OPEN WORLD CAR SIMULATOR

A PROJECT REPORT

Submitted by

Roll Number	Registration Number	Student Code	Student Name
20010301036	20013001171	BWU/BCA/20/045	BISHAL RAY
20010301148	20013001283	BWU/BCA/20/019	SUGNIK CHAKRABORTY
20010301147	20013001282	BWU/BCA/20/040	SUDIP CHANDRA DAS
20010301016	20013001151	BWU/BCA/20/017	ANKAN PRADHAN
20010301140	20013001275	BWU/BCA/20/013	SUBHAM DHAR

in partial fulfillment for the award of the degree

of

BACHELOR OF COMPUTER APPLICATION

Department of Computational Sciences

BRAINWARE UNIVERSITY
398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125
JULY 2023

OPEN WORLD CAR SIMULATOR

Submitted by

Roll Number	Registration Number	Student Code	Student Name
20010301036	20013001171	BWU/BCA/20/045	BISHAL RAY
20010301148	20013001283	BWU/BCA/20/019	SUGNIK CHAKRABORTY
20010301147	20013001282	BWU/BCA/20/040	SUDIP CHANDRA DAS
20010301016	20013001151	BWU/BCA/20/017	ANKAN PRADHAN
20010301140	20013001275	BWU/BCA/20/013	SUBHAM DHAR

in partial fulfillment for the award of the degree

of

BACHELOR OF COMPUTER APPLICATION

in

DEPARTMENT OF COMPUTATIONAL SCIENCES

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

DEPARTMENT OF COMPUTATIONAL SCIENCES

BONAFIDE CERTIFICATE

Certified that this project report on "**OPEN WORLD CAR SIMULATOR"** is the bonafide work of "**BISHAL RAY, SUGNIK CHAKRABORTY, SUDIP CHANDRA DAS, ANKAN PRADHAN, SUBHAM DHAR** " who carried out the project work under my supervision.

SIGNATURE
Dr. JAYANTA AICH
HEAD OF THE DEPARTMENT
Department of Computational Sciences
BRAINWARE UNIVERSITY
398, Ramkrishnapur Road, Near Jagadighata
Market, Barasat,Kolkata, West Bengal-700125

SIGNATURE
Mr. INDRANIL SARKAR
SUPERVISOR
Assistant Professor
Department of Computational Sciences

External Examiner

ABSTRACT

This project is a window based application especially for gaming sector. This application will be used by users to get full refreshment at the different physical time. This is a simulator application.

Simulation games are a genre of games that are designed to mimic activities you'd see in the real world. The purpose of the game may be to teach you something. For example, you could learn how to fish. Others simulation games take on operating a business such as a farm or a theme park.

TABLE OF CONTENT

CHAPTER 01
INTRODUCTION

1. INTRODUCTION

1.1 INTRODUCTION

Driving simulators are used to conduct experiments on for example driver behavior, road design, and vehicle characteristics. The results of the experiments often depend on the traffic conditions. One example is the evaluation of cellular phones and how they affect driving behavior. It is clear that the ability to use phones when driving depends on traffic intensity and composition, and that realistic experiment in driving simulators therefore has to include surrounding traffic. This paper describes a model that generates and simulates surrounding vehicles for a driving simulator. The proposed model generates a traffic stream, corresponding to a given target flow and simulates realistic interactions between vehicles. The model is built on established techniques for time-driven microscopic simulation of traffic and uses an approach of only simulating the closest neighborhood of the driving simulator vehicle. In our model this closest neighborhood is divided into one inner region and two outer regions. Vehicles in the inner region are simulated according to advanced sub-models for driving behavior while vehicles in the outer regions are updated according to a less time-consuming model. The presented work includes a new framework for generation and simulation of vehicles within a moving area. It also includes the development of an enhanced model for overtakings and a simple mesoscopic traffic model. The developed model has been tested within the VTI Driving simulator III. A driving simulator experiment has been performed in order to check if the participants observe the behavior of the simulated vehicles as realistic or not. The results were promising but they also indicated that enhancements could be made. The model has also been validated on the number of vehicles that catch up with the driving simulator vehicle and vice versa. The agreement is good for active and passive catch-ups on rural roads and for passive catch-ups on freeways, but less good for active catch-ups on freeways.

You just have to go into the information as asked in the program as well as you need to strike the search switch.

1.2 ORGANIZATION OF DOCUMENT

The reminder of this document is first providing the full description of the project. It lists all the functions performed by the system. And it also concerns the details of the system functions and actions of each function which was performed by the system.

Chapter 2: describes about the Literature survey.

Chapter 3: Describes about the Analysis of the Project

Chapter 4: Describes about the Design of the Project

Chapter 5: Describes about the Implementation of the Project

Chapter 6: Shows the Output Screens of the Project

Chapter 7: Describes about the Testing and Validation of the Project

Chapter 8: Gives the application's conclusion and future enhancement of the project

CHAPTER 02

LITERATURE SURVEY

2. LITERATURE SURVEY

This literature review is part of the "Driver performance in the OPEN WORLD CAR SIMULATOR : a validation study" project funded by the students for the group- . It focuses mainly on driving simulator validation studies with regard to driver behavior. Various approaches, methodologies and criteria have been proposed until today regarding the behavioral and physical validation of a driving simulator. At the same time, a number of behavioral validation studies have been conducted, with or without taking into account the proposed validation approaches. The author considered necessary this literature review because according to her knowledge there was no other published review which examined thoroughly the link between theory (proposed validation approaches and methodologies) and practice (validation studies cc driving simulators). Most of the recent behavioral validation studies have been focused on the absolute and relative validity of the simulator without taking into consideration the issue of the face validity. The fonnat of this paper will be as follows. A small introduction to driving simulators and their usefulness will be presented first followed by the necessity of validating them. The existing validation approaches, methodologies and criteria will be analyzed and earlier and recent behaviour validation studies will be reviewed and compared in detail. These studies will be classified according to the driver behaviour levels and driving performance (as they will be defined) and then will be assessed according to the validation criteria. Emphasis will be given to the interpretation of the findings from these comparisons, and in particular to their applicability in real road traffic situations.

2. THE NEED FOR DRIVING SIMULATORS

Driving simulators were fust developed for the training of a large number of personnel in the tactical use of war machinery during the Second World War. In the early 1960's, they were applied in the research field to study driver behaviour and hidher interaction with the vehicle and the road environment (Roberts, 1980). Due to rapid progress of the state of the art in visual displays and computer technology by 1975 at least sixteen driving simulators were operating throughout the United States using different techniques for the generation of the visual field and two in Europe (AUen, Klein and Ziedman, 1979). The latest trend to the development of driving simulators (after 1985) is to fulfil the specific needs of a particular group, whether this is an automotive industry, a private research institute or a university. The main application areas of today driving simulators have been to investigate acceptability issues of innovative transport elements (e.g. mad design, in-vehicle device), to evaluate the safety concept (e.g. possible increase of accidents due to new road design, in-vehicle device) as well as the credibility and transferability of the simulator results to the real world. Driving simulators have been used as research aids in a number of civil engineering, transport, psychology and ergonomics fields such as: innovative road design (e.g. testing the design of new tunnels, innovative highway design and road delineation, traffic calming); intelligent transport systems (e.g. new in-vehicle navigation systems, Head-Up-Displays, active pedals); impaired driver behaviour (driving behaviour affected by drugs, alcohol, severe brain damage, fatigue) and vehicle dynamics and layout (e.g. testing ABS, 4-wheel drive; vehicle interior design). The main advantage of driving simulators is that they can provide an inherently safe environment for driving research, which can be easily and economically configured to investigate a variety of human factors research problems. They make it possible to control experimental conditions over a wider range than field tests and can be easily changed from one condition to another. They are linked to digital computer systems which can further provide on-line data processing, formatting and storage and the reduction and compact arrangement of data. On the other hand, driving simulators provide drivers with an artificial environment which could never be the same as the real one. For example, even in the most advanced driving simulators the longitudinal and lateral accelerations are limited (e.g. VTI driving simulator: Nilsson, 1989; Daimler-Benz driving simulator: Drosdol and Panik, 1985) and only parts of the extremely complicated transport system can be simulated until today. The differences between the simulated and the real driving environment may influence subjects' driving behaviour and performance, hence any performance measures observed in a driving simulator may differ from the same measures observed during real driving. Therefore, the issue of evaluating the driving simulators emerges in order for them to produce transferable, reliable, and valid wults between the two environments.

3. EVALUATION OF DRIVING SIMULATORS

The evaluation of driving simulators could be separated into three stages: a) the transferability of the results obtained from a driving simulator to real world; b) the reliability of a driving simulator and c) the validity of a driving simulator. The first stage is crucial and rather necessary for the training simulators (either driving or flight simulators) and it has been extensively investigated for flight simulators. The reason why the second stage has not been given a lot of attention from the researchers and is mentioned very rarely is because a valid driving simulator is always reliable too, where the vice versa does not apply. The third and most important stage for any simulator, is the issue of validity and it is examined here thoroughly.

3.1. TRANSFERABILITY AND RELIABILITY OF RESULTS OF DRIVING SMULATORS

The issues of transferability and reliability of results obtained from a driving simulator will not be examined in detail since there are not the main topic of this paper, however definitions of these terms and examples relative to driving simulators will be given in the following paragraphs.

3.1.1. The issue of transferability

The issue of transferability of results from the simulated to the real environment has always been of critical importance for the training rather than the research simulators. Especially for the flight simulators, it was investigated since their first development and usage as pilot training devices in the aircraft and military industry. Caro (1973) in his investigation of different training techniques, he concluded that the potential training value of a flight simulator depends probably more on a proper training program than on the actual realism of the simulator. He also claimed that "transfer of training from a device to an aimaft is limited to the tasks which can be performed in the device. But, whether that limit is reached is a function of the way in which the device is used". Valverde (1973) in his review of flight simulator transfer of training studies concluded that in order to eliminate contradictory results from these studies, one must take into serious consideration the criterion measures; the individual differences between subjects as well as their motivation and attitudes towards the simulator; the attitude, ability and motivation of the instructor; and the instructional sequence. For a driving simulator in order to be valid it should allow at least the transfer of basic driving skills from a real driving environment to a simulated and at the same time it should present the subject with realistic visual and auditory cues and traffic scenarios. Since the objective of this paper is the behavioural validity of research driving simulators, transferability will not be examined further.

3.1.2. The issue of reliability

In terms of psychology reliability is the consistency with which a test measurn what it measures (Gleitman, 1991). In determining the reliability of a measuring instrument (e.g. a driving simulator), it is assumed that the instrument is measuring a relatively stable cmristic. Because a driving simulator is a very complicated system, an aggregation of a number of subsystems, it is clear that we cannot claim overall reliability of a simulator but reliability of each of its subsystems. For example, the vehicle dynamics model of a driving simulator measures the braking force via the braking system of the simulated vehicle. If the braking system of the simulated vehicle is half working then we will receive consistent results, the braking force, but not the correct ones. The braking system will be reliable but not valid. This means that high reliability alone does not guarantee that a test (here the braking system of the simulator) is a good measuring device. But, at the same time, a driving simulator is also a man-in-theloop device which means that one has to check both the physical as the behavioural reliability of the simulator. The author believes that although the physical reliability of the simulator and its subsystems can be measured and tested easily, the behavioural reliability is not only difficult to measure or test it, but even more to identify it. Unreliability can be a result of measurement errors produced by temporaty internal (e.g. for the behavioural reliability: low motivation, temporary indisposition of the subjects) or external (e.g. for the behavioural reliability: a distracting or uncomfortable testing environment) conditions (Aiken, 1971). More critical to a test's reliability is its validity, and since it is one of the objectives of this study, it will be thoroughly examined in the next paragraphs.

3.2. VALIDITY OF DRIVING SIMULATORS

Defining the validity of a driving simulator is a multi-disciplinary and complicated task. Mudd (1968) defined validity as the way in which the simulator "reproduces a behavioural environment" where according to AUen et al (1991) "Validity is only defined to a specific research question". Rolfe et al (1970) stated that "The value of a simulator depends on its ability to elicit from the operator the same sort of response that he would make in the real situation". According to Leonard and Wierwille (1975) "simulator validation is a problem of obtaining parallel measures in full-scale and in simulation and bringing these two sets of measures into correspondence". It is clear that the term "validity of a driving simulator" is not precisely defined and it needs further specification. On the other hand, validity from the standpoint of psychology is widely used for the assessment of psychological tests, it is clearly defined and there are already standards relative to the validity of a test (e.g. APA, 1985). However, there is a major problem in this instance. The driving simulators is a man-in-theloop system and human performance is confounded with system performance. This problem has been recognised from the early sixties. Ebel(1961) proved that very few psychological tests used in clinics and industry had satisfactory validity data when used in the simulators, which implied that the current concepts of validity may not be adequate. Still, the author judged as substantive to mention validity in terms of psychology and -%tempt to apply these terms to the driving simulator validity and confirm or not the above findings. The validity of a test is defined as the extend to which it measures what it is supposed to measure (Gleitman, 1991). Unlike reliability which can only be influenced by unsystematic errors of measurement, validity can be affected by both unsystematic and systematic (constant) errors, i.e. atest may be reliable without Wig valid, but it cannot be valid without being reliable. This means that reliability is a necessary but not a sufficient condition for validity. Primarily, all procedures for determining test validity are concerned with the relationships between performance on the test and other independently observable facts about the behaviour characteristics under consideration. (Anastasi, 1988) Validity has been classified into different categories. Tiffin and McCormick (1965) classified validity into four categories: a) concurrent, b) predictive, c) content and d) construct. The first two categories are tested using statistical or quantitative methods because they rely on numerical performance data for analysis whereas the last two by using logical or qualitative methods because they use judgement type data for analysis. Gleitman (1991) classified validity as previous but only using the terms of predictive and construct validity. The American Psychological Association (APA) (1985), in its proposal for the technical standards for test construction and evaluation, grouped validity evidence into three categories: a) the construct related, b) the content related and c) the criterion-related evidence of validity, which includes the predictive and concurrent validity. The same grouping is used by Anastasi (1988). According to APA (1989) standards, an ideal validation should include several types of evidence, comprising of all three traditional categories. The quality of the evidence is of primary importance and a single line of solid evidence is preferable to numerous lines of evidence of questionable quality. Gathering evidence may sometimes involve examining not only the present instrument in the present situation, but also the available evidence on the use of the same or similar instruments in similar situations. This working paper will refer to validity classification according to the APA and will try to associate this classification of validity with the validity of a driving simulator.

3.2.1. Criterion-related validity

According to McCoy (1963) the first and most important consideration in setting up an experimental investigation is the development of precise criteria. This is fundamental to selecting the parameters for measurement, and the measures to use, as well as providing the frame of reference in which validation will be attempted. The criterion-related validity indicate the effectiveness of a test in wcting an individual's performance in specified activities. For this purpose, performance on the test is checked against a criterion, that is, a direct and independent measure of that which the test is designed to predict (Anastasi, 1988). The choice of the criterion and the measurement procedures used to obtain criterion scores are of central importance. The value of a criterion-related study depends on the relevance of the criterion measure that is used. Whether a given degree of accuracy is judged to be high or low or useful or not useful depends on the context in which the decision is to be made (APA, 1985). When establishing the criterion-related validity of a driving simulator, we need to consider the existing driving simulator validity criteria and if they are not adequate, we have to develop new more precise criteria but more over to define precisely the experiment, which in a way plays the role of the test. But, before defining any criteria or test protocols, we should take into account the fact that there is both physical and behavioural validity of a simulator. The physical validity is absolute necessary condition for the behavioural validity. Behavioural validity can be checked if and only if the physical validity has already been tested and verified. Physical validity can not only enhance the behaviouml validity but also minimise the internal variables that may affect behavioural validity. Criterion-related validity is often used in local validation studies, in which the effectiveness of a test for a specific program is to be assessed. According to Anastasi (1988) the application of criterion-related validation is not technically feasible for small samples (40-50 cases). Aiken (1971) concluded that the correlation between the test and an external criterion measure can never be greater than the square mot of the parallel forms reliability coefficient of the test. Factors affecting criterion-related validity can be group differences, test length criterion contamination, base rate and incremental validity (Aiken, 1971). When applied to driving simulators, it can also be affW by group differences, i.e. if we only use male instead of both male and female subjects for the simulated experiments (there is evidence of different driving patterns between males and females on real mad); if we only use young instead of both young and old subjects (there is evidence of different driving patterns between young and old drivers on real road and lately in the simulators too), therefore it would be preferable if the sample size (n) is weighted for gender and age and could be greater than fifty (n=50) to avoid subjects' variability. It can also be affected by test length, i.e. the experiment in the simulator should not be so long in order to avoid fatigue, monotony, and simulator sickness.

The criterion-related validity can be distinguished to predictive and concurrent validity. A predictive study obtains information about the accuracy with which early test data can be used to estimate criterion scores that will be obtained in the future. A concurrent study serves the same purpose, but it obtains prediction and criterion information simultaneously. A decision theory framework can be used to judge the value or utility of a predictor test. One judgement can be that the most important error to avoid is a false positive - selecting someone who will subsequently fail. Another judgement focuses on avoiding a false negative -not selecting people who would have succeeded.
The relative cost assigned to each kind of error is again a value judgement; depending on that judgement, the subsequent interpretation of the utility of testing may differ (MA, 1985). The logical distinction between predictive and concurrent validation is based, not on time, but on the objectives of testing (Anastasi, 1988). Waive validity can be achieved by evaluating the effectiveness of the simulator in predicting certain driver performance in the future from present performance data is tested requiring a follow-up study (Leonard and WieMe, 1975). The simulator is used as the test or measure to predict future driver behaviour. This can be measured by the correlation coefficients. The appropriate criterion should be the genuine road used driving behaviour. Maybe it would be more practical to define a local or situation specific criterion related evidence of driving simulator validity. For example relative to driving speed, we should define: a) differences in measuring the speed on the road and in the simulator; b) the type of simulator used for the experiment; C) the type of subjects used, d) the time the real road and the simulated experiment took place. For driving simulators, concurrent validity can be achieved by comparing real road and simulated data (Leonard and Wierwille, 1975). Generalisation of a driving simulator validity is lid because validity is usually applicable for a particular task and a particular driving simulator.

<u>**3.2.1.1.Predictive validity**</u>

Predictive validity is evaluated by showing how will predictions made from a test or measure are confirmed by subsequent observation. Commonly, it is expressed in terms of a simple correlation between test scores or measures and some criterion scores or measures. An obvious problem here is the determination of precise criteria (McCoy, 1963). One index of a test's validity is the success with which it makes such prediction. This is usually measured by the correlation between the test score and some appropriate criterion. The correlation coefficient is a mathematical expression that sumrnarises both the direction and the strength of the relationship between the two measures. It varies between k1.00 and it is symbolised by r. The sign indicates the direction of the relationship where the strength of the correlation is expressed by its absolute value. When -1, i.e. when the correlation is perfect and prediction is emr-free, then there is no more variation at all around the line of best fit. But, the fact that two variables are correlated says nothing about the underlying causal relationship between them. Validity coefficients are just the correlations between test scores and criteria and their magnitude depends upon the range of ability within the group in which they are determined. As this range is narrowed, the correlation between test score and criterion declines. This %lationship holds for all correlation coefficients (Gleitman, 1991). The interpretation of a validity coefficient must take into account a number of concomitant circumstance, therefore the obtained correlation should be high enough to be statistically significant at some acceptable level, such as the 0.01 or 0.05 levels. Before drawing any conclusions about the validity of a test we should be reasonably certain that the obtained validity coefficient could not have arisen through chance fluctuations of sampling from a true correlation of zero. When a significant correlation between test scores and criterion has been established, we need to evaluate the size of the correlation on the light of the uses to be made of the test (Anastasi, 1988)

3.2.1.2.Concurrent validity

The only difference between concurrent and predictive validity is the point in time when the validating criteria are determined. The measure to be validated and the criterion measure are usually taken simultaneously or at about the same time. Generally the reason for seeking concurrent validity is to substitute one measure for another. According to McCoy (1963) concmnt validity can be evaluated by showing how well test scores or measures correlate with concun'eut status or performance.

3.2.1.3.Validity generalization

Earlier the judgements about generalisation were based upon non-quantitative reviews of the literature. Later, quantitative meta-analytic techniques were used. The results of validity generalisation studies can be used either to draw scientific conclusions andlor to use the results of validity evidence obtained from prior studies to support the use of a test in a new situation. The latter use raises questions about the degree to which validates are transportable to a specific new situation. If generalisation is limited, then local criterion-related evidence of validity may be necessary in most situations in which a test is used. If generalisation is extensive, then situation-specific evidence of validity may not be required. In conducting studies of the generalisability of validity evidence, the prior studies that are included may 'vary according to several situation facets. Some of the major facets are differences in the way the predictor construct is measured; the type of job or cufiiculum involved; the type of criterion measure; the type of test takers and the time period in which the study was conducted. A major objective of the study should be to decide whether variation in these facets affects the generalisability of validity evidence (MA, 1985).

3.2.1.4.Validity Standardisation

To evaluate a test, we need one further item of information in addition to its reliability and validity. We have to know something about the group on which the test was standardised. A clucial requirement in using tests is the comparability between the subjects who ze tested and the standardisation sample that yields the norm. If these two are drawn from different populations, the test scores may not be interpretable. The standardisation of any psychological or educational assessment instrument requires administering the instrument to a large sample of individuals (the standardisation sample) selects as representative of the target population of persons for whom the insment is intended (Aiken 1971). We are still too far from a driving simulator behavioural validity standardisation for the following reasons a) there are no standard tests to assess driver behaviour either on the road or in the simulator; b) although there are some standard tests used for the automotive industry for checking the capabilities of new cars, they are rarely applied for studies on the simulator; c) usually for each behavioural validation study, there are major differences on the type of simulators, subjects, test protocol, data collection methods. We need first to create the standard tests or improvelenhance the existing ones relative to driver performance, then apply them both on real road and in the simulator using large number of subjects on both environments (in order to avoid in-between subjects variability) and finally conclude about the validity standardisation.

3.2.2. Content-related validity

Content-related validity demonstrates the degree to which the sample of items, tasks, or questions on a test are representative of some defined universe or domain of content. The domain under consideration should be fully described in advance, rather than after the test has been prepared (Anastasi, 1988). The methods often rely on systematic observation of behaviour combined with the expert judgements but also certain logical and empirical procedures can be used to assess the relationship between parts of the test and the defined universe. Sometimes algorithms or lules can be constructed to generate items that differ systematically on various domain facets, thus assuring representiveness. Also. The first task of test developers is to specify adequately the universe of content that a test is intended to represent, given the proposed uses of the test (McCoy, 1963; APA, 1985). Content validity depends on the relevance of the individual's test responses to the behaviour area under consideration, rather than on the apparent relevance of item content. It is also important to guard against any tendency to overgeneralize regarding the domain sampled by the test. Another difficulty arises from the possible inclusion of irrelevant factors in the test scores. The end product of the testdevelopment procedures consists of the items actually included in the final version of the test. The manual should provide information on the content areas and the skills or instmctional objectives covered by the test, together with some indication of the number of items in each category.

When test users consider using an available test for a purpose other than that for which the test was developed originally, they need to judge the appropriateness of the original domain definition for the proposed new use. Another important task is to determine the degree to which the format and response properties of the sample of items or tasks in a test are representative of the universe. Methods classes in this category should often be concerned with the psychological construct underlying the test as well as with the character of test content. There is often no sharp distinction between test content and test construct. Contentrelated evidence of validity is a central concern during test development, thus inferences about content are linked to test construction as well as to establishing evidence of validity after a test has been developed and chosen for use. (Anastasi, 1988). The critical question for content-related validity is if the driver performance measures that we usually choose as representable of driving performance in real life are also representable of driving performance in a simulator. But before establishing the behavioural content-validity of a simulator, the physical content-validity has to be established, i.e. if the simulator as an aggregation of different subsystems is representative of a real car on a real road. Leonard and Wierwille (1975) suggested that content validity can be achieved by using the subjective opinion of how well the simulator resembles a real road automobile, which according to the author is face validity.

3.2.2.1.Face validity

Content validity should not be confused with face validity. The latter is not validity in the technical sense; it refers, not to what the test actually measures, but to what it appears superficially to measure. Face validity pertains to whether the test "looks valid" to the examinees who take it, the administrative ~ersonnel who decide on its use and other technically untrained observers. When refen& to driving simulators, it pertains to whether the simulator looks valid to the subjects driving it, to the people deciding . . on its use and on further investment, to the researchers who use it for their experiments.

A low face validity does not necessarily directly affect the validity of results. Yet, if it affects, e.g. subject's motivation, this in turn might affect validity. The general result of studies that compared different types and amounts of cues in driving simulators is that most cues increase face validity. A driving simulator is more realistic with more complex visual information, whereas the effect of a moving-base increases with the number of degrees of freedom. Yet, these studies do not relate face validity to absolute or relative validity of simulators to address different types of research questions. The functional validity questions the validity of the results regarding the resemblance between the behaviour in the simulator and the real task environment. Face validity should never be regarded as a substitute for objectively determined validity. As Harms et al(1996) concluded, increasing the face validity of the VTI driving simulator, it didn't necessarily enhance the overall behavioural validity of the simulator. Although this term of validity may cause some confusion, it is a desirable feature of tests. Face validity can often be improved by merely reformulating test items in terms that appear relevant and plausible in the particular setting in which they will be used. It cannot be assumed that improving the face validity of a test will improve its objective validity. Nor can it be assumed that when a test is modified so as to increase its face validity, its objective validity remains unaltered. (Anastasi, 1988)

3.2.3. Construct-related validity

The construct validity is the extent to which the performance on the test fits into a theoretical characteristic-or construct-about the attribute the test tries to measure (Aiken, 1971). This characteristic is called "construct" because it is a theoretical construction about the nature of human behaviour (APA, 1985). Each construct is developed to explain and orgaitise observed response consistencies. It derives from established interrelationships among behavioural measures and it requires the gradual accumulation of information from a variety of sources. Any data throwing light on the nature of the trait under consideration and the conditions affecting its development and manifestation represent appropriate evidence for this validation. (Anastasi, 1988).

Establishing the construct validity of a measure is a problem distinct from that of using that measure in predicting a second measure, although the lam can often contribute to construct validation. Evidence from content- and criterion-relation validation studies contributes to construct interpretations. The choice of which of one or more approaches to use to gather evidence for interpreting constructs will depend on the particular validation problem and the extent to which validation is focused on construct meaning (APA, 1985). Messick (1980) argued that the term validity, insofar as it designates the interpretative meaningfulness of a test, should be resewed for construct validity and other procedures with which the term validity has been traditionally associated should be designated by more specifically descriptive labels. Thus, content validity can be labelled content relevance and content coverage, to refer to domain specifications and domain representiveness, respectively. Criterion-related validity can be labelled predictive utility and diagnostic utility, to correspond to predictive and concurrent validation. Construct validity depends on: 1) Experts' judgements that the content of the test pertains to the construct of interest; 2) An analysis of the internal consistency of the test; 3) Studies of the relationships, in both experimentally contrived and naturally occurring groups, of test scores to other variables on which the group differ; 4) Correlations of the test with other tests and variables with which the test is expected to have a certain relationship and factor analyses of these intercorrelations; 5) Questioning examinees or raters in detail about their responses to a test or rating scale in order to reveal the specific mental processes that occurred in deciding to make those responses (Aiken, 1971). For driving simulators, construct validity can be achieved when we test if the simulator's data can be examined relative to identical hypothetical constructs used in other driving research (Breda et al, 1972; Leonard and Wierwille, 1975).

3.2.3.1.Convergent and discriminant validation

Construct validity, can be obtained if the test has high correlations with other measures (or methods of measuring) of the same construct (convergent validity) and low correlations with measures of different constructs (discriminant validity). Evidence regarding to the convergent and discriminant validity of an instrument can be possessed by comparing correlations between measures of the same construct using the same method, different constructs using the same method; the same construct using different methods; and different constructs using different methods.

Factor analysis is particularly relevant to construct validity because it is primarily used for analysing the interrelationships of behaviour data. A major purpose of factor analysis is to simplify the description of behaviour by reducing the number of categories from an initial multiplicity of test variables to a few common factors or traits. After the factors have been identified, they can be utilised in describing the factorial composition of a test. Each test can thus be characterised in terms of the major factors determining its score, together with the width or loading of each factor and the correlation of the test with each factor. Such a correlation is known as the factorial validity of the test. Correlations between a new test and similar earlier tests are sometimes cited as evidence that the new test measures approximately the same general area of behaviour as other tests designated by the same name. Unlike the correlations found in criterion-related validity, these correlations should be moderately high, but not too high. If the new test correlates too highly with an already available test, without such added advantages as brevity or ease of administration, then the new test represents needless duplication. Correlations with other tests could also be employed to demonstrate that the new test is relatively free from the influence of certain irrelevant factors (Anastasi, 1988).

Factor analysis is usually used in traffic engineering to determine which are the most important measures of driving performance on the real road. It could also be employed to identify the respective measures of driving performance on the simulator and then compare if these measures are the same for both environments (real road and simulator) and test if they are highly correlated.

3.2.4. CONCLUSIONS

McCoy (1963) investigated the applicability of the psychological terms of validity to manmachine systems and concluded that "the concepts of validity currently adhered are not of practical use to the human engineer interested in determining, quantitatively, the degree of validity attained in such a measurement system". Hoffman and Joubert (1966) concluded that opinion data, such as that used to determine content validity may not always be reliable indices of system performance. Leonard and Wienville (1975) concluded that the "ultimate method for determining validity is my] determining the degree of concurrent or predictive validity" and proposed a theory ''using the basic concept of concurrent validity applied to measured human performance". A literature review of the typical psychological measurement assessment theory and its application to driving simulators showed that it has been proven extremely difficult to apply the psychological definitions of validity to driving simulators. The author has to agree with EM (1961) and McCoy (1963) that these terms are not adequate for the simulators, because they are man-machine interacted systems and this interaction is too far complicated to be explained by these terms and new standards and procedures must be developed for the overall validation of a driving simulator.

4. REVIEW OF EARLIER AND RECENT BEHAVIOURAL VALIDATION STUDIES

The term 'bhavioural validity" of a driving simulator is defined as the comparison of driving performance indices from a particular experiment on real road with indices from an experiment in a driving simulator which is as close as it can be to the real experiment, i.e. the same road network, the same type of car, the same drivers, the same mad environment and other traffic. The issue of behavioural validity have not been addressed before 1975 for driving simulators because they were still in the developing stage but it was already a problem for the aircraft simulators. However validity had been addressed in terms of fidelity and its effects on transfer of training (Mudd, 1968; Blaiwes et al, 1973; Caro, 1973; Provenmire and Roscoe, 1973; Valverde, 1973; Williges et al, 1973). The first driving simulator validation studies examined more the physical rather than the behavioural correspondence of the simulator to the real world.

A number of behavioural validation studies have been examined here. For the early studies they are not exactly known all the technical characteristics of the simulators that they were used, or the type of statistical analysis was used and great detail about how the simulated and real road experiment have been conducted. The later validation studies have been described in greater detail and all the technical characteristics of the simulators used can be found in the relevant papers as well as in a survey about the most known driving simulators around the world by Blana (1996). Allthe details of the validation studies -both for the simulated and the real road experiments- are described in Table 9-1, Appendix A.

Before the review of any driving simulator behavioural validation study it was considered necessary from the author to give the definitions of driver behaviour and driver performance and the distinction between these two definitions as well as the definition of the driver behaviour levels as this terminology will be widely used in the following paragraphs.

4.1. DRIVER PERFORMANCE AND DRIVER BEHAVIOUR

"Driver performance... refers to the drivers' perceptual and motor skills, or what the driver can do whereas driver behaviour refers to what the driver in fact does do" (Evans, 1991 p.133). Driver performance focuses on capabilities and skills and can be investigated by many methods, including laboratory tests, simulator experiments, tests using instrumented vehicles and observations of actual traffic. On the other hand, driver behaviour cannot be investigated in laboratory, simulator or instrumented vehicles studies. Therefore, information on driver behaviour tends to be more uncertain than that of driver performance. The distinction between driver behaviour and driver performance is one of the most central concepts in traffic safety because according to NWen and Summala (1976) driving is a "self-paced" task. In other words, drivers choose their own desired levels of task diffculty. The driving task is a closed-loop compensatory feedback control process, meaning that the driver makes inputs (to the steering wheel, brake and accelerator pedal), receives feedback by monitoring the results of the inputs, and in response to the results, makes additional inputs; an open loop process is one, such as throwing a baseball, in which once the process is initiated no corrections are possible based on later knowledge about the trajectory (Evans, 1976 p.109). Relative to driving simulators, Crawford's (1961) statement thirty six years ago, that "it has proved extremely difficult to define what is meant by driving performance and to develop adequate techniques of measuring it" still stands today although according to the author's opinion Crawford is referred to driver behaviour rather than driver performance. Nilsson (1989) has also pointed out it is not exactly known which of these cues are really necessary and essential for a successful representation of the real road environment in the simulator. Nilsson defined the traffic system for traffic safety purposes as accidents, physiological measurements and driving performance. The way these measures are actually chosen in a study are strongly dependent on the hypothesis to be tested in that specific study and can be any variable that is available in the simulator model. Physiological measures include the monitoring of physical and mental stress of the body from the environment and the driving tasks, as well as the level of arousal (e.g. pulse rate, blood pressure etc.). Other miscellaneous measures include questionnaires and interviews to detect the participants' subjective opinions and evaluations concerning the tested tasks, conditions, etc. Driving performance includes, most fresuently, forward speed, lateral vehicle position and different stimulus-induced reaction times. Less frequently lateral or longitudinal accelerations, steering wheel angle and steering wheel torque are measured (Nilsson, 1989).

Limitations in simulator validity are directly related to the cues that a specific simulator provides. On one hand drivers rarely use all the available cues to perform a task. Thus according to Flexman and Stark (1987), it is not always necessary to provide in the simulator identical cues to those of real life.

On the other hand it is assumed that all types of real mad environment cues (e.g. visual information, sound, self-motion) are provided more or less to the simulator. Thus it can also be assumed that the results observed in the simulator can be successfully compared to the results obtained fmm real life.

This problem of identifying which cues are the most important for the real-world and the simulated driving could be attributed to our limited knowledge of how drivers perceive and understand the mad environment and how exactly they behave and interact with other road users while driving. Therefore, a first approach to the solution of this problem cezld be an attempt to &fine driver behaviour and the levels that a driver progresses through (consciously or unconsciously) when she drives from a point A to a point B.

4.2. DRIVER BEHAVIOUR LEVELS

According to Janssen (1979) and adopted by Michon (1985), driver's behaviour can be described by three levels: the strategic, the tactical and the control. Each level is &fined by different action patterns and a different "preview" which is the time in which the events, that are correlated with and dependent on the behaviour in the actual situation, will take place. Figure 4.1 gives diagrammaticaly the three driver behaviour levels.

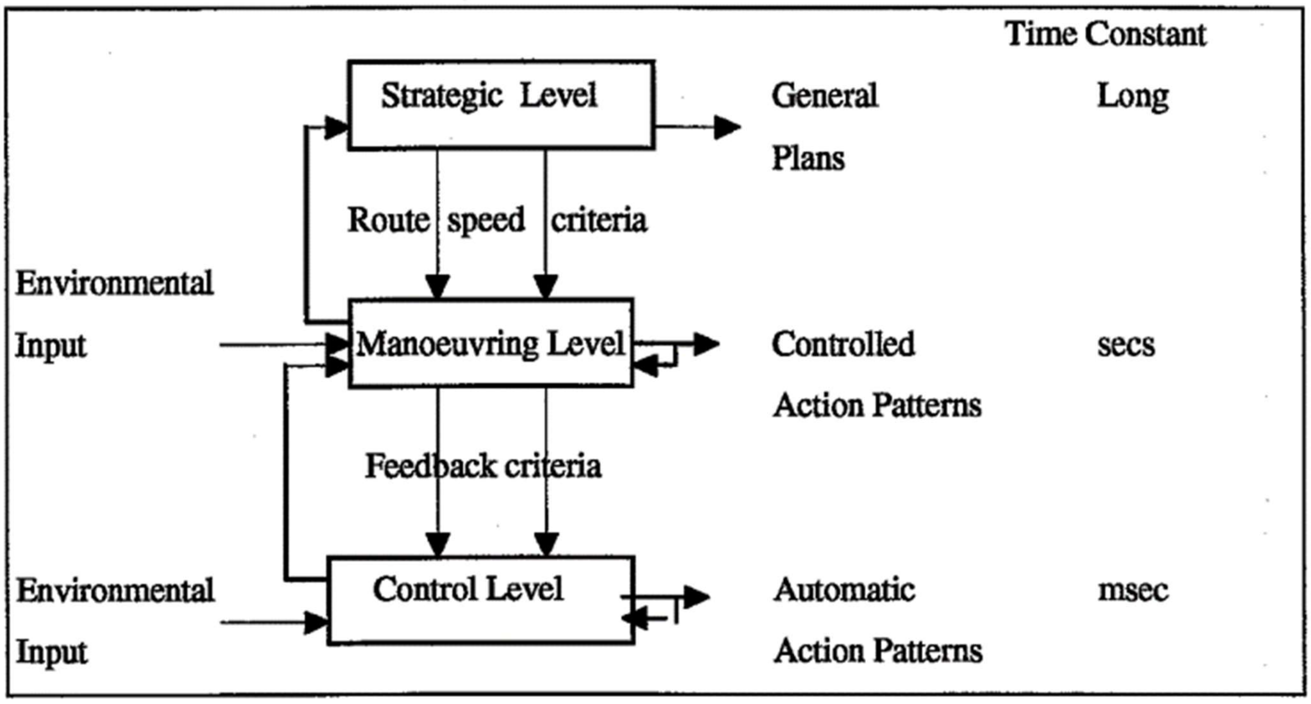


Figure 4.1 The hierchical structure of the road user task

The strategic level is mainly related to the process of mute planning, and following of a mute using various means of route information. It includes the determination of trip goals, route and modal choice, plus an evaluation of the costs and risks involved. F'lans derive further from general considerations about transport and mobility and also from common factors such as aesthetic satisfaction and comfort. At this level, the preview can be as long as the whole drive. The driver is fully aware of the different tasks. Usually in-vehicle navigation systems are tested in the simulator at this level.

The tactical level is mainly characterised by the manoeuvring behaviour (e.g. overtaking, crossing and turning, obstacle manoeuvring and gap acceptance). These patterns must meet the criteria derived from general goals set at the strategic level. Conversely these goals may occasionally be adapted to fit the outcome of certain manoeuvres. In this case, the preview is of the order of seconds to a few minutes. The assimilation of information, and decision making, are more conscious than at the control level. Simplified in-vehicle infodon systems, mobile phones, speed limiters, automatic-cruise controllers are tested in the simulator at this level. (the steering-wheel movements demonstrate a difference in control tactics).

The last level, the control level defines the automatic action patterns. The tasks which are situated here have the purpose of adjusting the position of the vehicle on the road both in longitudinal and lateral directions. Steering of the vehicle and steering it on the road and choosing speeds and gears are the relevant tasks. Two important things about the control level are that the "preview" is of the order of a few seconds or less and the different tasks are accomplished in an automatic way: the driver is hardly aware of the visual infonnation she assimilates and of the way in which this information results in decisions and actions. Traffic calming measures, new tunnel design, impaired driving and experiments which are directly related to the longitudinal and lateral control of the vehicle are tested in the simulator at this level.

In order to be able to perform the different tasks at each of these three levels, the driver needs to get information about the conditions of the surroundings. If the preview time is shorter, the need to update the information available is more frequent. This means that at the micro level, an almost continuous information stream is necessary, while at the strategic level, the information can be spread over a longer timescale and it doesn't have to be continuous.

How are these driver behaviour levels related to the behavioural validation of driving simulators? When conducting experiments on research driving simulators, a combination of different action patterns out of the three different driver behaviour levels is often used to describe the experiment and more often these are the control patterns.

CHAPTER 3

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

System analysis procedures as applied to driving simulators must accomplish several bask tasks: (1) they must identify the system components and interactions that are to be simulated; (2) they must specify the degree of simulation of each component in the system. In this case, the extent to which the automobile and the environment are to be reproduced by the simulator must be adequately established; (3) the input and output to the simulator, plus the process of introducing variability and measuring the response to it, must be subjected to careful analysis; (4) finally, the simulation process requires that a major portion of any driver-related systems analysis be devoted to an examination of each component in the system, its input, functioning, and response characteristics.

The basic components of the driving situation were analyzed from a systems point of view. In order to summarize some existing simulation attempts, the particular focus of each study with respect to the component or method of handling a component, was utilized. The driving simulation literature was reviewed to provide an integrated conceptual framework of the accomplishments of research in this field

The basic component relations were explored and the primary functions of simulation were then analyzed. These primary functions were: (1) the introduction of experimental variation into the system; (2) the representation of the components of the system; (3) the measurement of component response; and (4) the measurement of system performance.

FEASIBILTY STUDY

Feasibility Study is an important concept in any System development. It should be noted always how for the system is feasible on the way in its development and after development. It is made mainly into 3 aspects.

Technical Feasibility

The feasibility study involves knowing whether the project can be developed with the help of current equipment, existing software technologies, employees etc.
The organization contains required equipment and required software and the employees. No other special requirements are to be created just for the system development. The system is technically feasible.

Economic Feasibility

Economic Feasibility involves cost to develop and install the system. Here, the check is done whether the organization can meet the costs involved in developing the project or not.

Operational Feasibility

The study involves in finding whether the system will be used if it is developed and implemented or there will be any resistance from the user due to development of system.

Introduction to UNITY

Unity is an engine for creating games on multiple platforms. Unity was released by Unity Technologies in 2005. The focus of Unity lies in the development of both 3D and 2D games and interactive content. Unity now supports 27 different target platforms for deploying. The most popular platforms are Android, PC, and iOS systems.

What is Unity

Unity is an engine for creating games on multiple platforms. Unity was released by Unity Technologies in 2005. The focus of Unity lies in the development of both 3D and 2D games and interactive content. Unity now supports 27 different target platforms for deploying. The most popular platforms are Android, PC, and iOS systems.

- Unity is an integrated platform that is used as a gaming engine and framework.
- Unity allows you to develop once and publish everywhere.
- Although unity is considered to be more appropriate for creating 3D games, it can also be equally used to develop 2D games.
- In Unity, it is possible to develop games with heavy assets without depending on the additional frameworks or engines. It really enhances the experience of users.
- With the help of Unity, our game developers can access a wealth of resources like intuitive tools, ready-made assets, clear documentation, online community, etc. free of cost for creating exciting 3D contents in the games.
- Asset tracking and rendering, scripting are some of the features of Unity game development that we use in reducing time and cost.

Factors that Enhances the Efficiency of Unity

Unity Multiplayer: Multiplayer experience in unity is unparalleled. Unity enables the users to play for the traffic that uses the relay servers and matchmakers.

IDE: Unity provides the text editor for writing the code. To reduce confusion, sometimes, a distinct code editor is also used by our developers. As the IDE (Integrated development editor) of the unity engine supports C# and Unity Script (JavaScript), we use it in our game development process for creating immersive and exciting games.

Platform Support: The Unity engine is highly acceptable due to its ability to support a total of 27 different platforms. It is used for developing and deploying gaming apps that can be easily shared among personal computers, mobile, and web platforms.

Debugging: Debugging and Tweaking is extremely easier with Unity game development. During the gameplay, the game variables are displayed, and it allows the developers to debug the process at run-time.

Unity Analytics: The built-in analytics of Unity offers indispensable insights regarding your game. It is necessary during the release of a game. Information related to the distribution of the game and feedback of the players can be easily obtained through unity analytics.

Graphics: The unity engine provides high-quality visual effects and audio. The visuals developed by Unity are adaptable on every device and screen without any compromise or distortion with the quality of the images.

Unity is the most preferred gaming engine among today's developers. On Unity, the coding part of the game development process is only about 20%. Hence extensive programming skills are not required.

The free license offered by Unity makes it open for game developers all over the world. The developers can access the resources from the asset store of Unity, which is used to enrich the process of mobile game development.

Prerequisite

It is important to have access to the machine that meets Unity's minimum requirements. Prerequisite knowledge of basic C# is required for a full understanding of this series.

Audience

This tutorial is created for those who find the world of gaming exciting and creative. This tutorial will help learners who aspire to learn game-making.

HISTORY OF UNITY

The Unity game engine launched in 2005, aiming to "democratize" game development by making it accessible to more developers. The next year, Unity was named runner-up in the Best Use of Mac OS X Graphics category in Apple Inc.'s Apple Design Awards. Unity was initially released for Mac OS X, later adding support for Microsoft Windows and Web browsers.

Unity 2.0 (2007)

Unity 2.0 launched in 2007 with approximately 50 new features. The release included an optimized terrain engine for detailed 3D environments, real-time dynamic shadows, directional lights and spotlights, video playback, and other features. The release also added features whereby developers could collaborate more easily. It included a Networking Layer for developers to create multiplayer games based on the User Datagram Protocol, offering Network Address Translation, State Synchronization, and Remote Procedure Calls. When Apple launched its App Store in 2008, Unity quickly added support for the iPhone. For several years, the engine was uncontested on the iPhone and it became well-known with iOS game developers.

Unity 3.0 (2010)

Unity 3.0 launched in September 2010 with features expanding the engine's graphics features for desktop computers and video game consoles. In addition to Android support, Unity 3 featured integration of Illuminate Labs' Beast Lightmap tool, deferred rendering, a built-in tree editor, native font rendering, automatic UV mapping, and audio filters, among other things. In 2012 VentureBeat wrote, "Few companies have contributed as much to the flowing of independently produced games as Unity Technologies. [...] More than 1.3 million developers are using its tools to create gee-whiz graphics in their iOS, Android, console, PC, and web-based games. Unity wants to be the engine for multi-platform games, period." A May 2012 survey by Game Developer magazine indicated Unity as its top game engine for mobile platforms.

Unity 4.0 (2012)

In November 2012, Unity Technologies delivered Unity 4.0. This version added DirectX 11 and Adobe Flash support, new animation tools called Mecanim, and access to the Linux preview.

Facebook integrated a software development kit for games using the Unity game engine in 2013. This featured tools that allowed tracking advertising campaigns and deep linking, where users were directly linked from social media posts to specific portions within games, and easy in-game-image sharing. In 2016, Facebook developed a new PC gaming platform with Unity. Unity provided support for Facebook's gaming platforms, and Unity developers could more quickly export and publish games to Facebook.

Unity 5 (2015)

The Verge said of 2015's Unity 5 release: "Unity started with the goal of making game development universally accessible. [...] Unity 5 is a long-awaited step towards that future." With Unity 5, the engine improved its lighting and audio. Through WebGL, Unity developers could add their games to compatible Web browsers with no plug-ins required for players. Unity 5.0 offered real-time global illumination, light mapping previews, Unity Cloud, a new audio system, and the Nvidia PhysX 3.3 physics engine. The fifth generation of the Unity engine also introduced Cinematic Image Effects to help make Unity games look less generic. Unity 5.6 added new lighting and particle effects, updated the engine's overall performance, and added native support for Nintendo Switch, Facebook Gameroom, Google Daydream, and the Vulkan graphics API. It introduced a 4K video player capable of running 360-degree videos for virtual reality.

However, some gamers criticized Unity's accessibility due to the high volume of quickly produced games published on the Steam distribution platform by inexperienced developers. CEO John Riccitiello said in an interview that he believes this to be a side-effect of Unity's success in democratizing game development: "If I had my way, I'd like to see 50 million people using Unity – although I don't think we're going to get there any time soon. I'd like to see high school and college kids using it, people outside the core industry. I think it's sad that most people are consumers of technology and not creators. The world's a better place when people know how to create, not just consume, and that's what we're trying to promote."

Unity (2017–present)

In December 2016, Unity Technologies announced that they would change the versioning numbering system for Unity from sequence-based identifiers to year of release to align the versioning with their more frequent release cadence; Unity 5.6 was therefore followed by Unity 2017. Unity 2017 tools featured a real-time graphics rendering engine, color grading and worldbuilding, live operations analytics and performance reporting. Unity 2017.2 underscored Unity Technologies' plans beyond video games. This included new tools such as Timeline, which allowed developers to drag-and-drop animations into games, and Cinemachine, a smart camera system within games. Unity 2017.2 also integrated Autodesk's 3DS Max and Maya tools into the Unity engine for a streamlined asset sharing in-game iteration process.

Unity 2018 featured the Scriptable Render Pipeline for developers to create high-end graphics. This included the High-Definition Rendering Pipeline for console and PC experiences, and the Lightweight Rendering Pipeline (later renamed to the Universal Render Pipeline) for mobile, virtual reality, and augmented reality.

Unity 2018 also included machine learning tools, such as Imitation Learning, whereby games learn from real player habits, support for Magic Leap, and templates for new developers.

The C# source code of Unity was published under a "reference-only" license in March 2018, which prohibits reuse and modification.

As of 2020, software built with Unity's game engine was running on more than 1.5 billion devices. According to Unity, apps made with their game engine account for 50 percent of all mobile games, and are downloaded more than 3 billion times per month, and approximately 15,000 new projects are started daily with its software. Financial Times reported that Unity's engine "powers some of the world's most lucrative mobile games", such as Pokémon Go and Activision's Call of Duty Mobile.

In June 2020, Unity introduced the Mixed and Augmented Reality Studio (MARS), which provides developers with additional functionality for rules-based generation of augmented reality (AR) applications. Unity released Unity Forma, an automotive and retail solution tool, on December 9, 2020.

Unity acquired Finger Food Advanced Technology Group in 2020, as it aimed to bolster its non-video game uses and offer additional design help to customers. The company went public in September 2020, to further expand use of its game engine into industries outside of gaming.

In June 2020, Unity announced the Unity Editor will support Apple Silicon. The first beta version shipped later that year.

Unity 2021 brought multiple new features such as Bolt, Unity's Visual Scripting system, a new multiplayer library to support multiplayer games, improved Il2cpp runtime performance, Volumetric clouds for the High Definition Render pipeline. Shadow caching and Screen Space Global Illumination for HDRP. For the Universal Render Pipeline it added new features such as point light shadows, Deferred renderer and general core engine improvements and fixes. Full Apple Silicon support was also added in Unity 2021.2. Unity Hub support for Apple Silicon editors arrived in version 3.0 in January 2022.

Changes to Unity 2022 were intended to improve productivity by reducing the time required to enter play mode and import files, and implementing visual search queries and multiselection in the package manager. For 2D projects, changes focused on accelerating core software, import, animation, and physics. Sprite atlasing was revised. Support for PSD extension files and layer management were added to the 2D PSD Importer, and Delaunay tessellation for 2D physics was added.

FEATURES

Unity gives users the ability to create games and experiences in both 2D and 3D, and the engine offers a primary scripting API in C# using Mono, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality. Prior to C# being the primary programming language used for the engine, it previously supported Boo, which was removed with the release of Unity 5, and a Boo-based implementation of JavaScript called UnityScript, which was deprecated in August 2017, after the release of Unity 2017.1, in favor of C#.

Within 2D games, Unity allows importation of sprites and an advanced 2D world renderer. For 3D games, Unity allows specification of texture compression, mipmaps, and resolution settings for each platform that the game engine supports, and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects.

Two separate render pipelines are available, High Definition Render Pipeline (HDRP) and Universal Render Pipeline (URP, previously LWRP), in addition to the legacy built-in pipeline. All three render pipelines are incompatible with each other. Unity offers a tool to upgrade shaders using the legacy renderer to URP or HDRP.

USAGE SHARE OF UNITY

Top Competitors and Alternatives of Unity

The top three of Unity's competitors in the Game Development category are Discord with 20.04%, Unreal Engine with 14.82%, Blender with 11.35% market share.

Technology	Domains	Market Share	Versus page
Discord	8998	20.04%	Unity vs Discord
Unreal Engine	6655	14.82%	Unity vs Unreal Engine
Blender	5093	11.35%	Unity vs Blender
Google Gaming	2306	5.14%	Unity vs Google Gaming

Technology	Domains	Market Share	Versus page
Kahoot!	1615	3.60%	Unity vs Kahoot!
Godot	1604	3.57%	Unity vs Godot
BabylonJS	406	0.90%	Unity vs BabylonJS
GameMaker	399	0.89%	Unity vs GameMaker
GameAnalytics	397	0.88%	Unity vs GameAnalytics
Mod.io	320	0.71%	Unity vs Mod.io
LibGDX	276	0.61%	Unity vs LibGDX
Phaser	275	0.61%	Unity vs Phaser
Bandicam	261	0.58%	Unity vs Bandicam
PlayFab	248	0.55%	Unity vs PlayFab
CRYENGINE	242	0.54%	Unity vs CRYENGINE
Stencyl	212	0.47%	Unity vs Stencyl
BeamNG	199	0.44%	Unity vs BeamNG
GameSalad	164	0.37%	Unity vs GameSalad

Technology	Domains	Market Share	Versus page
Chartboost	160	0.36%	Unity vs Chartboost
Kivy	132	0.29%	Unity vs Kivy

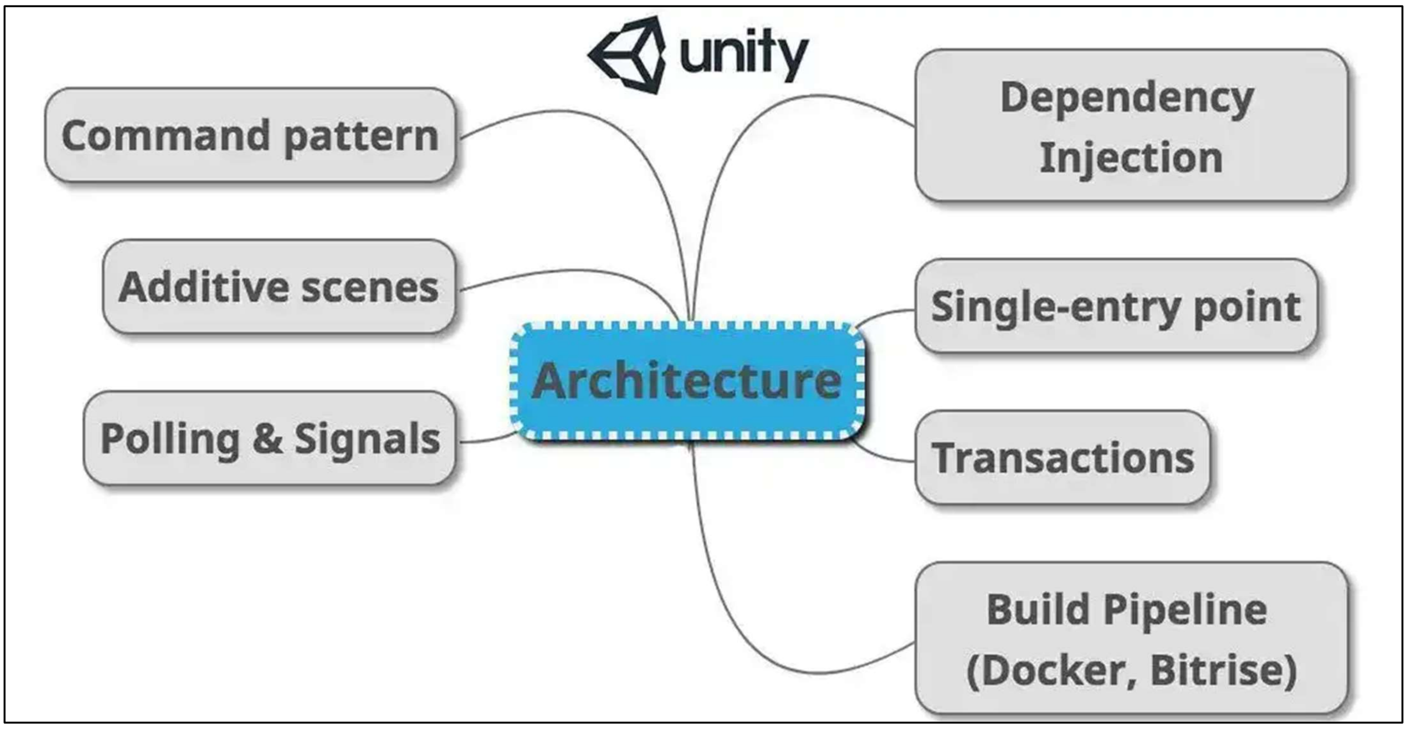
ARCHITECTURE OF UNITY

The Unity engine is built with native C/C++ internally, however it has a C# wrapper that you use to interact with it. As such, you need to be familiar with some of the key concepts of scripting in C#. This section of the User Manual contains information on how Unity implements .NET and C#, and any exceptions you might encounter as you code.

For information on how to get started scripting in Unity, and the fundamentals you need to know, see the documentation on Getting started scripting in Unity.

This section covers the following topics:

Page	Description
Overview of .NET in Unity	How the Unity engine uses the .NET framework, and any differences you might encounter if you have used .NET outside of Unity before. This area also contains information on how Unity manages memory, and how to reference additional profiles in your Project.
Scripting backends	Unity has two main scripting backends: Mono and IL2CPP. This section describes the differences between the backends and how and when to use them, plus their restrictions. It also contains information on managed code stripping, which removes unused code from your build.
Code reloading in the Editor	Information on domain reloads and how they impact on the performance of your application. Also contains information on running code on Editor launch, and how to quickly enter and exit Play mode with Configurable Enter Play Mode.
Script serialization	Serialization is the automatic process of transforming data structures or object states into a format that Unity can store and reconstruct later. This contains information on how to use serialization in your Project in an effective way.
Script compilation	How Unity compiles and in what order. Also contains information on Assembly Definitions and best practices around using them.


unity
Command pattern
Additive scenes
Polling & Signals
Architecture
Dependency Injection
Single-entry point
Transactions
Build Pipeline (Docker, Bitrise)

CHAPTER 04
IMPLEMENTATION

4.IMPLEMENTATION

##Starting_screen.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Starting_screen : MonoBehaviour
{
   public void play(){
      SceneManager.LoadScene(2);
   }
   public void exit(){
      Application.Quit();
   }
   public void control(){
      SceneManager.LoadScene(5);
   }
   public void credit(){
      SceneManager.LoadScene(4);
   }
}
```

##CarController.cs

```
using System;
using UnityEngine;

#pragma warning disable 649
namespace UnityStandardAssets.Vehicles.Car
{
   internal enum CarDriveType
   {
      FrontWheelDrive,
      RearWheelDrive,
      FourWheelDrive
   }

   internal enum SpeedType
   {
      MPH,
      KPH
   }

   public class CarController : MonoBehaviour
   {
      [SerializeField] private CarDriveType m_CarDriveType = CarDriveType.FourWheelDrive;
      [SerializeField] private WheelCollider[] m_WheelColliders = new WheelCollider[4];
      [SerializeField] private GameObject[] m_WheelMeshes = new GameObject[4];
      [SerializeField] private WheelEffects[] m_WheelEffects = new WheelEffects[4];
```

```
    [SerializeField] private Vector3 m_CentreOfMassOffset;
    [SerializeField] private float m_MaximumSteerAngle;
    [Range(0, 1)] [SerializeField] private float m_SteerHelper; // 0 is raw physics , 1 the car will grip in the
direction it is facing
    [Range(0, 1)] [SerializeField] private float m_TractionControl; // 0 is no traction control, 1 is full
interference
    [SerializeField] private float m_FullTorqueOverAllWheels;
    [SerializeField] private float m_ReverseTorque;
    [SerializeField] private float m_MaxHandbrakeTorque;
    [SerializeField] private float m_Downforce = 100f;
    [SerializeField] private SpeedType m_SpeedType;
    [SerializeField] private float m_Topspeed = 200;
    [SerializeField] private static int NoOfGears = 5;
    [SerializeField] private float m_RevRangeBoundary = 1f;
    [SerializeField] private float m_SlipLimit;
    [SerializeField] private float m_BrakeTorque;

    private Quaternion[] m_WheelMeshLocalRotations;
    private Vector3 m_Prevpos, m_Pos;
    private float m_SteerAngle;
    private int m_GearNum;
    private float m_GearFactor;
    private float m_OldRotation;
    private float m_CurrentTorque;
    private Rigidbody m_Rigidbody;
    private const float k_ReversingThreshold = 0.01f;

    public bool Skidding { get; private set; }
    public float BrakeInput { get; private set; }
    public float CurrentSteerAngle{ get { return m_SteerAngle; }}
    public float CurrentSpeed{ get { return m_Rigidbody.velocity.magnitude*2.23693629f; }}
    public float MaxSpeed{get { return m_Topspeed; }}
    public float Revs { get; private set; }
    public float AccelInput { get; private set; }

    // Use this for initialization
    private void Start()
    {
      m_WheelMeshLocalRotations = new Quaternion[4];
      for (int i = 0; i < 4; i++)
      {
        m_WheelMeshLocalRotations[i] = m_WheelMeshes[i].transform.localRotation;
      }
      m_WheelColliders[0].attachedRigidbody.centerOfMass = m_CentreOfMassOffset;

      m_MaxHandbrakeTorque = float.MaxValue;

      m_Rigidbody = GetComponent<Rigidbody>();
      m_CurrentTorque           =           m_FullTorqueOverAllWheels           -
(m_TractionControl*m_FullTorqueOverAllWheels);
```

```
    }


    private void GearChanging()
    {
        float f = Mathf.Abs(CurrentSpeed/MaxSpeed);
        float upgearlimit = (1/(float) NoOfGears)*(m_GearNum + 1);
        float downgearlimit = (1/(float) NoOfGears)*m_GearNum;

        if (m_GearNum > 0 && f < downgearlimit)
        {
            m_GearNum--;
        }

        if (f > upgearlimit && (m_GearNum < (NoOfGears - 1)))
        {
            m_GearNum++;
        }
    }


    // simple function to add a curved bias towards 1 for a value in the 0-1 range
    private static float CurveFactor(float factor)
    {
        return 1 - (1 - factor)*(1 - factor);
    }


    // unclamped version of Lerp, to allow value to exceed the from-to range
    private static float ULerp(float from, float to, float value)
    {
        return (1.0f - value)*from + value*to;
    }


    private void CalculateGearFactor()
    {
        float f = (1/(float) NoOfGears);
        // gear factor is a normalised representation of the current speed within the current gear's range of
speeds.
        // We smooth towards the 'target' gear factor, so that revs don't instantly snap up or down when
changing gear.
        var targetGearFactor = Mathf.InverseLerp(f*m_GearNum, f*(m_GearNum + 1),
Mathf.Abs(CurrentSpeed/MaxSpeed));
        m_GearFactor = Mathf.Lerp(m_GearFactor, targetGearFactor, Time.deltaTime*5f);
    }


    private void CalculateRevs()
    {
```

```
    // calculate engine revs (for display / sound)
    // (this is done in retrospect - revs are not used in force/power calculations)
    CalculateGearFactor();
    var gearNumFactor = m_GearNum/(float) NoOfGears;
    var revsRangeMin = ULerp(0f, m_RevRangeBoundary, CurveFactor(gearNumFactor));
    var revsRangeMax = ULerp(m_RevRangeBoundary, 1f, gearNumFactor);
    Revs = ULerp(revsRangeMin, revsRangeMax, m_GearFactor);
}


public void Move(float steering, float accel, float footbrake, float handbrake)
{
    for (int i = 0; i < 4; i++)
    {
        Quaternion quat;
        Vector3 position;
        m_WheelColliders[i].GetWorldPose(out position, out quat);
        m_WheelMeshes[i].transform.position = position;
        m_WheelMeshes[i].transform.rotation = quat;
    }

    //clamp input values
    steering = Mathf.Clamp(steering, -1, 1);
    AccelInput = accel = Mathf.Clamp(accel, 0, 1);
    BrakeInput = footbrake = -1*Mathf.Clamp(footbrake, -1, 0);
    handbrake = Mathf.Clamp(handbrake, 0, 1);

    //Set the steer on the front wheels.
    //Assuming that wheels 0 and 1 are the front wheels.
    m_SteerAngle = steering*m_MaximumSteerAngle;
    m_WheelColliders[0].steerAngle = m_SteerAngle;
    m_WheelColliders[1].steerAngle = m_SteerAngle;

    SteerHelper();
    ApplyDrive(accel, footbrake);
    CapSpeed();

    //Set the handbrake.
    //Assuming that wheels 2 and 3 are the rear wheels.
    if (handbrake > 0f)
    {
        var hbTorque = handbrake*m_MaxHandbrakeTorque;
        m_WheelColliders[2].brakeTorque = hbTorque;
        m_WheelColliders[3].brakeTorque = hbTorque;
    }


    CalculateRevs();
    GearChanging();
    AddDownForce();
    CheckForWheelSpin();
    TractionControl();
}


private void CapSpeed()
{
    float speed = m_Rigidbody.velocity.magnitude;
    switch (m_SpeedType)
    {
        case SpeedType.MPH:

            speed *= 2.23693629f;
            if (speed > m_Topspeed)
                m_Rigidbody.velocity = (m_Topspeed/2.23693629f) * m_Rigidbody.velocity.normalized;
            break;

        case SpeedType.KPH:
            speed *= 3.6f;
            if (speed > m_Topspeed)
                m_Rigidbody.velocity = (m_Topspeed/3.6f) * m_Rigidbody.velocity.normalized;
            break;
    }
}


private void ApplyDrive(float accel, float footbrake)
{

    float thrustTorque;
    switch (m_CarDriveType)
    {
        case CarDriveType.FourWheelDrive:
            thrustTorque = accel * (m_CurrentTorque / 4f);
            for (int i = 0; i < 4; i++)
            {
                m_WheelColliders[i].motorTorque = thrustTorque;
            }
            break;

        case CarDriveType.FrontWheelDrive:
            thrustTorque = accel * (m_CurrentTorque / 2f);
            m_WheelColliders[0].motorTorque = m_WheelColliders[1].motorTorque = thrustTorque;
            break;

        case CarDriveType.RearWheelDrive:
            thrustTorque = accel * (m_CurrentTorque / 2f);
            m_WheelColliders[2].motorTorque = m_WheelColliders[3].motorTorque = thrustTorque;
            break;
    }

    for (int i = 0; i < 4; i++)
    {
        if (CurrentSpeed > 5 && Vector3.Angle(transform.forward, m_Rigidbody.velocity) < 50f)
        {
            m_WheelColliders[i].brakeTorque = m_BrakeTorque*footbrake;
        }
        else if (footbrake > 0)
        {
            m_WheelColliders[i].brakeTorque = 0f;
            m_WheelColliders[i].motorTorque = -m_ReverseTorque*footbrake;
        }
    }
}


private void SteerHelper()
{
    for (int i = 0; i < 4; i++)
    {
        WheelHit wheelhit;
        m_WheelColliders[i].GetGroundHit(out wheelhit);
        if (wheelhit.normal == Vector3.zero)
            return; // wheels arent on the ground so dont realign the rigidbody velocity
    }

    // this if is needed to avoid gimbal lock problems that will make the car suddenly shift direction
    if (Mathf.Abs(m_OldRotation - transform.eulerAngles.y) < 10f)
    {
        var turnadjust = (transform.eulerAngles.y - m_OldRotation) * m_SteerHelper;
        Quaternion velRotation = Quaternion.AngleAxis(turnadjust, Vector3.up);
        m_Rigidbody.velocity = velRotation * m_Rigidbody.velocity;
    }
    m_OldRotation = transform.eulerAngles.y;
}


// this is used to add more grip in relation to speed
private void AddDownForce()
{
    m_WheelColliders[0].attachedRigidbody.AddForce(-transform.up*m_Downforce*
                                  m_WheelColliders[0].attachedRigidbody.velocity.magnitude);
}


// checks if the wheels are spinning and is so does three things
// 1) emits particles
// 2) plays tiure skidding sounds
```

```
    // 3) leaves skidmarks on the ground
    // these effects are controlled through the WheelEffects class
    private void CheckForWheelSpin()
    {
        // loop through all wheels
        for (int i = 0; i < 4; i++)
        {
            WheelHit wheelHit;
            m_WheelColliders[i].GetGroundHit(out wheelHit);

            // is the tire slipping above the given threshhold
            if (Mathf.Abs(wheelHit.forwardSlip) >= m_SlipLimit || Mathf.Abs(wheelHit.sidewaysSlip) >=
m_SlipLimit)
            {
                m_WheelEffects[i].EmitTyreSmoke();

                // avoiding all four tires screeching at the same time
                // if they do it can lead to some strange audio artefacts
                if (!AnySkidSoundPlaying())
                {
                    m_WheelEffects[i].PlayAudio();
                }
                continue;
            }

            // if it wasnt slipping stop all the audio
            if (m_WheelEffects[i].PlayingAudio)
            {
                m_WheelEffects[i].StopAudio();
            }
            // end the trail generation
            m_WheelEffects[i].EndSkidTrail();
        }
    }

    // crude traction control that reduces the power to wheel if the car is wheel spinning too much
    private void TractionControl()
    {
        WheelHit wheelHit;
        switch (m_CarDriveType)
        {
            case CarDriveType.FourWheelDrive:
                // loop through all wheels
                for (int i = 0; i < 4; i++)
                {
                    m_WheelColliders[i].GetGroundHit(out wheelHit);

                    AdjustTorque(wheelHit.forwardSlip);
                }
                break;
            case CarDriveType.RearWheelDrive:
                m_WheelColliders[2].GetGroundHit(out wheelHit);
                AdjustTorque(wheelHit.forwardSlip);

                m_WheelColliders[3].GetGroundHit(out wheelHit);
                AdjustTorque(wheelHit.forwardSlip);
                break;

            case CarDriveType.FrontWheelDrive:
                m_WheelColliders[0].GetGroundHit(out wheelHit);
                AdjustTorque(wheelHit.forwardSlip);

                m_WheelColliders[1].GetGroundHit(out wheelHit);
                AdjustTorque(wheelHit.forwardSlip);
                break;
        }
    }


    private void AdjustTorque(float forwardSlip)
    {
        if (forwardSlip >= m_SlipLimit && m_CurrentTorque >= 0)
        {
            m_CurrentTorque -= 10 * m_TractionControl;
        }
        else
        {
            m_CurrentTorque += 10 * m_TractionControl;
            if (m_CurrentTorque > m_FullTorqueOverAllWheels)
            {
                m_CurrentTorque = m_FullTorqueOverAllWheels;
            }
        }
    }


    private bool AnySkidSoundPlaying()
    {
        for (int i = 0; i < 4; i++)
        {
            if (m_WheelEffects[i].PlayingAudio)
            {
                return true;
            }
        }
        return false;
    }
  }
}
```

##PauseMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public GameObject pausemenu;
    public bool ispaused;
    private bool ismute;
    // Start is called before the first frame update
    void Start()
    {
        pausemenu.SetActive(false);
        ismute = PlayerPrefs.GetInt("MUTED") == 1;
        AudioListener.pause = ismute;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape)){
            if(ispaused){
               resumegame();
            }
            else{
                pausegame();
            }
        }
    }
    public void pausegame(){
        pausemenu.SetActive(true);
        Time.timeScale = 0f;
        ispaused = true;
        AudioSource[] audios = FindObjectsOfType<AudioSource>();
        foreach (AudioSource a in audios){
            a.Pause();
        }
    }
    public void resumegame(){
        pausemenu.SetActive(false);
        Time.timeScale = 1f;
        ispaused = false;
        AudioSource[] audios = FindObjectsOfType<AudioSource>();
        foreach (AudioSource a in audios){
            a.Play();
        }
    }
    public void backtomain(){
        Time.timeScale = 1f;
        SceneManager.LoadScene(0);
    }
    public void mapselect(){
        Time.timeScale = 1f;
        SceneManager.LoadScene(2);
    }
    public void volume(){
        ismute = !ismute;
        AudioListener.pause = ismute;
        PlayerPrefs.SetInt("MUTED", ismute ? 1 : 0);
    }
    public void restart(){
        Time.timeScale = 1f;
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
}
```

<u>##Back.cs</u>

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Back : MonoBehaviour
{
    public void back_menu(){
        SceneManager.LoadScene(0);
    }
}
```

CHAPTER 05
OUTPUT SCREENS

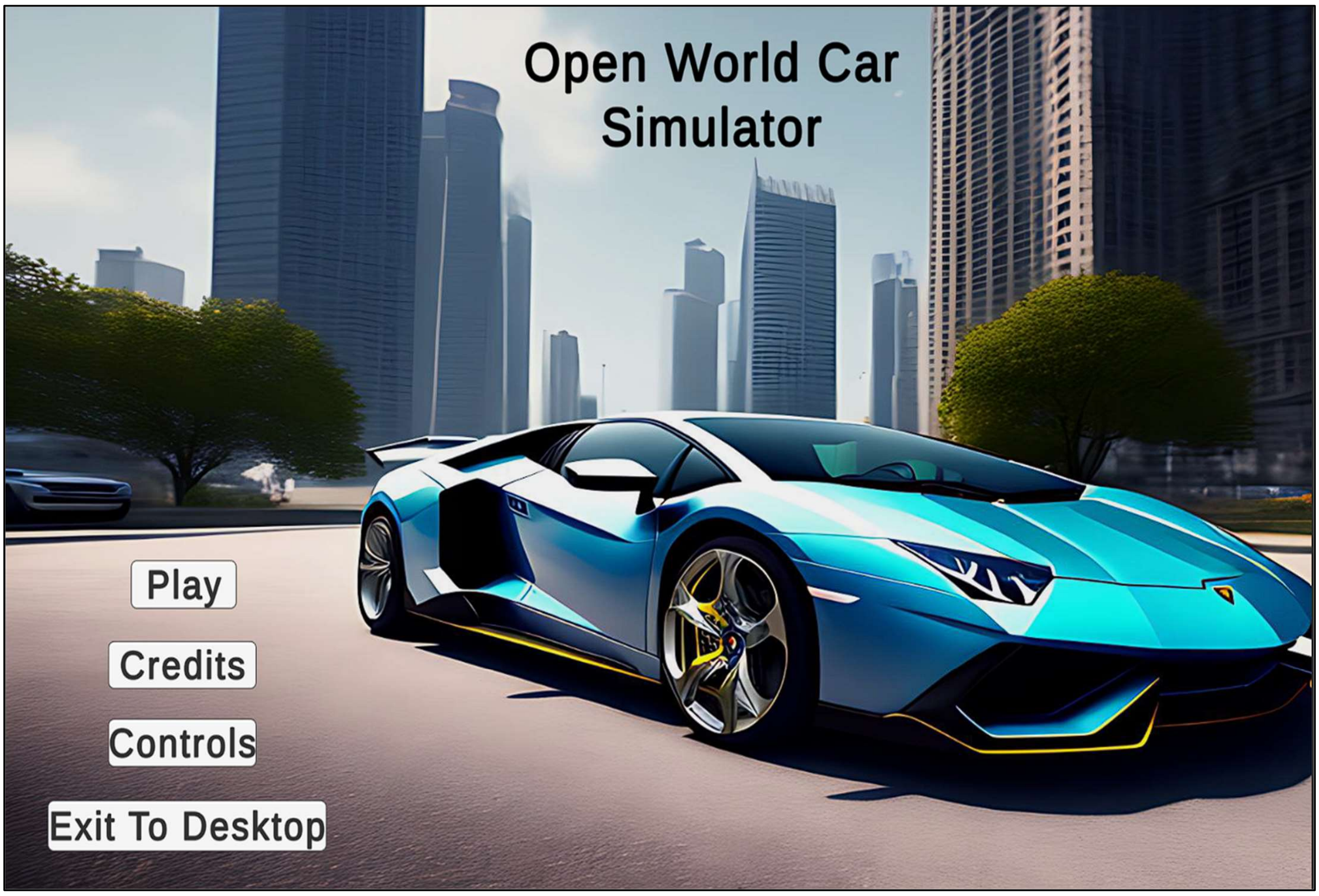
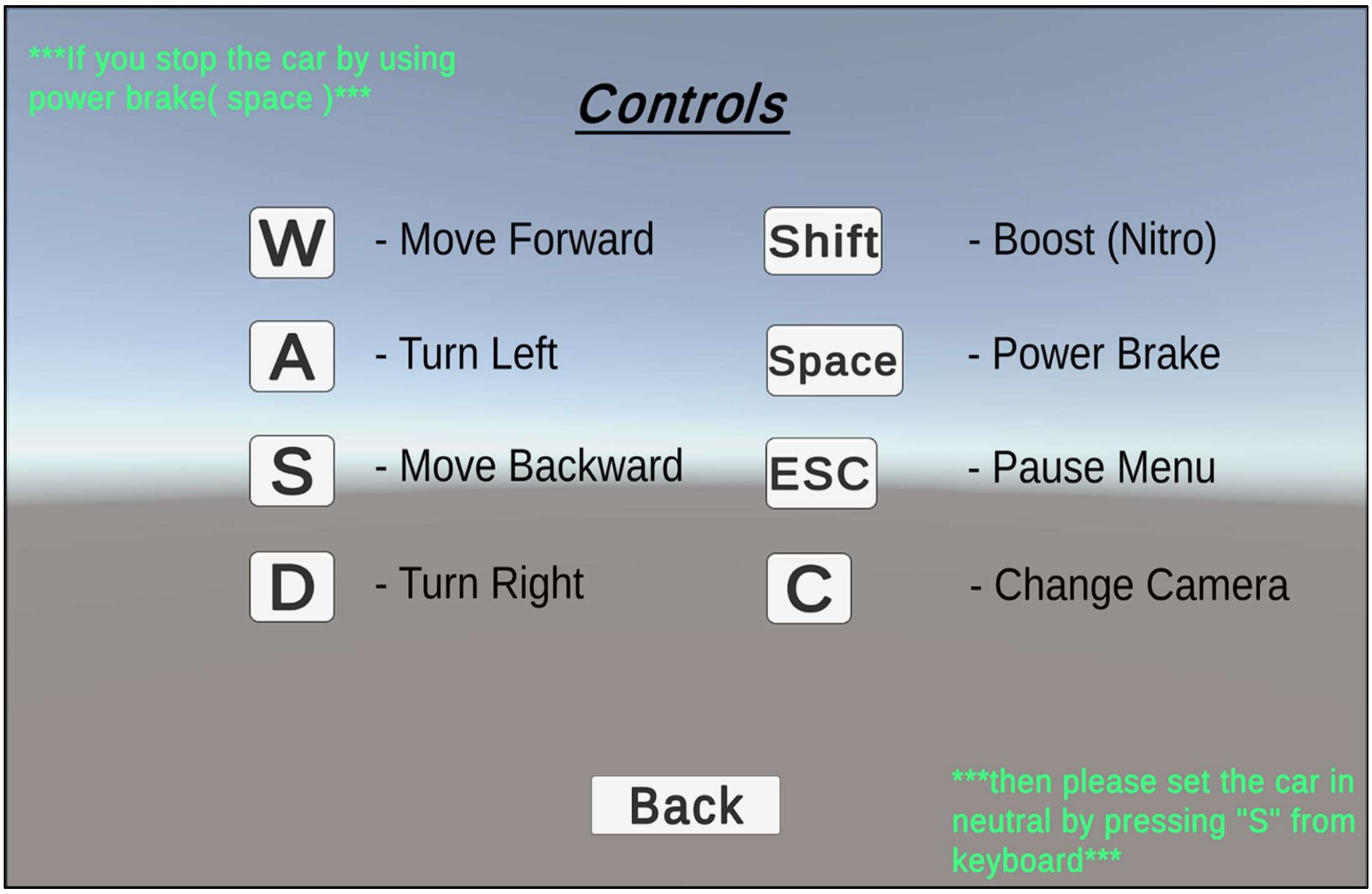


Fig 5.1 Starting Window


If you stop the car by using power brake(space)

Controls

W	- Move Forward	Shift	- Boost (Nitro)
A	- Turn Left	Space	- Power Brake
S	- Move Backward	ESC	- Pause Menu
D	- Turn Right	C	- Change Camera

Back

then please set the car in neutral by pressing "S" from keyboard

Fig 5.2 Controls

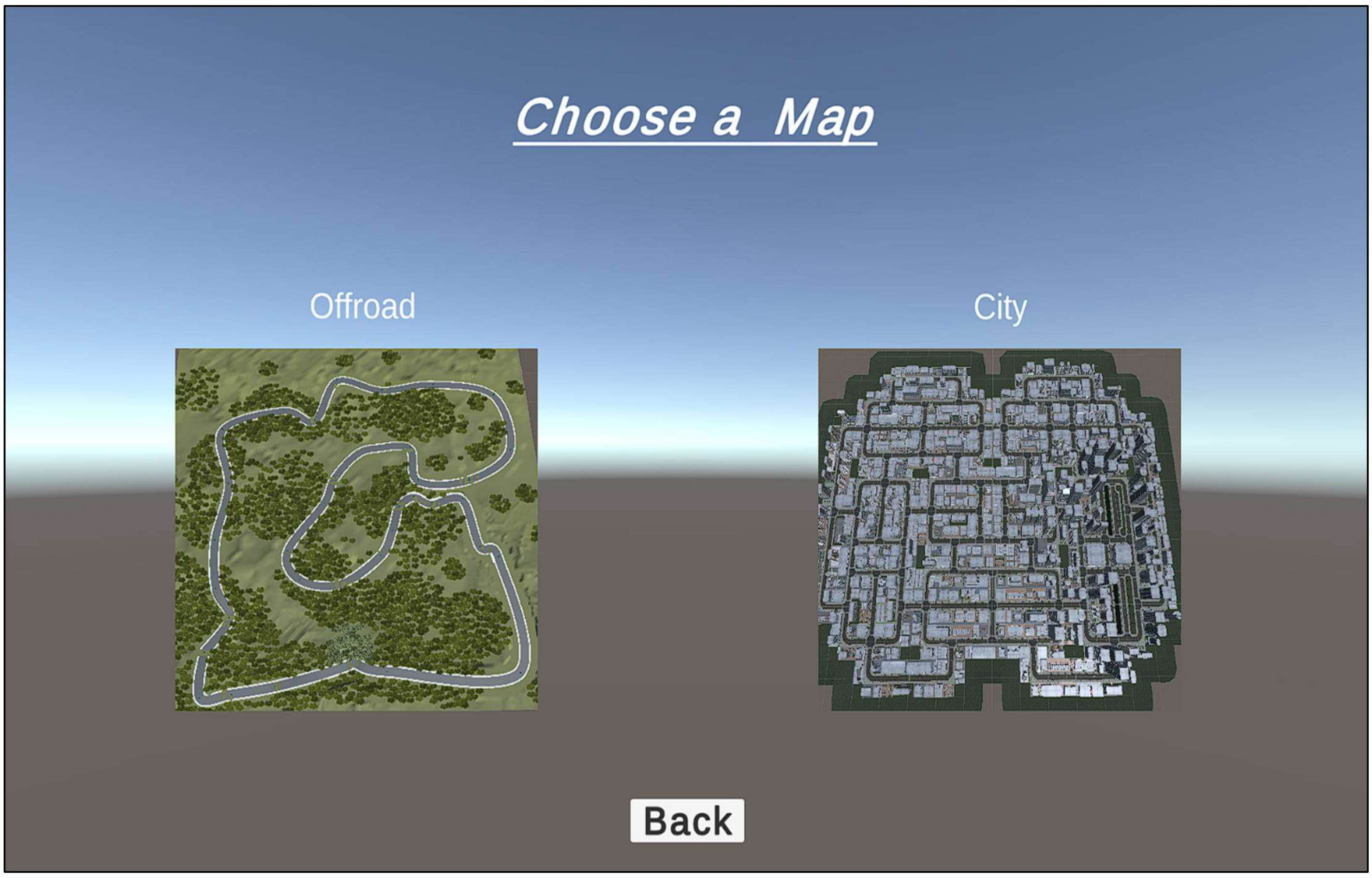


Fig 5.3 Maps

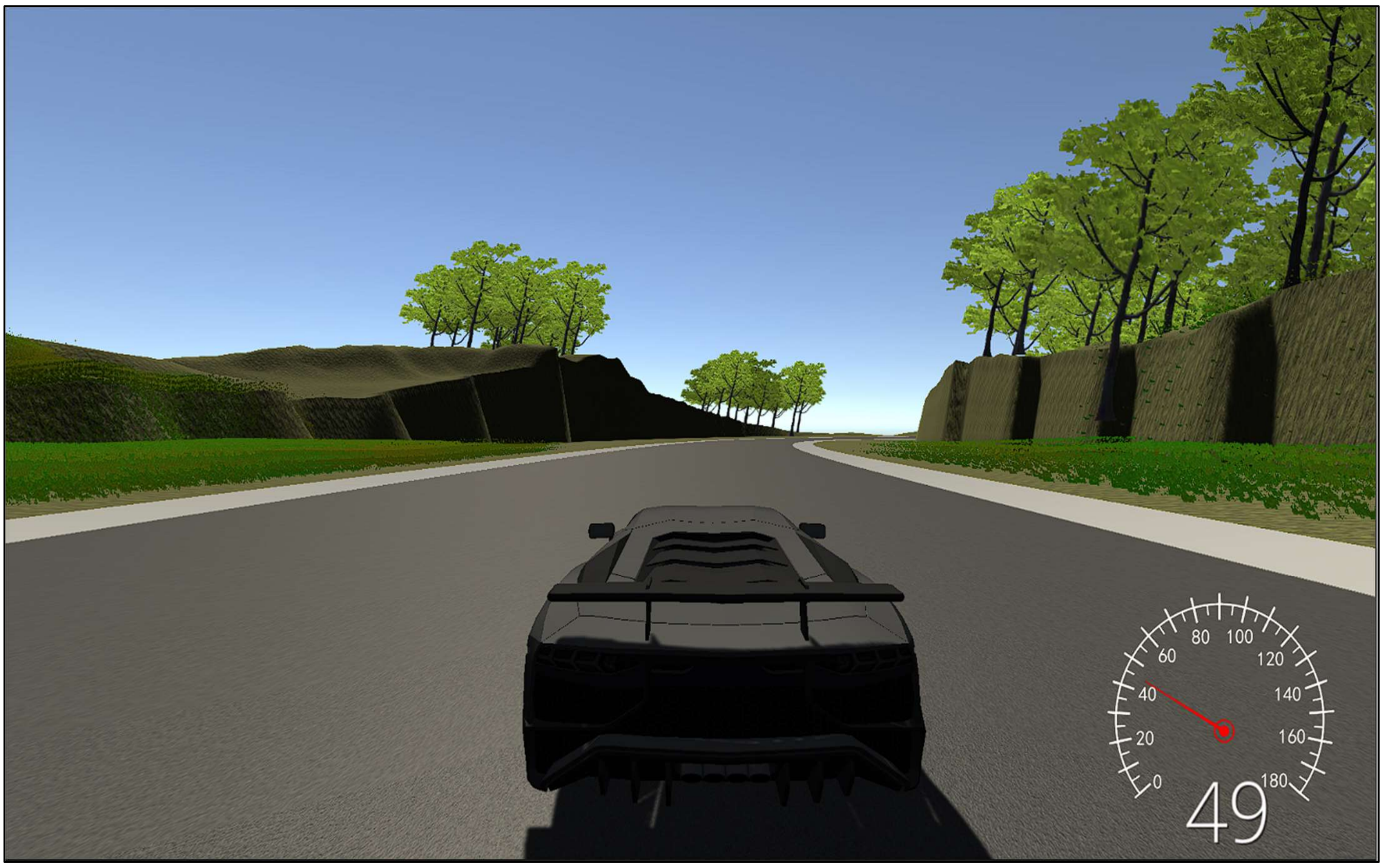


Fig 5.4 Off-Raod Map

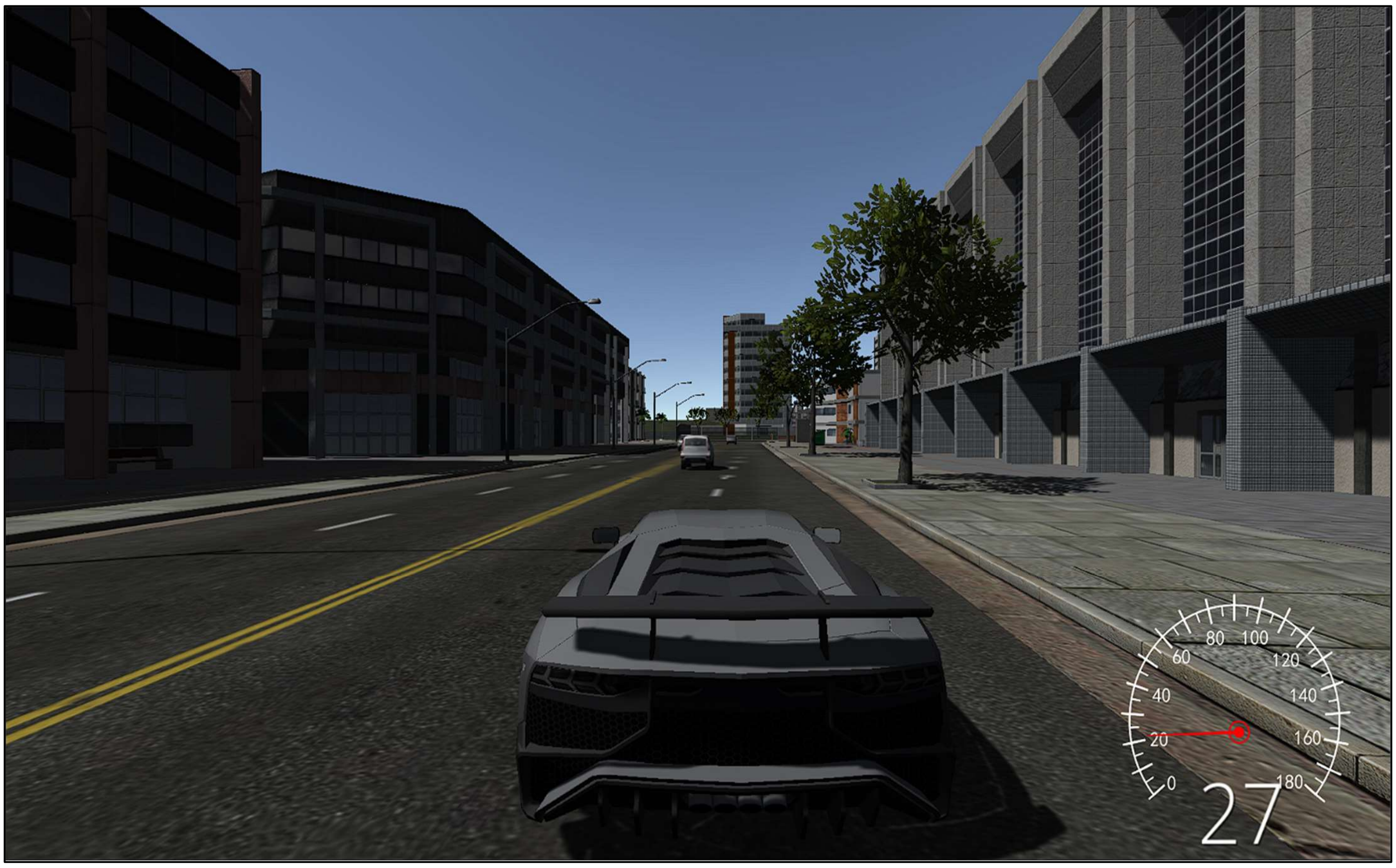


Fig 5.5 On-Rod Map

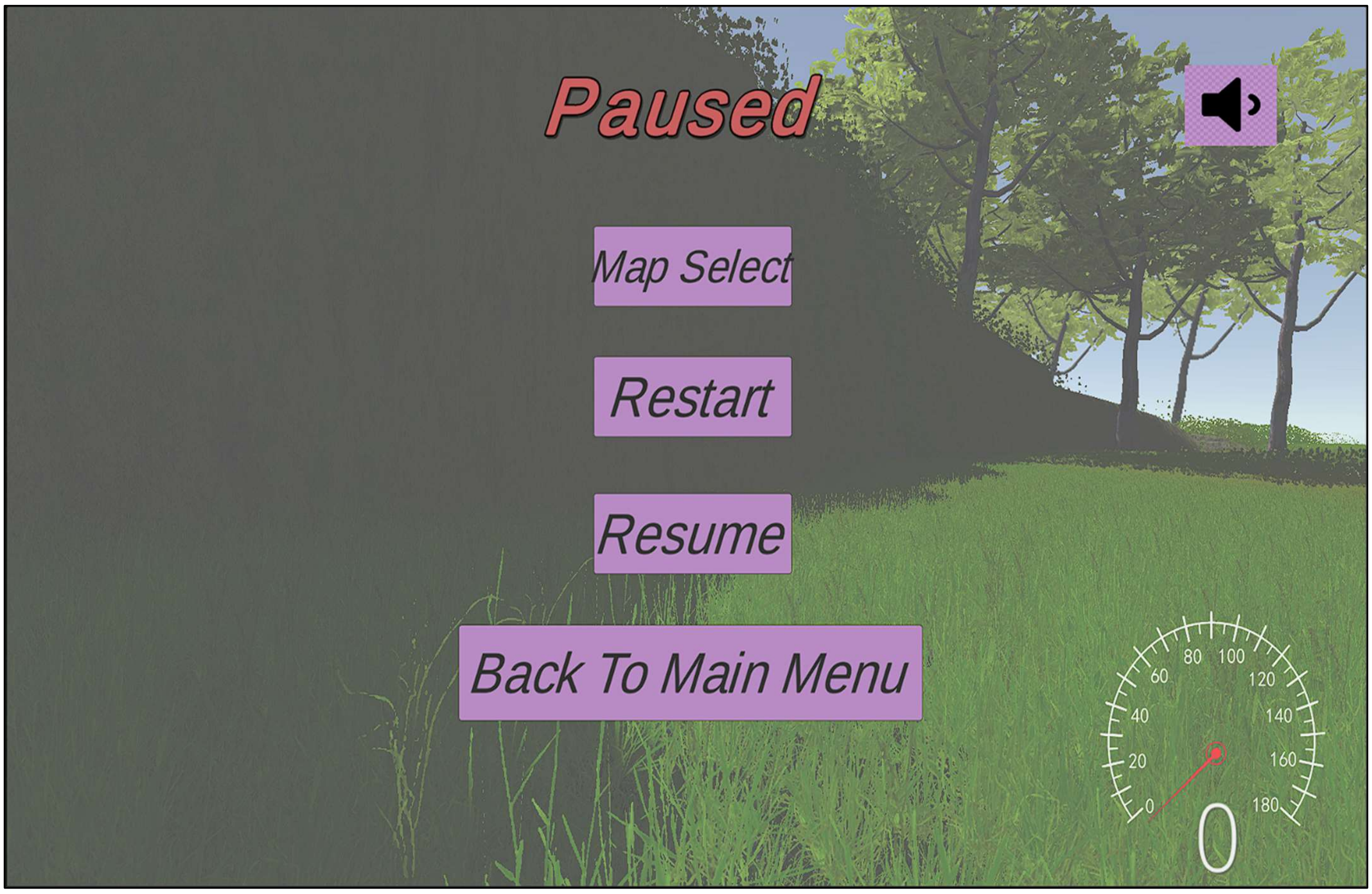


Fig 5.6 Pause Menu

CHAPTER 06

TESTING AND DEBUGGING

6. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover each and every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test addresses a specific testing requirement.

6.1 Black Box Testing

Black Box Testing is also known as behavioral, opaque-box, closed-box, specification-based or eye-to-eye testing.

It is a Software Testing method that analyzes the functionality of a software/application without knowing much about the internal structure/design of the item that is being tested and compares the input value with the output value.

The main focus of Black Box Testing is on the functionality of the system as a whole. The term 'Behavioral Testing' is also used for Black Box Testing.

Behavioral test design is slightly different from the black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged. Each testing method has its own advantages and disadvantages. There are some bugs that cannot be found using black box or white box technique alone.

A majority of the applications are tested using the Black Box method. We need to cover the majority of test cases so that most of the bugs will get discovered by the Black-Box method.

This testing occurs throughout the Software Development and Testing Life Cycle i.e in Unit, Integration, System, Acceptance, and Regression Testing stages.

6.2 White Box Testing

White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

The term "WhiteBox" was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "Black Box Testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

System Requirement

Minimum Requirement :-

Display Resolution: 1280 x 720

CPU: Intel I3 5th gen or Ryzen 3 3rd gen

Ram: 8 GB

Storage: 2 GB

Graphics: Integrated GPU

OS: Windows 7

DirectX Version: 10

Recommended Requirement :-

Display Resolution: 1920 x 1080 or higher

CPU: Intel I5 8th gen or Ryzen 5 5th gen

Ram: 12 GB or higher

Storage: 2 GB

Graphics: Nvidia GTX 1060 or AMD Radeon (TM) R5 M330 2GB or higher

OS: Windows 10 or higher

DirectX Version: 12

Test Cases

Sl. No.	Test Cases	Result	Description	Issues
1.	Test Case 1 (V 1.0)	PASS	First time we make this game which is our prototype.	Car is flipped and car coding doesn't work properly.
2.	Test Case 2 (V 1.1)	PASS	Car is changed and code is improved.	Starting window and other windows is not added.
3.	Test Case 3 (V 1.2)	PASS	We add buttons and some windows like starting window.	Graphics and some buttons are not working.
4.	Test Case 4 (V 1.3)	PASS	Add functionality to all buttons.	Credits are boring type.
5.	Test Case 5 (V 1.4)	PASS	Game comes for improve to look better.	Sometime game crashed.
6	Test Case 6 (V 1.5)	PASS	Noting to improve just fix the crashes problem.	Same as Test Case 5.
7.	Test Case 7 (V 1.6)	PASS	Add improve graphics and fix all previous problems.	In the offroad map car don't stop to collision with trees.

CHAPTER 07

CONCLUSION

7. CONCLUSION

Driving simulators are probably the most sophisticated application of computer-aided kinematic and dynamic simulations, as well as one of the biggest triumphs in their development. Similar to flight simulators, driving simulators place the driver in an artificial environment believed to be a valid substitute for one or more aspects of the actual driving experience. However, unlike flight simulators developed mainly for pilot training, driving simulators support much more than driver training. Advanced driving simulators today are used by engineers and researchers in vehicle design, intelligent highway design, and human factors studies such as driver behaviors under the influence of drugs, alcohol, and severe weather conditions. They provide a safe environment for testing in which controlled, repeated measurements can be undertaken cost-effectively. Researchers and engineers believe that the measurements obtained can help them predict equivalent measurements in the real world that lead to a better understanding of the complex driver–vehicle–roadway interaction in critical driving situations. The results of such studies will ultimately lead to reductions in the number of traffic-related deaths and injuries on the nation's highways.

From its appearance, a driving simulator—for example, the University of Iowa's National Advanced Driving Simulator (NADS) in Iowa City—consists of a dome on top of a Stewart platform mounted on longitudinal and lateral rails on the ground, as shown in Figure 8.38(a) (Center for Computer-Aided Design, National Advanced Driving Simulator 1994). The motion system, on which the dome is mounted, provides horizontal and longitudinal travel and rotation in either direction, so the driver feels acceleration, braking, and steering cues as if he or she were actually driving a real car, truck, or bus. Inside the dome is a vehicle cab (or a full-sized vehicle body) equipped electronically and mechanically with instrumentation specific to its make and model. The 360-degree visual displays offer traffic, road, and weather conditions; a high-fidelity audio subsystem completes the driving experience. The driver is immersed in sight, sound, and movement so real that impending crash scenarios can be convincingly presented without the driver being endangered.

The key technology in a driving simulator is the real-time vehicle dynamic simulation (Bae and Haug 1987). The driver's interaction with the system through the steering wheel and gas and brake pedals is captured by sensors and electronics. The signals are converted into inputs to the underlying vehicle dynamic model. The equations of motion of the vehicle dynamic model must be solved faster than in real time, so that the actuators underneath the Stewart platform can perform the pushes or pulls that mimic various driving conditions. In the meantime, the simulation results provide large excursions in longitudinal and lateral directions that are used to give acceleration and motion cues to the driver inside. Without the real-time dynamic simulation, the driving experience cannot claim to be truly physics-based.

There are several notable driving simulators and associated research groups around the world, including those in the United Kingdom, France, Sweden, and the United States, in addition to the NADS in Iowa City (Figures 8.39(a) through (d)). Car manufacturers have built their own simulators in the past 20 years, among them the pioneering Daimler-Benz Driving Simulator that was put into operation in 1985 and which has been enhanced since that time in many details and has been used in many different applications. Located in Berlin, this simulator went through a major modernization in 1994 (Figure 8.39(e)), including extension of the platform motion in a lateral direction to improve motion simulation quality. Ford Motor Company developed a motion-based driving simulator, called Virtual Test Track Experience (VIRTTEX) to test the reactions of sleepy drivers

The use of driving simulators has evolved from the first applications related to driver training or analysis of the effect of different substances such as alcohol or drugs, to more complex research studies focused on human factors and driver behavior during primary and secondary driving tasks. Nowadays, driving simulators are used not only in research studies, but also in the different stages of the design, development, and validation of in-vehicle systems, as well as in the design of infrastructure elements (Paul et al., 2009).

Regarding human factors, the studies in the driving simulators have allowed examination of the human–machine interface (HMI), i.e., the communication between the user and the vehicle or its technologies covering the following aspects:

- Workload analysis.
- Usability.
- Distraction due to secondary tasks.
- Reaction times (e.g., to avoid a collision when offering a warning).
- The effects of driver information on driving performance.
- The location of HMI elements on the dashboard.
- Selection of communication channel (acoustic, visual, haptic).

The evaluation of In-Vehicle Systems (IVIS) and Advanced Driver Assistance Systems (ADAS) have been broadly applied in several research and development projects, showing that driving simulator studies are, together with tests on test tracks and tests in real life, valid tools (Engen et al., 2009). In fact, dynamic driving simulators play an important role in the initial phases of development, since driver behavior and driving performance (lateral control, longitudinal control, interaction with other vehicles, etc.) provide early valuable results in a virtual environment.

Several European Projects have introduced experimentation with driving simulators to analyze new in-vehicle functionalities. For example, in the AIDE project, funded by the Sixth Framework Programme, the effect of the combination of warnings coming from different ADAS functions were tested (Paul et al., 2008). Thus, four ADAS were selected for this study, Frontal Collision Warning (FCW), Lane Departure Warning (LDW), Curve Speed Warning (CSW), and Blind Spot Detection (BSD), in order to analyze user reaction and possible conflicts when simultaneous ADAS warnings were presented

In this study the warning conditions were manipulated to create different levels of theoretical mental workload. Two strategies were considered (independent variable) in a conflict situation: simultaneous activation of warning signals for the driver and prioritization of warnings. Such critical situations can only be reproduced in high performance driving simulators, with the required levels of safety and repeatability.
Another example of the application of driving simulators to the design of new ADAS is the INTERACTIVE Project (2013), launched under the Seventh Framework Programme, where information, warning, and intervention strategies were developed, according to aspects such as: layer of driving task, level of assistance and automation, situation awareness, mental workload, sequence of interaction, etc.
part from the evaluation of the HMI and warning strategies conducted in driving simulators, the vehicle HIL simulations have proven an added value in several phases of the development process of ADAS, such as sensor verification, rapid control prototyping, model validation, functional level validation, fine-tuning of control algorithms, production sign-off tests, and preparation of test drives (Gietelink et al., 2006). This is possible thanks to the representative environment obtained, where test scenarios can be varied very easily in accurate and reproducible conditions.
Moreover, in the development of ADAS, different driving simulation platforms may be used with the combination of HIL and driver-in-the-loop (DIL), in order to create special testing scenarios that include high speed traffic flow, low-frictional load, etc., with high controllability and repeatability (Jianqiang et al., 2010). This allows to speed up the development process and to reduce the associated development costs.
Finally, HIL simulation can be used to support the development, testing, and verification of many functions and algorithms related to autonomous driving, by extending conventional HIL simulation to vehicle interaction with other vehicles in traffic and with a simulated surrounding environment sensed by simulated sensors. (Deng et al., 2008).
In this sense, there is no doubt that driving simulation is an essential and powerful tool in the design, development, and validation of current and future in-vehicle technologies, especially in the field of connected and automated road transport, where new challenges regarding user behavior, vehicle operation, complex scenarios, and innovative interaction capabilities are rapidly arising.