SOFTWARE ENGINEERING OF THE RACEGEN AUTOMATIC PROGRAM GENERATOR

A THESIS

Presented to the Department of

Computer Engineering and

Computer Science

California State University, Long Beach

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By Lawrence Cohen

B.S., 2003, California State University, Long Beach

May 2006

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 1434631

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 1434631

Copyright 2006 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

WE, THE UNDERSIGNED MEMBERS OF THE COMMITTEE,

HAVE APPROVED THIS THESIS

SOFTWARE ENGINEERING OF

THE RACEGEN AUTOMATIC

PROGRAM GENERATOR

By

Lawrence Cohen

COMMITTEE MEMBERS

Frank Murgolo, Ph.D. (Chair)	Computer Engineering and Computer Science
Shui Lam, Ph.D.	Computer Engineering and Computer Science
Dar-Biau Liu, Ph.D.	Computer Engineering and Computer Science

ACCEPTED AND APPROVED ON BEHALF OF THE UNIVERSITY

Wayne Dick, Ph.D.
Department Chair, Department of Computer Engineering and Computer Science

California State University, Long Beach

May 2006

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ABSTRACT

SOFTWARE ENGINEERING OF THE RACEGEN AUTOMATIC PROGRAM GENERATOR

By

Lawrence Cohen

May 2006

This thesis details the Object-Oriented software engineering process used to develop a parameter-driven automatic program generator called RaceGen. RaceGen is based on the combination of an Application Program Interface (API) generation pattern and a Code Attributes generation pattern.

Parameterization was accomplished through the use of a Graphical User Interface (GUI). The GUI allows the user to customize the output of the generator by entering selections on a series of questionnaire forms. The content of the questionnaire forms was based on the Software Requirements Definition, which began the software engineering process.

The following stages of the software engineering process (Software Life Cycle) are covered in this thesis: Analysis, Design, and Implementation. Testing was not treated as a separate step. I performed testing at every step. I did not consider the remainder of the lifecycle, i.e., Integration, Maintenance, and Retirement.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE OF CONTENTS

Page

LIST OF TABLES v

CHAPTER

1. INTRODUCTION 1

2. AUTOMATIC PROGRAM GENERATION 3
   - Background 3
   - Patterns for Automatic Program Generation 3
   - Choosing a Pattern 5
   - Using the Patterns in RaceGen 8

3. THE SOFTWARE LIFE CYCLE 9
   - Analysis 9
   - Requirements Definition 9
   - Functional Specification 12
   - Design 13
   - Implementation 15
   - Reworking the Documentation 17

4. CODING STANDARDS 18
   - File Standards 18
   - Method Standards 18
   - Naming Standards 19
   - Format Standards 20

5. WHAT WOULD I DO DIFFERENT NEXT TIME? 21

6. CONCLUSION 22
   - Development Model 22
   - Why Automatic Program Generation? 22
   - Extension of the Software 23
   - Maintenance 24

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER Page

APPENDICES 25

A. SOFTWARE DEVELOPMENT DOCUMENTATION 26

B. PREGENERATED HEADER FILES 76

C. POSTGENERATED HEADER FILES 115

D. TOOLS USED 152

E. GLOSSARY 154

F. USER'S MANUAL 156

REFERENCES 162

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF TABLES

TABLE Page

1. Patterns for Automatic Program Generation 4

2. Patterns Which I Chose to Use 6

3. Patterns Which I Chose to not Use 7

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 1

INTRODUCTION

This thesis describes the software engineering process used to develop RaceGen. This thesis is not intended to explore the game development process. It is intended to describe how an automatic program generator was developed using software engineering techniques.

RaceGen is a parameter-driven [1] automatic program generator [2] that can produce a wide variety of games. The only restriction on the game produced is that it be based on a race between a number of objects. A collection of user selected settings are used to determine the code that is generated. In the case of RaceGen, the method of collecting this data is through the use of a series of questionnaire forms. The user selects various options from the GUI and when finished generates the code. The code is then compiled by the user and may be executed. For this project, I decided to write the GUI in Visual Basic for its programming ease. I decided to have RaceGen generate C++ code, because it is the Object-Oriented language which I am most familiar.

RaceGen was developed using the Waterfall model of software engineering in which certain phases of development are completed and "signed off" prior to the start of the next phase [3]. This thesis covers the first three phases of the Waterfall model: Analysis, Design, and Implementation. The first step was the Requirements Definition. The Requirements Definition was used to determine what options would be available on the GUI, which was used to determine what code to generate. Once the Requirements

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Definition was complete, the Functional Specification was written to fulfill the Requirements. The design is based on the Functional Specification and was the last step before the code was written.

The GUI was written in Visual Basic, due to the ease of writing GUI code using the Visual Basic API. The RaceGen generator generates code in C++, using the Open GL library for graphics. The reasons for choosing C++ were to be able to use Object Oriented Programming and to be able to use the Open GL libraries to implement the graphics.

The purpose of the RaceGen automatic program generator is to generate a complete program without any programming by the user. In this way, people who know absolutely nothing about programming will be able to generate their own custom racing games, according to how they want it. This is in contrast to other automatic program generators which generate a template to which additional code must be added, which requires programming knowledge in order to produce a functional program.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 2

AUTOMATIC PROGRAM GENERATION

Background

The purpose of Automatic Program Generation is to produce code by a means which is simpler than typing it in by hand. Various methods are used to generate code. Since RaceGen is parameter-driven, a GUI is used for the user to define what code will be generated. That way, the user may customize the generated code by selecting whatever design he or she wants for the final product.

The basis of RaceGen's automatic program generation is that it selectively takes pieces of code from a Library folder and places them in a Code folder. RaceGen modifies most of the code in the Library folder according to the user's questionnaire selections prior to transfer into the Code folder. When the generation is complete, the Code folder will contain all of the code. The user will then compile the code and it will then be ready for execution by the player.

Patterns for Automatic Program Generation

Research has revealed seven basic patterns for program generation. Some of them are more suited for use by this thesis than others. This chapter is to describe each of the patterns, identify the ones that are best suited for this project and document why the others will not be used.

The following are the patterns [4]:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE 1. Patterns for Automatic Program Generation

Pattern	Description
TEMPLATES + FILTERING	Describes the simplest way of generating code. Code is generated by applying templates to textual model specifications (often XML/XMI), typically after filtering some parts of the specification. The code to be generated is embedded in the templates.
TEMPLATES + METAMODEL	An extension of the TEMPLATES + FILTERING pattern. Instead of applying patterns directly to the model, we first instantiate a metamodel from the specification. The templates are then specified in terms of the metamodel.
FRAME PROCESSING	Describes a way of generating code by means of so-called frames. Frames can be seen as programs (functions) that generate code as the result of their evaluation. Frames can be parameterized by number and string literals as well as other frame instances.
API-BASED GENERATORS	Provides an API against which code-generating programs are written. This API is typically based

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	on the metamodel/syntax of the target language.
INLINE CODE GENERATION	Describes a technique where code generation is done implicitly during interpretation or compilation of a regular, hand-written program. This process typically modifies the program that is then subsequently compiled or interpreted.
CODE ATTRIBUTES	Describes a means by which normal, handwritten program code contains annotations, or attributes, that specify things that are not contained in the code. Based on these attributes, additional code can be generated.
CODE WEAVING	Is about combining, or weaving, different parts of program text together. These different parts typically specify different independent aspects which are then combined in the woven program. Weaving is based on specifications, how the different aspects fit together, so-called join-points.

Choosing a Pattern

In order to develop an automatic program generator, patterns were chosen among these seven. Listed below are the patterns which I have selected for RaceGen and the reasons why I have chosen them.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE 2. Patterns Which I Chose to Use

Pattern	Reason for choosing
API-BASED GENERATION	It provides a way to generate a small amount of code that needs to handle a well-defined task, in this case a racing game. The general task of generating the code is well-defined, while the specifics are customized, through the use of parameterization. In addition to API-BASED GENERATION, CODE WEAVING may also have been used to implement modifications to the source code by the user.
CODE ATTRIBUTES	Its purpose is to generate code, in addition to existing code. It uses annotation in the code to identify where the additional code must go. In this case, the annotations are comments.

Next, I have listed the other patterns and for each pattern are the reasons why I chose to not use them.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE 3. Patterns Which I Chose to not Use

Pattern	Reasons for not choosing
TEMPLATES + FILTERING	It requires a powerful filtering mechanism and it is most useful when the source specification is highly structured and uses well-defined meta syntax.
TEMPLATES + METAMODEL	It is more suited for larger, model-based systems.
FRAME PROCESSING	It is too complicated for this project. At this point, I am unfamiliar with its use and would require more in-depth research to establish a level of understanding necessary to include it as the best choice.
INLINE CODE GENERATION	It is more suited for generating source code that requires operating system flexibility. To do this it needs to have a lot of preprocessing which should not be necessary for this project.
CODE WEAVING	Used for more complex projects. It would require detailed specifications and a meta-model not necessary for this project.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Using the Patterns in RaceGen

RaceGen uses a combination of the API-based pattern and the Code Attributes pattern. The API-based pattern is used to select which sections of code will be included in the final product. For example, if the user selects the boat vehicle from the questionnaire and then later clicks the Generate button, the Boat.h and Boat.cpp files will be transferred from the Library folder to the Code folder. This direct transfer is possible, because there is nothing in the boat class which is defined by the user's questionnaire selections. The files for boat's parent class, Vehicle, however cannot be transferred over as is, because the user has selected the vehicle's top speed, damage tolerance, etc. So, the Code Attributes pattern must first be applied to the Vehicle class's files before applying the API-based pattern, which is to transfer the files from the Library folder to the Code folder, based on the user's questionnaire selections.

The method used to employ the Code Attributes method is to place comments in the source code where the code is determined by the user's questionnaire selections. When the generator encounters such a comment, it will determine the code that should be there. For example, when the following comment is encountered:

```
//Declare controls
```

the generated code will be the following if keyboard controls are selected by the user:

```
//Declare controls
Keyboard kb;
```

Once the Code Attributes pattern has been applied to the entire section of code, it is transferred to the Code folder.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 3

THE SOFTWARE LIFE CYCLE

Analysis

The Analysis was a means to produce documentation which could be shown to the client to show them that their needs will be met by the software to be developed. It includes the Requirements Definition and the Functional Specification. The purpose of the Analysis is to determine what can be done to meet the client's needs. Once the Analysis was complete, it could have been shown to the client who decides if it will satisfy their requirements. (In my case, I am the client as well as the developer). They would probably have wanted changes to be made. Once the changes were complete, so was the Analysis phase and then the Design was worked on.

Requirements Definition

The requirements in the Requirements Definition are determined by the client in a real world case, but in this case, I came up with them myself. I had to play the part of the client, since there was not a real client for this product. First, I came up with the idea for an Automatic Program Generator which is capable of generating a wide variety of racing games. That became the first requirement. Thinking as a client, I started thinking of what the racing game should do. This is the basis of the remaining requirements.

The wording of the requirements are such that they are non-technical (since they are supposed to have been written by a client). Considering that the client has requested software that requires no additional programming by the user, there is a good chance that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

they do not know anything about programming and the Requirements reflect that type of client.

The requirements do need to be specific and unambiguous. This is so that the software engineers will understand what is to be done and minimize the amount of communication that is required with the client. Communication with the client is important, but discussion which is intended to clarify carelessly written Requirements only adds to the development time and ultimately could result in either additional cost or lower quality. So, to avoid this, much care was taken in the writing of the Requirements so that they are clear and specific. The more detailed and specific the Requirements are, the less the software engineers will need to establish on their own to satisfy the Requirements. In RaceGen's case, the Requirements are not extremely detailed. Since there are differing levels of knowledge among clients, Requirements specifications will vary in the amount of detail they contain. Although more customer detail is always desirable, experienced software engineers should be able to work with the client to develop a set of requirements that will lead to the development of the product the client desires. This should be true even though the client can not articulate their desires on their own.

The first step was to create Scenarios which are closely connected to the Use Case Diagrams [5] and describe expected usages of the system. The Scenarios are intended to describe the use of RaceGen, such that every possible situation is covered by the Scenarios. The Scenarios are specific. They show exactly what is being selected by the user. For example, in the first Scenario, the actual title that the user enters is listed,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

"Grand Prix Pro." I listed the type of vehicle, "car", rather than just saying that the user selects "a vehicle." All the scenarios follow the same format, but using different settings.

To create each scenario, I thought of the possible usages and recorded a specific action for each step of an occurrence. The objective of the Scenarios is to record every possible action by the actors. Due to RaceGen's generality, this set is too large to consider in its entirety. To minimize the number of scenarios, I made as many different selections between scenarios as possible. Although it was necessary to list every possible selection in the scenarios, it was not necessary to list each vehicle (i.e. car, motorcycle, airplane, etc.) and each track (i.e. street, dirt, sky, etc.). That is because, from the standpoint of the scenarios, there is no difference between a car and a motorcycle, or between a street track and a dirt track.

Once the Scenarios were complete for RaceGen, I began to design Use Case Diagrams. "Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on what a system does rather than how" [5]. The first Use Case Diagram shows what the user does to generate the code using RaceGen. It shows that the user initiates code generation which is handled by the Code Generator. It also shows that the user compiles the generated code. Once the code is compiled, then it may be used by the player. The second Use Case Diagram shows what the Code Generator does to generate the code. Once the code is generated, it is saved and may be accessed by the user. The third Use Case Diagram shows how the player uses the racing game that was generated by the user using RaceGen.

Having completed the Requirements Definition, Use Case Diagrams, and Scenarios, I created a Glossary to define application specific terms which needed to be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

clarified, such as user, developer, and player. Then I began to write the Functional Specification.

Functional Specification

The Functional Specification was written to show what will be done to satisfy each Requirement. When the Requirements were written, they were numbered. In the Functional Specification, each functional component was accompanied by a list of the Requirements which that component satisfies. One component may satisfy more than one requirement. In that case, all of the Requirements that are satisfied by that component were listed.

The Requirements Definition was "tested" against the Functional Specification. Each requirement in the Requirements Definition must appear at least once in the Functional Specification and each component in the Functional Specification must correspond to one or more requirements in the Requirements Definition.

Each component describes *what* was to be done, but not *how* it was going to be done. The components include screen shots of the GUI where applicable. In that case, there is a description of how the GUI works, what all of the buttons do, etc. Depending on how detailed the requirements were, the software engineer must originate some ideas of their own. This is not desirable, because it leads to opportunities (in the negative sense) for there to be discrepancies between what the software engineer is proposing and what the client really wants.

When the Functional Specification is complete, it is reviewed with the client. If the components do not do what the clients want, the Functional Specification

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

documentation will need to be updated. Sometimes the reason that the components do not do what the client wants is because the Requirements Definition was not written correctly. In that case, the Requirements Definition would need to be updated and the Functional Specification would also need to be updated. Once a Functional Specification is complete and the client agrees that the components satisfy the requirements, we create a list of the nouns and verbs in the Functional Specification. This list eventually became the classes, attributes and methods in the Class Diagram. I next created the Data Dictionary, which has a definition of each noun and verb from the Nouns and Verbs list. Finally, I created the Class Definition which was the starting point for the header files.

Design

The first step in the Design was the GUI Design. The GUI Design actually began in the Functional Specification, because the GUI needed to be designed for the screen shots. However, limited functionality was described in the Functional Specification, since its only purpose was for screen shots to show the client. Validation needed to be added for text entries. Functionality needed to be added to the buttons, especially the Generate button, which is used to begin generation of the code.

The GUI has three forms. The first form is used to enter information, such as the title of the game and laps per race. The second form is used to select track, vehicle, controls, and weather conditions. The last form is to select details of how damage is handled and also to set up the interactions between various components of the game, such as how a vehicle interacts with speed. For example, a higher number entered here affects

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the handling of the vehicle more when it has a high speed. Also, the third form has a Generate button which the user clicks to begin code generation.

In addition to the design of the GUI, I also created a complete list of classes, attributes, and methods. I started with the Nouns and Verbs list. For each entry in the list, I marked whether it is a class, attribute, or method. Each verb is a method. Each noun is either a class or an attribute. If a noun seemed complex, having methods and attributes of its own, I made it as a class. The simpler nouns I marked as attributes. This step provided the foundation for the Class Diagrams.

The Class Diagrams show the classes and the relationships between the classes. For example, for the Track class, the composition relationship is shown. One Track is composed of many Obstacles and one Finish Line. Also for the Track class, inheritance is shown. A Velodrome [6] is a Track and also a Street track is a Track. There are two separate Class Diagrams: one for the Generator, and one for the Generated Code. Since there is so much detail in the Class Diagrams, I listed the attributes and methods in the Class Diagram Catalog instead of in the diagrams themselves. The Class Diagram Catalog lists attributes and methods for each class. Without the Class Diagram Catalog, the attributes and methods would have to have been listed on the Class Diagram and the Class Diagram would have been too difficult to read, due to its large size.

I developed a Methods list from the Class Diagram Catalog. For each class in the Class Diagram Catalog, I listed the class name, followed by its methods. For each method I listed its signature, preconditions and postconditions. The signature shows the method name, its return type, and its parameters. The signature also shows if the method

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is const, meaning that they do not modify the object. This is important for eliminating many bugs [7].

"Preconditions and postconditions are used to specify precisely what a function does. However, ..., a precondition/postcondition specification does not indicate anything about how a function accomplishes its work." The preconditions and postconditions are like a contract, stating that if you do this for me (precondition), I will do this for you (postcondition). I will not say how I will satisfy the contract. That is left up to me. If the preconditions are not met, then the results will be unpredictable, and it will not be the responsibility of the function to check that the preconditions are being met, [8].

Implementation

The Implementation phase was when the actual code is written. The Class Diagram was used as a guide with the detailed list of methods in the Class Diagram Catalog becoming the header file for each class. If care is taken in the Analysis and Design phases to be accurate and detailed, the Implementation phase will not be difficult. Everything should fall into place as planned.

To code RaceGen, I started with the GUI, since it was needed for screen shots for the Functional Specification. There are two main parts of the GUI that needed to be coded, the questionnaire forms and the code generator. At first, I coded the forms. Using Visual Basic for this was easy. However, deciding where to put the controls was not a simple task. In addition to being functional, the GUI needed to be visually pleasing. There should not be too many nor too few controls on one form. I made the first form, the Title form, simple so that the user would have an easy introduction to the system. All

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the forms have a combination of different settings which are not necessarily related, but are together so there would not be too few controls on one form. Each form has a Submit button to accept the settings and advance to the next form and a cancel button to return to the previous form. The last form has a Generate button instead of the Submit button which generates the code.

The next step was to generate some simple code. The idea I used was to verify that the automatic program generation would work to generate a simple program before continuing with the implementation. So, I had the generator generate a window with the title which was entered on the Title form. Since none of the code had been written yet, there was nothing else to generate.

Being confident that the automatic code generator would work, I began to write the code according to the design. The code would later be used by the generator to put together a racing game, according to the user's parameter selections. Due to the Object-Oriented Design, I was confident that if the program worked with any one track and any one vehicle, then it would work for any other track and any other vehicle. The generator generates calls to methods that are defined at the abstract class level. As long as the concrete classes implement these methods as defined by their contracts, the system architecture will function as designed. I decided to implement the code for the street track and car vehicles. I wrote the code for keyboard control, mouse control, damage handling, high score lists, and weather. Although some of these (such as high score list) may not be needed for a particular game, I still needed to have it available in the library for use, in case it is selected to be used. Similarly, I would need to have the other tracks and other vehicles available too.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reworking the Documentation

One of the basic principles of applying the standard formal Software Engineering methodology is that there will be less time spent on Implementation if more care is taken on the Analysis and Design. For this reason, it was important to be thorough and detailed in the Analysis phase so that less time was spent on Implementation. Once Implementation began, however, I found that the Design was not always correct and changes had to be made for the Implementation to work.

Whenever there had been a flaw or omission in the Design or something that was left out, the Design documentation needed to be updated so that errors did not propagate from a faulty design. If this were a real-life project, communication with the customer would also be necessary before going back and making changes to the documents of the Analysis phase.

A feedback loop [3] was used to rework the documentation. Once a change was made, to the Design for example, the change would be carried over to the Analysis through the feedback loop. Through this process, all of the documentation was kept correct and in agreement.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 4

CODING STANDARDS

The Coding Standards "refer to how programming language features are used..." [9]. The following are benefits of adhering to Coding Standards [10]:

- Programmers can go into any code and figure out what is going on.
- New people can get up to speed quickly.
- People new to C++ are spared the need to develop a personal style and defend it to the death.
- People new to C++ are spared making the same mistakes over and over again.
- People make fewer mistakes in consistent environments.

File Standards

There is exactly one header file for each class. The header file contains all of the data and method declarations for that class. In addition to the header file, each class has a source file which contains the definitions of all of its methods.

Method Standards

Each method is commented with the method name, preconditions and postconditions [8]. Preconditions are what the state of the program must be prior to calling the method. This includes, but is not limited to, initialization of variables, other methods being called, and certain GUI functions being activated.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Postconditions may include return values, and side effects of the method being called.

Example: void AddDamage()

//preconditions: Damage is enabled.

//postconditions: Damage is increased for the vehicle.

Naming Standards

Class names begin with a capital letter. The first letter of each word in the class name will be capitalized.

Example: FinishLine

File names are named as the class name, followed by .h for the header file and by .cpp for the source file.

Example: FinishLine.h

FinishLine.cpp

Attribute names are all lower-case. Words are separated by an underscore.

Example: top_speed

Method names begin with a capital letter. The first letter of each word in the function name is capitalized.

Example: UpdatePosition

When an attribute name is part of the method name, only the first letter of the attribute name is capitalized.

Example: ReduceTop_speed.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Format Standards

Indentations are four spaces.

Braces are placed on their own lines.

Blank lines are inserted to separate related lines of code and make the code easier to read.

Line length is limited to eighty characters.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 5

WHAT WOULD I DO DIFFERENT NEXT TIME?

For the most part, everything went well on this project. If I were to do anything different next time, it would be to be more careful with the Design before going into the Implementation. It is much easier to make changes in the Design than it is to rearrange code and then go back and make the Design match it. What is even worse is when the problem goes back to the Analysis. In that case the Design and the Analysis documentation would all need to be updated. In a real life situation, this would be compounded by meetings with the client and concerns about meeting the deadline.

Next, I learned to establish a naming convention for attributes, methods, classes, and files before coming up with the names themselves. Not doing this on this project led to a lot of time spent changing names to be in agreement. The most difficult name was for a method which includes the name of a two word attribute. For example, top_speed is the correct format for the attribute name, according to the naming standards. Accessor utility functions start with the word "Get" and then the attribute name that it gets. Since method names have the first letter of each word capitalized and the first letter of attribute names capitalized, the correct name for the method would be GetTop_speed. However, this did not look right to me, and before I have established a naming convention, some methods may have been named such as GetTop_Speed, or Gettop_speed. It would have been good to have the coding standard in place early on in the project to avoid this confusion.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 6

CONCLUSION

Development Model

Using the Waterfall model of software engineering, I developed the RaceGen automatic program generator. RaceGen allows the user to produce a variety of racing games without having any programming knowledge. I decided to use the Waterfall model of software engineering because it is divided up into separate phases which get signed off as they are completed and then the process continues with the next phase. This works well with the RaceGen project since there is only one software engineer. If there were more software engineers, then possibly a different model would have been chosen. Rapid Prototyping was not chosen because there was no need to have a version of the software which has limited functionality [11]. Since this is a thesis project, only the final, fully functional version is of use. Another model, the Spiral model combines the features of the prototyping model and the Waterfall model [12]. This model was not used, because it is favored for large, expensive and complicated projects.

Why Automatic Program Generation?

Using RaceGen is different from using the parameters as inputs to control the programs execution, because it actually generates only the code which is necessary for the program to execute according to the parameters. This not only saves storage space, but it also improves execution speed due to the avoidance of decision making which would be required if the parameters controlled the programs execution. In addition, RaceGen allows for easy expansion of the list of vehicles and track types. By providing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the definition of the component to be added, the user merely needs to add the name to the appropriate list on the questionnaire form for the component to be available in a generated game. For example, if someone wants to add a kart vehicle, they need to create a "kart.h" file and a "kart.cpp" file which contains the definition of a kart. Then just add "kart" to the list of vehicles on the questionnaire form.

Extension of the Software

In order to improve RaceGen, the generated code would be made more graphically realistic. However, since the focus of this thesis is the software engineering process, not the appearance or physics of the gameplay of the generated code, these improvements are not really considered as part of the scope of this project, yet will be covered anyway. The tracks would be made more graphically intense as would the vehicles. There could be a vehicle editor which the user could use to design a vehicle. Moving a cursor around a grid and selecting the color of each cell in the grid, the user would be able to design the appearance of the vehicle. The tracks could be designed using a similar editor.

In addition to graphical enhancements, the artificial intelligence (or more accurately, artificial stupidity) of the computer controlled vehicles could be made to be more realistic. As it is, the computer vehicles do not collide, since the collision effects of the player controlled vehicle are sufficient to verify that the code has been generated properly. If the computer vehicles are allowed to react to collision effects, then there would need to be code implemented which recovers the vehicles from a collision. This recovery would include change of direction according to their position relative to the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

obstacle and acceleration. If the computer vehicles are made to be artificially stupid (able to collide), then it seems fitting for them to have the capability to avoid collisions. This would include detection of collisions with obstacles and other vehicles. Therefore, this would provide the ability of the computer controlled vehicles to steer around the course. Once again, this is not a game programming thesis, so this functionality has been left as extensions of this project.

Maintenance

In order for future developers to maintain the software they, they need to know that the code to be edited is located in the "Library" folder. Editing the code in the "Code" folder is useful only to test the changes to the generated code. However, since the files in the "Code" folder are generated by RaceGen, they will be lost (written over) once new code is generated by RaceGen. Any code that is dependent on questionnaire form selections needs to be included in or excluded from the generated code based on those selections. Therefore, the RaceGen code itself needs to be modified. This portion of the code is located in the code for the Damage form which is called when the Generate button is clicked by the user.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDICES

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX A

SOFTWARE DEVELOPMENT DOCUMENTATION

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

REQUIREMENTS DEFINITION

Requirement Number	Requirements
1	The software being developed will need to make racing games on its own that can be played without having to do any computer programming.
2	It has to be able to make a variety of games that are easy to build.
3	There needs to be a way for someone to select different things to be in the game, like vehicles, tracks, and weather.
4	We should also be able to decide what kind of controls can be used to play a game.
5	As far as the types of vehicles go, there will be cars, motorcycles, airplanes, bicycles, boats, horses, and dogs available to select from.
6	There should be tracks available that those types of vehicles would race on, like a street track, dirt track, a track in the sky, water, and a bicycle racing track (called a "velodrome").
7	The weather conditions should be optional for each game being made. If the option is not wanted, then there will not be weather conditions in the game. However, if the weather conditions are selected, whoever is making the game should be able to select which kind of weather conditions will be in the game, such as rain, snow, hail, and clouds.
8	There should be a way to make the chances of the weather being different for each race. In that case, maybe there could be a way to make one type of weather more common in a game. Like maybe I want to have a lot of rainy races and not much hail.
9	Someone should be able to specify what controls are available for each game. Possible controls to select from would be keyboard and mouse.
10	If a game is made that allows more than one type of controller, then the person playing the game should have an option to change it.
11	One type of controller will be set as the default by whoever is designing the game.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

12	If keyboard controls are available, there needs to be a way to assign accelerate, brake, and steering to different keys by the game designer and by the player if that is how the designer wants to make the game.
13	They could just be made a fixed set of keys.
14	Whoever is making a game should be able to name the game and display the title for everyone who plays it to see.
15	When someone gets done with a race, they should be able to see a list of high scores, if that is how the designer wants the make the game.
16	The tracks and vehicles should be chosen from a list so that the person making a game does not have to design their own vehicles and tracks.
17	They should be able to set how many other vehicles are in the race.
18	They should be able to set the top speed of the vehicle.
19	There should be obstacles in the game for vehicles to crash into.
20	The user will assign the effects of obstacles on vehicles.
21	The user may choose to enable vehicle damage.
22	If vehicle damage is enabled, it will be incremental, such that each amount of damage done to the vehicle will be kept track of.
23	When a vehicle has encountered an obstacle, damage will be assigned to the vehicle.
24	Incremental damage may be opted to affect vehicle performance.
25	The effects of incremental damage on the performance of the vehicle will be handling and top speed.
26	The user will select which sections of the vehicle affect speed or handling when damaged.
27	The user will assign the damage tolerance for each vehicle. This is the amount of damage that will cause the vehicle to breakdown and no longer be able to drive at all.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

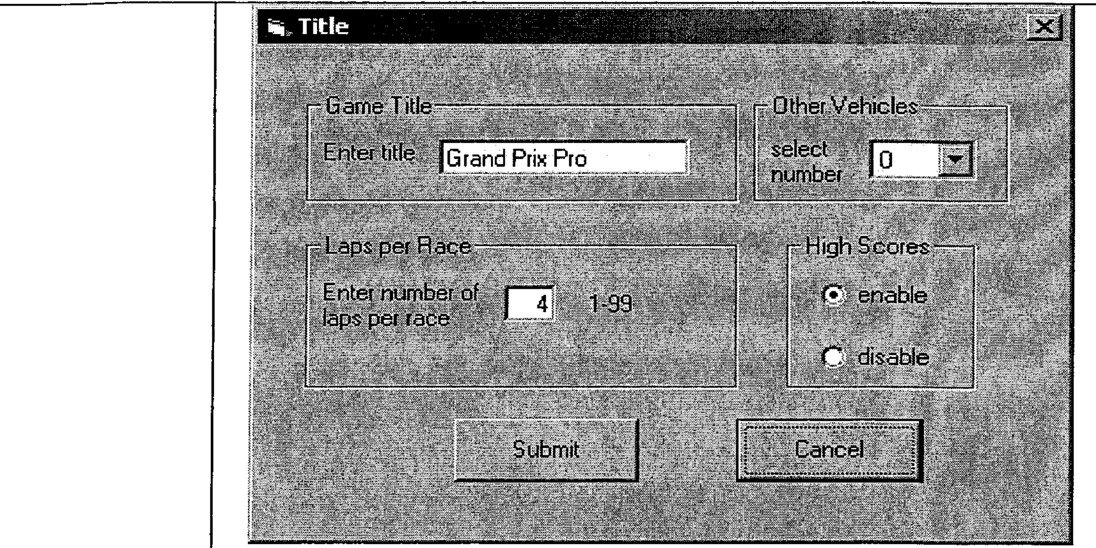
28	The game designer will have the ability to set the effect of interaction between different settings of the parameters. The settings that will interact, thus affecting game play, are the following: • Vehicle and track type • Track and weather condition • Vehicle and obstacles – concerning how the vehicle reacts to the collision • Front end of vehicle and obstacles – concerning what permanent effect the collision has on the vehicle when hit in front. • Rear end of vehicle and obstacles – concerning what permanent effect the collision has on the vehicle when hit in the rear. • Vehicle and speed
29	The game designer will be able to specify the number of laps per race.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

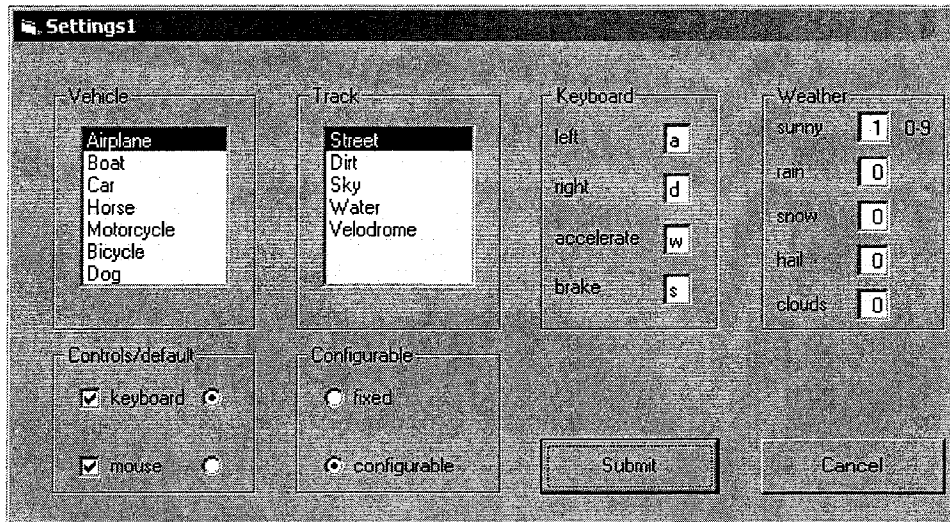
FUNCTIONAL SPECIFICATION

Requirement Number	Functional Components
1	The RaceGen racing game program generator will provide a means for the user to generate code for a racing game by entering selections from a questionnaire. The code will be generated based on these selections without any programming done by the user. Everything in the functional specification, i.e. ease of use, user-friendliness is an overall goal of the specification and will be addressed by all specifications. This requirement will be satisfied by everything in the product.
2	The questionnaire forms will provide the user with a variety of choices for game generation. The forms will be easy for the user to complete. They will allow for easy navigation and be user-friendly.
14,15,17,29	The user will be able to enable or disable the high scores list, according to the following GUI. The player will be able to enter initials to be displayed with the score. The high scores will be stored in a file and be listed in order at the end of a race. The following is how the user will enter the title of the game. This title will appear as the title of the window while the game is being played. The following GUI will be used to assign the number of laps per race. The allowable range is 1-99. The following GUI will be used to allow the user to select how many vehicles will be in the race, in addition to the player's vehicle. The user may enter from 0-5 additional vehicles.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	 Pressing the "Submit" button will accept the selections and continue to the next page. Pressing the "Cancel" button will reject the selections and exit RaceGen.

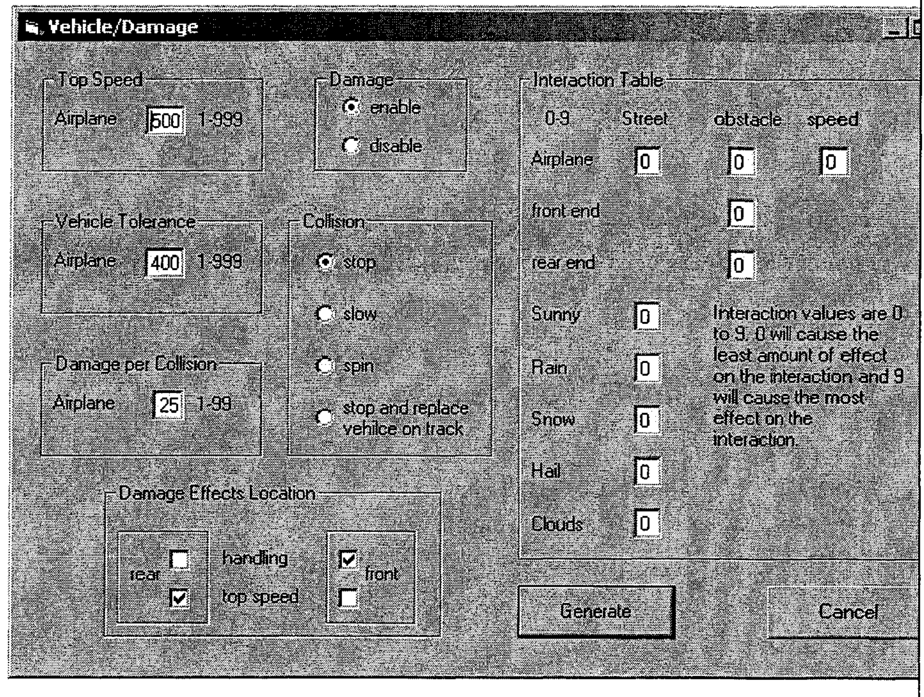
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3,4,5,6,7,8,9, 11,12,13,16	The following Graphical User Interface (GUI) is used to select the vehicles, tracks, weather, and controls:  Select the controls which are to be made available to the user by marking the checkboxes to the left of the control name. Use the radio buttons to the right to select the default control. If more than one control type is selected from the GUI, then the player will have the option of changing the controller. For weather: The numbers entered are used to determine the probability that the weather condition will come up for a race. The weather conditions will be selected before each race, based on these probabilities. Only one weather condition will be applied to each race. The selected weather condition will be applied for the entire race. A higher number represents a more probable weather condition. If only one entry is non-zero, then that weather condition will be applied to each race. There must be at least one non-zero entry. Pressing the “Submit” button will accept the selections and continue to the next page. Pressing the “Cancel” button will reject the selections and return to the previous page.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

10, 12	The following menu is used by the player to start a race, assign keys for keyboard control if it is available and configurable is selected, and change controls if both are available: 1) Start race 2) Configure keyboard 3) Change controls These are the prompts for the user to configure the keyboard controls: Enter Left: Enter Right: Enter Accelerate: Enter Brake: Once entries are made for each prompt, the keyboard keys will be assigned. When the user opts to change controls, mouse controls will be used if keyboard was previously selected, and vice versa.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

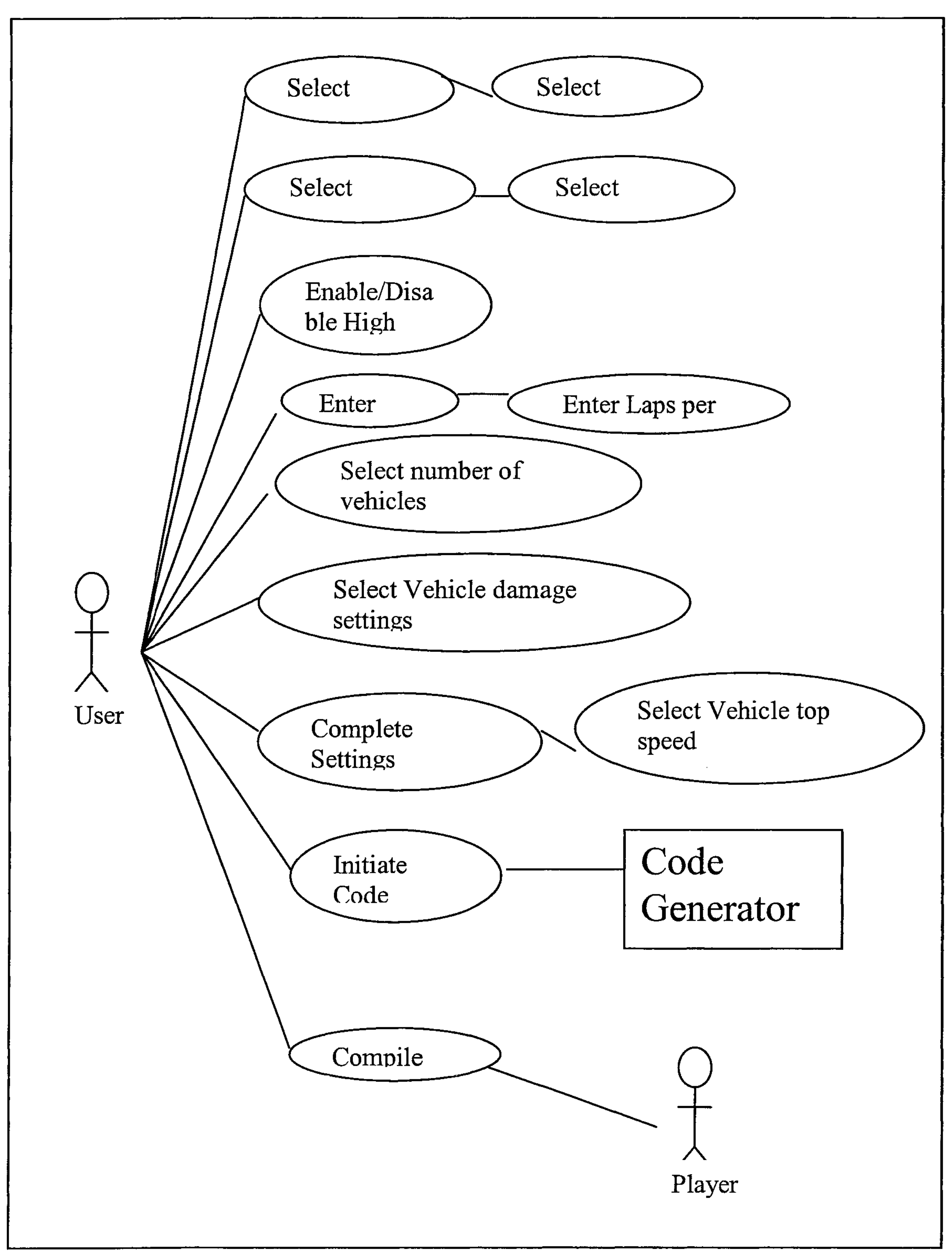
18,20,21,22, 23,24,26,27	The following GUI will be used for the user to set the top speed of the vehicles for the selected vehicle type. The allowable range is 1-999. It will be used by the user to enable or disable vehicle damage. It will be used for the user to select either handling or top speed to be affected if damage is assigned to the rear or front of the vehicle. The user will use it to select the affect of a collision and to assign the damage tolerance for each vehicle. The allowable range is 1-999. When a vehicle is created, it will be initialized with zero damage. Each time it is involved in a collision, if damage is enabled, an amount of damage will be added to the vehicle's total damage, according to the following GUI. The amount entered is how much damage is done to the vehicle. The allowable range is 1-99. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

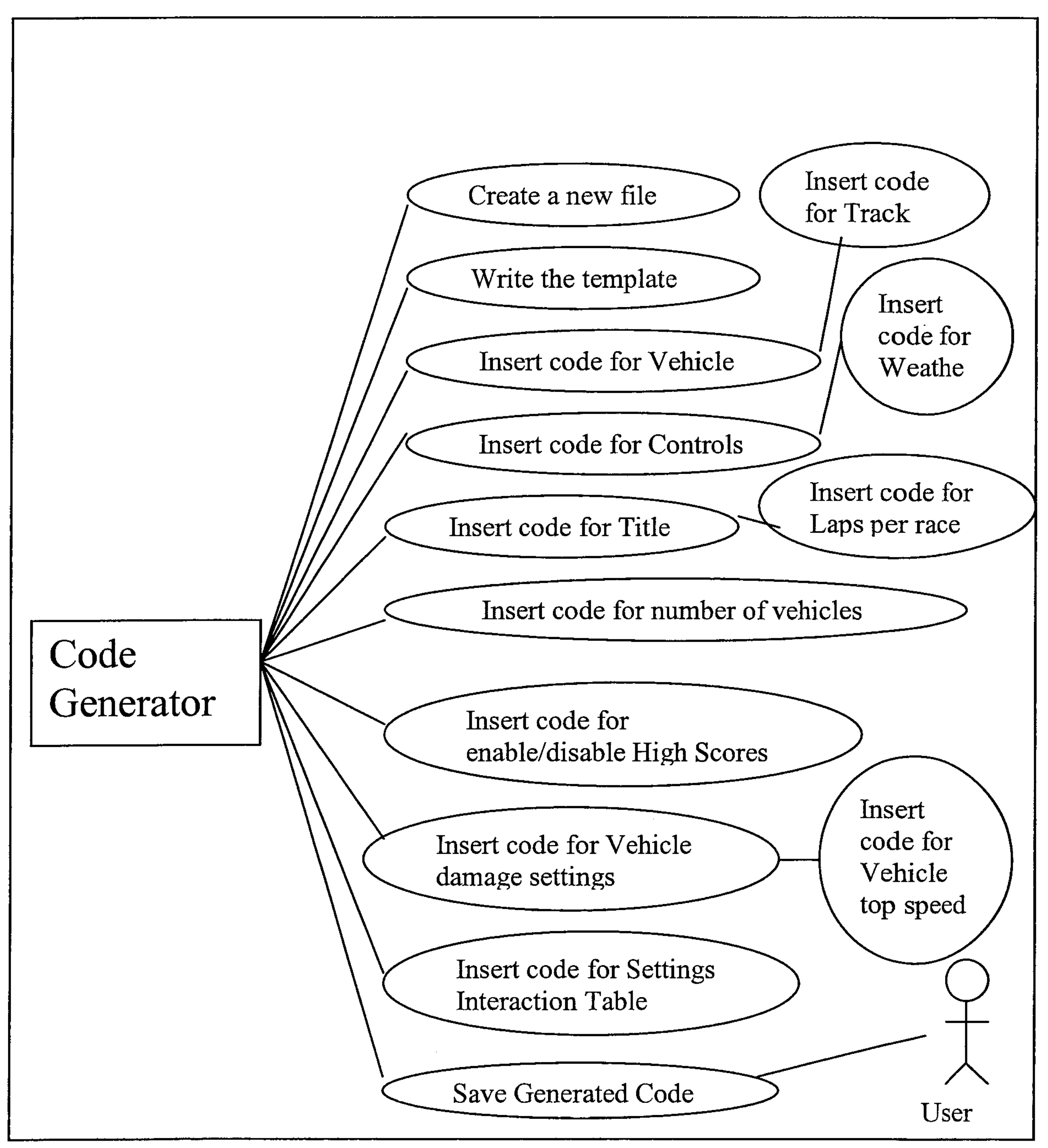
23,28	The user will use the GUI table to set the effect of interaction between different settings of the parameters. Vehicle / track, track / weather, vehicle / speed: The numbers represent the effect on vehicle handling. A higher number will make steering more difficult. The vehicle will tend to slide to the outside of turns. Vehicle / obstacles: If the spin or slow options are selected (see #21), then this is the amount of spin or slow. The above settings will affect the performance and handling of the vehicles. A probability will be determined, based on the user defined settings, which will be used to update the vehicle positions. When the vehicle positions are updated, a check will be done to determine if each vehicle has encountered an obstacle. Once the damage to the car has been assigned, a check will be made to determine if the car has reached its maximum damage tolerance. When the vehicle positions are updated, a check will be done to determine if each vehicle has crossed the finish line. Pressing the "Generate" button will accept the selections and generate the code. Pressing the "Cancel" button will reject the selections and return to the previous page.
19	Obstacles will be placed alongside the track at various locations so that a vehicle which stays on the track will not collide with one and a vehicle which strays off the course must try to avoid them.
23,25 (see #26)	Whenever a vehicle is damaged and effects of damage on performance are selected, either the handling or top speed will be reduced, based on the location of the damage on the vehicle and the user's selection of what is affected by damage to that section. Each vehicle will have a front section and a rear section. When the vehicle is damaged, if top speed is assigned to that section by the user, the top speed will be reduced. If handling is assigned to the damaged section, then the vehicle will be more difficult to steer.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

USE CASE DIAGRAM



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

SCENARIOS

The following are scenarios for RaceGen in which the **user** is a person who wants to generate a racing game program and the **player** is a person who plays the game designed by the user.

1) Scenario 1
   a) User's actions
      i) Title form:
         The user enters the title as "Grand Prix Pro," enables high scores and assigns 4 laps per race. The user selects 0 other vehicles. The user clicks on the submit button.
      ii) Settings1 form:
         The user selects a car to race on a street track. The user enables keyboard and mouse controls and selects keyboard as the default. The user selects fixed keyboard controls and assigns A to left, D to right, W to accelerate, and S to brake. The user selects weather to be 1 for sunny and 0 for the others. The user clicks on the submit button.
      iii) Vehicle/Damage form:
         The user selects top speed of 200 and enables damage. The user selects tolerance of 400, damage per collision of 25, stop and replace vehicle on track when a collision occurs, and selects rear damage to affect top speed and front damage to affect handling. The user enters car interacts with street track of 0, speed of 3, and obstacle of 4. The user enters sunny interacts with street track of 0. The user clicks on the generate button.
      iv) The user compiles and saves "Grand Prix Pro."
   b) Player's actions
      i) Player runs the "Grand Prix Pro" game.
      ii) The weather is sunny.
      iii) Player completes 4 laps without colliding with an obstacle.
      iv) Player gets the high score and enters his/her initials.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2) Scenario 2
   a) User's actions
      i) Title form:
         The user enters the title as "Motocross Racer," disables high scores and assigns 8 laps per race. The user selects 3 other vehicles. The user clicks on the submit button.
      ii) Settings1 form:
         The user selects a motorcycle to race on a dirt track. The user enables mouse controls. Mouse is automatically the default. The user cannot edit keyboard controls. The user selects weather to be 0 for everything. The user clicks on the submit button. The cursor moves to the sunny field, because the entries may not be all zeroes. The user changes rain to 3. The user clicks the submit button.
      iii) Vehicle/Damage form:
         The user selects top speed of 50 and disables damage. The user selects for the motorcycle to slow when after a collision with an obstacle. The user cannot edit the other settings. The user enters motorcycle interacts with dirt track of 2, speed of 2, and obstacle of 2. The user enters rain interacts with dirt track of 6. The user clicks on the generate button.
      iv) The user compiles and saves "Motocross Racer."
   b) Player's actions
      i) Player runs the "Motocross Racer" game.
      ii) The weather is raining.
      iii) Player completes 8 laps with many collisions with obstacles. Damage is disabled, so the motorcycle is not damaged, but it does slow down.
      iv) High scores are disabled, so the player's score is not saved.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3) Scenario 3
   a) User's actions
      i) Title form:
         The user enters the title as "Air Rally," enables high scores and assigns 3 laps per race. The user selects 2 other vehicles. The user clicks on the submit button.
      ii) Settings1 form:
         The user selects an airplane to race in the sky. The user enables keyboard controls. Keyboard is automatically the default. The user selects configurable keyboard controls and assigns A to left, D to right, W to accelerate, and S to brake as the defaults. The user selects weather to be 1 for sunny and 0 for the others. The user clicks on the submit button.
      iii) Vehicle/Damage form:
         The user selects top speed of 300 and enables damage. The user selects tolerance of 200, damage per collision of 20, spin when a collision occurs, and decides not to have damage effects, and so unchecks damage effects locations. The user enters airplane interacts with sky of 0, speed of 1, and obstacle of 9. The user enters sunny interacts with sky of 1. The user clicks on the generate button.
      iv) The user compiles and saves "Air Rally."
   b) Player's actions
      i) Player runs the "Air Rally" game.
      ii) The player does not like the keyboard configuration and changes A,W,D,S to J,I,L,K.
      iii) The weather is sunny.
      iv) Player completes 2 laps, colliding with 10 obstacles. The airplane now has reached its damage tolerance and the race is over.
      v) Player does not get the high score, and does not get to enter initials.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4) Scenario 4
   a) User's actions
      i) Title form:
         The user enters the title as "Grand Prix Pro," enables high scores and assigns 4 laps per race. The user selects 0 other vehicles. The user clicks on the submit button.
      ii) Settings1 form:
         The user selects a car to race on a street track. The user enables keyboard and mouse controls and selects keyboard as the default. The user selects fixed keyboard controls and assigns A to left, D to right, W to accelerate, and S to brake. The user selects weather to be 1 for sunny, 1 for snow, and 0 for the others. The user enters car interacts with street track of 0, speed of 3, and obstacle of 4. The user enters sunny interacts with street track of 0. The user clicks on the submit button.
      iii) Vehicle/Damage form:
         The user selects top speed of 200 and enables damage. The user selects tolerance of 400, damage per collision of 25, stop when a collision occurs, and selects rear damage to affect top speed and handling and front damage to affect top speed and handling also. The user clicks on the generate button.
      iv) The user compiles and saves "Grand Prix Pro."
   b) Player's actions
      i) Player runs the "Grand Prix Pro" game.
      ii) The player does not want to use keyboard controls, so changes to mouse control.
      iii) The weather is snowing.
      iv) Player completes 4 laps while hitting many obstacles. Each time an obstacle is hit, damage is assigned to the vehicle. The damage done to the front of the vehicle causes a decrease in handling and a lower top speed. Damage done to the rear of the vehicle does the same.
      v) Player gets the high score and enters his/her initials.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

NOUNS AND VERBS

Nouns
accelerate
airplane
available control
bicycle
boat
brake
car
clouds
code
collision effect
configurable controls
controls
coordinate
current control
damage
damage effect
damage per collision
damage tolerance
developer
direction
dirt
dog
fixed controls
front of vehicle
GUI
hail
high score list
high scores file
horse
initials
interaction parameter
laps per race
left
motorcycle
mouse
obstacle
player
position
questionnaire

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

race
RaceGen
rain
range
rear of vehicle
right
score
sky
snow
speed
start / finish line
street
time
title
tolerance
top speed
track
track and weather interaction parameter
user
vehicle
vehicle and obstacle interaction parameter
vehicle and speed interaction parameter
vehicle and track type interaction parameter
vehicle damage
vehicle damage tolerance
vehicle front end and obstacle interaction parameter
vehicle handling
vehicle position
vehicle rear end and obstacle interaction parameter
velodrome
water
weather
weather condition probabilities
weather condition weight

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Verbs
add damage
affect handling
affect top speed
assign damage
calculate score
calculate next vehicle position
calculate weather condition probability
cancel
cause vehicle to breakdown
change control
check collision
check if vehicle crossed finish line
check maximum damage
collide with an obstacle
determine weather condition
disable high scores
disable vehicle damage
display high scores
enable high scores
enable vehicle damage
generate code
reduce handling
reduce top speed
set available controls
set configurable controls
set current control
set damage effects
set damage per collision
set default control
set initials
set interaction parameter
set keyboard keys
set number of laps per race
set number of other vehicles
set title
set top speed
set vehicle damage tolerance
set weather condition
slide
slow
sort high scores

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

spin
stop and replace vehicle on track
store high scores
submit
update vehicle positions
validate damage per collision
validate number of laps per race
validate vehicle damage tolerance
validate weather condition probabilities

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

DATA DICTIONARY

Nouns	Definition
accelerate	A key to make the vehicle go faster.
airplane	A vehicle that would typically be in the sky.
available control	A control which the user allows the player to use during a race.
bicycle	A vehicle that would typically be in a velodrome.
boat	A vehicle that would typically be in the water.
brake	A key to make the vehicle go slower.
car	A vehicle that would typically be on a street track.
clouds	A weather condition which may limit visibility.
code	The instructions which the computer executes for a player to play a game.
configurable control	A control which the player may assign functions to.
controls	What the player uses to drive a vehicle.
coordinate	The x and y position on the track.
current control	The control that the player is using.
damage	See vehicle damage.
damage effects	What is affected by damage to the front or rear of a vehicle. Either top speed, handling or neither.
damage per collision	See vehicle damage per collision.
damage tolerance	See vehicle damage tolerance.
developer	A person who is designing the RaceGen software.
direction	The angle that the vehicle is moving.
dog	A small animal which is considered to be a vehicle.
fixed controls	A control which the player is not allowed to configure.
front of vehicle	The section of the vehicle forward of the midpoint.
GUI	Graphical User Interface. It is used for the user or player to interact with the computer and make selections that will be used by the computer.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

hail	A weather condition of icy chunks falling from the sky.
high score list	The highest scores and the initials of the players who scored them.
high scores file	Where the high scores are stored.
horse	A large animal which is considered to be a vehicle.
initials	The first letter of the first, middle, and last name of a player.
interaction parameter	A value entered by the user to determine the outcome of two factors acting on one another.
laps per race	How many times the vehicle must complete a circuit of the track to complete a race.
left	A key to steer the vehicle to the left.
motocross	A hilly, dirt track.
motorcycle	A vehicle that would typically be on a street track or a dirt track.
mouse	A controller with a ball and two or three buttons.
obstacle	Something which a player must try to avoid while racing.
player	A person who plays the game which the user generated with RaceGen.
position	The coordinates of the location on the track.
questionnaire	A form requiring a series of selections to be made by the user in order to generate a game.
race	Driving a vehicle around a track.
RaceGen	Software used to automatically generate code for racing games.
rain	A weather condition of water droplets falling from the sky.
range	Used to identify the set of valid inputs.
rear of vehicle	The section of the vehicle behind the midpoint.
right	A key to steer the vehicle to the right.
score	A value assigned at the end of a race which represents the skill of the player.
sky	A track in the air.
snow	A weather condition of snowflakes falling from the sky.
speed	See vehicle speed.
start / finish line	The location across the track where a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	vehicle starts behind and must cross to complete a lap.
street	A flat, paved track.
time	The length of the race. Used to calculate the score.
title	A name which the user assigns to a game being generated.
top speed	The maximum velocity of a vehicle.
total probability	The sum of the weather condition probabilities.
tolerance	See vehicle damage tolerance.
track	The course which the vehicles must follow in order to complete laps.
track and weather interaction parameter	A value entered by the user to determine the outcome of weather acting on a track.
user	A person who uses RaceGen to generate code for a racing game.
vehicle	An object driven on a track by the player in order to complete a race.
vehicle and obstacle interaction parameter	A value entered by the user to determine the outcome of an obstacle acting on a vehicle.
vehicle and speed interaction parameter	A value entered by the user to determine the outcome of speed acting on a vehicle.
vehicle and track type interaction parameter	A value entered by the user to determine the outcome of a track type acting on a vehicle.
vehicle damage	A value which represents the amount suffering an obstacle inflicts upon a vehicle.
vehicle damage per collision	The amount of damage which would be assigned to the vehicle for each collision.
vehicle damage tolerance	The amount of damage that will cause a vehicle to breakdown and no longer be driven.
vehicle front end and obstacle interaction parameter	A value entered by the user to determine the outcome of an obstacle acting on the front end of a vehicle.
vehicle handling	The ability of the vehicle to steer.
vehicle position	The location of a vehicle with respect to the track
vehicle rear end and obstacle interaction parameter	A value entered by the user to determine the outcome of an obstacle acting on the rear end of a vehicle.
vehicle speed	The distance a vehicle travels per unit of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	time.
velodrome	A banked oval track.
water	An aquatic track.
weather	Elemental effects which could influence how a vehicle performs.
weather condition probabilities	Values for a weather condition which are used to determine how commonly it will occur in a race.
weather condition weight	Values for a weather condition which are used to calculate the probabilities.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Verbs	
Accelerate	Make the vehicle go faster.
add damage	Increase a damage value assigned to a vehicle due to a collision.
affect handling	Influence the handling of a vehicle due to a collision.
affect top speed	Influence the top speed of a vehicle due to a collision.
assign damage	Affect either handling or top speed or neither, based on the front and rear damage effects.
brake	Make the vehicle go slower.
calculate next vehicle position	Determine the new location of a vehicle after it moves.
calculate score	Determine the score, based on the player's performance.
calculate weather condition probability	Determine the probability value for a weather condition, based on the value entered for it and the other weather conditions by the user.
cancel	Reject the current questionnaire form selections and return to the previous form.
cause vehicle to breakdown	A vehicle is so damaged that it can no longer be driven.
change control	Use a control other than the current control.
check collision	Determine if the vehicle collides with an obstacle.
check if vehicle crossed finish line	Determine if the vehicles new location will place it on the other side of the finish line, indicating a completed lap.
check maximum damage	Determine if the amount of damage assigned to the vehicle has equaled or exceeded the vehicle damage tolerance.
collide with an obstacle	An intersection of a vehicle and an obstacle.
collision effect	How the vehicle reacts to a collision.
determine weather condition	Determine which weather condition will be applied to a race.
disable high scores	Do not allow a high score list to be generated or displayed.
disable vehicle damage	Do not allow a vehicle to be damaged.
display high scores	Show the player a list of the highest scores

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	and the initials of the player who got them.
enable high scores	Generate and display a high score list after each race.
enable vehicle damage	Allow a vehicle to be damaged.
generate code	Create a computer program, based on the selections made by the user.
reduce handling	Make the vehicle more difficult to steer.
reduce top speed	Make the maximum speed of the vehicle lower.
set available controls	Set which controls the player will be able to use in a race.
set configurable controls	Set whether controls will be fixed or configurable.
set current control	Set which control will be used in the race. This is automatically done if there is only one available control and the user will not be able to set it.
set damage effects	Set whether handling and/or top speed, or neither will be affected when damage is done to the front or rear of the vehicle.
set damage per collision	Set a value which will be added to the vehicle damage after a collision.
set default control	Set a control to be the default. If there is only one available control, then it will be the default.
set initials	Initials are entered into the high score list.
set interaction parameter	Values are entered into the interaction table.
set keyboard keys	Set which keyboard keys will be used to control the vehicle.
set number of laps per race	Set how many laps will complete a race.
set number of other vehicles	Set how many vehicles will be in the race in addition to the player's vehicle.
set title	Set the name for the game being generated.
set top speed	Set a maximum value for the speed of the vehicle.
set vehicle damage tolerance	Set the amount of damage that will cause a breakdown of the vehicle.
set weather condition	Apply the selected weather condition to a race.
slide	The vehicle moves toward the outside of a turn
slow	The speed of a vehicle is reduced.
sort high scores	Alphabetize the list of high scores.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

spin	The vehicle rotates upon a vertical axis through its center.
steer left	Steer the vehicle to the left.
steer right	Steer the vehicle to the right.
stop and replace vehicle on track	The vehicle speed is set to zero and its location is updated to a position on the track.
store high scores	The high score list is saved on a disk.
submit	Accept the current questionnaire form selections and advance to the next form.
unused control	The control which is not the current control
update vehicle positions	Move the vehicle's location.
validate damage per collision	Determine if the user's input is within range.
validate number of laps per race	Determine if the user's input is within range.
validate vehicle damage tolerance	Determine if the user's input is within range.
validate weather condition probabilities	Determine if the user's input is within range.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CLASSES, ATTRIBUTES, AND METHODS

Nouns	Class	Attribute	Method
accelerate	X		
airplane	X		
available control		X	
bicycle	X		
boat	X		
brake	X		
car	X		
clouds	X		
code	X		
configurable controls		X	
controls	X		
current control		X	
damage per collision		X	
dirt	X		
dog	X		
fixed controls		X	
front of vehicle	X		
GUI	X		
hail	X		
high score list	X		
horse	X		
initials		X	
interaction parameter		X	
laps per race		X	
left	X		
motorcycle	X		
mouse	X		
obstacle	X		
rain	X		
rear of vehicle	X		
right	X		
score		X	
sky	X		
snow	X		
start / finish line		X	
street	X		
title		X	
top speed		X	
track	X		

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

track and weather interaction parameter		X	
vehicle	X		
vehicle and obstacle interaction parameter		X	
vehicle and speed interaction parameter		X	
vehicle and track type interaction parameter		X	
vehicle damage		X	
vehicle damage tolerance		X	
vehicle front end and obstacle interaction parameter		X	
vehicle handling		X	
vehicle position		X	
vehicle rear end and obstacle interaction parameter		X	
velodrome	X		
water	X		
weather	X		
weather condition probabilities		X	

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

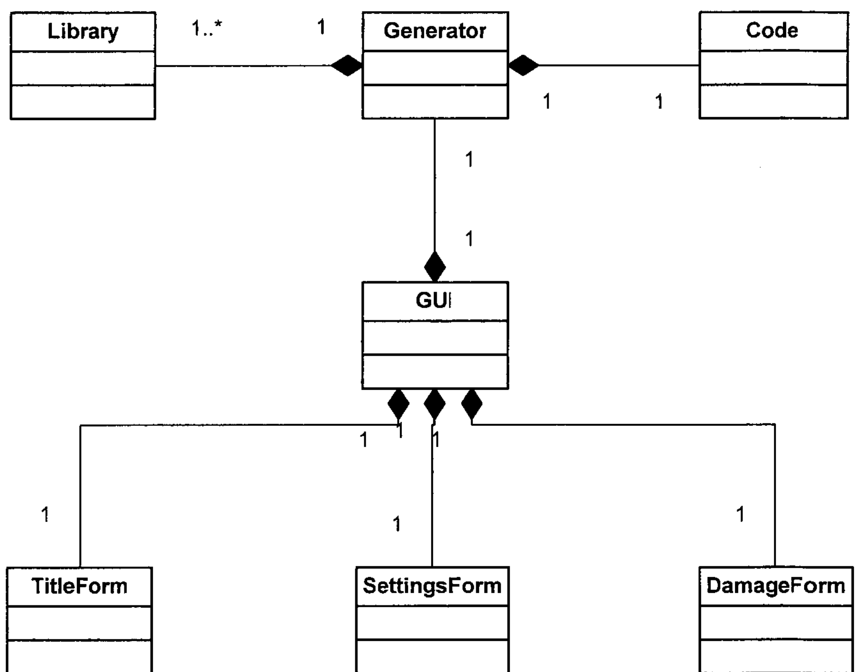
Verbs	Class	Attribute	Method
add damage			X
affect handling			X
affect top speed			X
calculate score			X
calculate vehicle position probability			X
calculate weather condition probability			X
cancel			X
cause vehicle to breakdown			X
change control			X
check collision			X
check if vehicle crossed finish line			X
check maximum damage			X
collide with an obstacle			X
determine weather condition			X
disable high scores			X
disable vehicle damage			X
display high scores			X
enable high scores			X
enable vehicle damage			X
generate code			X
reduce handling			X
reduce top speed			X
set available controls			X
set configurable controls			X
set current control			X
set damage effects			X
set damage per collision			X
set default control			X
set initials			X
set interaction parameter			X
set keyboard keys			X
set number of laps per race			X
set number of other vehicles			X
set title			X
set top speed			X
set vehicle damage tolerance			X
set weather condition			X
slide			X

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

slow			X
sort high scores			X
spin			X
stop and replace vehicle on track			X
store high scores			X
submit			X
update vehicle positions			X
validate damage per collision			X
validate number of laps per race			X
validate vehicle damage tolerance			X
validate weather condition probabilities			X

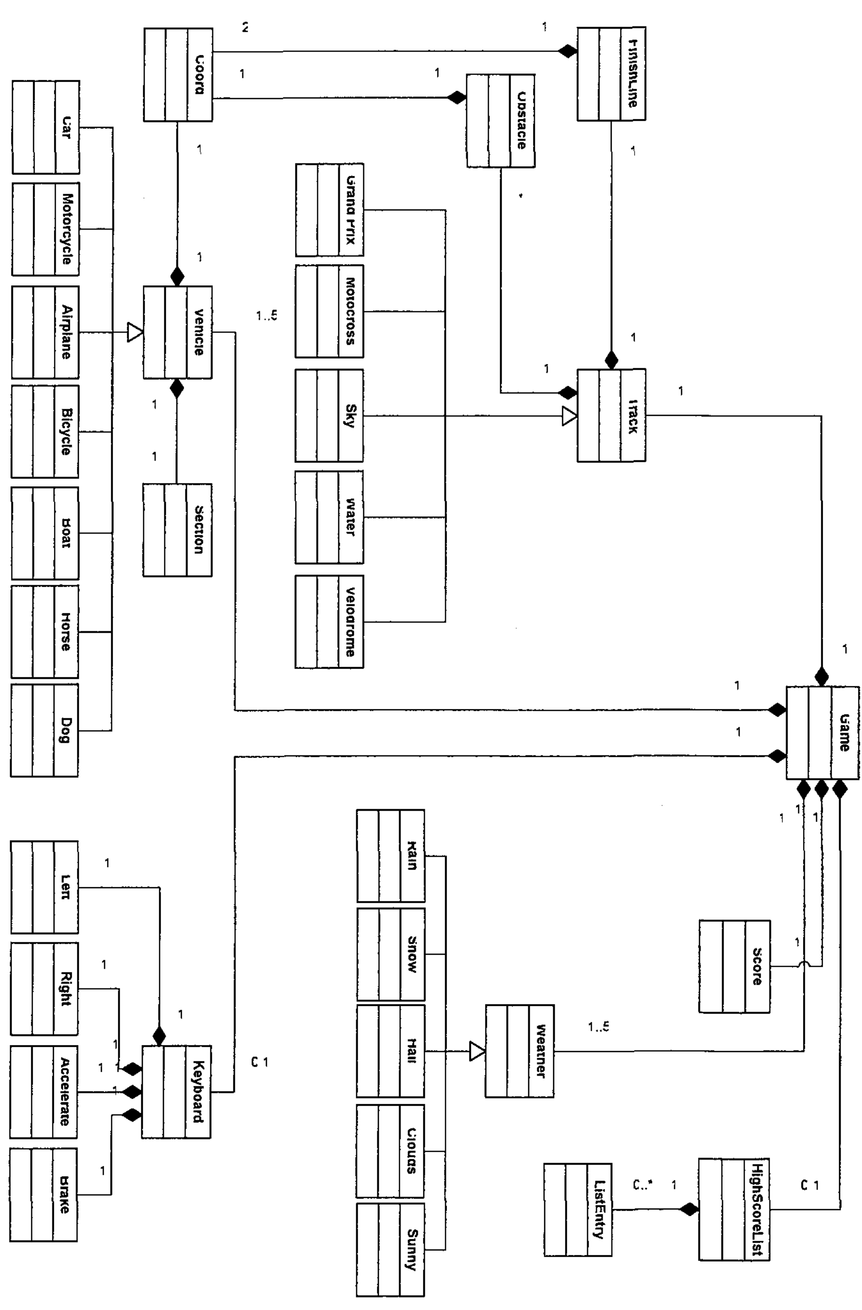
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CLASS DIAGRAM OF CODE GENERATOR



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CLASS DIAGRAM OF GENERATED CODE



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CLASS DIAGRAM CATALOG

Class	Attributes	Methods
Game	title	IsRaceOver
	laps_per_race	End
	vehicles_in_race	
	track_sunny_interaction	
	track_rain_interaction	
	track_snow_interaction	
	track_hail_interaction	
	track_clouds_interaction	
	vehicle_obstacle_interaction	
	vehicle_track_interaction	
	front_obstacle_interaction	
	rear_obstacle_interaction	
	vehicle_speed_interaction	
	track	
	high_score_list	
	score	
	keyboard	

Class	Attritbutes	Methods
Track	comp	DisplayLaps
	Finish_line	IsCollision
	obst	CheckLap

Class	Attritbutes	Methods
GrandPrix		

Class	Attritbutes	Methods
Dirt		

Class	Attritbutes	Methods
Sky		

Class	Attritbutes	Methods

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Water		

Class	Attritbutes	Methods
Velodrome		

Class	Attributes	Methods
Obstacle	xpos	Display
	ypos	GetXpos
		GetYpos
		SetPos

Class	Attributes	Methods
Coord	x	GetX
y	y	GetY
		SetX
		SetY

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Class	Attributes	Methods
Vehicle	race_time	Accelerate
	damage	Brake
	position	SteerLeft
	speed	SteerRight
	direction	CalculateNextVehiclePosition
	next_position	HitObstacle
	top_speed	AddDamage
	top_speed_min	IsMaxDamage
	handling	Slide
	handling_min	Stop
	damage_per_collision	Spin
	damage_tolerance	Slow
	rear_effect	StopReplace
	front_effect	AssignDamage
	turn	IsTopSpeedMin
	laps	IsHandlingMin
	new_lap	UpdatePosition
	collision	GetDamage
		GetPos
		GetSpeed
		GetNext_pos
		GetTop_speed
		GetHandling
		GetDamage_tolerance
		GetTurn
		GetNew_lap
		GetCollision
		SetDirection
		SetTurn
		IncLaps
		SetNew_lap
		SetSpeed
		SetCollision
		Display

Class	Attributes	Methods
Car		Display

Class	Attributes	Methods

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Motorcycle		Display

Class	Attritbutes	Methods
Airplane		Display

Class	Attritbutes	Methods
Bicycle		Display

Class	Attritbutes	Methods
Boat		Display

Class	Attritbutes	Methods
Horse		Display

Class	Attritbutes	Methods
Dog		Display

Class	Attritbutes	Methods
Section		ReduceHandling
		ReduceTopSpeed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Class	Attributes	Methods
Weather	total_probability	CalculateWeatherConditionProbability
	probability	DetermineWeatherCondition
	weight	SumWeights
		Display

Class	Attributes	Methods
Rain		Display

Class	Attributes	Methods
Snow		Display

Class	Attributes	Methods
Hail		Display

Class	Attributes	Methods
Clouds		Display

Class	Attributes	Methods
Sunny		Display

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Class	Attributes	Methods
Keyboard	current_control	ConfigureKeys
	dfault	
	left_key	
	right_key	
	accelerate_key	
	brake_key	

Class	Attributes	Methods
Left	assigned	GetAssigned
		SetAssigned

Class	Attributes	Methods
Right	assigned	GetAssigned
		SetAssigned

Class	Attributes	Methods
Accelerate	assigned	GetAssigned
		SetAssigned

Class	Attributes	Methods
Brake	assigned	GetAssigned
		SetAssigned

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Class	Attributes	Methods
Score	points	GetPoints
	prev_clock	CalculatePoints

Class	Attributes	Methods
HighScoreList	entry	EnterInitials
		Display
		Load
		Store
		Sort
		Update
		GetScore

Class	Attributes	Methods
ListEntry	points	
	initials	

Class	Attributes	Methods
FinishLine	endpt1	Display
	endpt2	SetEndpt1
		SetEndpt2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Class	Attributes	Methods
Generator	GUI	
	Code	
	Library	

Class	Attributes	Methods
GUI	TitleForm	
	SettingsForm	
	DamageForm	

Class	Attributes	Methods
TitleForm	cmdSubmit	Form_Load
	cmdCancel	cmdSubmit_Click
	txtLaps	cmdCancel_Click
	txtTitle	txtLaps_Validate
	optEnable	txtTitle_Validate
	optDisable	

Class	Attributes	Methods
SettingsForm	cmdSubmit	Form_Load
	cmdCancel	cmdSubmit_Click
	lstVehicle	cmdCancel_Click
	lstTrack	chkKeyboard_Click
	chkKeyboard	chkMouse_Click
	chkMouse	optConfigurable_Click
	optFixed	txtSunny_Validate
	optConfigurable	txtRain_Validate
	txtLeft	txtSnow_Validate
	txtRight	txtHail_Validate
	txtAccelerate	txtClouds_Validate
	txtBrake	
	txtSunny	
	txtRain	
	txtSnow	
	txtHail	
	txtClouds	

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Class	Attributes	Methods
DamageForm	cmdGenerate	Form_Load
	cmdCancel	cmdGenerate_Click
	txtSpeed	cmdCancel_Click
	txtTolerance	optDisable_Click
	txtDamage	optEnable_Click
	optEnable	txtDamage_Validate
	optDisable	txtSpeed_Validate
	optStop	txtTolerance_Validate
	optSlow	txtVehicleTrack_Validate
	optSpin	txtVehicleSpeed_Validate
	optReplace	txtFront_Validate
	chkRearHandling	txtRear_Validate
	chkRearTopSpeed	txtSunny_Validate
	chkFrontHandling	txtRain_Validate
	chkFrontTopSpeed	txtSnow_Validate
	txtVehicleTrack	txtHail_Validate
	txtVehicleObstacle	txtClouds_Validate
	txtVehicleSpeed	
	txtFront	
	txtRear	
	txtSunny	
	txtRain	
	txtSnow	
	txtHail	
	txtClouds	
	lblVehicle	
	lblSpeed	
	lblTolerance	
	lblDamage	
	lblTrack	

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

METHODS

Game

void End()
//preconditions: none
//postconditions: Message is displayed on screen to play again.

bool IsRaceOver(Vehicle) const
//preconditions: none
//postconditions: Returns true if the race is over.

Track

DisplayLaps()
//preconditions: none
//postconditions: Number of laps for the vehicle is displayed.

bool IsCollision(Obstacle , Vehicle)
//preconditions: Next vehicle position has been calculated.
//postconditions: Returns true if collision, false otherwise.

bool IsCollision(Vehicle,Vehicle)
//preconditions: Next vehicle position has been calculated.
//postconditions: Returns true if collision, false otherwise.

bool CheckLap(Vehicle &)
//preconditions: none
//postconditions: Return true if the vehicle has just completed a lap.

Obstacle

void Display()
//preconditions: none
//postconditions: The obstacle is displayed.

float GetXpos() const
//preconditions: none
//postconditions: Returns the x position of the obstacle.

float GetYpos() const
//preconditions: none
//postconditions: Returns the y position of the obstacle.

void SetPos(float, float)
//preconditions: The x and y values must be between -140 and 140.
//postconditions: Sets the x and y values for the obstacle.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Coord

```
float GetX() const
//preconditions:  none
//postconditions:  Returns the X position.

float GetY() const
//preconditions:  none
//postconditions:  Returns the Y position.

void SetX(float)
//preconditions:  The x  value must be between -140 and 140.
//postconditions:  Sets the X position.

void SetY(float)
//preconditions:  The y  value must be between -140 and 140.
//postconditions:  Sets the Y position.
```

Vehicle

```
void Accelerate()
//preconditions:  none
//postconditions:  Speed is increased.

void Brake()
//preconditions:  none
//postconditions:  Speed is decreased.

void SteerLeft()
//preconditions:  none
//postconditions:  Direction is decreased.

void SteerRight()
//preconditions:  none
//postconditions:  Direction is increased.

Coord CalculateNextVehiclePosition()
//preconditions:  The vehicle is moving.
//postconditions:  Returns the next position for the vehicle to move.

void HitObstacle()
//preconditions:  Vehicle position is the same as the obstacle position.
//postconditions:  Damage is handled.

void AddDamage()
//preconditions:  Damage is enabled.
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

//postconditions: Damage is increased for the vehicle.

bool IsMaxDamage()
//preconditions: none
//postconditions: Returns true if damage equals or exceeds tolerance, false otherwise.

void Slide()
//preconditions: Collision effect is Slide.
//postconditions: Vehicle is moved along a straight line.

void Stop()
//preconditions: Collision effect is Stop.
//postconditions: Vehicle speed is zero.

void Spin()
//preconditions: Collision effect is Spin.
//postconditions: Vehicle is rotated.

void Slow()
//preconditions: Collision effect is Slow.
//postconditions: Vehicle speed is reduced.

void StopReplace()
//preconditions: Collision effect is StopReplace.
//postconditions: Vehicle is centered on track and speed is zero.

void AssignDamage(bool)
//preconditions: The value passed in is true if damage is to be assigned to the front.
//postconditions: Damage is assigned to the appropriate section.

bool IsTopSpeedMin()
//preconditions: none
//postconditions: Returns true if top_speed is at the minimum.

bool IsHandlingMin()
//preconditions: none
//postconditions: Returns true if handling is at the minimum.

void UpdatePosition()
//preconditions: next_pos has been calculated.
//postconditions: Sets pos equal to next_pos.

Car

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

void Display(float, float, float)
//preconditions: Color is passed in as red, green, and blue values.
//postconditions: Vehicle is displayed on the track.

Motorcycle

void Display(float, float, float)
//preconditions: Color is passed in as red, green, and blue values.
//postconditions: Vehicle is displayed on the track.

Airplane

void Display(float, float, float)
//preconditions: Color is passed in as red, green, and blue values.
//postconditions: Vehicle is displayed on the track.

Bicycle

void Display(float, float, float)
//preconditions: Color is passed in as red, green, and blue values.
//postconditions: Vehicle is displayed on the track.

Boat

void Display(float, float, float)
//preconditions: Color is passed in as red, green, and blue values.
//postconditions: Vehicle is displayed on the track.

Horse

void Display(float, float, float)
//preconditions: Color is passed in as red, green, and blue values.
//postconditions: Vehicle is displayed on the track.

Dog

void Display(float, float, float)
//preconditions: Color is passed in as red, green, and blue values.
//postconditions: Vehicle is displayed on the track.

Section

void ReduceHandling(float &)
//preconditions: The value must be between 0 and 1.
//postconditions: handling is reduced.

void ReduceTopSpeed(int &)
//preconditions: The value must be between 1 and 999.
//postconditions: top_speed is reduced.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

void SetFactor(fint)
//preconditions: The value must be between 0 and 9.
//postconditions: the factor is set.

Weather

int DetermineWeatherCondition(std::vector<float>)
//preconditions: Probabilities have been calculated to values between 0 and 1. Values are pushed into the vector.
//postconditions: A weather condition is determined.

Rain

void Display()
//preconditions: This weather condition is active.
//postconditions: Weather condition is displayed.

Snow

void Display()
//preconditions: This weather condition is active.
//postconditions: Weather condition is displayed.

Hail

void Display()
//preconditions: This weather condition is active.
//postconditions: Weather condition is displayed.

Clouds

void Display()
//preconditions: This weather condition is active.
//postconditions: Weather condition is displayed.

Sunny

void Display()
//preconditions: This weather condition is active.
//postconditions: Weather condition is displayed.

Keyboard

void ConfigureKeys()
//preconditions: Keyboard is available and configurable.
//postconditions: Keys are assigned to Left, Right, Accelerate, and Brake.

Left

char GetAssigned() const
//preconditions: Keyboard is current control. Value must be a valid keyboard key.
//postconditions: Returns the key assigned to Left.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

void SetAssigned(char)
//preconditions: Keyboard is current control.
//postconditions: Assigns a key to Left.

Right

char GetAssigned() const
//preconditions: Keyboard is current control. Value must be a valid keyboard key.
//postconditions: Returns the key assigned to Right.

void SetAssigned(char)
//preconditions: Keyboard is current control.
//postconditions: Assigns a key to Right.

Accelerate

char GetAssigned() const
//preconditions: Keyboard is current control. Value must be a valid keyboard key.
//postconditions: Returns the key assigned to Accelerate.

void SetAssigned(char)
//preconditions: Keyboard is current control.
//postconditions: Assigns a key to Accelerate.

Brake

char GetAssigned() const
//preconditions: Keyboard is current control. Value must be a valid keyboard key.
//postconditions: Returns the key assigned to Brake.

void SetAssigned(char)
//preconditions: Keyboard is current control.
//postconditions: Assigns a key to Brake.

Score

void CalculatePoints(int)
//preconditions: Race is ended. Time is positive.
//postconditions: Points are updated.

int GetPoints() const
//preconditions: none
//postconditions: Points are returned.

HighScoreList

void Load()
//preconditions: High score file exists.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

//postconditions: High scores are loaded.

void AddListEntry(int)
//preconditions: The score is passed in as positive number.
//postconditions: A list entry is added to the list.

void Display() const
//preconditions: High Score List is enabled.
//postconditions: none

void Store() const
//preconditions: High Score List is enabled and contents of high score list file has been input
//postconditions: High scores and initials are stored in HighScores.txt.

void Sort()
//preconditions: High Score List is enabled.
//postconditions: High scores are sorted in descending order.

int GetLowScore() const
//preconditions: none
//postconditions: Returns the low score on the list.

ListEntry

void SetPoints(int)
//preconditions: Points must be positive.
//postconditions: Points are set.

void SetInitials(string)
//preconditions: Initials must not be more than three characters.
//postconditions: Initials are set.

FinishLine

void Display()
//preconditions: none
//postconditions: Finish Line is displayed on the track.

void SetEndpt1(Coord)
//preconditions: x and y coordinates are between -140 and 140.
//postconditions: X and Y positions are set for the endpoint.

void SetEndpt2(Coord)
//preconditions: x and y coordinates are between -140 and 140.
//postconditions: X and Y positions are set for the endpoint.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B

PREGENERATED HEADER FILES

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Accelerate.h

#ifndef ACCELERATE_H
#define ACCELERATE_H

class Accelerate
{
      public:
            Accelerate();
            ~Accelerate();

            char GetAssigned() const;
            //preconditions: none
            //postconditions: Returns the key assigned to Accelerate.

            // set assigned

      private:
            char assigned;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Airplane.h

#ifndef AIRPLANE_H
#define AIRPLANE_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Airplane:public Vehicle
{
	public:
		Airplane();
		Airplane(int, int, int, int, int, int, int, int, int);
		~Airplane();

		void Display(float, float, float);
		//preconditions: Colors passed in are valid red, green and
blue
		//		values.
		//postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Bicycle.h

#ifndef Bicycle_H
#define Bicycle_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Bicycle:public Vehicle
{
      public:
            Bicycle();
            Bicycle(int, int, int, int, int, int, int, int, int);
            ~Bicycle();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //                values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Boat.h

#ifndef Boat_H
#define Boat_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Boat:public Vehicle
{
      public:
            Boat();
            Boat(int, int, int, int, int, int, int, int, int);
            ~Boat();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //                values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Brake.h

#ifndef BRAKE_H
#define BRAKE_H

class Brake
{
      public:
            Brake();
            ~Brake();

            char GetAssigned() const;
            //preconditions: none
            //postconditions: Returns the key assigned to Brake.

            // set assigned

      private:
            char assigned;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Car.h

#ifndef CAR_H
#define CAR_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Car:public Vehicle
{
      public:
            Car();
            Car(int, int, int, int, int, int, int, int, int);
            ~Car();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //                values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Clouds.h

#ifndef CLOUDS_H
#define CLOUDS_H

#include "Weather.h"

class Clouds : public Weather
{
      public:
            Clouds(float);
            ~Clouds();

            void Display() const;
            //preconditions: none
            //postconditions: Displays the weather condition.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
 // File: Coord.h

#ifndef COORD_H
#define COORD_H

class Coord
{
	public:
		Coord();
		Coord(float, float);
		~Coord();

		float GetX() const;
		//preconditions: none
		//postconditions: Returns the x position.

		float GetY() const;
		//preconditions: none
		//postconditions: Returns the x position.


		void SetX(float);
		//preconditions: Value must be between -140 and 140.
		//postconditions: Sets the x position.

		void SetY(float);
		//preconditions: Value must be between -140 and 140.
		//postconditions: Sets the x position.


	private:
		float x;
		float y;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Dirt.h

#ifndef DIRT_H
#define DIRT_H

#include "Track.h"

#include<vector>

class Dirt : public Track
{
	public:
		Dirt();
		~Dirt();

		std::vector<Obstacle> obst;

	private:
		Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Dog.h

#ifndef Dog_H
#define Dog_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Dog:public Vehicle
{
      public:
            Dog();
            Dog(int, int, int, int, int, int, int, int, int);
            ~Dog();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //                values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Finish_line.h

#ifndef FINISH_LINE_H
#define FINISH_LINE_H

#include "Coord.h"

class Finish_line
{
      public:
            Finish_line();
            ~Finish_line();

            void SetEndpt1(Coord);
            //preconditions: Vaild x and y coordinates are passed in.
            //postconditions: x and y coordinates set.

            void SetEndpt2(Coord);
            //preconditions: Valid x and y coordinates are passed in.
            //postconditions: x and y coordinates set.

            void Display();
            //preconditions: none
            //postconditions: Finish line is displayed on the track.

      private:
            Coord endpt1;
            Coord endpt2;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Game.h

#ifndef GAME_H
#define GAME_H

// track header file
#include "Score.h"
#include "Weather.h"
// header files

class Game
{
      public:
            Game();
            Game(char*, int, int, float, float, float, float,
                  float, float, float, int, int, float);
            ~Game();

            char* GetTitle() const;
            //preconditions: none
            //postconditions: Returns title.

            int GetLaps_per_race() const;
            //preconditions: none
            //postconditions: Returns laps per race.

            int GetVehicle_in_race() const;
            //preconditions: none
            //postconditions: Returns number of vehicles in race.

            float GetVehicle_speed_interaction() const;
            //preconditions: none
            //postconditions: Returns factor for interaction between
vehicle
                                  and speed.

            float CalculateInteraction();
            //preconditions: factors vector contains weather vs. track
            //                interaction factors
            //postconditions: Calculates and returns interaction.

            float GetInteraction() const;
            //preconditions: none
            //postconditions: Returns interaction.

            bool IsRaceOver(Vehicle) const;
            //preconditions: none
            //postconditions: Returns true if the race is over.

            void End();
            //preconditions: none
            //postconditions: Message is displayed on screen to close
window.

            // track declaration
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
		// high score declaration
		Score scr;
		// keyboard declaration

	private:
		char* title;
		int laps_per_race;
		int vehicles_in_race;
		float track_sunny_interaction;
		float track_rain_interaction;
		float track_snow_interaction;
		float track_hail_interaction;
		float track_clouds_interaction;
		float vehicle_obstacle_interaction;
		float vehicle_track_interaction;
		int front_obstacle_interaction;
		int rear_obstacle_interaction;
		float vehicle_speed_interaction;
		int wthr_sel;
		float interaction;
		std::vector<float> factors;

};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Hail.h

#ifndef HAIL_H
#define HAIL_H

#include "Weather.h"

class Hail : public Weather
{
      public:
            Hail(float);
            ~Hail();

            void Display() const;
            //preconditions: none
            //postconditions: Displays the weather condition.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: HighHighScoreListList.h

#ifndef HighScoreList_H
#define HighScoreList_H

#include<vector>
#include<string>
#include "ListEntry.h"

using namespace std;

class HighScoreList
{
      public:
            HighScoreList();
            ~HighScoreList();

            void AddListEntry(int);
            //preconditions: Value must be a positive number.
            //postconditions: A list entry is added to the list.

            void Display();
            //preconditions: none
            //postconditions: High score list is displayed.

            void Load();
            //preconditions: High score list file exists.
            //postconditions: High scores are loaded.

            void Store();
            //preconditions: none
            //postconditions: High scores are saved.

            void Sort();
            //preconditions: none
            //postconditions: High scores are sorted.

            int GetLowScore() const;
            //preconditions: none
            //postconditions: Low score on the list is returned.

      private:
            std::vector<ListEntry> scores;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Horse.h

#ifndef Horse_H
#define Horse_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Horse:public Vehicle
{
      public:
            Horse();
            Horse(int, int, int, int, int, int, int, int, int);
            ~Horse();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //                values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Keyboard.h

#ifndef KEYBOARD_H
#define KEYBOARD_H

#include"Left.h"
#include"Right.h"
#include"Accelerate.h"
#include"Brake.h"

class Keyboard
{
      public:
            Keyboard();
            ~Keyboard();

            // configure keys
            // control methods

            Left left_key;
            Right right_key;
            Accelerate accelerate_key;
            Brake brake_key;

      // private
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Left.h

#ifndef LEFT_H
#define LEFT_H

class Left
{
	public:
		Left();
		~Left();

		char GetAssigned() const;
		//preconditions: none
		//postconditions: Returns the key assigned to Left.

		// set assigned

	private:
		char assigned;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: ListEntry.h

#ifndef LISTENTRY_H
#define LISTENTRY_H

#include<string>

using namespace std;

class ListEntry
{
	public:
		ListEntry();
		~ListEntry();

		int GetPoints() const;
		//preconditions: none
		//postconditions: Return points.

		string GetInitials() const;
		//preconditions: none
		//postconditions: Return initials.

		void SetPoints(int);
		//preconditions: Value must be positive.
		//postconditions: Points are set.

		void SetInitials(string);
		//preconditions: No more than three characters may be
passed in.
		//postconditions: Initials are set.


	private:
		int points;
		string initials;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Motorcycle.h

#ifndef Motorcycle_H
#define Motorcycle_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Motorcycle:public Vehicle
{
      public:
            Motorcycle();
            Motorcycle(int, int, int, int, int, int, int, int, int);
            ~Motorcycle();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //               values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Obstacle.h

#ifndef OBSTACLE_H
#define OBSTACLE_H

class Obstacle
{
      public:
            Obstacle();
            Obstacle(float, float);
            ~Obstacle();

            float GetXpos() const;
            //preconditions: none
            //postconditions: Returns the x position of the obstacle.

            float GetYpos() const;
            //preconditions: none
            //postconditions: Returns the y position of the obstacle.

            void SetPos(float, float);
            //preconditions: Values must be between -140 and 140.
            //postconditions: Sets the x and y values for the obstacle.

            void Display();
            //preconditions: none
            //postconditions: The obstacle is displayed.


      private:
            float xpos;
            float ypos;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Rain.h

#ifndef RAIN_H
#define RAIN_H

#include "Weather.h"

class Rain : public Weather
{
      public:
            Rain(float);
            ~Rain();

            void Display() const;
            //preconditions: none
            //postconditions: Displays the weather condition.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Right.h

#ifndef RIGHT_H
#define RIGHT_H

class Right
{
	public:
		Right();
		~Right();

		char GetAssigned() const;
		//preconditions: none
		//postconditions: Returns the key assigned to Right.

		// set assigned

	private:
		char assigned;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Score.h

#ifndef SCORE_H
#define SCORE_H

class Score
{
      public:
            Score();
            ~Score();

            int GetPoints() const;
            //preconditions: none
            //postconditions: Points are returned.

            void CalculatePoints(int);
            //preconditions: Race is ended. Time is positive.
            //postconditions: Points are updated.

      private:
            int points;
            int prev_clock;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Section.h

#ifndef SECTION_H
#define SECTION_H

class Section
{
      public:
            Section();
            ~Section();

            // handling
            // top speed
            // set factor

      // private
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Sky.h

#ifndef SKY_H
#define SKY_H

#include "Track.h"

#include<vector>

class Sky : public Track
{
	public:
		Sky();
		~Sky();

		std::vector<Obstacle> obst;

	private:
		Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: SNOW.h

#ifndef SNOW_H
#define SNOW_H

#include "Weather.h"

class Snow : public Weather
{
      public:
             Snow(float);
             ~Snow();

             void Display() const;
             //preconditions: none
             //postconditions: Displays the weather condition.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Street.h

#ifndef STREET_H
#define STREET_H

#include "Track.h"

#include<vector>

class Street : public Track
{
      public:
            Street();
            ~Street();

            std::vector<Obstacle> obst;

      private:
            Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Sunny.h

#ifndef SUNNY_H
#define SUNNY_H

#include "Weather.h"

class Sunny : public Weather
{
      public:
            Sunny(float);
            ~Sunny();

            void Display() const;
            //preconditions: none
            //postconditions: Displays the weather condition.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Track.h

#ifndef TRACK_H
#define TRACK_H

#include"Obstacle.h"
#include"Finish_line.h"
// vehicle header file

#include<vector>

class Track
{
	public:
		Track();
		~Track();

		bool IsCollision(Obstacle, Vehicle);
		//preconditions: Next vehicle position has been calculated.
		//postconditions: Returns true if collision, false
otherwise.

		bool IsCollision(Vehicle, Vehicle);
		//preconditions: Next vehicle position has been calculated.
		//postconditions: Returns true if collision, false
otherwise.

		bool CheckLap(Vehicle&);
		//preconditions: none
		//postconditions: Returns true if the vehicle has just
completed
					a lap.

		void DisplayLaps();
		//preconditions: none
		//postconditions: Number of laps for the vehicle is
displayed.

		int GetTurn0() const;
		//preconditions: none
		//postconditions: Returns the position where turn is made.

		int GetTurn1() const;
		//preconditions: none
		//postconditions: Returns the position where turn is made.

		int GetTurn2() const;
		//preconditions: none
		//postconditions: Returns the position where turn is made.

		int GetTurn3() const;
		//preconditions: none
		//postconditions: Returns the position where turn is made.

		int GetObstNum() const;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
		//preconditions: none
		//postconditions: Returns the number of obstacles on the
track.

		// vehicle declaration

	protected:
		float handling;
		int turn0;
		int turn1;
		int turn2;
		int turn3;
		int obst_num;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Vehicle.h

#ifndef VEHICLE_H
#define VEHICLE_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"

#include "glut.h"
#include "math.h"

class Vehicle
{
	public:
		Vehicle();
		Vehicle(int, int, int, int, int, int, int, int, int);
		~Vehicle();

		int GetDamage() const;
		//preconditions: none
		//postconditions: Returns the damage of the vehicle.

		float GetHandling() const;
		//preconditions: none
		//postconditions: Returns the handling of the vehicle.

		int GetTop_speed() const;
		//preconditions: none
		//postconditions: Returns the top_speed of the vehicle.

		int GetDamage_tolerance() const;
		//preconditions: none
		//postconditions: Returns the damage_tolerance of the
vehicle.

		Coord GetPos() const;
		//preconditions: none
		//postconditions: Returns the position of the vehicle.

		int GetSpeed() const;
		//preconditions: none
		//postconditions: Returns the speed of the vehicle.

		Coord GetNext_pos() const;
		//preconditions: none
		//postconditions: Returns the next position of the vehicle.

		int GetTurn() const;
		//preconditions: none
		//postconditions: Returns the turn number of the vehicle.

		int GetLaps() const;
		//preconditions: none
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        //postconditions: Returns the number of laps the vehicle
has
        //                completed.

        bool GetNew_lap() const;
        //preconditions: none
        //postconditions: Returns true if the vehicle has just
crossed
        //                the finish line.

        bool GetCollision() const;
        //preconditions: none
        //postconditions: Returns true if the vehicle is colliding.

        float GetRadius() const;
        //preconditions: none
        //postconditions: Returns the radius of the vehicle.

        void SetHandling(float);
        //preconditions: none
        //postconditions: Sets the handling of the vehicle.

        void SetDirection(float);
        //preconditions: none
        //postconditions: Sets the direction of the vehicle.

        void SetTurn(int);
        //preconditions: none
        //postconditions: Sets the turn of the vehicle.

        void IncLaps();
        //preconditions: none
        //postconditions: Increments the number of laps the vehicle
has
        //                completed.

        void SetNew_lap(bool);
        //preconditions: none
        //postconditions: Sets the value of new_lap.

        void SetSpeed(int);
        //preconditions: none
        //postconditions: Sets the speed of the vehicle.

        void SetCollision(bool);
        //preconditions: none
        //postconditions: Sets the value of collision.

        void Accelerate();
        //preconditions: none
        //postconditions: Speed is increased.

        void Brake();
        //preconditions: none
        //postconditions: Speed is decreased.
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
            void SteerLeft(float);
            //preconditions: none
            //postconditions: Direction is decreased.

            void SteerRight(float);
            //preconditions: none
            //postconditions: Direction is decreased.

            float UpdateSpeedFactor(float);
            //preconditions: none
            //postconditions: Updates the speed factor.

            void CalculateNextPosition();
            //preconditions: The vehicle is moving.
            //postconditions: Returns the next position for the vehicle
to
            //                move.

            void HitObstacle();
            //preconditions: The Vehicle position is the same as the
obstacle
            //               position.
            //postconditions: Damage is handled.

            // max damage

            void AssignDamage(bool);
            //preconditions: The value passed in is true if damage is
to be
            //               assigned to the front
            //postconditions: Damage is increased for the vehicle,
sections
            //                are affected.

            void UpdatePosition();
            //preconditions: next_pos has been calculated.
            //postconditions: Sets pos equal to next_pos.

            Section front;
            Section rear;

      protected:
            Coord pos;
            float radius;

      private:
            int race_time;
            int damage;
            int speed;
            double direction;
            Coord next_pos;
            int top_speed;
            int top_speed_min;
            float handling;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
                float handling_min;
                int damage_per_collision;
                int damage_tolerance;
                int rear_effect;
                int front_effect;
                int turn;
                int laps;
                bool new_lap;
                bool collision;

                void Stop();
                //preconditions: none
                //postconditions: Vehicle speed is set to zero.

                void Spin();
                //preconditions: none
                //postconditions: Spinning is displayed.

                void Slow();
                //preconditions: none
                //postconditions: Slow is displayed.

                void StopReplace();
                //preconditions: none
                //postconditions: Vehicle is replaced on the track and the
speed
                //                 is set to zero.

                // add damage

                bool IsTopSpeedMin();
                //preconditions: none
                //postconditions: Returns true if top_speed is at the
minimum.

                bool IsHandlingMin();
                //preconditions: none
                //postconditions: Returns true if handling is at the
minimum.

};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Velodrome.h

#ifndef VELODROME_H
#define VELODROME_H

#include "Track.h"

#include<vector>

class Velodrome : public Track
{
	public:
		Velodrome();
		~Velodrome();

		std::vector<Obstacle> obst;

	private:
		Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Water.h

#ifndef WATER_H
#define WATER_H

#include "Track.h"

#include<vector>

class Water : public Track
{
      public:
            Water();
            ~Water();

            std::vector<Obstacle> obst;

      private:
            Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Weather.h

#ifndef WEATHER_H
#define WEATHER_H

#include <vector>

class Weather
{
	public:
		Weather();
		~Weather();

		int DetermineWeatherCondition(std::vector<float>);
		//preconditions: probabilities have been calculated to
values
		//		between 0 and 1 and are pushed into the
vector.
		//postconditions: Returns the index of the selected
condition.

		float GetProbability() const;
		//preconditions: none
		//postconditions: Returns the probability.


	protected:
		float probability;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX C

POSTGENERATED HEADER FILES

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Accelerate.h

#ifndef ACCELERATE_H
#define ACCELERATE_H

class Accelerate
{
      public:
            Accelerate();
            ~Accelerate();

            char GetAssigned() const;
            //preconditions: none
            //postconditions: Returns the key assigned to Accelerate.

            void SetAssigned(char);
            //preconditions: A valid keyboad key must be passed in.
            //postconditions: Assigns a key to Accelerate.

      private:
            char assigned;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Airplane.h

#ifndef AIRPLANE_H
#define AIRPLANE_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Airplane:public Vehicle
{
      public:
            Airplane();
            Airplane(int, int, int, int, int, int, int, int, int);
            ~Airplane();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //                values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Bicycle.h

#ifndef Bicycle_H
#define Bicycle_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Bicycle:public Vehicle
{
      public:
            Bicycle();
            Bicycle(int, int, int, int, int, int, int, int, int);
            ~Bicycle();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //              values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Boat.h

#ifndef Boat_H
#define Boat_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Boat:public Vehicle
{
      public:
            Boat();
            Boat(int, int, int, int, int, int, int, int, int);
            ~Boat();

            void Display(float, float, float);
};

#endif
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Brake.h

#ifndef BRAKE_H
#define BRAKE_H

class Brake
{
      public:
            Brake();
            ~Brake();

            char GetAssigned() const;
            //preconditions: none
            //postconditions: Returns the key assigned to Brake.

            void SetAssigned(char);
            //preconditions: A valid keyboad key must be passed in.
            //postconditions: Assigns a key to Brake.

      private:
            char assigned;
};

#endif
```

```
// File: Car.h

#ifndef CAR_H
#define CAR_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Car:public Vehicle
{
      public:
            Car();
            Car(int, int, int, int, int, int, int, int, int);
            ~Car();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //              values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Coord.h

#ifndef COORD_H
#define COORD_H

class Coord
{
      public:
            Coord();
            Coord(float, float);
            ~Coord();

            float GetX() const;
            //preconditions: none
            //postconditions: Returns the x position.

            float GetY() const;
            //preconditions: none
            //postconditions: Returns the x position.


            void SetX(float);
            //preconditions: Value must be between -140 and 140.
            //postconditions: Sets the x position.

            void SetY(float);
            //preconditions: Value must be between -140 and 140.
            //postconditions: Sets the x position.


      private:
            float x;
            float y;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Dirt.h

#ifndef DIRT_H
#define DIRT_H

#include "Track.h"

#include<vector>

class Dirt : public Track
{
      public:
            Dirt();
            ~Dirt();

            std::vector<Obstacle> obst;

      private:
            Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Dog.h

#ifndef Dog_H
#define Dog_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Dog:public Vehicle
{
      public:
            Dog();
            Dog(int, int, int, int, int, int, int, int, int);
            ~Dog();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Finish_line.h

#ifndef FINISH_LINE_H
#define FINISH_LINE_H

#include "Coord.h"

class Finish_line
{
      public:
            Finish_line();
            ~Finish_line();

            void SetEndpt1(Coord);
            //preconditions: Vaild x and y coordinates are passed in.
            //postconditions: x and y coordinates set.

            void SetEndpt2(Coord);
            //preconditions: Valid x and y coordinates are passed in.
            //postconditions: x and y coordinates set.

            void Display();
            //preconditions: none
            //postconditions: Finish line is displayed on the track.

      private:
            Coord endpt1;
            Coord endpt2;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Game.h

#ifndef GAME_H
#define GAME_H

#include "Street.h"
#include "Score.h"
#include "Weather.h"
#include "Keyboard.h"
#include "Sunny.h"

class Game
{
      public:
            Game();
            Game(char*, int, int, float, float, float, float,
                  float, float, float, int, int, float);
            ~Game();

            char* GetTitle() const;
            //preconditions: none
            //postconditions: Returns title.

            int GetLaps_per_race() const;
            //preconditions: none
            //postconditions: Returns laps per race.

            int GetVehicle_in_race() const;
            //preconditions: none
            //postconditions: Returns number of vehicles in race.

            float GetVehicle_speed_interaction() const;
            //preconditions: none
            //postconditions: Returns factor for interaction between
vehicle
            //                  and speed.

            float CalculateInteraction();
            //preconditions: factors vector contains weather vs. track
            //                interaction factors
            //postconditions: Calculates and returns interaction.

            float GetInteraction() const;
            //preconditions: none
            //postconditions: Returns interaction.

            bool IsRaceOver(Vehicle) const;
            //preconditions: none
            //postconditions: Returns true if the race is over.

            void End();
            //preconditions: none
            //postconditions: Message is displayed on screen to close
window.
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        Street track;
        Score scr;
        Keyboard kb;

    private:
        char* title;
        int laps_per_race;
        int vehicles_in_race;
        float track_sunny_interaction;
        float track_rain_interaction;
        float track_snow_interaction;
        float track_hail_interaction;
        float track_clouds_interaction;
        float vehicle_obstacle_interaction;
        float vehicle_track_interaction;
        int front_obstacle_interaction;
        int rear_obstacle_interaction;
        float vehicle_speed_interaction;
        int wthr_sel;
        float interaction;
        std::vector<float> factors;

};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Grand_Prix.h

#ifndef GRAND_PRIX_H
#define GRAND_PRIX_H

#include "Track.h"

#include<vector>

class Grand_Prix : public Track
{
	public:
		Grand_Prix();
		~Grand_Prix();

		std::vector<Obstacle> obst;

	private:
		Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: HighHighScoreListList.h

#ifndef HighScoreList_H
#define HighScoreList_H

#include<vector>
#include<string>
#include "ListEntry.h"

using namespace std;

class HighScoreList
{
      public:
            HighScoreList();
            ~HighScoreList();

            void AddListEntry(int);
            //preconditions: Value must be a positive number.
            //postconditions: A list entry is added to the list.

            void Display();
            //preconditions: none
            //postconditions: High score list is displayed.

            void Load();
            //preconditions: High score list file exists.
            //postconditions: High scores are loaded.

            void Store();
            //preconditions: none
            //postconditions: High scores are saved.

            void Sort();
            //preconditions: none
            //postconditions: High scores are sorted.

            int GetLowScore() const;
            //preconditions: none
            //postconditions: Low score on the list is returned.

      private:
            std::vector<ListEntry> scores;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Horse.h

#ifndef Horse_H
#define Horse_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Horse:public Vehicle
{
      public:
            Horse();
            Horse(int, int, int, int, int, int, int, int, int);
            ~Horse();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //               values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Keyboard.h

#ifndef KEYBOARD_H
#define KEYBOARD_H

#include"Left.h"
#include"Right.h"
#include"Accelerate.h"
#include"Brake.h"

class Keyboard
{
      public:
            Keyboard();
            ~Keyboard();

            void ConfigureKeys();
            //preconditions: none
            //postconditions: Keys are assigned to Left, Right,
Accelerate,
            //                and Brake.

            bool IsCurrentControl() const;
            //preconditions: none
            //postconditions: Returns true is keyboard is the current
            //                control.

            void ToggleCurrentControl();
            //preconditions: none
            //postconditions: Changes controls from keyboard to mouse
or vice
            //                versa.

            Left left_key;
            Right right_key;
            Accelerate accelerate_key;
            Brake brake_key;

      private:
            bool current_control;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Left.h

#ifndef LEFT_H
#define LEFT_H

class Left
{
	public:
		Left();
		~Left();

		char GetAssigned() const;
		//preconditions: none
		//postconditions: Returns the key assigned to Left.

		void SetAssigned(char);
		//preconditions: A valid keyboad key must be passed in.
		//postconditions: Assigns a key to Accelerate.

	private:
		char assigned;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: ListEntry.h

#ifndef LISTENTRY_H
#define LISTENTRY_H

#include<string>

using namespace std;

class ListEntry
{
      public:
            ListEntry();
            ~ListEntry();

            int GetPoints() const;
            //preconditions: none
            //postconditions: Return points.

            string GetInitials() const;
            //preconditions: none
            //postconditions: Return initials.

            void SetPoints(int);
            //preconditions: Value must be positive.
            //postconditions: Points are set.

            void SetInitials(string);
            //preconditions: No more than three characters may be
passed in.
            //postconditions: Initials are set.


      private:
            int points;
            string initials;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Motorcycle.h

#ifndef Motorcycle_H
#define Motorcycle_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"
#include "Vehicle.h"

#include "glut.h"
#include "math.h"

class Motorcycle:public Vehicle
{
      public:
            Motorcycle();
            Motorcycle(int, int, int, int, int, int, int, int, int);
            ~Motorcycle();

            void Display(float, float, float);
            //preconditions: Colors passed in are valid red, green and
blue
            //                values.
            //postconditions: Vehicle is displayed on the track.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Obstacle.h

#ifndef OBSTACLE_H
#define OBSTACLE_H

class Obstacle
{
	public:
		Obstacle();
		Obstacle(float, float);
		~Obstacle();

		float GetXpos() const;
		//preconditions: none
		//postconditions: Returns the x position of the obstacle.

		float GetYpos() const;
		//preconditions: none
		//postconditions: Returns the y position of the obstacle.

		void SetPos(float, float);
		//preconditions: Values must be between -140 and 140.
		//postconditions: Sets the x and y values for the obstacle.

		void Display();
		//preconditions: none
		//postconditions: The obstacle is displayed.


	private:
		float xpos;
		float ypos;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Rain.h

#ifndef RAIN_H
#define RAIN_H

#include "Weather.h"

class Rain : public Weather
{
      public:
            Rain(float);
            ~Rain();

            void Display() const;
            //preconditions: none
            //postconditions: Displays the weather condition.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Right.h

#ifndef RIGHT_H
#define RIGHT_H

class Right
{
      public:
            Right();
            ~Right();

            char GetAssigned() const;
            //preconditions: none
            //postconditions: Returns the key assigned to Right.

            void SetAssigned(char);
            //preconditions: A valid keyboad key must be passed in.
            //postconditions: Assigns a key to Right.

      private:
            char assigned;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Score.h

#ifndef SCORE_H
#define SCORE_H

class Score
{
	public:
		Score();
		~Score();

		int GetPoints() const;
		//preconditions: none
		//postconditions: Points are returned.

		void CalculatePoints(int);
		//preconditions: Race is ended. Time is positive.
		//postconditions: Points are updated.

	private:
		int points;
		int prev_clock;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Section.h

#ifndef SECTION_H
#define SECTION_H

class Section
{
      public:
            Section();
            ~Section();

            void ReduceHandling(float &);
            //preconditions: Value must be between 0 and 1.
            //postconditions: Handling is reduced.

            void ReduceTopSpeed(int &);
            //preconditions: Value must be between 1 and 999.
            //postconditions: Top Speed is reduced.

            void SetFactor(int);
            //preconditions: Value must be between 0 and 9
            //postconditions: Factor is set.

      private:
            int factor;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Sky.h

#ifndef SKY_H
#define SKY_H

#include "Track.h"

#include<vector>

class Sky : public Track
{
	public:
		Sky();
		~Sky();

		std::vector<Obstacle> obst;

	private:
		Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: SNOW.h

#ifndef SNOW_H
#define SNOW_H

#include "Weather.h"

class Snow : public Weather
{
      public:
            Snow(float);
            ~Snow();

            void Display() const;
            //preconditions: none
            //postconditions: Displays the weather condition.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Street.h

#ifndef STREET_H
#define STREET_H

#include "Track.h"

#include<vector>

class Street : public Track
{
      public:
            Street();
            ~Street();

            std::vector<Obstacle> obst;

      private:
            Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Sunny.h

#ifndef SUNNY_H
#define SUNNY_H

#include "Weather.h"

class Sunny : public Weather
{
      public:
            Sunny(float);
            ~Sunny();

            void Display() const;
            //preconditions: none
            //postconditions: Displays the weather condition.
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Track.h

#ifndef TRACK_H
#define TRACK_H

#include"Obstacle.h"
#include"Finish_line.h"
#include "Bicycle.h"

#include<vector>

class Track
{
	public:
		Track();
		~Track();

		bool IsCollision(Obstacle, Vehicle);
		//preconditions: Next vehicle position has been calculated.
		//postconditions: Returns true if collision, false
otherwise.

		bool IsCollision(Vehicle, Vehicle);
		//preconditions: Next vehicle position has been calculated.
		//postconditions: Returns true if collision, false
otherwise.

		bool CheckLap(Vehicle&);
		//preconditions: none
		//postconditions: Returns true if the vehicle has just
completed
		//		a lap.

		void DisplayLaps();
		//preconditions: none
		//postconditions: Number of laps for the vehicle is
displayed.

		int GetTurn0() const;
		//preconditions: none
		//postconditions: Returns the position where turn is made.

		int GetTurn1() const;
		//preconditions: none
		//postconditions: Returns the position where turn is made.

		int GetTurn2() const;
		//preconditions: none
		//postconditions: Returns the position where turn is made.

		int GetTurn3() const;
		//preconditions: none
		//postconditions: Returns the position where turn is made.

		int GetObstNum() const;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
		//preconditions: none
		//postconditions: Returns the number of obstacles on the
track.

		std::vector<Bicycle> comp;

	protected:
		float handling;
		int turn0;
		int turn1;
		int turn2;
		int turn3;
		int obst_num;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Vehicle.h

#ifndef VEHICLE_H
#define VEHICLE_H

#include "Coord.h"
#include "Obstacle.h"
#include "Section.h"

#include "glut.h"
#include "math.h"

class Vehicle
{
      public:
            Vehicle();
            Vehicle(int, int, int, int, int, int, int, int, int);
            ~Vehicle();

            int GetDamage() const;
            //preconditions: none
            //postconditions: Returns the damage of the vehicle.

            float GetHandling() const;
            //preconditions: none
            //postconditions: Returns the handling of the vehicle.

            int GetTop_speed() const;
            //preconditions: none
            //postconditions: Returns the top_speed of the vehicle.

            int GetDamage_tolerance() const;
            //preconditions: none
            //postconditions: Returns the damage_tolerance of the
vehicle.

            Coord GetPos() const;
            //preconditions: none
            //postconditions: Returns the position of the vehicle.

            int GetSpeed() const;
            //preconditions: none
            //postconditions: Returns the speed of the vehicle.

            Coord GetNext_pos() const;
            //preconditions: none

            //postconditions: Returns the next position of the vehicle.

            int GetTurn() const;
            //preconditions: none
            //postconditions: Returns the turn number of the vehicle.

            int GetLaps() const;
            //preconditions: none
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        //postconditions: Returns the number of laps the vehicle
has
        //                completed.

        bool GetNew_lap() const;
        //preconditions: none
        //postconditions: Returns true if the vehicle has just
crossed
        //                the finish line.

        bool GetCollision() const;
        //preconditions: none
        //postconditions: Returns true if the vehicle is colliding.

        float GetRadius() const;
        //preconditions: none
        //postconditions: Returns the radius of the vehicle.

        void SetHandling(float);
        //preconditions: none
        //postconditions: Sets the handling of the vehicle.

        void SetDirection(float);
        //preconditions: none
        //postconditions: Sets the direction of the vehicle.

        void SetTurn(int);
        //preconditions: none
        //postconditions: Sets the turn of the vehicle.

        void IncLaps();
        //preconditions: none
        //postconditions: Increments the number of laps the vehicle
has
        //                completed.

        void SetNew_lap(bool);
        //preconditions: none
        //postconditions: Sets the value of new_lap.

        void SetSpeed(int);
        //preconditions: none
        //postconditions: Sets the speed of the vehicle.

        void SetCollision(bool);
        //preconditions: none
        //postconditions: Sets the value of collision.

        void Accelerate();
        //preconditions: none
        //postconditions: Speed is increased.

        void Brake();
        //preconditions: none
        //postconditions: Speed is decreased.
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
            void SteerLeft(float);
            //preconditions: none
            //postconditions: Direction is decreased.

            void SteerRight(float);
            //preconditions: none
            //postconditions: Direction is decreased.

            float UpdateSpeedFactor(float);
            //preconditions: none
            //postconditions: Updates the speed factor.

            void CalculateNextPosition();
            //preconditions: The vehicle is moving.
            //postconditions: Returns the next position for the vehicle
to
            //                 move.

            void HitObstacle();
            //preconditions: The Vehicle position is the same as the
obstacle
            //                position.
            //postconditions: Damage is handled.

            bool IsMaxDamage();
            //preconditions: none
            //postconditions: Returns true if damage equals or exceeds
            //                 tolerance, false otherwise.

            void AssignDamage(bool);
            //preconditions: The value passed in is true if damage is
to be
            //                assigned to the front
            //postconditions: Damage is increased for the vehicle,
sections
            //                 are affected.

            void UpdatePosition();
            //preconditions: next_pos has been calculated.
            //postconditions: Sets pos equal to next_pos.

            Section front;
            Section rear;

      protected:
            Coord pos;
            float radius;

      private:
            int race_time;
            int damage;
            int speed;
            double direction;
            Coord next_pos;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
		int top_speed;
		int top_speed_min;
		float handling;
		float handling_min;
		int damage_per_collision;
		int damage_tolerance;
		int rear_effect;
		int front_effect;
		int turn;
		int laps;
		bool new_lap;
		bool collision;

		void Stop();
		//preconditions: none
		//postconditions: Vehicle speed is set to zero.

		void Spin();
		//preconditions: none
		//postconditions: Spinning is displayed.

		void Slow();
		//preconditions: none
		//postconditions: Slow is displayed.

		void StopReplace();
		//preconditions: none
		//postconditions: Vehicle is replaced on the track and the
speed
		//                 is set to zero.

		void AddDamage();
		//preconditions: none
		//postconditions: Increases the damage of the vehicle.

		bool IsTopSpeedMin();
		//preconditions: none
		//postconditions: Returns true if top_speed is at the
minimum.

		bool IsHandlingMin();
		//preconditions: none
		//postconditions: Returns true if handling is at the
minimum.

};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Water.h

#ifndef WATER_H
#define WATER_H

#include "Track.h"

#include<vector>

class Water : public Track
{
	public:
		Water();
		~Water();

		std::vector<Obstacle> obst;

	private:
		Finish_line fl;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
// File: Weather.h

#ifndef WEATHER_H
#define WEATHER_H

#include <vector>

class Weather
{
      public:
            Weather();
            ~Weather();

            int DetermineWeatherCondition(std::vector<float>);
            //preconditions: probabilities have been calculated to
values
            //               between 0 and 1 and are pushed into the
      vector.
            //postconditions: Returns the index of the selected
condition.

            float GetProbability() const;
            //preconditions: none
            //postconditions: Returns the probability.



      protected:
            float probability;
};

#endif
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX D

TOOLS USED

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following tools were used in the development of the software:

Microsoft Visual Basic ® - GUI

Microsoft Visual C++ ® - generated code

OpenGL library ® - graphics in the generated code

Microsoft Visio ® - Use Case Diagrams and Class Diagrams

Microsoft Word ® - All text and tables

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX E

GLOSSARY

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Term	Definition
Developer	A person who is designing the RaceGen software.
User	A person who uses RaceGen to generate code for a racing game.
Player	A person who plays the game which the user generated with RaceGen.
GUI	Graphical User Interface. It is used for the user to interact with the computer and make selections that will be used by the computer.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

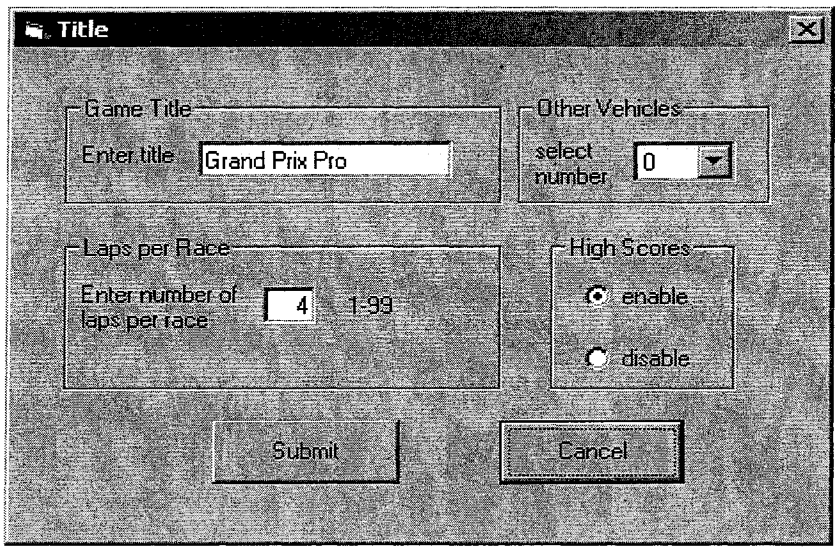
APPENDIX F

USER'S MANUAL

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The RaceGen racing game program generator provides a means for the user to generate code for a racing game by entering selections from a questionnaire. The code is generated based on these selections without any programming done by the user. The questionnaire forms provide the user with a variety of choices for game generation. The forms are easy for the user to complete. They allow for easy navigation and are user-friendly.

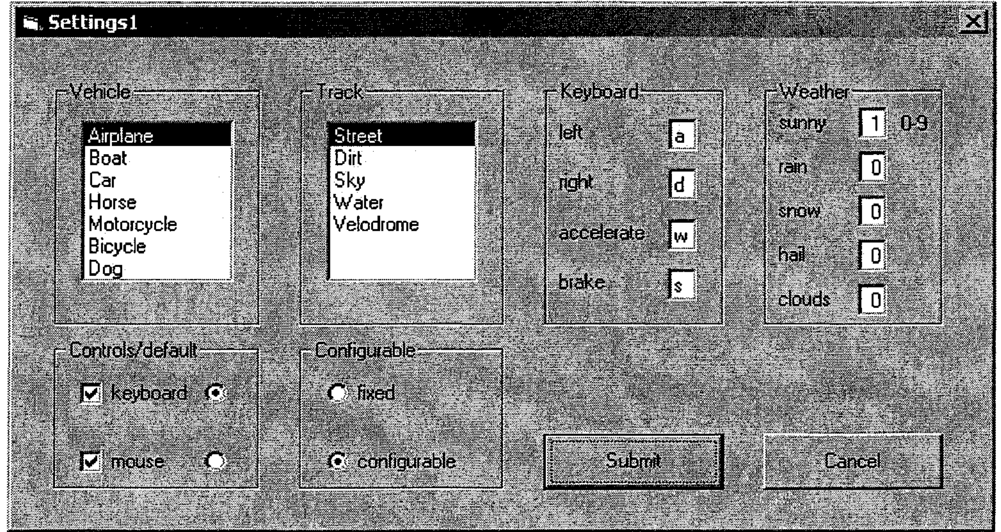
The user is able to enable or disable the high scores list, according to the following GUI. The player will be able to enter initials to be displayed with the score. The high scores will be stored in a file and be listed in order at the end of a race. The following is where the user enters the title of the game. This title will appear as the title of the window while the game is being played. The following GUI is used to assign the number of laps per race. The allowable range is 1-99. The following GUI allows the user to select how many vehicles will be in the race, in addition to the player's vehicle. The user may enter from 0-5 additional vehicles. Pressing the "Submit" button accepts the selections and continues to the next page. Pressing the "Cancel" button rejects the selections and exits RaceGen.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following Graphical User Interface (GUI) is used to select the vehicles, tracks, weather, and controls. Select the controls which are to be made available to the user by marking the checkboxes to the left of the control name. Use the radio buttons to the right to select the default control. If more than one control type is selected from the GUI, then the player will have the option of changing the controller.

For weather: the numbers entered are used to determine the probability that the weather condition will come up for a race. The weather conditions will be selected before each race, based on these probabilities. Only one weather condition will be applied to each race. The selected weather condition will be applied for the entire race. A higher number represents a more probable weather condition. If only one entry is non-zero, then that weather condition will be applied to each race. There must be at least one non-zero entry. Pressing the "Submit" button will accept the selections and continue to the next page. Pressing the "Cancel" button will reject the selections and return to the previous page.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following menu is used by the player to start a race, assign keys for keyboard control if it is available and configurable is selected, and change controls if both are available:

1) Start race

2) Configure keyboard

3) Change controls

These are the prompts for the user to configure the keyboard controls:

Enter Left:

Enter Right:

Enter Accelerate:

Enter Brake:

Once entries are made for each prompt, the keyboard keys will be assigned.

When the user opts to change controls, mouse controls will be used if keyboard was previously selected, and vice versa.

The following GUI is used for the user to set the top speed of the vehicles for the selected vehicle type. The allowable range is 1-999. It is used by the user to enable or disable vehicle damage. It is used for the user to select either handling or top speed to be affected if damage is assigned to the rear or front of the vehicle. The user uses it to select

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the affect of a collision and to assign the damage tolerance for each vehicle. The allowable range is 1-999.

When a vehicle is created, it will be initialized with zero damage. Each time it is involved in a collision, if damage is enabled, an amount of damage will be added to the vehicle's total damage, according to the following GUI. The amount entered is how much damage is done to the vehicle. The allowable range is 1-99.

The user uses the GUI table to set the effect of interaction between different settings of the parameters.

Vehicle / track, track / weather, vehicle / speed:

The numbers represent the effect on vehicle handling. A higher number will make steering more difficult. The vehicle will tend to slide to the outside of turns.

Vehicle / obstacles:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

If the spin or slow options are selected, then this is the amount of spin or slow.

The above settings will affect the performance and handling of the vehicles. A probability will be determined, based on the user defined settings, which will be used to update the vehicle positions. When the vehicle positions are updated, a check will be done to determine if each vehicle has encountered an obstacle. Once the damage to the car has been assigned, a check will be made to determine if the car has reached its maximum damage tolerance. When the vehicle positions are updated, a check will be done to determine if each vehicle has crossed the finish line. Pressing the "Generate" button will accept the selections and generate the code. Pressing the "Cancel" button will reject the selections and return to the previous page. Obstacles will be placed alongside the track at various locations so that a vehicle which stays on the track will not collide with one and a vehicle which strays off the course must try to avoid them. Whenever a vehicle is damaged and effects of damage on performance are selected, either the handling or top speed will be reduced, based on the location of the damage on the vehicle and the user's selection of what is affected by damage to that section. Each vehicle will have a front section and a rear section. When the vehicle is damaged, if top speed is assigned to that section by the user, the top speed will be reduced. If handling is assigned to the damaged section, then the vehicle will be more difficult to steer.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

REFERENCES

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

REFERENCES

[1] Farlex. "parameter-driven"
http://computing-dictionary.thefreedictionary.com/parameter-driven (2005).

[2] Myers, Bill. "Info Boss Automatic Program Generator"
http://www.winsite.com/bin/Info?500000015244 (1998).

[3] Sommerville, Ian 2001. *Software Engineering.* New York, Addison – Wesley.

[4] Voelter, Markus. "A Catalog of Patterns for Program Generation"
http://www.voelter.de/data/pub/ProgramGeneration.pdf (2003).

[5] Miller, Randy. "Borland Developer Network"
http://bdn.borland.com/article/0,1410,31863,00.html (2005).

[6] Williams, Bob. "National Sports Center Velodrome"
http://www.nscsports.com/velo/ (2005).

[7] Deitel and Deitel, H.M. and P.J. 2000. *C++ How to Program.* New Jersey, Prentice Hall.

[8] Longman, Addison Wesley. "Preconditions and Postconditions"
http://www.cs.ucf.edu/~reinhard/classes/cop3503/notes01.pdf (1997).

[9] Magestro, Dan. "C++ Programming Guidelines"
http://www.star.bnl.gov/STAR/comp/train/standards.html (2004).

[10] Hoff, Todd. "Standardization is Important"
http://www.possibility.com/Cpp/CppCodingStandard.html#important (2005).

[11] Scacchi, Walt 2001. "Process Models in Software Engineering." New York – John Wiley
and Sons, Inc.

[12] Tech Target. "spiral model"
http://searchvb.techtarget.com/sDefinition/0,,sid8_gci755347,00.html (2006).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.