuse and modularity have very important role. Generic programming divides the program into little meaningful pieces and develops them separately. In this approach, interfaces are in a vital role. By using identical interfaces, it promotes code re-use and modularity (Denhert, J). We can give C++ templates and C# templates as examples (Garcia R. et al).

In Dynamic programming, programmer can interfere the program during runtime. That means, while program is running its code and structure can be changed. With this way a program can be changed fast using more simple grammar and methods. This approach can give much more flexibility but it does not convenient for every kind of situation. Well-known examples of this approach are Perl, Ruby, Python and JavaScript (Paulson, L).

If we show these programming approaches together:

A. **Object Oriented Programming**
   1. Java
   2. C#
   3. Smalltalk

B. **Functional Programming**
   1. F#
   2. Scala

C. **Procedural Programming**
   1. C
   2. Pascal

D. **Generic Programming**
   1. C++ Templates
   2. C# Generics

E. **Dynamic Programming Languages**
   1. Perl
   2. Ruby
   3. Python
   4. JavaScript

There are more examples of programming paradigms but these are among the most popular of programming paradigms. Among these programming language paradigms object oriented and dynamic programming languages are most popular. We will try to use Java Programming Language, which is object oriented, statically typed and
Compiled and generic language, to teach fundamentals of programming. According to this programming language paradigms there are a lot of programming language features. We can give some examples of these features as below.

A. **Variable Type Identifiers**
   1. **Numbers**
      1. Integers
      2. Floating Point Numbers
      3. Complex Numbers
      4. Decimal Numbers (Only in some of them)
   2. Characters and Strings

B. **Control Flow Structures**
   1. Conditional Statements
   2. Loop Statements
   3. Exceptions

C. **Functions – Sub Procedures**

D. **Object Oriented Features**
   1. Inheritance
   2. Virtual – Abstract Methods
   3. Polymorphism
   4. Interfaces

E. **Functional Programming Features**
   1. Recursive Programming
   2. Lambda

(Gosling, J et al).

To fully teach all of these features are very difficult. Some of the features of most programming languages are geared towards advanced programmers. Teaching some of these features to students even to graduate students, is very hard thing to do. Therefore as we said before we will try to teach fundamental and basic features of programming languages. With keeping these things in mind I decided to choose below fundamental features to teach to children.

3.4 **Selected Features from Programming Languages**

Below, selected features to use in the tutorials can be seen. It was important to select these features before creating the tutorials and applying them to the students. Because, the tutorials will be designed according to this selection.

A. **Variable Type Identifiers**
   1. Strings
   2. Numbers
      1. Integers

B. **Control Flow Structures**
   1. Conditional Flow Statements
   2. Loop Statements

C. **Object Oriented Features**
   1. Inheritance

Inheritance is an object-oriented concept which relate the classes with each other in a
hierarchical order (java.sun.com). It is one of the main concepts which make the object-oriented programming different from other programming paradigms. Some of the important properties in an object-oriented programming language comes from inheritance concept, such as re-usability (Taivalsaari, A).

I have selected these selected features because they are common and most-known features of any language. Using these features even fairly complicated programs can be written. Also a language which carry these features are turing complete which means that theoretically it is possible to write any program (Kepser, S).

In section Tutorial Contents I explained which of these programming language features introduced in which tutorial.
4 Game Programming Requirements

In this chapter, before designing tutorials, I will discuss what does game programming require? And analyze if our java and Greenfoot choices satisfy these requirements.

4.1 Programming Language Requirements

Although most games are written using general purpose programming languages, there exists some specialized languages, which are mostly based on general purpose programming languages, for writing games, like (Kahn, Kenneth : ToonTalk [...] ), (Kodu : XBox Game Engine), (Lego Mindstorm NXT : Lego Game Engine), (Squeak), (Robocode), (Phrogram), (Greenfoot).

Sophisticated games need more powerful Language elements but, since the target group is consist of children aged between 13 – 15 and since they are new to programming concepts and also since purpose of the project is just to teach the fundamentals of programming logic, it is better to use a specialized, modest programming language from the examples which I gave above.

Most necessary features are already supported by almost all languages. Like control and flow structures, data types, functions (methods, subroutines, etc), arrays.

Since we choose Greenfoot and Java all of these requirements are amply supported. I mentioned about Greenfoot, Java and their properties in the previous chapters.

4.2 Specialized Requirements

Apart from Programming Language Requirements, game programming requires some specialized features. These features must be supplied by a game framework of selected platform or must be coded by programmers. It is these specialized requirements which make game programming hard. Some of the specialized Requirements of Game Programming can be listed as

- Collision Detection
- Keyboard and Mouse Support
- Sound Support
- Graphic Support
- Vector Programming
- Mathematics

This list should not be thought as complete list but as a requirement and the most important features for basic platform games. Java and Greenfoot with its programming API easily support these requirements. Even though it is possible to do all of these things with only Java as tool, Greenfoot with its simplified API approach make this very easy to do (Java API), (Greenfoot API). I will use keyboard and mouse support, graphic support and mathematics in my tutorials.

To create a game firstly rules for all in-game interactions should be specified. And also, controlled elements by users (characters, vehicles, etc.), environment of the game, relations between controlled elements and environment and how to show this relation (animations) should be determined (Ian Parberry, et al).
5  Game Programming Tutorials

So far, I have decided about the requirements to select the target group and which programming language and framework I will use to achieve my purpose. According to my methodology, the next thing to do is to develop tutorials (scenarios) to represent the concepts we have decided to teach.

5.1  Tutorial Like Approach

Like (Kahn et al : A Computer Game to Teach Programming), we follow tutorial like teaching. This student-centered approach designs a dynamic, interactive and creative environment. Also tutorial like approach can be customizable according to the target or the target group (Bork, Alfred). Another reason for choosing the tutorial based approach is that with this approach we will be able to use divide and conquer learning algorithm more effectively. Separate (divide) and conquer learning algorithm is one of the well-known and widely used ways to teach about programming and computer technology. According to this learning algorithm, first a meaningful part of a subject to teach should be determined and this part should be analyzed separately. Then, the same thing should be applied recursively on the subject till there is no part left (Fürnkranz, J).

With tutorials we are separating the parts of the concepts to teach into tutorials and trying to conquer these parts separately. We prepared some restrained programming tutorials to teach children programming. In these programming tutorials we introduce programming language concepts gradually. Since concepts are introduced one at time, the target group were able to follow and learn some sophisticated programming language concepts like inheritance.

5.2  Tutorial Contents

We followed example of (Mohammed et al Using Greenfoot [..]), their method favors tutorials.

[...] students were taught many of the standard elementary programming concepts: control structures (if/else, for loops), classes, objects, members, methods, and arrays.

Although they use only one game we tried to make multiple games.

In each tutorial I tried to use recalls. Since the target groups is new to the this concept it is good to use recalls to reinforce the knowledge gained from previous tutorials. Recalls are really successful tools in learning (Trow, C.).

I have designed these tutorials to cover all the chosen concepts to teach. Some of the classes and methods can be found hard to understand by the target group but according to aim of this project it is not necessary to understand all the classes and methods in every detail. In those cases, It is enough to know functionality of them.

For example; students probably will not understand the most of the lines in the final code of fifth tutorial. For this tutorial it will be enough to make them understand the nested “if” control structure logic and the types of variables used by these “if” structures. If we tried to use only very simple codes to teach them the selected features, it would be not possible to create scenarios which are that entertaining, gripping and motivating. For this reason, as you can see in the tutorials, all the codes given directly to the students with a step by step order without expecting that student will understand all of them.

We do not expect that students will be able to repeat the same steps without looking
the tutorials. They will just be responsible about the concepts we are trying to teach them. I used this code piece in the last tutorial as an another example:

```java
for(int i=0; i<10; i++)
{
    int randomX = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
    int randomY = Greenfoot.getRandomNumber(getHeight() - 40) + 20;
    addObject(new Money(), randomX, randomY);
}
```

They will write this code directly by looking at (simply copying), but they do not have to know or understand about how to use method `Greenfoot.getRandomNumber()` here, it is enough to learn that this code generates a random x and y as a location of a money object and add this object into the greenfoot world and repeat this process 10 times. Here, important thing is to teach them the logic of for structure.

You can find these tutorials in details in the appendices chapter, A1.

5.2.1 Greenfoot Introduction Tutorial

This is a simple tutorial which taught fundamentals of Greenfoot Environment. Students will be familiar with this environment. Information about two main classes of Greenfoot will be given. Students will learn how to create new classes, how to change size of the world of Greenfoot. Coordinate structure of the world and depending on this location of the objects in this world will be introduced. They will also learn how to make an object act depending on the coordinate system. Students create a simple car object which moves automatically to right of screen. They use if control structure to change direction of simple car. In this tutorial, the car will not be controlled with keyboard.

With this tutorial, students will be familiar with integer type variable, “if” control structure and inheritance concept from the features we selected to teach.

![Figure 5.1: Tutorial 1 - Greenfoot Introduction](image)
5.2.2 Simple Car Game 1 Tutorial
In this tutorial students introduced to simple car which moves around the world. It moves faster in the road, slower in the earth. Car moves according to keyboard arrows and also it has complete freedom of movement. The main problem in this tutorial is to control the speed of the car object and make it move according to the keyboard commands.

With this tutorial, students will use integer type variable, more than one “if” control structure and inheritance concept from the features we selected to teach.

![Figure 5.2](image)

Figure 5.2: Tutorial 2 - Greenfoot Simple Car Race 1

5.2.3 Simple Car Game 2 Tutorial
Students start from where they left in second tutorial. We introduce more car like movement. Car change direction when they press left-right by buttons. It moves forward or reverses when they press up-down keys. In this tutorial we repeat the same steps with the previous tutorial until the last step to strengthen the previous knowledge. The main problem in this tutorial is to supply a rotation movement to the car.

With this tutorial, students will use integer type variable, more than one “if” control
structure and inheritance concept from the features we selected to teach.

**Figure 5.3:** Tutorial 3 - Greenfoot Simple Car Race 2

### 5.2.4 How to Write onto The World of Greenfoot Tutorial

In this tutorial students will learn how to write something on the world of Greenfoot. Actually, here, we are working with images not the writing itself. We are taking the given or the default writing as an image and show it on the screen using these images as background image of the objects, and this is the main problem in this tutorial.

This process can be used as a basis of some applications in a game, such as scoreboards, speedometers. The knowledge learned from this tutorial will be very useful for the following tutorials.

With this tutorial, students will use string, integer, float type variables and inheritance concept from the features we selected to teach.
5.2.5 Find the Correct Traffic Sign Tutorial

This is a simple classic memory/concentration game. Players try to find the same traffic sign among hidden signs. Our scenario for this game is that, when the world is created, 16 objects are placed in it with a size 4x4. We have 8 different images and each image is used twice, and users try to find the same two images with consecutively two selections. Here, we are using the knowledge learned from previous tutorial in a more detailed shape; when we click on an object, its background image changes. So the main problem in this tutorial is to check the changed backgrounds and make them stay like that or return to the default condition after comparing them with each other.

With this tutorial, students will use string, boolean type variables, “if/else” and nested “if” control structures, inheritance concept from the features we selected to teach.
5.2.6 Collect the Objects Tutorial

In this last tutorial, our purpose is to create a game in which a user controls a car, trying to collect the money and gasoline signs while avoiding fire signs. Our car has a default amount of gasoline at the beginning and this gasoline decrease when our car object moves. By collecting gasoline signs we are earning time and collecting signs we are earning points. Fire signs make our car lose gasoline (time). All the experience from previous tutorials will be used in this last one.

In this tutorial, in addition to the all other thing learned in previous tutorials, “for” loop structure will be presented.
5.3 Programming Language Features Learned in Tutorials

In this section a general view about, which programming features are represented in tutorials, will be given.

Control Flow Structures
In the first tutorial an introduction to “the if” statement is represented with a single “if” structure. In the other tutorials more than one if structure is used by the students. In the collect objects and traffic sings tutorials nested if structures are used and again in the traffic signs tutorial if-else structure is mentioned.

Primitive data types
Integer data type is used in all tutorials. Double data type is mentioned in the simple car game 2 tutorial, float data type represented in writing tutorial and boolean data type is used by students in traffic sign tutorial.

Constructors
Since we are using a Greenfoot world in all tutorials, we used constructors in all tutorials because to change the size of the world we have to give the new size in a constructor. In addition to this, in the writing tutorial and collect objects tutorials, constructors for user-defined classes are used.

Methods/Functions
In all tutorials, predefined methods, which are belong to two main class of Greenfoot, are used. Starting form the simple car game 2 tutorial, user defined methods used by the students.

Looping structures
In the last tutorial, the “for” loop structure represented.
Inheritance

In the Greenfoot environment, already all the classes inherit from the two main classes (“world” and “actor”) of the Greenfoot. In all tutorials, students used the “super” keyword, which change the size of the world, in the classes they created. Since, by doing this they used a constructor in the super class of the user-defined class, we can say that inheritance is represented in all tutorials. In addition to this, in the last tutorial we create our car class as subclass of the move class and the car object used the methods defined in the move class. So in the last tutorial inheritance concept is represented in details.

5.4 Results and Discussion

After finishing the tutorials, the next step is to obtain results of our work and to interpret them. As I mentioned in my methodology, to evaluate the improvements in the programming skills, mathematical logic and coordinate system concept knowledge of the target group, I have applied a survey and a test; you can find them in the appendix chapter A.2 and A.3. The target group consists of 6 students who fulfill the requirements I mentioned before.

I applied the survey after the last tutorial to evaluate the programming skills gained during the tutorials. And I applied the test both before the tutorials and after the tutorials (the same test) to evaluate the mathematical logic and coordinate system knowledge gained during the tutorials.

If you look at the survey, you can see that I selected the questions to cover the programming features we selected to teach. While selecting the questions I inspired from “Using Greenfoot and Games to Teach Rising 9th and 10th Grade Novice Programmers” (Mohammed et al Using Greenfoot […]). Some of the questions are taken directly from this work.

Result of the survey is:

<table>
<thead>
<tr>
<th>Student</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>excellent</td>
<td>good</td>
<td>not good</td>
<td>good</td>
<td>good</td>
<td>able</td>
<td>not exactly able</td>
</tr>
<tr>
<td>2</td>
<td>excellent</td>
<td>good</td>
<td>not good</td>
<td>good</td>
<td>good</td>
<td>able</td>
<td>not exactly able</td>
</tr>
<tr>
<td>3</td>
<td>excellent</td>
<td>good</td>
<td>not good</td>
<td>good</td>
<td>not good</td>
<td>able</td>
<td>not exactly able</td>
</tr>
<tr>
<td>4</td>
<td>good</td>
<td>excellent</td>
<td>good</td>
<td>excellent</td>
<td>good</td>
<td>able</td>
<td>not exactly able</td>
</tr>
<tr>
<td>5</td>
<td>excellent</td>
<td>good</td>
<td>not good</td>
<td>good</td>
<td>good</td>
<td>able</td>
<td>not exactly able</td>
</tr>
<tr>
<td>6</td>
<td>excellent</td>
<td>good</td>
<td>good</td>
<td>excellent</td>
<td>good</td>
<td>able</td>
<td>not exactly able</td>
</tr>
</tbody>
</table>

Figure 5.7: Results of the Survey

By looking the results I can say that in overall our purpose, to teach the logic of programming to the students, is achieved. The students reacted well in overall to the survey so now they are really familiar to the programming concepts.

If we investigate answers one by one, we can say that now they know now what is Greenfoot framework and how to use it, they are familiar with class, method and object concepts, they know why we are using “if-else” and “for” statements and they can construct an “if” statement themselves.

But there are problems at two points. According to the answers, data type concept is not understood, I think this is because of the design of the tutorials. I used data types to teach the other concepts, I did not designed a tutorial directly to represent the data type
concept.

Another problem is at the last question which is about to give an example of for loop. All the answers are indicating just the condition part in the “for” loop not the all “for” loop elements. If I generalize the answers they look like this “as long as this condition is satisfied, do that”. They did not mention about the control variable. I think the reason behind this is that I only used “for” loop in one tutorial. I can say this looking the success in “if” statement. Because I used “the if” statement in 5 different tutorials. In conclusion, by looking the overall, I can say that our purpose about the logic of programming is achieved.

When we come to the results of the test, it is like that:

<table>
<thead>
<tr>
<th>Student</th>
<th>Before Tutorials</th>
<th>After Tutorials</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coordinate system correct answers</td>
<td>Algebra of propositions correct answers</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 5.8:** Results of the Test

In this test there are 7 questions, which are prepared according to the test books of the target group. 4 of the questions are related with the coordinate system and 3 of the questions are related with the algebra of propositions which is mentioned in mathematical logic courses. Exactly the same test is applied both before and after the tutorials.

If we look at the number of correct answers given before the tutorials and the number of correct answers given after tutorials we can interpret this result.

For the coordinate system concept, the number of correct answers given before the application of tutorials is 21 and correct answers given after the application of tutorials is 23. This means that there is an 8.3% improvement. Looking at this data I can not say that there is an obvious improvement in the knowledge of coordinate system concept because this improvement is not that big and the correct answers are close to each other. I think the questions were easy for the target group because the number of correct answers is really high and numbers before and after the tutorials are close to each other. Also to talk about a success for a small improvement like that we need more students in the target group and more questions. The result is promising but we need more data to conclude that the purpose about coordinate system is achieved.

For the mathematical logic concept, the number of correct answers given before the application of tutorials is 8 and after the application of tutorials is 11. Looking at this data I can make the same comments with the previous one: I can not say that there is an obvious improvement in the knowledge of mathematical logic because this improvement is not that big and the correct answers are close to each other. Again, also to talk about a success for a small improvement like that we need more students in the
target group and more questions. The result is promising (more promising than the previous one) but we need more data to conclude that the purpose about coordinate system is achieved.
6 Conclusion and Further Studies

For this project, my aim was to improve and tame the programming skills and the programming logic of the students aged between 13 and 15 creating games on Greenfoot Framework. Also it is aimed to improve the knowledge of the students about the basic concepts of physics and mathematics such as coordinate system and mathematical logic.

To achieve my purpose, I used computer game creation to catch the interest of the students and to increase the motivation of them. I can say that this approach worked well. Students get very excited and eager when they hear the computer game creation. Computer games can be used in education to attract the students and to motivate them. In this project I tried to improve their knowledge about the coordinate system and the mathematical logic but computer games can be used for other subjects as well.

Greenfoot environment which uses java programming language satisfied my requirements well in the process of game creation. Greenfoot is really a great tool for teaching the logic of programming to the children aged between 13 and 15 because of its simplicity and its ability to show the classes and objects in an understandable and visualized way.

I have selected the core programming features to represent them to the students. And I have designed game scenarios on Greenfoot Framework according to these selections. Then I applied these scenarios to the students as tutorials.

Lastly, to evaluate the purpose of this project, I applied a survey and a test to the students. According to the results, I can say that with these six tutorials, students gained familiarization to the programming logic and the Greenfoot Framework. Also, according to the results of the tests, students improved their knowledge about coordinate system and mathematical logic.

Looking to the result, it can be said that main purpose, to tame and improve the programming logic of the target group, of the project is achieved but for the other purposes I can not say the same thing. There is a slight improvement.

As a further study, other games can be designed to represent the other concepts of programming such as polymorphism and abstract structures or to improve other selected subjects of mathematics, physics or another discipline. Also population of the target group can be increased to make the results more significant.
7 References


23

[Java API] . . http://download.oracle.com/javase/1.5.0/docs/api/.


game making activity. (2006) :

[M.H.M. Yatim et al] Maizatul H.M. Yatim/Lennart Nacke/Maic Masuch. Improving Game Design by Understanding the Gender Differences The Cognitive Approach. () :

[Mohammed et al Using Greenfoot [..]] Mohammed Al-Bow. Using Greenfoot and Games to Teach Rising 9th and 10th Grade Novice Programmers . (2006) :


A  Appendices

A.1  Tutorials
In this section, the tutorials applied to the target group will be given directly in details. They are prepared as numbered articles. I tried to use figures for each article to make the tutorials more explanatory.

A.1.1  Greenfoot Introduction Tutorial
In this tutorial fundamentals of Greenfoot Environment will be mentioned. Students will be familiar with this environment. Information about two main classes of greenfoot will be given. Students will learn how to create new classes, how to change size of the world of greenfoot. Coordinate structure of the world and depending on this location of the objects in this world will be introduced. They will also learn how to make an object act depending on the coordinate system. Students create a simple car object. Which moves automatically to right of screen. They use if control structure to change direction of simple car. In this tutorial, the car will not be controlled with keyboard.

**Tutorial:**
1) Start with selecting Greenfoot from Program Files Menu.

![Greenfoot Launcher]

2) If greenfoot have not been operated before, selection of the virtual machine can be needed. Continue with selecting one of the virtual machines on your computer.

![Greenfoot Launcher]

3) Start with new from scenario menu.
4) By selecting a suitable folder, create a scenario called SimpleCar.

5) You will face with the screen below.

In this screen, class "World" includes the base java class which will be used as a base for environment in which the car will move. We will create a more suitable environment for our purpose by inheriting from this "World" class. In the same way, the class "Actor" is the base class for objects moving in our environment.
6) By putting the cursor on the "World" class and using the second button of the mouse, create a subclass which inherits from the class "World".

7) On the new coming screen;
   Select:
   a) New class name: MyCarWorld
   b) New class image: backgrounds/cell.jpg
8) The new class we have created will be seen like this:

9) By double clicking on the MyCarWorld class, open the java code. Change the constructor code below;
public MyCarWorld()
{
    // Create a new world with 20x20 cells with a cell size of 10x10 pixels.
    super(20, 20, 10);
}

// In this process, grid with size 20x20 is created. Each square of the grid will be at size 10x10.

Into the code below;

public MyCarWorld()
{
    // Create a new world with 20x20 cells with a cell size of 10x10 pixels.
    super(10, 10, 60);
}

/// In this process, grid with size 10x10 is created. Each square of the grid will be at size 60x60.

10) By returning the main screen push the button Compile All.

![Image of Greenfoot Simple Car scenario with a grid and a Compile All button highlighted.]

11) You will see that world will change into 10x10 width and height with pixel size 60x60. Below you can see a sample of 3x3 of this world.
12) Putting the cursor on the actor superclass create a subclass of actor by using second button of the mouse.

13) On the emerging screen
Select:
   a) New class name: BlueCar
   b) New class image: transport car01.png
14) Push the button "Compile All"
15) By putting the cursor on the bluecar class, select new BlueCar() using the second second button of the mouse.

16) Put the new created blue car on the world by draggin and dropping.
17) Open the BluCar source code by double clicking on the blue car icon. You will see a screen like below.

Using `act` method, actors can move on the world. Since this method is empty now, nothing will happen when we run the scenario.

```java
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class BlueCar here.
 *
 * @author (your name)
 *
 * @version (a version number or a date)
 */

public class BlueCar extends Actor {

    /**
     * Act - do whatever the BlueCar wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```
18) Close the code editor by clicking on the close button.
19) By putting the cursor on the car object on the world click the second button of the mouse. By selecting the inspect, explore the properties and variables of the car object.

![Inspecting Car Object](image)

20) Properties of the bluecar object should look like below:

![Object Inspector](image)

21) Using the mouse move the blue car object to another location on the world. Again using the second button of the mouse inspect the properties of the blue car object. Especially observe how x and y coordinates are changing.

a) Upper left corner:
b) Upper right corner:

![Greenfoot Object Inspector](image)

```
int x: 0
int y: 0
private int rotation: 0
World world:
private GreenfootImage image:
```

Show static fields

Close

---

c) Lower left corner:

![Greenfoot Object Inspector](image)

```
int x: 9
int y: 0
private int rotation: 0
World world:
private GreenfootImage image:
```

Show static fields

Close
d) Lower right corner:
22) In this situation we can see that when we move the car by using the mouse, x and y variables are changing between 0 and 9. If you remember first time when we create the world we have used the code segment below:

```java
public MyCarWorld()
{
    // Create a new world with 20x20 cells with a cell size of 10x10 pixels.
    super(10, 10, 60);
}
```

In here, our world has been created according to 10x10 grids. We can show it like this:

```
\[\begin{array}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}\]
```

As an example, the red circle is in the coordinate, x = 7, y = 4.

23) Putting the cursor on the blue car object on the world push the second button of the mouse. And run the setLocation method which is inherited from the actor superclass.
Entering the values 7 and 4 on the emerging screen click on the ok button.

24) Observe the change of the location of the blue car object. Giving random values observe how the blue car move to other locations. Like:
   setLocation(3,3);
   setLocation(8,8);
   setLocation(1,9);
   setLocation(0,0);
   setLocation(9,9);
25) Since we have this information the car class can be moved on the world. By clicking on the blue car object open the code editor. Write the code below in the act method.
public class BlueCar extends Actor {

    /**
     * Act - do whatever the BlueCar wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        setLocation(getX() + 1, getY());
    }
}

26) Compile the scenario by using Compile All button.

27) Like we did before, run the new BlueCar() by using the second button of the mouse on the BlueCar class.

28) Drag the blue car object on a location on the world and drop.

29) By clicking on the act button observe how the blue car object moving on the direction of X, like in the example below.
30) If we continue to click on the act button we can observe that blue car object will stop at the edge of the world. Greenfoot do not let the blue car move through outside of the world borders
31) By putting the car object on a starting point, click on the run button.

32) To click on the run is the same thing to click on the act button sequentially.
While the pause button will cut the acting till pressing the run button again, Speed indicator will determine how fast the simulation will be, in other words it determine how fast the act method will be called.

33) Now, we want to make our car object move to the left when it reaches the border of our world. We know from previous tries that border coordination for our world is 9.
   We can do this by decreasing our x coordinate by 1 instead of increasing it by 1
   IF our object's x coordinate reaches the value 9.

34) Stop our simulation by clicking on the pause button and put our blue car object somewhere in the middle of our world.

35) Open the code editor of the bluecar class by double clicking on it.

36) Edit the code like below

```java
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class BlueCar extends Actor
{
    /**
     * Act - do whatever the BlueCar wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    int movementRate = 1;
    public void act()
    {
        if (getX() == 9)
        {
            movementRate = -1;
        }
        setLocation(getX() + movementRate, getY());
    }
}
```

What we have done here is that, instead of always increase the x coordinate by 1, we save this rate in a variable called movementRate.
Instead of `setLocation(getX()+1,getY())` we use `setLocation(getX() + movementRate, getY())`

That means we increase the x coordinate by 1 till it reaches 9 then we start to decrease it(to move in the left direction).

The code piece
```
if (getX() == 9)
```
{ 
    movementRate = -1;
}
means if getx() equals to 9 make the movementRate -1.

37) Compile the scenario again by clicking compile all button.

38) Create a blue car again by using second button of the mouse on blue car class.

39) Put the blue car object into somewhere in the middle of the world.

40) By clicking the run button observe that our car object firstly comes to the right border of the world then by moving the opposite direction it reaches the left border of the world.

A.1.2 Simple Car Game 1
In this tutorial students introduced to simple car. Which moves around the world. It moves faster in the road, slower in the earth. Car moves according to keyboard arrows and also it has complete freedom of movement.

Tutorial:
1) Start with selecting Greenfoot from Program Files.

2) If you have never run greenfoot before, it may be necessary to choose a virtual machine. Continue with selectig a virtual machine from your computer.
3) Start with new from Scenario Menu

4) Selecting a suitable folder, create a scenario called CarRacing1.
5) You will see the screen below:

6) By right clicking on the World class create a subclass.
7) On the emerging screen
Select:
   a) New class name: MyCarWorld
   b) New class image: Browse for more images

8) From the image selecting screen, select the CarRacingPlatform.png from the same directory with this tutorial.

9) You will see a screen like below:
10) The new created class will be seen like below in the greenfoot world:

11) If you click on the Compile All button:
12) You will see a screen like below:

13) The reason for that; Our picture is big and just a part of it shown on the screen. So situation is like that:
14) Only upper left section of the picture can be seen. If we look at the .png file in windows we can see its properties.

15) According to this our picture file has dimensions 501x445

16) Open the code editor by double clicking on the MyCarWorld class.

   We can see the code below:

   ```java
   public MyCarWorld()
   {
      // Create a new world with 20x20 cells with a cell size of 10x10 pixels.
      super(20, 20, 10);
   }
   // The process in here is that to create a grid with size 20x20. Each square of the grid is in the size of 10x10 pixel. Because of the written code we can see only the section with
public MyCarWorld()
{
    super(501, 445, 1);
}

Here, We are creating a grid with size 501x445. Each square of the grid is 1x1 pixel size.

17) Returning the greenfoot main screen push the compile all button.

18) Create a new subclass of Actor class by clicking on it with the second button of the mouse.
19) On the emerging screen:
   Select:
   a) New class name: BlueCar
   b) New class image: Browse for more images

20) Select the car01.png file from the directory of this tutorial.
    You should see this screen:
21) When you click on the ok button, you should see a screen like this:
22) Click on the compile all button

23) clicking with second button of the mouse on BlueCar class, select newBlueCar().
24) drag the new created car with mouse on the world and drop it

25) In this situation, nothing will happen when you click on run button.
26) The reason for that there is nothing written in the act behaviour of the bluecar.

27) Open the code editor by double clicking on the BlueCar class. Our objects can move on the world using their act methods. Since, this method is empty now nothing happen when we click on act or run button.
28) Use the code belowe for the act method:
29) Here, by using `Greenfoot.isKeyDown("right")`, the car object is controlled with the right arrow button. When we press the right button our car object moves to the right direction. For example our car object at the position \((x,y) = (100,100)\). When the right arrow key is pressed, it moves according to the value which is stored in MOVEMENT integer. So it will come to the position \((102, 100)\).

30) Compile the scenario by pressing the compile all button. And try the object by creating a new instance of it. Drag and drop the car on the world and observer its movement pressing the right arrow button.

31) Changing our code, make the car moves in the four direction.

32) Here, only difference from making the object move in the x direction is that y value is increasing when down arrow key pressed and y value is decreasing when up arrow key pressed.

33) Lastly, we should make the car move slower at green areas.

34) Add the code below into the bluecar class code. Here we are relating the color value of the current location of the car with the grass color. If they are same we reduce the movement speed of the car half of its current speed.
A.1.3 Simple Car Game 2

Students start from where they left in second tutorial. We introduce more car like movement. Car change direction when they press left-right by buttons. It moves forward or reverse when they press up-down keys. In this tutorial we repeat the same steps with the previous tutorial until the last step to strengthen the previous knowledge.

**Tutorial:**

1) Start with selecting Greenfoot from Program Files.

2) If you have never run greenfoot before, it may be necessary to choose a virtual
machine. Continue with selecting a virtual machine from your computer.

3) Start with new from Scenario Menu

4) Selecting a suitable folder, create a scenario called CarRacing1.
5) You will see the screen below:

![Screen Shot](image1.png)

6) By right clicking on the World class create a subclass.

![Screen Shot](image2.png)
7) On the emerging screen
   Select:
   a) New class name: MyCarWorld
   b) New class image: Browse for more images

8) From the image selecting screen, select the CarRacingPlatform.png from the same directory with this tutorial.

9) You will see a screen like below:
10) The new created class will be seen like below in the greenfoot world:

11) If you click on the Compile All button:
12) You will see a screen like below:

13) The reason for that; Our picture is big and just a part of it shown on the screen. So situation is like that:
14) Only upper left section of the picture can be seen. If we look at the .png file in windows we can see its properties.

15) According to this our picture file has dimensions 501x445

16) Open the code editor by double clicking on the MyCarWorld class.

    We can see the code below:

    ```java
    public MyCarWorld()
    {
        // Create a new world with 20x20 cells with a cell size of 10x10 pixels.
        super(20, 20, 10);
    }
    ```
public MyCarWorld()
{
    super(501, 445, 1);
}

Here, We are creating a grid with size 501x445. Each square of the grid is 1x1 pixel size.

17) Returning the greenfoot main screen push the compile all button.

18) Create a new subclass of Actor class by clicking on it with the second button of the mouse.
19) On the emerging screen:
   Select:
   a) New class name: BlueCar
   b) New class image: Browse for more images

20) Select the car01.png file from the directory of this tutorial.
    You should see this screen:
21) When you click on the ok button, you should see a screen like this:
22) Click on the compile all button
23) clicking with second button of the mouse on BlueCar class, select newBlueCar().

24) drag the new created car with mouse on the world and drop it
25) In this situation, nothing will happen when you click on run button.
26) The reason for that there is nothing written in the act behaviour of the bluecar.

27) Open the code editor by double clicking on the BlueCar class. Our objects can move on the world using their act methods. Since, this method is empty now nothing happen when we click on act or run button.

28) Use the code below for the act method:

```java
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and HouseInfo)

/**
 * Write a description of class BlueCar here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */

public class BlueCar extends Actor
{
    /**
     * Act - do whatever the BlueCar wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```
29) Here, by using `Greenfoot.isKeyDown("right")`, the car object is controlled with the right arrow button. When we press the right button our car object moves to the right direction. For example our car object at the position (x,y) = (100,100). When the right arrow key is pressed, it moves according to the value which is stored in MOVEMENT integer. So it will come to the position (102, 100)

30) Compile the scenario by pressing the compile all button. And try the object by creating a new instance of it. Drag and drop the car on the world and observer its movement pressing the right arrow button.

31) Changing our code, make the car moves in the four direction.
32) Here, only difference from making the object move in the x direction is that y value is increasing when down arrow key pressed and y value is decreasing when up arrow key pressed.

33) Lastly, we should make the car move slower at green areas.

34) Add the code below into the bluecar class code.

35) Here we are relating the color value of the current location of the car with the grass color. If they are same we reduce the movement speed of the car half of its current speed.
Comparing to a real car game, our problem is here that it can move any direction it wants. But in real a car can move only in the direction it faces. So what we have to do is to change the direction of the car by using right and left arrow keys like a steering wheel. Using the up and down arrow keys we will make the car move in the direction of it faces or in the opposite direction.

The information which controls the direction of our car is the rotation information. By using the getRotation method we get the rotation information and using the setLocation method we can change the car's direction.

```java
import greenfoot.*;
import java.awt.*;
public class BlueCar extends Actor {
    private final int MOVEMENT = 2;
    public void act()
    {
        Color currentColor = getWorld().getColorAt(getX(), getY());
        Color grassColor = getWorld().getColorAt(0, 0);
        int speed = MOVEMENT;
        if (currentColor.equals(grassColor))
        {
            speed = speed / 2;
        }

        if (Greenfoot.isKeyDown("right"))
        {
            setLocation(getX() + speed, getY());
        }

        if (Greenfoot.isKeyDown("left"))
        {
            setLocation(getX() - speed, getY());
        }

        if (Greenfoot.isKeyDown("up"))
        {
            setLocation(getX(), getY() - speed);
        }

        if (Greenfoot.isKeyDown("down"))
        {
            setLocation(getX(), getY() + speed);
        }
    }
}
```
import greenfoot.*;
import java.awt.*;
public class BlueCar extends Actor
{
    private final int MOVEMENT = 2;
    private final int ROTATION = 3;
    public void act()
    {
        Color currentColor = getWorld().getColorAt(getX(), getY());
        Color grassColor = getWorld().getColorAt(0, 0);
        int speed = MOVEMENT;
        if (currentColor.equals(grassColor))
        {
            speed = speed / 2;
        }

        if (Greenfoot.isKeyDown("right"))
        {
            setRotation(getRotation() + ROTATION);
        }

        if (Greenfoot.isKeyDown("left"))
        {
            setRotation(getRotation() - ROTATION);
        }

        if (Greenfoot.isKeyDown("up"))
        {
            move(speed);
        }

        if (Greenfoot.isKeyDown("down"))
        {
            move(-speed);
        }
    }

    public void move(double distance)
    {
        double direction = Math.toRadians(getRotation());
        int x = (int) Math.round(getX()) + Math.cos(direction) * distance;
        int y = (int) Math.round(getY()) + Math.sin(direction) * distance;
        setLocation(x, y);
    }
}
A.1.4 Writing Tutorial

In this tutorial students will learn how to write something on the world of greenfoot. Actually, here, we are working with images not the writing itself. We are taking the given or the default writing as an image and show it on the screen using these images as background image of the objects.

This process can be used as a basis of some applications in a game, such as scoreboards, speedometers. The knowledge learned from this tutorial will be very useful for the following tutorials.

**Tutorial:**
1) Select “Wordpad” from start Menu

![](image1.png)

2) When the Wordpad starts, you will see a screen like this:

![](image2.png)

3) Selecting suitable properties from font menu, write “Hello” on the screen.

![](image3.png)
4) Selecting:
   • Font = Times New Roman
   • Font Size = 24
     Write “Hello” on the screen

   Hello

5) Since Greenfoot is using Java Programming Language, it uses similar libraries to Java while writing something on the screen. The logic of writing something on the screen is like this:
   • Select the font type (e.g. “Arial”).
   • Select the font size (e.g. “24”).
   • Selection of effect (e.g. “bold, italic”).
   • Selection of position the writing.
   • Selection of what to write.

6) Open the Greenfoot and add a new project.

7) Name it as “WritingTutorial”.

76
8) You will see the main Greenfoot screen.

9) Right click on the World Class and add a new world selecting new subclass.
10) Name our new world as “WritingTutorialWorld” and select the sand as background. Then select OK.

11) Clicking on the compile all button, see the current situation of our world.
12) By double clicking on the WritingTutorialWorld see the code of it.

13) Like we see in the previous tutorials, this code creates a world with the size 20x20. And each square(cell) in this world is in size 10x10. Since we want more control on the world, we will work with different sized world. Change the code like below.
14) Observe the new world clicking on compile all button.

15) Now we want to write something on the screen. We will do this like we did with the Wordpad. Create a new subclass of Actor by right-clicking on it.
16) Name this subclass as Writing and do not assign a background or any other thing.

17) Compile the project again and add a Writing object into the world by right clicking on Writing and selecting new Writing().
18) We can add lots of Writing object into our world by creating them with new Writing().

19) As you can see, with the default code we are adding just green foots into the world. It takes its background image from its parent class Actor. We want to change this background with a writing we want.

20) By double-clicking on the Writing class open the default code of it.
21) If we repeat the necessary information on this code.

```java
public class Writing extends Actor
    {
        public void act()
        {
            // Add your action code here.
        }
    }
```

With this piece of code, our Writing class inherits the properties of the Actor class.

In this code piece, making changes on act method, which defined in the actor class, we make our object to act differently.

22) Like act method, there are other methods predefined in the actor class. By using these methods we can make our objects do different things according to the situations. Double-click on the actor class on the Greenfoot main screen. A browser window will be opened. We can see the other methods predefined in the actor class.
23) These methods can be used our Writing class directly. If we investigate the methods in the browser window we can find 2 methods we need in this tutorial:
   protected void addedToWorld(World world)
   This method is called by the Greenfoot system when the object has been inserted into the world.
   void setImage(GreenfootImage image)
   Set the image for this object to the specified image.

24) Our logic will be this: To change the background image when the object inserted into the world. In other words;
   addedToWorld {
   ....
   setImage(..)
   }

25) Double-clicking on it, open the writing class and add the code below:
26) What we are doing with this code is that, firstly we determine the size of our image as 60x40. Secondly with;

```java
GreenfootImage image = new GreenfootImage(imageWidth, imageHeight);
```

code piece, we are creating a new greenfoot image. Then with;

```java
image.setImage(new java.awt.Color(0, 0, 0, 160));
```

determine the color we will use. After that with;

```java
image.fillRect(0, 0, imageWidth, imageHeigth);
```

we are filling the selected image with the selected color. And finally with,

```java
setImage(image);
```

we are changing our actor's background image with the new created image.

27) In other words we are doing the same thing with this:
28) Close the code editor and click on the compile all. If we create a new writing object and add it into the world. We will see this:
29) You can make other tries changing the color.

```java
image.setColor(new java.awt.Color(0, 128, 40, 121));
```

30) Now we want add a writing like we add color. We will do this by selecting the font color as white.

```java
public class Writing extends Actor {

    protected void addedToWorld(World world) {
        int imageWidth = 60;
        int imageHeight = 40;
        Greenfoot.Image image = new Greenfoot.Image(imageWidth, imageHeight);
        image.setColor(new java.awt.Color(0, 0, 0, 0));
        image.fillRect(0, 0, imageWidth, imageHeight);

        float fontSize = 12.0f;
        java.awt.Font font = image.getFont();
        font = font.deriveFont(fontSize);
        image.setFont(font);
        image.setColor(new java.awt.Color(0, 255, 255, 255));
        image.drawString("Demo", imageWidth/10, imageHeight/2);

        setImage(image);
    }
}
```

31) If you compile the project and add a new writing object into the world (right click on the writing class and select new Writing() ) you will see this:
32) This process is very similar to the process we did with Wordpad. We are selecting 12 as font size, we are taking this font as our image's normal font. After setting up the size we are writing using drawString selecting the color.

33) It will be good if we select what we write while creating a writing object. Like we said earlier, objects can take information from constructors when they are created. To use this property change the code like below:
34) Click on compile all button. When we try to create a new writing object we will see this:
35) If we select to create a new object using `new Writing()`, we will see the same “demo” word on our world, when we add this object into our world.

36) If we try to add a new object by selecting `new Writing(String pWritingToWrite)` , we will see a screen like this:

37) When we write something into this box, this writing object will be created in the world. For example, if we write “Success”, we will see this screen:
A.1.5 Find The Correct Traffic Sign Tutorial

This is a simple classic memory/concentration game. Players try to find the same traffic sign among hidden signs. Our scenario for this game is that, when the world is created, 16 objects are placed in it with a size 4x4. We have 8 different images and each image is used twice, and users try to find the same two images with a consecutive two selections. Here, we are using the knowledge learned from previous tutorial in a more detailed shape; when we click on an object, its background image is changing.

**Tutorial:**

1) We will try to develop the famous game “find the same two picture” with Greenfoot Framework. Start with running the Greenfoot
2) Open a new scenario and name it as TrafficSignTutorial.
3) Create a new world by right-clicking on the world class.

4) Set the name of our new world as “SignWorld” and do not assign any background image.
5) Compile the project. You will see a screen like this:

![New class interface](image)

6) Open the code of SignWorld class by double-clicking on it. You will see the default code.
7) For this tutorial we need a world with size 4x4 and cells with size 120x120. So, if we change the code according to this:

```java
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class SignWorld here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class SignWorld extends World {

    /**
     * Constructor for objects of class SignWorld.
     *
     */
    public SignWorld()
    {
        // Create a new world with 20x20 cells with a cell size of 10x10 pixels.
        super(20, 20, 10);
    }
}
```
8) Compile the project again. You will see that our world is larger now:
9) Open the directory which you saved this project at the beginning.

10) Copy the images we will use in this tutorial into the images folder. You can find these images in the same folder with this tutorial.
11) Right-clicking on actor class, create a new actor object and name it as TrafficSign. Choose board.jpg as background image.

12) Right-clicking on TrafficSign, Create a new TrafficSign object.
13) When you add this new object into our world, you will see this screen:

14) We can create lots of TrafficSign objects in our world.
15) In this situation, we have to drag and drop 16 TrafficSign objects to fill our world with TrafficSign objects completely. It is better to make these objects to be added when our world is created. To do this, we should add this code into our world (SignWorld class):

```java
public SignWorld()
{
    // Create a new world with 20x20 cells with a cell size of 10x10 pixels.
    super(4, 4, 120);
    addObject(new TrafficSign(), 0, 0);
    addObject(new TrafficSign(), 0, 1);
    addObject(new TrafficSign(), 0, 2);
    addObject(new TrafficSign(), 0, 3);
    addObject(new TrafficSign(), 1, 0);
    addObject(new TrafficSign(), 1, 1);
    addObject(new TrafficSign(), 1, 2);
    addObject(new TrafficSign(), 1, 3);
    addObject(new TrafficSign(), 2, 0);
    addObject(new TrafficSign(), 2, 1);
    addObject(new TrafficSign(), 2, 2);
    addObject(new TrafficSign(), 2, 3);
    addObject(new TrafficSign(), 3, 0);
    addObject(new TrafficSign(), 3, 1);
    addObject(new TrafficSign(), 3, 2);
    addObject(new TrafficSign(), 3, 3);
}```

16) With this, we are adding our objects into our world automatically when our world created. As you remember when we created our SignWorld we arranged it as 4x4 squares. When we write addObject(new TrafficSign(), 2, 3); we are adding a new TrafficSign object into our world at a position 2x3 (Our world is 4x4).
17) When we compile our project, we will see this screen as expected:
18) As you can see, our world is filled completely with TrafficSign objects. Since we did not write anything for acting of our TrafficSign objects when we click the run button, nothing will happen. What we want is to show images when we click on the TrafficSign objects.

19) To do this, we should write these codes in our TrafficSign class.

```java
public class TrafficSign extends Actor {

    public void act() {
        if (Greenfoot.mouseClicked(this)) {
            GreenfootImage image = new GreenfootImage("Stop.png");
            setImage(image);
        }
    }
}
```

20) What we are doing here is that, to change the background image of our object with the Stop.png.

21) Click on compile all button and then click on run (as you know from earlier tutorials, we should click on run button to make the objects we created act). When you click on random objects you will see that background image is changing with Stop.png.
22) Now, all the objects are changing their background images with Stop.png. If we want them to show different images we should replace the Stop.png with a general variable. And to give the information to the object about which image it will show using this variable. You are familiar with this logic from previous tutorial “WritingTutorial”. We did this tutorial to give information about this situation.

23) To make these changes we should start with doing this:

```java
public class TrafficSign extends Actor {
    private String imageName;

    public TrafficSign(String pImageName) {
        imageName = pImageName;
    }

    public void act() {
        if (Greenfoot.mouseClicked(this)) {
            GreenfootImage image = new GreenfootImage(imageName);
            setImage(image);
        }
    }
}
```

24) Now, if we click on compile all button, we will get an error like this in our SignWorld class:
The reason for this error is that, now our objects want to be informed what to show when we click on them.

25) To solve this problem we should write the names of the image files, we want show when we click on our objects, one by one. So, we should replace the code in SignWorld class like this:

```java
public SignWorld()
{
    // Create a new world with 20x20 cells with a cell size of 10x10
    super(4, 4, 120);
    addObject(new TrafficSign("Stop.png"), 0, 0);
    addObject(new TrafficSign("FirstAid.png"), 0, 1);
    addObject(new TrafficSign("DoNotTurnLeft.png"), 0, 2);
    addObject(new TrafficSign("ForbiddenToOvertake.png"), 0, 3);
    addObject(new TrafficSign("Forbidden.png"), 1, 0);
    addObject(new TrafficSign("UForsbidden.png"), 1, 1);
    addObject(new TrafficSign("Stop.png"), 1, 2);
    addObject(new TrafficSign("ForbiddenToOvertake.png"), 1, 3);
    addObject(new TrafficSign("NoVehicles.png"), 2, 0);
    addObject(new TrafficSign("UForsbidden.png"), 2, 1);
    addObject(new TrafficSign("DoNotTurnLeft.png"), 2, 2);
    addObject(new TrafficSign("NoVehicles.png"), 3, 0);
    addObject(new TrafficSign("FirstAid.png"), 3, 1);
    addObject(new TrafficSign("GiveWay.png"), 3, 2);
    addObject(new TrafficSign("FirstAid.png"), 3, 3);
}
```

As you can see, according to our purpose, there should be 2 copy of each images. Since we created our world as 4x4 we have 16 square so we need 8 images.

26) Now, let's compile the project. After clicking on the run button, randomly click on our objects. As expected, you will see:
27) Now our work opening the boxes. What we want to do now, after opening 2 images they should get closed if they are not same. So we need a information about which image is open at the moment. We should add this information into our SignWorld class:
Now, our world will know which image is opened.

28) We want that, if the new opened image and previously opened image is same they will be remained opened. But if, they are not same, both of them will be closed (closed means that their traffic-sign image will turn into “board.jpg” again). So we need to change the image of traffic-sign image into opened image. Now open the code of the trafficsign object by double-clicking on it and edit it like this:

```java
public SignWorld()
{
    // Create a new world with 20x20 cells with a cell size of 10x10 pixels.
    super(4, 4, 120);
    addObject(new TrafficSign("Stop.png"), 0, 0);
    addObject(new TrafficSign("FirstAid.png"), 0, 1);
    addObject(new TrafficSign("Police.png"), 0, 2);
    addObject(new TrafficSign("DoNotTurnLeft.png"), 0, 3);
    addObject(new TrafficSign("ForbiddenToOvertake.png"), 1, 0);
    addObject(new TrafficSign("UForbidden.png"), 1, 1);
    addObject(new TrafficSign("Stop.png"), 1, 2);
    addObject(new TrafficSign("ForbiddenToOvertake.png"), 1, 3);
    addObject(new TrafficSign("NoVehicles.png"), 2, 0);
    addObject(new TrafficSign("UForbidden.png"), 2, 1);
    addObject(new TrafficSign("GiveWay.png"), 2, 2);
    addObject(new TrafficSign("DoNotTurnLeft.png"), 2, 3);
    addObject(new TrafficSign("Police.png"), 3, 0);
    addObject(new TrafficSign("NoVehicles.png"), 3, 1);
    addObject(new TrafficSign("GiveWay.png"), 3, 2);
    addObject(new TrafficSign("FirstAid.png"), 3, 3);
}

private TrafficSign openedSign;

public TrafficSign getOpenedSign()
{
    return openedSign;
}
public void setOpenedTrafficSign(TrafficSign pTrafficSign)
{
    openedSign = pTrafficSign;
}
```
29) At this point, we still do not know which background of the object is shown at any particular time. We have to get this information. So with this purpose we should edit TrafficSign class like this:

```java
public class TrafficSign extends Actor {
    private String imageName;
    private String normalImageName = "board.jpg";

    public TrafficSign(String pImageName)
    {
        imageName = pImageName;
    }

    public void act()
    {
        if (Greenfoot.mouseClicked(this))
        {
            trafficSignShow();
        }
    }

    public void trafficSignShow()
    {
        GreenfootImage image = new GreenfootImage(imageName);
        setImage(image);
    }

    public void normalImageShow()
    {
        GreenfootImage image = new GreenfootImage(normalImageName);
        setImage(image);
    }
}
```
public class TrafficSign extends Actor
{
    private String imageName;
    private String normalImageName = "board.jpg";
    private boolean isImageOpen = false;

    public TrafficSign(String pImageName)
    {
        imageName = pImageName;
    }

    public void act()
    {
        if (Greenfoot.mouseClicked(this))
        {
            if(isImageOpen == true)
            {
                normalImageShow();
                isImageOpen = false;
            }
            else
            {
                trafficSignShow();
                isImageOpen = true;
            }
        }
    }
}

30) Now, click on compile all button. Now you can see that, when we click on an object a traffic sign will appear (changing background image of the object with the traffic-sign we put at this particular square). And observe that when we click on the opened object again it will get closed. What we did in here is that, If the object is opened (clicked), traffic sign will be showed and the isImageOpened variable will be marked as true, else normal (board.jpg) image will be showed and the isImageOpened variable will be marked as false.

31) Now we should make that there will be two opened image at the same time and they will remain opened if they are same. Now we will use Greenfoot.delay(5) function. This will give 5 seconds before go to the next step.

32) Now edit the TrafficSign class like this:
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

class TrafficSign extends Actor
{
    private String imageName;
    private String normalImageName = "board.jpg";

    private boolean isImageOpened = false;

    public TrafficSign(String pimageName)
    {
        imageName = pimageName;
    }

    public String getImageName()
    {
        return imageName;
    }

    public void act()
    {
        if (isImageOpened == true)
        {
            return;
        }

        if (Greenfoot.mouseClicked(this))
        {
            showTrafficSign();
            Greenfoot.delay(5);
            TrafficSign openedSign = getSignWorld().getOpenedSign();
            if (openedSign != null)
            {
                if (openedSign.getImageName() != getImageName())
                {
                    openedSign.normalImageShow();
                    this.normalImageShow();
                }
                else
                {
                    openedSign.isImageOpened = true;
                    this.isImageOpened = true;
                }
                getSignWorld().setOpenedTrafficSign(null);
            }
            else
            {
                getSignWorld().setOpenedTrafficSign(this);
            }
        }
    }

    public SignWorld getSignWorld()
    {
        return (SignWorld) getWorld();
    }

    public void showTrafficSign()
    {
        GreenfootImage image = new GreenfootImage(imageName);
        setImage(image);
    }
}
A.1.6 Collect The Objects Tutorial

In this last tutorial, our purpose is to create a game in which a car, controlled by a user, try to collect the money and gasoline signs while avoiding fire signs. Our car has a default amount of gasoline at the beginning and this gasoline decrease by time. By collecting gasoline signs we are earning time and collecting signs we are earning points. Fire signs make our car lose gasoline (time). At the end of the tutorial our game will look like this:
Tutorial:
1) We start with opening a new Greenfoot project:

![Greenfoot interface](image1)

2) Name this project as CollectMoney:

![New Scenario dialog](image2)

3) And our project is opened:
4) Right click on the world class create a new class subclass:

5) Name our new class as TrafficWorld and assign the bricks2.jpg from background images:
6) When you click on compile all button you will see the greenfoot world like this:
7) With this shape our world is small for the game we want to create. So, as we did in the previous tutorials we need to change the size of our world. Double-click on the TrafficWorld class and open the code editor:

8) A world with a size (20, 20, 10) is small for our game. It's better to change this with (900, 600, 1). This means, our world will be in size 900x600 and consist of squares with size 1x1. As we know previous tutorials TrafficWorld() constructor is give necessary information while our world is created:

9) If we close the editor and click on compile all button, we will see this result:
10) Now, it is time to decide about the objects we need. If you look at the image in the beginning of the tutorial, you will see that we have 4 objects:
   - Car: This object will be controlled by a user with keyboard keys.
   - Money: This object will increase the point of the user when it is collected with the car object.
   - Gasoline: This object will increase the gasoline level of the car when it is collected with the car object. This means that remaining time will increase.
   - Fire: This object will decrease the gasoline level of the car when it is collected with the car object. This means that remaining time will decrease.

11) Let's start with money object. Copy the images which are in the same folder with this tutorial into the images folder which is in the our CollectMoney greenfoot project folder.
12) Right click on the actor class and create a new subclass.

13) Name this class as Money and select the money.png as image of this object and click on Ok.

14) After clicking on compile all button you can create money objects on our world. As you know, to do this, we should right-click on the Money class and select new Money() and then place the object onto our world.
15) We want that these money objects (also other objects) should be placed in our world when the world is created. To do this, we will use addObject method in our world class like we did in previous tutorials.
16) Double-click on the TrafficWorld class and open the editor and edit it like this:
Here, we are creating objects with specified locations in our world. As you know from previous tutorials these numbers indicates the locations in our world.

17) Close the editor and click on compile all button. You will see this screen:

![Image of objects at specified locations]

18) Our problem is that, these money objects will appear always at the same locations. In this situation, a user will memorize the locations of the objects. What we need is that, these money objects should appear on the screen in random locations. To do this we will use getRandomNumber method supplied by Greenfoot Environment. Click on Help menu of the Greenfoot and select Greenfoot Class Documentation:
19) Select the Greenfoot class from from the menu and you will see the information about the method we are talking about. I will try to describe what this method is doing.

20) This method gives us random numbers in a given range. That means we will obtain different numbers when this method is used. We will use this method to give random locations to our objects instead of specific locations. Open the editor double-clicking on the TrafficWorld class. Our world was like that:
21) Now, we will use random numbers instead of specific locations. To do this we will use random numbers for all X and Y coordinates. But since our world has a size we need to be careful about that these random numbers should be in the border of our world. We will edit our world code like this:
public TrafficWorld()
{
    // Create a new world with 900x600 cells with a cell size of 1x1 pixels.
    super(900, 600, 1);

    int randomX1 = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
    int randomY1 = Greenfoot.getRandomNumber(getHeight() - 40) + 20;

    int randomX2 = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
    int randomY2 = Greenfoot.getRandomNumber(getHeight() - 40) + 20;

    int randomX3 = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
    int randomY3 = Greenfoot.getRandomNumber(getHeight() - 40) + 20;

    addObject(new Money(), randomX1, randomY1);
    addObject(new Money(), randomX2, randomY2);
    addObject(new Money(), randomX3, randomY3);
}

Here, to obtain images in the borders of our world but not in the corners (thinking about the size of the images itself) we are taking maximum width and height and then subtract 40 and then add 20.

4) Close the editor and click on compile all button:

23) Try to click on compile all button several times and observe that position of the objects is changing in each try.
24) We have another problem here. When we want to place more objects in our world we need to create them separately. To overcome this kind of problems,
programming languages present loop structures. With these structures we can repeat a piece of code as much as we want. To do this we need to find which code is repeated. Here, these 3 lines are repeated:

```java
int randomX1 = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
int randomY1 = Greenfoot.getRandomNumber(getHeight() - 40) + 20;
addObject(new Money(), randomX1, randomY1);
```

25) If we write this code with a loop structure it will be like this:

```java
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
public class TrafficWorld extends World
{

    public TrafficWorld()
    {
        // Create a new world with 900x600 cells with a cell size of 1x1 pixels.
        super(900, 600, 1);

        for(int i=0; i<10; i++)
        {
            int randomX = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
            int randomY = Greenfoot.getRandomNumber(getHeight() - 40) + 20;
            addObject(new Money(), randomX, randomY);
        }
    }
}
```

What we are doing here is to increase the i number by 1 till the value of i equals to 10. And to create a money object with a random location each time value of i is increased. As a result we have 10 money objects with a random location in our world.

26) Close the editor and click on compile all button:
27) Click on compile button several times and observe that location of money objects is changing in each try.
28) Now, our money object is ready. Let's continue with fire and fuel objects. To create a fire object right click on the Actor class and select new subclass.

29) Name this new object as Fire and select the fire.png as image of this object and then click on Ok button.
30) To create a fuel object right click on the Actor class and select new subclass.
31) Name this new object as Fuel and select the fuel.png as image of this object and then click on Ok button.

![New class dialog box]

32) Now with the same way we will create fire and fuel objects. Open the editor again by double-clicking on TrafficWorld class and edit the code like this to add fire and fuel objects.
33) Close the editor and click on compile all button. You will see a screen like this:

![Screen with objects](image)

34) As you can see, now, we have 10 money, 10 fire and 10 fuel objects randomly located in our world.

35) Now, it is time to create the car object. We will use the codes which we used in previous tutorials without changing them. To create the car object firstly we will

```java
public TrafficWorld()
{
    super(900, 600, 1);

    for(int i=0; i<10; i++)
    {
        int randomX = Greenfoot.getRandomNumber(getWidth()) - 40 + 20;
        int randomY = Greenfoot.getRandomNumber(getHeight()) - 40 + 20;
        addObject(new Money(), randomX, randomY);
    }

    for(int i=0; i<10; i++)
    {
        int randomX = Greenfoot.getRandomNumber(getWidth()) - 40 + 20;
        int randomY = Greenfoot.getRandomNumber(getHeight()) - 40 + 20;
        addObject(new Fire(), randomX, randomY);
    }

    for(int i=0; i<10; i++)
    {
        int randomX = Greenfoot.getRandomNumber(getWidth()) - 40 + 20;
        int randomY = Greenfoot.getRandomNumber(getHeight()) - 40 + 20;
        addObject(new Fuel(), randomX, randomY);
    }
}
```
create another object which will help our car object. Right click on actor class again and select new subclass.

36) Name this new class as Mover and do not assign any images.
37) To open editor double-click on the mover class, clear all the lines inside our mover class, just copy and paste the code below, this code will help our car to move in our world. You do not need to know about this code in detail. This class is written by Michael Kölling in July 2007 to help objects move in greenfoot world.

```java
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * A Mover is an actor that also has 'move' and 'turn' ability. Both moving and turning
 * are relative to its current position. When moving, the Mover will move in the direction
 * it is currently facing.
 *
 * Both 'move' and 'turn' methods are available with or without parameters.
 *
 * The 'Mover' class is a subclass of Actor. It can be used by creating subclasses, or
 * copied into scenarios and edited inline.
 *
 * The initial direction is to the right. Thus, this class works best with images that
 * face right when not rotated.
 *
 * This class can also check whether we are close to the edge of the world.
 *
 * @author Michael Kölling
 * @version 1.0 (July 2007)
 */
public class Mover extends Actor
{
    private static final double WALKING_SPEED = 5.0;

    /**
     * Turn 90 degrees to the right (clockwise).
     */
    public void turn()
    {
        turn(90);
    }

    /**
     * Turn 'angle' degrees towards the right (clockwise).
     */
    public void turn(int angle)
    {
        setRotation(getRotation() + angle);
    }

    /**
     * Move a bit forward in the current direction.
     */
    public void move()
    {
        move(WALKING_SPEED);
    }

    /**
     * Move a specified distance forward in the current direction.
     */
```
*/
public void move(double distance)
{
    double angle = Math.toRadians( getRotation() );
    int x = (int) Math.round(getX() + Math.cos(angle) * distance);
    int y = (int) Math.round(getY() + Math.sin(angle) * distance);

    setLocation(x, y);
}

/**
* Test if we are close to one of the edges of the world. Return true if we are.
*/
public boolean atWorldEdge()
{
    if(getX() < 20 || getX() > getWorld().getWidth() - 20)
        return true;
    if(getY() < 20 || getY() > getWorld().getHeight() - 20)
        return true;
    else
        return false;
}

38) Using this class our car will move in the greenfoot world. Now it is time to
create our car object. This time right click on Mover class instead of Actor class
and select create new subclass. By doing this, we will use Mover class directly to
make our car move in the greenfoot world. As you know previous tutorials, that
means Car class inherits from Mover class.

39) Name our new class as Car and select car01.png from transport menu as image
of our object.

127
40) Double-clicking on the car class open the editor and edit the code like below. This is very similar to the process we did at the tutorials Simple Car Game 1 and 2. With this code we are trying to control our car object with keyboard keys and Mover class help us for rotations and for other things.
41) Close the editor. Click on compile all button. Right-click on Car class and selecting create new Car(), create a new car and place it on the world. Click on run button. You can see that now you can control the car object with keyboard keys.
42) Like fire, fuel and money objects we can place our car object in the greenfoot world when the world is created. Open the editor by double-clicking on the
TrafficWorld class and edit the code like this:

```java
public TrafficWorld()
{
    // Create a new world with 900x600 cells with a cell size of 1x1 pixels.
    super(900, 600, 1);

    for(int i=0; i<10; i++)
    {
        int randomX = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
        int randomY = Greenfoot.getRandomNumber(getHeight() - 40) + 20;
        addObject(new Money(), randomX, randomY);
    }

    for(int i=0; i<10; i++)
    {
        int randomX = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
        int randomY = Greenfoot.getRandomNumber(getHeight() - 40) + 20;
        addObject(new Fire(), randomX, randomY);
    }

    for(int i=0; i<10; i++)
    {
        int randomX = Greenfoot.getRandomNumber(getWidth() - 40) + 20;
        int randomY = Greenfoot.getRandomNumber(getHeight() - 40) + 20;
        addObject(new Fuel1(), randomX, randomY);
    }

    Car car = new Car();
    addObject(car, 40, 40);
}
```

43) With this edit we created a new car object and placed it in specific location (40, 40) in our world. Now our car can move in the greenfoot world but cannot collect the objects. Because we did not write anything about the interaction of our object.

44) Now, we want to make our car collect the objects when it encounters them. To do this we should edit our car class. Double-clicking on the car class open the editor and edit the code like this:
public void act()
{
    if (Greenfoot.isKeyDown("right"))
    {
        turnRight();
    }
    if (Greenfoot.isKeyDown("left"))
    {
        turnLeft();
    }
    if(Greenfoot.isKeyDown("up"))
    {
        move();
    }
    Actor intersectingActor = getOneIntersectingObject(null);
    if (intersectingActor != null)
    {
        getWorld().removeObject(intersectingActor);
    }
}

45) The code marked as yellow looks for if any objects intersecting with our car object. And if there is(are) object(s) intersecting with our car objects it removes these objects from world like we add them into the world.

    addObject → to add an object into greenfoot world
    removeObject → to remove an object from greenfoot world.

46) Click on compile all button and then click on run button. Control the car with keyboard keys and observe that objects which our car object encounters are removed from the world.
47) Now we want that when our car face with a money object our score will increase
when our car face with a fuel object our fuel(time) will increase and when our
car face with a fire our fuel(time) will decrease.

48) To show the score and fuel level we should add two variables into the car class.
Double-clicking on the car class open the editor and add these variables into the
class:

```java
private int fuel = 1000;
private final int fuel_bonus = 100;
private final int money_bonus = 10;
private final int fire_penalty = 100;
private int money = 0;
```

49) Our car class should look like this:

```java
public class Car extends Mover {
    private int fuel = 1000;
    private final int fuel_bonus = 100;
    private final int money_bonus = 10;
    private final int fire_penalty = 100;
    private int money = 0;

    private final int ROTATION_SPEED = 3;

    public void turnRight()
    {/
        turn(ROTATION_SPEED);
    }

    public void turnLeft()
    {
        turn(-ROTATION_SPEED);
    }

    public void act()
    {/
```
50) Now we need to make fuel decrease by time. To do this double-click on the car object and open the editor. Edit the code like this:

```java
public void act()
{
    fuel = fuel - 1;
    if (fuel < 0)
    {
        return;
    }

    if (Greenfoot.isKeyDown("right"))
    {
        turnRight();
    }
}
```

51) What we did here is that, when our car moves, fuel will decrease and when the fuel decreased to zero car will not move any longer. Close the editor and click on the compile all button. Observe that our car will not move when its fuel decreased to zero.

52) Open the editor again by double-clicking on the car class. Now we want to make our car behave differently when it encounters different objects. Add this code into the class:

```java
Actor intersectingActor = getOneIntersectingObject(null);
if (intersectingActor != null)
{
    if (intersectingActor instanceof Fuel)
    {
        fuel = fuel + fuel_bonus;
    }
    if (intersectingActor instanceof Fire)
    {
        fuel = fuel - fire_penalty;
    }
    if (intersectingActor instanceof Money)
    {
        money = money + money_bonus;
    }

    getWorld().removeObject(intersectingActor);
}
}
```

53) What we are doing here is that when our car object encounters with a fuel object we increase our fuel by fuel_bonus (100). When our car object encounters with a fire object we decrease our fuel by fire_penalty (100). When our car object encounters with a money object we increase our money(score) by money_bonus (10).
Close the editor and click on compile button. Click on the run button, play the game and observe that when we collect the objects and our car stops when its fuel is zero.

54) Click on pause button and right-click on car object. Select the inspect:

55) A window called object inspector will be opened. From this window you can see the value of variables at the current time. You can check your money (score) from this menu. For example, if you collect 7 money objects you should have 70 points.
Lastly, we need to show our score (money) when the game is over (when the fuel becomes zero). We did this already in the previous tutorial (Read again the related parts of the previous tutorial). To do this add the code below into the car class and call this method when the game is over.

```java
import greenfoot.*;

public class Car extends Mover {
    private int fuel = 1000;
    private final int fuel_bonus = 100;
    private final int money_bonus = 10;
    private final int fire_penal...y = 100;
    private int money = 0;
    private final int ROTATION_SPEED = 3;

    public void turnRight() {
        turn(ROTATION_SPEED);
    }
    public void turnLeft() {
        turn(-ROTATION_SPEED);
    }

    private final float fontSize = 72.0F;
    public void GameOver() {
        int worldWidth = getWorld().getWidth();
        int worldHeight = getWorld().getHeight();

        GreenfootImage image = new GreenfootImage(worldWidth, worldHeight);

        java.awt.Font font = image.getFont();
        font = font.deriveFont(fontSize);
        image.setFont(font);
    }
}
```
57) Close the editor and click on the compile all button. Play the game and observe that the score will be shown when the game is over.
Your Score: 60
A.2 Test
I prepared this test to evaluate the development of mathematical logic and coordinate system knowledge of the target group. These questions are similar to the questions in their textbooks. To evaluate the development in mathematical logic concept I have used questions about the algebra of propositions. According to the curriculum the target group study the algebra of propositions in their mathematical logic courses.

1) What is the closest distance between the point \((7, 11)\) and the x axis
   a) 7       b) 11       c) 18       d) 4

2) What is the coordinates of the point which is at the intersection of \(y = 2\) and \(x + 9 = 0\) lines.
   a) (2, 9)   b) (9, 2)   c) (-2, 9)   d) (-9, 2)

3) At what points \(y = 2x + 12\) line intersects the x and y axis.
   a) -6 and 12  b) 2 and 12  c) 12 and -6  d) 12 and -2

4) What is the distance between two points \(A(2, 4)\) and \(B(-9, 4)\) given on a coordinate system.
   a) 6       b) 8       c) 11       d) 5

5) What is the equivalent of this expression: \(p \land (p \lor q)\)'
   a) p       b) q       c) 0       d) 1

6) What is the equivalent of this expression: \(p \land (p \land q)'\)
   a) p       b) q       c) 0       d) 1

7) What is the equivalent of this expression: \([p \leftrightarrow (p \lor q)] \rightarrow p\)
   a) p       b) q       c) 0       d) 1
A.3 Survey

I prepared this survey to evaluate the development of programming knowledge of the target group. While selecting the questions I inspired from “Using Greenfoot and Games to Teach Rising 9th and 10th Grade Novice Programmers” (Mohammed et al. Using Greenfoot [..]). Some of the questions are taken directly from this work.

1) Rate your understanding of Greenfoot Environment
   a) not good  b) good  c) excellent

2) Rate your understanding of classes, objects and methods
   a) not good  b) good  c) excellent

3) Rate your understanding of data types
   a) not good  b) good  c) excellent

4) Rate your understanding of “if-else” statement
   a) not good  b) good  c) excellent

5) Rate your understanding of “for” loop structure
   a) not good  b) good  c) excellent

6) Give an example of a “if” statements (You do not have to write this as codes, you can write this with words like speaking).

7) Give an example of a “for” loop (You do not have to write this as codes, you can write this with words like speaking).