MEKELLE UNIVERSITY
ETHIOPIAN INSTITUTE OF TECHNOLOGY
MEKELLE

3D CAR RACING GAME BY DAF GAMING

` Developers
Dawit getachew--------------96248/08
Amanuel Kiros---------------96489/08
Fraol abebe -------------------96671/08

In particular fulfillments for the requirements of the semester project for 4th year students of computer stream

SUBMISSION DATE HAMLE 08 2018

ACKNOWLEDGEMENT

We express our sincere gratitude to our college Mekelle Institute of Engineering & Management, for providing us an opportunity to undertake and complete such an interesting project report.

We are very grateful to our respected faculty, Mr. Teklay who was our backbone and our guide throughout this project. Without his assistance we wouldn't have completed this project within a short period of time.

It would be really very unfair without the mention of our friends and families. The immense love and moral support they have given is truly unmeasurable.

PROJECT REPORT ON
"3D CAR RACING GAME"

Has successfully completed by
DAWIT GETACHEW (Roll no. - 96248/08)
AMANUEL KIROS (Roll no. - 96489/08)
FRAOL ABEBE(Roll no. - 96671/08)

Under the esteemed guidance of Mr. Teklay. Faculty of Computer Science & Engineering dept. Mekelle Institute of Engineering & Management, Mekelle

Index

Abstract

This project describes a case study, where we are focusing on developing a 3D racing car game, using a process based upon agile development; an evolutionary development method. The project will cover implementation of real-time graphics, physics engine, network support, as well as sound effects and background music. In the end, our case study will show that this development process was an appropriate choice for our game development project.

CHAPTER 1.Introduction

Developing software applications is a time-consuming process, and with time-consuming processes come high costs. During the last years, several software development methodologies, often known as agile software development, have become widely used by software developers to address this issue. Many different development methodologies can be more or less good, depending of the task and application type.

One of the software development methodologies is the evolutionary software method, which, as the name hints, takes on an evolutionary approach to the problem, and allows the project to evolve through different stages of the project. Our case study will show how well this evolutionary approach worked on our project where we choose to develop a 3D graphic computer game.

1.1 Over view

Games are produced through the software development process.[1] Games are developed as a creative outlet[2] and to generate profit.[3] Development is normally funded by a publisher.[4] Well-made games bring profit more readily.[5] However, it is important to estimate a

game's financial requirements,[6] such as development costs of individual features.[7] Failing to provide clear implications of game's expectations may result in exceeding allocated budget.[6] In fact, the majority of commercial games do not produce profit.[8][9][10] Most developers cannot afford changing development schedule and require estimating their capabilities with available resources before production.[11]

The game industry requires innovations, as publishers cannot profit from constant release of repetitive sequels and imitations.[12][*neutrality is disputed*] Every year new independent development companies open and some manage to develop hit titles. Similarly, many developers close down because they cannot find a publishing contract or their production is not profitable.[13] It is difficult to start a new company due to high initial investment required.[14] Nevertheless, growth of casual and mobile game market has allowed developers with smaller teams to enter the market. Once the companies become financially stable, they may expand to develop larger games.[13] Most developers start small and gradually expand their business.[14] A developer receiving profit from a successful title may store up capital

to expand and re-factor their company, as well as tolerate more failed deadlines.[15]

An average development budget for a multiplatform game is US$18-28M, with high-profile games often exceeding $40M.[16]

In the early era of home computers and video game consoles in the early 1980s, a single programmer could handle almost all the tasks of developing a game — programming, graphical design, sound effects, etc.[17][18][19] It could take as little as six weeks to develop a game.[18] However, the high user expectations and requirements[18] of modern commercial games far exceed the capabilities of a single developer and require the splitting of responsibilities.[20] A team of over a hundred people can be employed full-time for a single project.[19]

1.2 Problem statement of the Game Development(Design)

The reason we want to develop a game is because

- Is to entertain people

- DAF our team is very interseted in developing game.

Some requirements for the computer game were given from the beginning, such as:

3D graphics – The game must contain 3D models, and render these in the game. 3D environments were never a requirement, and platform games with 2D environment could still open up for 3D objects.

Impressive result – The game result must impress whoever plays the game. It should last long, and make the players come back and play it over and over again.

Graphical effects – To achieve an impressive result, we would need to add modern graphical effects, such as real-time rendered soft shadows, motion blur, and ambient occlusion.

Working with these requirements, we decided to use Unity 3D as our platform to develop our 3D game with. This decision was made with regard to that the platform had many in-built tools and provided a good framework for us to get started with the development as fast as possible. The fact that Unity 3D also used javascript as development language was also in consideration, since we wanted to learn this newly developed javascript language.

1.3 Objective

1.3.1General objective

The requirement for the game to contain 3D graphics introduced an interesting challenge for the project group, since all had none or little experience in 3D modelling. Spending time learning how to model proper 3D models

for our game was therefore necessary. During the research to find out what 3D modelling program to use, we found that we could use different studios to create models that we could later import to our game project.

In this project, we were left free to decide what type of game we wanted to develop. The suggestion was that a racing game would be suitable, since such a game usually do not depend on advanced assets, e.g. animated models. After some brainstorming, it was decided that a racing game should be developed.

1.3.2 Specific Objective

However, there were two different racing game ideas, which will be described below.

The first game was 2D racing game that has some features and uses a Studio called Visual Studio in it but these features don't make players entertained or satisfied .

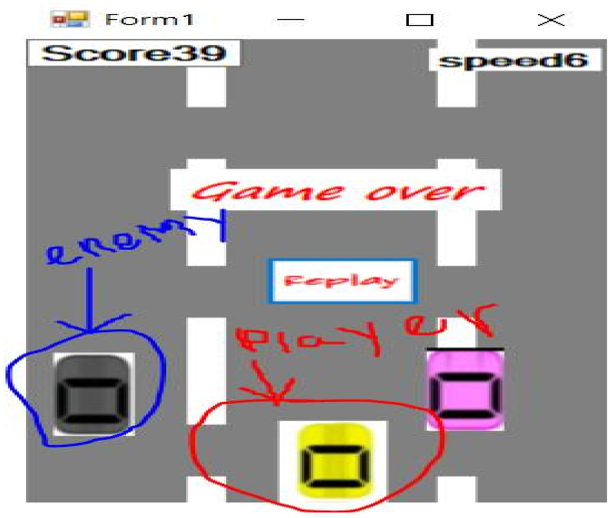
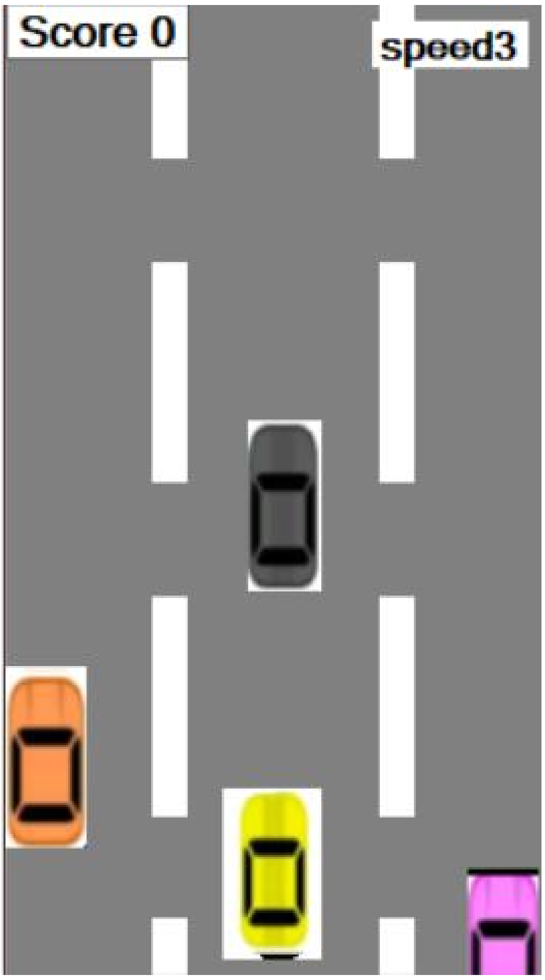


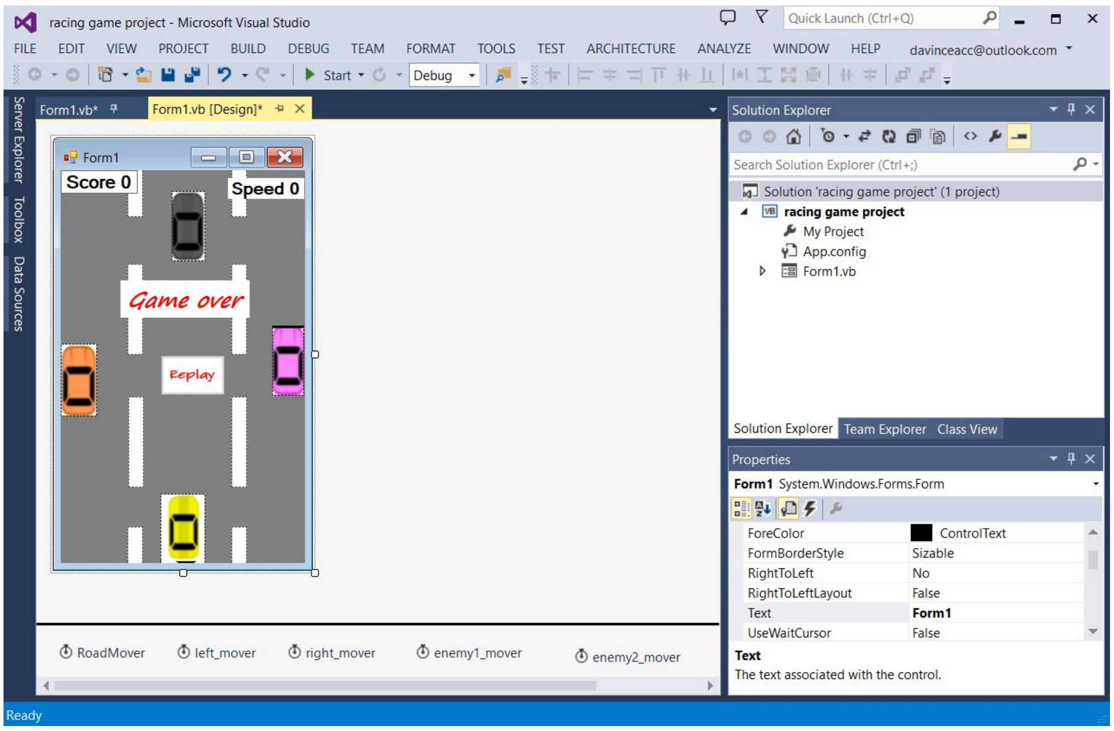
Fig 1.1

Because:-

* low Graphics Design

* Scripting algorithms Some captures of the game are displayed below





Form1 System.Windows.Forms.Form

ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	**Form1**
UseWaitCursor	False

Text
The text associated with the control.

As u see from the figures it is such a low graphics display and even the studio(Fig 1.3) is not having many tools ,effects and other additional features that must be added to make a game more interesting. But the squad(DAF) discussed this disadvantage and Decide to use a A more interactive and best tool called ***UNITY*** as pointed out in the introduction part.

What is **unity**

Unity is a world wide accepted game engine that is used to develop a game. In it it has many features that make a game more interactive and realistic and also it uses different software with it to :-code Scripts(Monodevelop,visualstudio,SublimeTxt),photoshop editing tools etc.…

This game engine is available online for free but with limited features .But any one can buy and use the premium version that has more features with it.

Our team then decided that this software tools is going to be the best choice with out any doubt.

---The version we used Unity2018.3.4f1

So Now we Starterted to develop 3D car racing game that is best in:-

Graphics

The game we are going to develop must be best in graphics mean it must display its characters,players,things objects clearly and and for this purpose the unity Software give us a path of finding the amazing graphics

Alogorithm usage

We need our game to have good algorithm to use on our virtual objects in there in unity since this objects are controlled by a user **Playercars** and some other are controlled by the computer it self and they are AI(Artifical)

Making real

This requirment   relies on the above 2 requirments and when we design this game it   must be Real as possible as we can that means the

--First we need to have Ground for the cars to move Ground,then objects on the ground,

LikeBanners,trees,containers,mountains,grass,sky and also The cars it self that is controlled by the player and making the enemy cars compete and try to destroy the player car.

-And when the car collides with the other objects the car losses its balances. And the game has laps to go and timers counting for the player cars and AI cars.

CHAPTER 2.Literature review

The idea that we proposed has been worked in different part of the world,in different ways and by different organizations.

There website is **:**

- **Need for speed**

(i. https://www.ea.com/games/needforspeed)

(ii. https://en.wikipedia.org/wiki/Need_for_Speed

Need for Speed (*NFS*) is a racing video game franchise published by Electronic Arts and currently developed by Ghost Games. The series centers around illicit street racing and in general tasks players to complete various types of races while evading the local law enforcement in police pursuits. The series released its first title, *The Need for Speed*, in 1994. The most recent game, *Need for Speed Payback*, was released in 2017.

The series has been overseen and had games developed by multiple notable teams over the years including EA Black Box andCriterion Games, the developers of *Burnout*.The franchise has been critically well received and is one of the most successful video game franchises of all time, selling over 150 million copies of games.Due to its strong sales, the franchise has expanded into other forms of media including a film adaptation and licensed Hot Wheels toys.

Almost all of the games in the *NFS* series employ the same fundamental rules and similar mechanics: the player controls a race car in a variety of races, the goal being to win the race. In the tournament/career mode, the player must win a series of races in order to unlock vehicles and tracks. Before each race, the player chooses a vehicle and has the option of selecting either an automatic or manual transmission. All games in the series have some form of multiplayer mode allowing players to race one another via a split screen, a LAN or the Internet. Since *Need for Speed: High Stakes*, the series has also integrated car body customization into gameplay.

Although the games share the same name, their tone and focus can vary significantly. For example, in some games the cars can suffer mechanical and visual damage, while in other games the cars cannot be damaged at all; in some games, the software simulates real-car behavior (physics), while in others there are more forgiving physics.

With the release of *Need for Speed: Underground*, the series shifted from racing sports cars on scenic point-to-point tracks to an import/tuner subculture involving street racing in an urban setting. To date, this theme has remained prevalent in most of the following games.

Need for Speed: Shift and its sequel took a simulator approach to racing, featuring closed-circuit racing on real tracks like the Nürburgring and the Laguna Seca, and fictional street circuits in cities like London and Chicago. The car lists include a combination of exotics, sports cars, and tuners in addition to special race cars.

- **Blur**

(i. https://www.gamespot.com/blur)

(ii. https://en.wikipedia.org/wiki/Blur_(video_game))

Blur (stylized as ***blur***) is an arcade racing video game released on May 2010 for Microsoft Windows, PlayStation 3 and Xbox 360. It was developed by Bizarre Creations and published by Activision in North America and Europe. The game features a racing style that incorporates real world cars and locales with arcade style handling and vehicular combat. *Blur* was the penultimate game developed by Bizarre Creations before they were shut down by Activision on February 18, 2011.

In *Blur*'s career mode, the player will encounter numerous characters and many licensed cars ranging from Dodge Vipers to Lotus Exiges to Ford Transit and vans fitted with F1 engines, all of which have full damage modeling and separate traits such as Acceleration, Speed, Drift, Grip and Stability. Some special car models have been designed by Bizarre Creations themselves. Albeit simplified, the tracks are also based on real-world environments, such as the Los Angeles river canals and several parts of London. Depending on the character(s) the player races against or tags along with in team races, they will have their own racing styles, power-up setups, match types, locales and cars. As the player reaches the podium in races, performs stunts and uses power-ups in certain ways, they will gain 'fan points'. These points help the player progress through the career, purchase more cars and parts and earn more fans for the user base. Also, during the career, players will encounter fan icons along the tracks. Driving through these will trigger short challenges (e.g. shooting another car with a certain weapon, or performing a long drift), which will reward the player with a fan points boost. During the career mode, each challenge features a final boss, which, once defeated will yield access to their specific mods (mods being upgrades that provide enhanced functionality to a standard powerup e.g. Khan's titanium shield), and customized cars. At the final boss challenge, all the bosses meet together for a final race.

● **GRAND THEFT AUTO SAN ANDERIAS (i. e www.rockstargames.com/gta)**

Grand Theft Auto is an action-adventure video game developed by DMA Design and published by BMG Interactive. It was first released in Europe and North America in October 1997 for MS-DOS and Microsoft Windows. It was later re-released on 12 December 1997in Europe and 30 June 1998 in North America for the PlayStation. It is the first instalment of the *Grand Theft Auto* series, to be followed by 1999's *Grand Theft Auto 2*. The series, which has led to five main entries and several special edition games over 16 years, has sold more than 150 million units as of September 2013.The story follows a group of criminals in three fictionalised versions of US cities as they perform bank robberies, assassinations, and other illegal activities for their respective crime syndicates.

The game was originally intended to be named *Race'n'Chase* and to be developed for the Commodore Amiga, starting in 1996.However, it was nearly cancelled due to production issues.

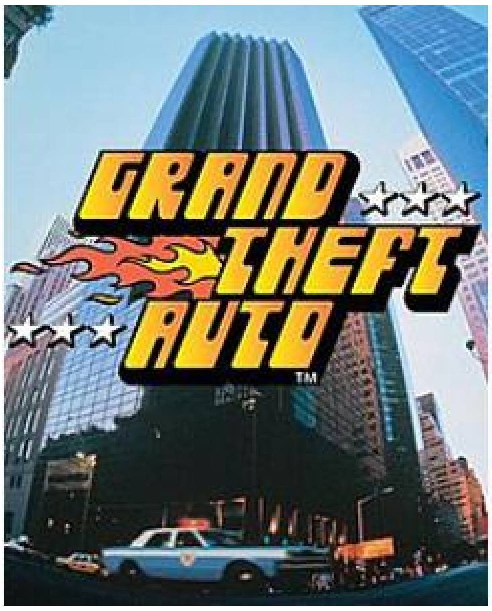
Its successor, *Grand Theft Auto 2*, was released in October 1999.

Grand Theft Auto is made up of six levels split between the three main cities. In each level, the player's ultimate objective is to reach a target number of points, which is typically achieved by performing tasks for the city's local crime syndicate. Each level is initiated at atelephone box and has its own unique set of tasks.[7] Successful completion of a mission rewards the player with points and opens the opportunity to attempt harder missions for higher rewards, while failure awards fewer points and may permanently seal off opportunities for more tasks. Completing missions also increases the player's "multiplier", which increases the points the player gets for doing other tasks. When the player amasses the target score (which starts at $1,000,000, but becomes higher in the later levels), the next chapter is unlocked.

There are eight playable characters in the game, four male and four female: Travis, Troy, Bubba, Kivlov, Ulrika, Katie, Divine, and Mikki (the PlayStation version only includes the four male characters, however). In actual gameplay, there is no real difference, since all player-characters wear the identical yellow jumper, although they do wear different coloured trousers and hair colours to each other and have the correct skin colours. Players may also name their character which, with the correct name, acts like a cheat code and alters gameplay.

The player is free to do whatever they want, but have limited lives upon doing so. The player can gain points by causing death and destruction amid the traffic in the city, or steal and sell cars for profit. To get to the large target money required to complete a level, players will usually opt to complete at least some missions to build up their multiplier. Some criminal acts have an inherent multiplier; for example, using a police car for running over people doubles the number of points received. If the player is arrested then their multiplier is halved. Unlike in later games in the series, the player can be killed, or "wasted", in one hit without body armour. If the player is wasted then they lose a life. In both cases the player loses their current equipment. If the player is wasted too many times, they must restart the level.

Even during missions there is still some freedom as most of the time the player is free to choose the route to take, but the destination is usually fixed. It was this level of freedom which set *Grand Theft Auto* apart from other action based computer games at the time. The PC releases of the game allowed networked multiplayer gameplay using the IPX protocol. Some places in the game have to be unlocked by completing missions.



There goal is to deliver enjoyable up to date game for users.Here we are going to to compact these different games a single interface and customizing them in order to satisfy user needs ,that users can get them all in one interface.

Chapter 3 methodology

Development Process

In a software development project, the resulting product is required to fulfil many different qualities. Examples of such quality requirements are:
robustness, availability, maintainability, dependability and usability. To meet such varying demands, it is important to base the work on a well prepared strategy. In software engineering, the term for such a strategy is commonly known as software process, which is built on one or several software process models.

What Is a Software Process Model?

A software process model is a theoretical philosophy that describes the best way of developing software. Based on one or several models, a software process is formed providing guidance on how to operate. A software process model may also be described as an abstract representation of a soft-ware process. The concept of the process model is similar to an abstract java class, which cannot be instantiated, but it can be implemented by another class, thus providing basic guidelines for that other class. A model may for example demand customer involvement, but it does not state exactly how. A process implementing that model should involve the customer in the process' activities, but is free to choose how.

There is not only one type of process model, but two. The first one is the most common, and described above. The second type of process model is called a process paradigm, which is a model even more general than an ordinary process model. Such a model does not hold any details on how the activities that lead to the completion of a project should be performed, but what it does hold is basic guidelines of how to develop software, and assumptions about what sort of project could benefit from implementing a particular model. With this in regard, one can conclude that a process paradigm provides a framework that may be adapted to form a process which suits a particular project.

There are three major process paradigms that are commonly used today in software engineering practice; the waterfall model, component-based software engineering and evolutionary development.

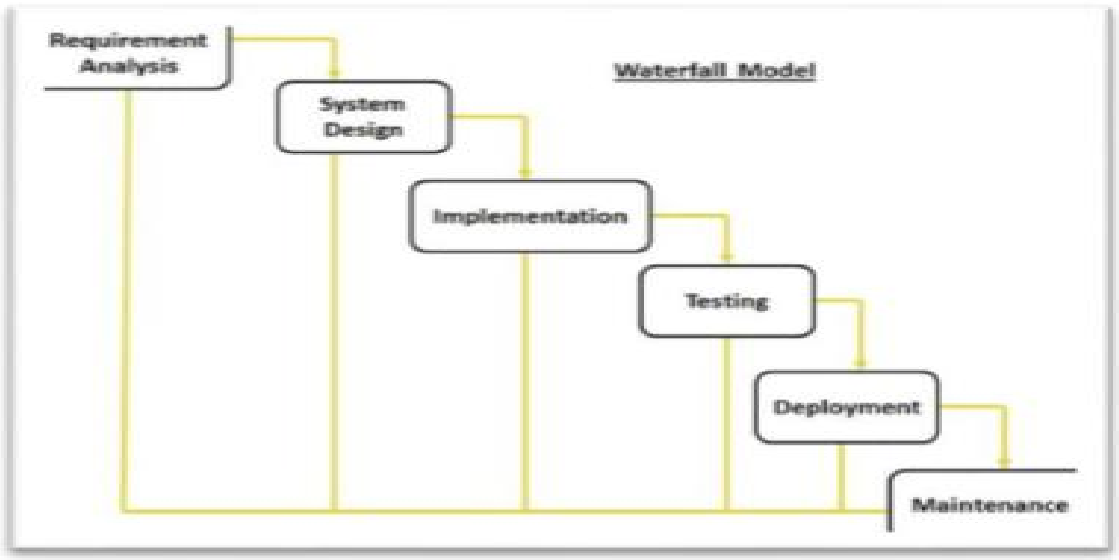
The Waterfall Model

The Waterfall Model is recommended for large and complex systems that have a long life-time2. Some systems which carry these attributes are also critical systems. This means that if a fault would occur, it may result in:

- Threat to human life (death or injury)
- Economic loss

- Environmental damage

It is believed that the waterfall model would be an appropriate choice when developing a critical system, since the model emphasizes on thoroughness. The basic concept is to take all the activities and treat them separately. One activity is always followed by another, in the same way water travels down some falls. This description becomes even more obvious when looking at a visualization of the model.



1. **Requirements definition:**

All requirements on the system are found by talking to system users. Example of requirements can be services, constraints and goals, such as “We want a webpage that color-blind people can enjoy”.

2. **System and software design:**

In this activity, the overall architecture of the system is established.

3. **Implementation and unit testing:**

The software is implemented in units which also are tested.

4. **Integration and system testing :**

The units are merged together into a complete system. Further testing is required.

5. **Operation and maintenance:**

The system is delivered to the customer and put into operation.

"**Bugs**" are almost always found, and therefore the system required bugfixing and maintenance.

In each of the activities described above, one or several documents are produced. The idea is not to start on a new activity until the documents belonging to the previous activity is signed off, but in practice, this is not how it is done. Instead most of the activities overlap and all the documents feed information to the different activities. Although these documents provide a good overlook.

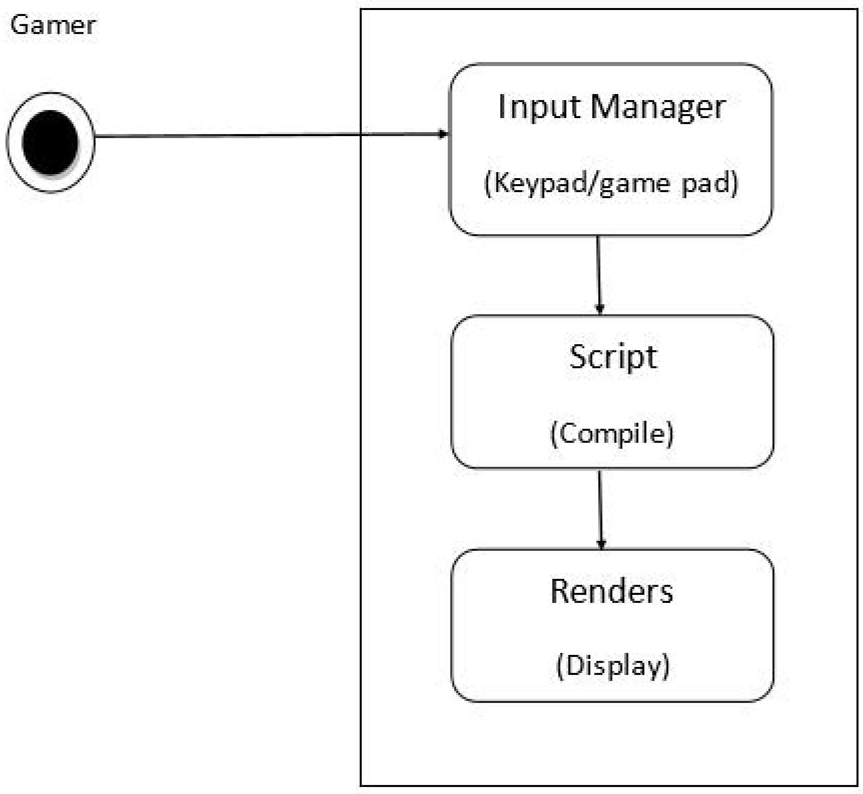
3.1 Game Engine

3.1.1 Game Framework

To save time in our development process, we choose to use Unity 3d, when developing our game. 2018.3.14f1 .Unity 3D . Unity is a cross-platform game engine developed by Unity Technologies and used to develop video games for PC, consoles, mobile devices and websites. First announced only for Mac OS, at Apple's Worldwide Developers Conference in 2005, it has since been extended to target more than fifteen platforms. It is now the default software development kit (SDK) for the Wii U. Unity Pro is available for a fee and Unity Personal has no fee; it is available for any use to individuals or companies with less than US$100,000 of annual gross revenue. On March 3, 2015 with the release of Unity 5.0, Unity Technologies made the complete engine available for free including all features, less source code and support. Unity is noted for an ability to target games to multiple platforms.

Five versions of Unity have been released. In 2006 at the 2006 WWDC trade show, Apple, Inc. named Unity as the runner up for its Best Use of Mac OS X Graphics category.

3.1.2 System Environment



Gamer can interact with system by giving input (press key to start game) to the system. System give those inputs to script, if any change occur (if the value is changed) this object send to renders to display the things (a character can change its place).

3.1.3 Quality Function Deployment of “CAR RACING”

Quality Function Deployment is a technique that translates the needs of the customer into technical requirements for software/game. It concentrates on maximizing customer satisfaction from the Software engineering process .With respect to our project the following requirements
are identified by a QFD.

o Normal Requirements.
o Expected Requirements.

Normal Requirements

Normal requirements consist of objectives and goals that are stated during the meeting with the actor/gamer/relevant people. Normal requirements of our project are:-

1. User friendly efficient and lucrative system.

2. Minimum maintenance cost (may be graphics definition).

3. Availability of expected requirements within the PC/mobile configuration.

4. Easy to operate.

5. They observe our game as this is build with professional manner.

6. The game with measured coding, professional thinking.

Expected Requirements

These requirements are implicit to the system and may be so fundamental that the actor/gamer/ relevant people does not explicitly state them .Their absence will be a cause for dissatisfaction.

1. Develop system within limited cost.

2. Maximum high definition.

3. Minimum hardware requirements which is relevant for this game.

4. Design whole system with efficient manner.

3.2 Specific Requirements

This section covers the project external requirements of our game and also indicates the user characteristics for this project.

3.2.1 External Interface Requirements of the Game

3.2.1.1 User Interfaces

Every game must has a menu so is can be user friendly enough and gamers can easily fulfill their need. Menu is also an important thing while creating the SRS document section. In this SRS document part; we have used the menu snapshots in the user manual part. These snapshots are based on the menu of the game.

3.2.1.2 Hardware Interfaces

“DAF Gaming ” is a windows   gaming application designed specifically for the windows
platform. Gaming application data is stored locally on the game engine elements. “DAF GAMING” has been developed for Android developed Version and all subsequent releases. In the future we released in the
android platform. Now the windows platform is graphically adaptable with a 2&3 dimensional
graphics library .

3.2.1.3 Software Interface

“DAF GAMING” has been developed using a series of game development tools.

Working tools and platform

Unity 3D
Mono Develop
VisualBasics

3.2.2 User Characteristics for the System

There is only one user at a time in this software and the user interacts with the game (system) in different manner.
So, Gamer is the only one who communicates with the system through playing the game. And this gamer can be any person. The primary requirement is that, the gamer must read the playing procedure provided by us (developers).

3.3 Analysis Model of Our Game Project

This section describes the Software Requirements Specification of our project by analyzing the proper models of requirement engineering.

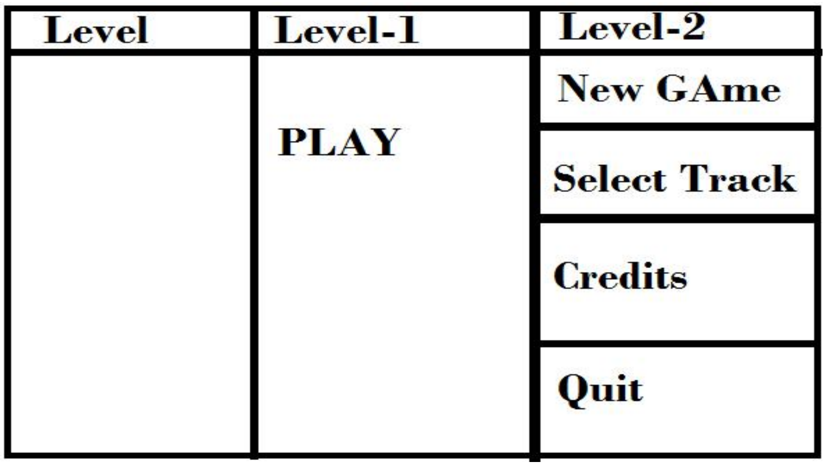
3.3.1 Scenario Based Model

This Model depicts how the user interacts with the system and the specific sequence of activities that occur as the software is used.

3.3.1.1 Use Case Scenario

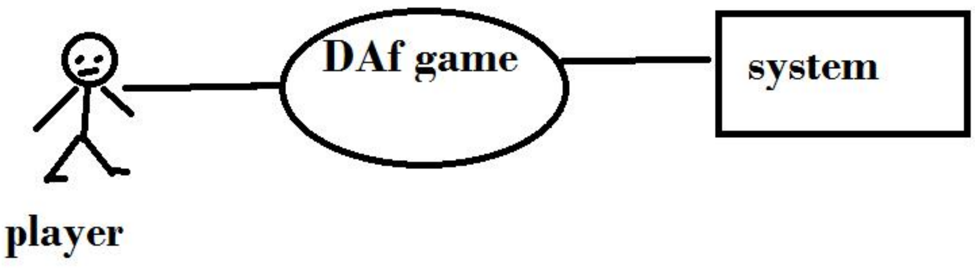
The following table summarizes the use cases of the system. We have created the use cases based on the UX view

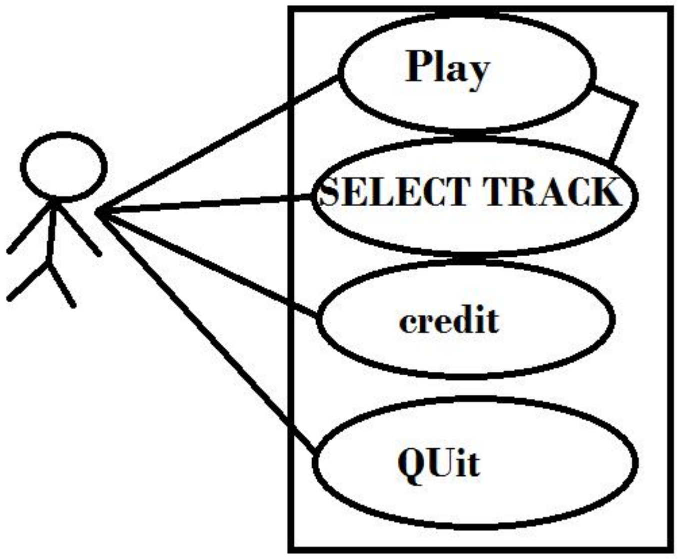
(mentioned in “User Story Part”) of the game. The swimlane diagram
connects UX with background programming which are the two important views of a game SRS



Level	Level-1	Level-2
	PLAY	New GAme
		Select Track
		Credits
		Quit

3.3.1.2 Use Case Diagram with Use Case





Play

i.

Use case: New Game

Primary Actors: Any one playing the game

Goal in context: To start a new game

Precondition:

1. System supports the game configuration
2. The file has been triggered to run and the game screen has appeared

Triggers: The player needs to start a new game

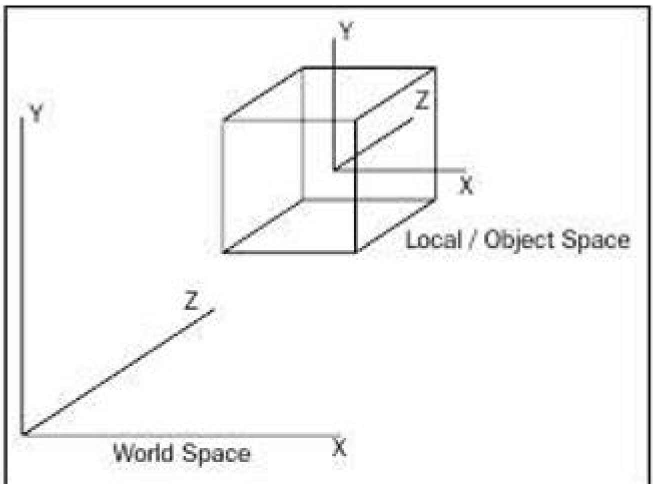
3.4 IMPLEMENTING THE DESIGN

3.4.1.Creating Terrain

3.4.1.1. Basic Concepts

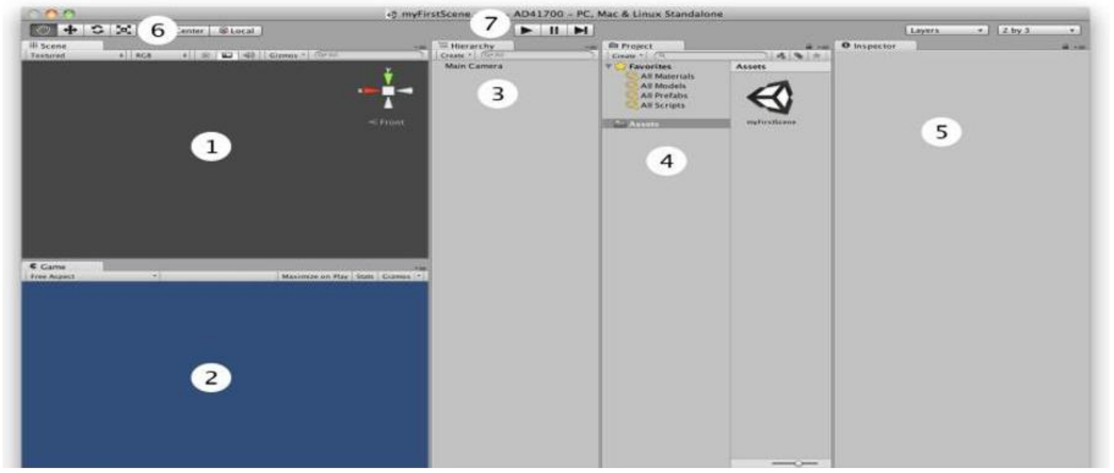
A. Coordinate Space

Three axes X, Y and Z – based on the Cartesian coordinate system (René Descartes, 1637)



B. User Interface

File > New Project
We don’t have to import any of the packages at this point, but do specify the save to location in the dialog window. We kept all our project related assets in this location to avoid missing files and broken links later in the game development process.
File > Save Scene as…



Window > Layouts > 2 by 3
User Interface Components (see screenshot on previous page):
1. Scene: This is where you will place any visual assets in your Unity environment. It will update in real-time when you are previewing the game. Note the manipulator on the top right; this allows you to switch between a number of standard views. We are currently in the perspective view (toggle between isometric (2D) and perspective (3D)). Although this doesn't matter too much, it allows us to view our scene with a vanishing point, which is the standard way Unity games will display.
2. Game: When you're not actively running the game, it will show a rendering of how the game will look, ignoring graphical effects that need to be computed at run-time, from the point of view of the main camera. When you're previewing the game, you'll be playing through this window. Since our scene is currently empty, all this window is showing is the background color.
3. Hierarchy: This lists all the objects in the currently loaded scene, and any children they may have. Children are objects

that can be thought of as subordinate to the parent object; wherever the top object moves, they'll follow, keeping the current offset they have to this object. This is an important concept for Unity beginners to understand; we'll cover it more in detail later and in the workshops.

4. Project/Assets view: This is a list of all custom assets for our game, including graphical assets, sound, scripts (more on these later), prefabs (pre- assembled game objects), and much more. Our current game is currently using only one empty scene (titled “myFirstScene”).

5. Inspector: Since we currently don't have any objects selected in the Hierarchy or the Project/Assets view, it's completely blank. The inspector allows us to look at and tweak individual settings of various game objects and assets, as well as adjust some global settings. The Inspector is contentsensitive and changes its parameters based on which game object/asset is selected. This is also a place to show you your project settings and preferences by choosing them from the Edit menu.

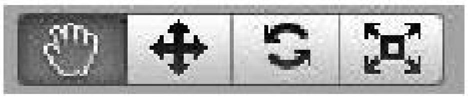
6. Graphical icons for moving the scene and its contents. The hand allows us to pan around the scene; when combined with other scene camera controls, Unity becomes very easy to navigate (see below). The icon on its right, which looks like four

arrows, allows you to move a selected object around. We call this transforming the object. The next icon allows for rotation of the object, and the final one allows for uniform scaling of the object.

7. Playback bar. This allows us to play, pause, and stop running our game in the Unity editor. This is the quickest and easiest way to test and tweak the game.

C.Navigating the scene window

The scene view is what allows us to look around and move the visual assets we import into Unity. It's how you'll assemble your levels and place important things like lighting, trigger zones, audio, and much more. Being able to control the camera is important if we want to do anything at all with it.



Hand Tool (shortcut Q): drag around in the scene to pan our view. Holding down alt+drag will rotate the view, Ctrl.+drag will allow you to zoom. It is important to remember that this doesn't move anything in the scene, just your point of view. **Translate Tool (shortcut X):** active selection tool, enables to drag an object's axis handles in order to reposition it. Rotate **Tool (shortcut E):** using handles to allow us to rotate an object around either of its axes.
Scale Tool (shortcut R): works the same as the previous two tools, allows scaling of an object.

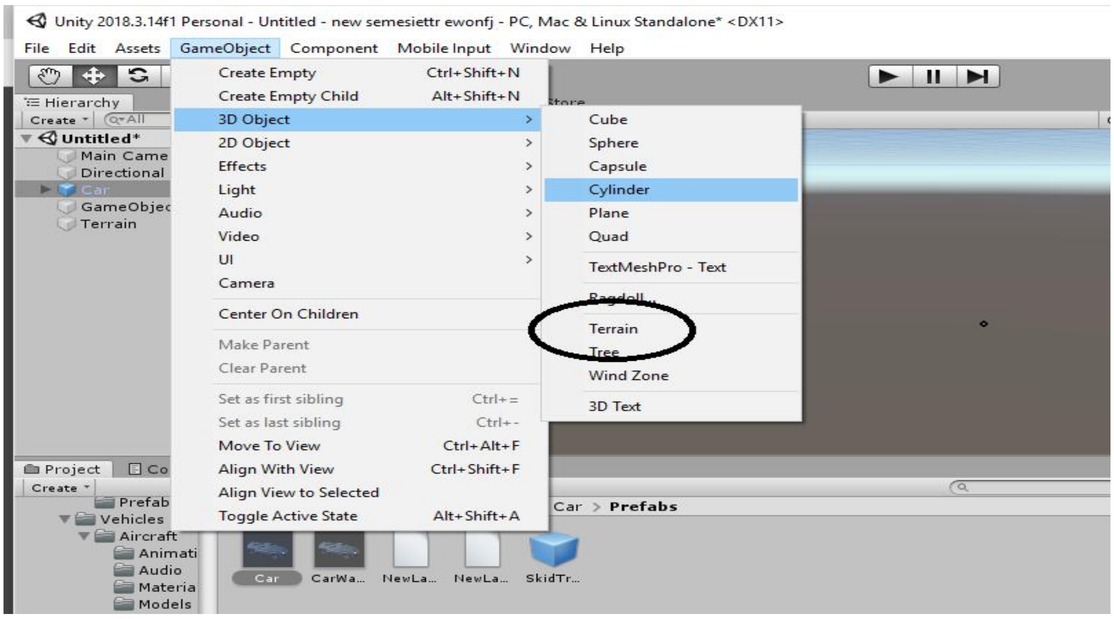
D. Our First Unity3D Scene

Now that we can look around the scene, let's learn a few ways we can place things in it. First we'll take a look at the basic game objects Unity can create without importing external assets. Unity 3 has geometric primitives (cubes, spheres, planes, capsule, etc.), lights, particle systems, cameras, and more that it can create without needing external assets. To access these, go to the top menu bar, select Game Object ->Create Other and make a choice. To begin with, try making a simple scene with a cube (functioning as a floor), a sphere, and a light. There's three different types of light - for now, a directional should work just fine, as light travels in rays with the direction of the arrows of the light, a good way to simulate the sun in Unity.
Start by creating a cube:
Game Object > Create Other > Cube
We can already use the Inspector window (after selecting the cube in the hierarchy) to modify its scale properties: Scale: X: 25, Y: 1, Z: 25 and to translate it Position: X: 0, Y: - 10, Z: 0

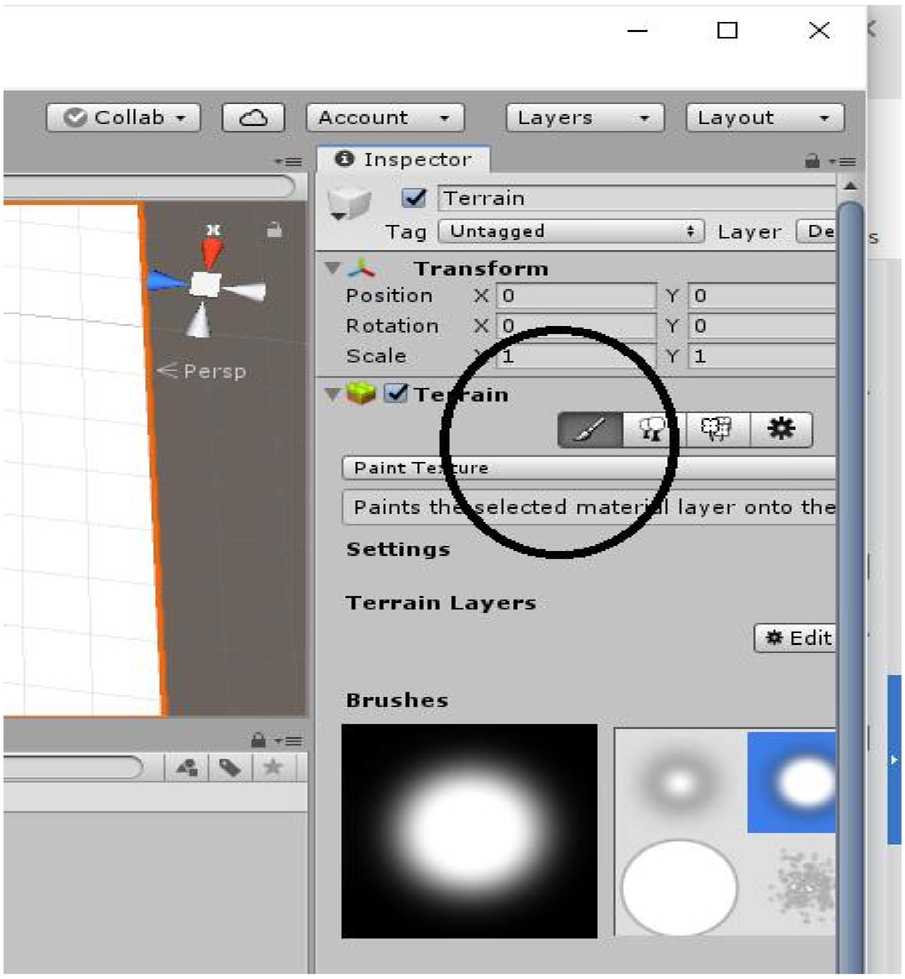
Now we just need to move our camera a little back and point it downward to see the newly created box (it looks more like a plane now, see screenshot on the following page – we changed the view in the Scene window to left-isometric)
Position: X: 0, Y: 0, Z: -25; Rotation: X: 20, Y: 0, Z: 0;

3.5.Enviroment Design and Track Design

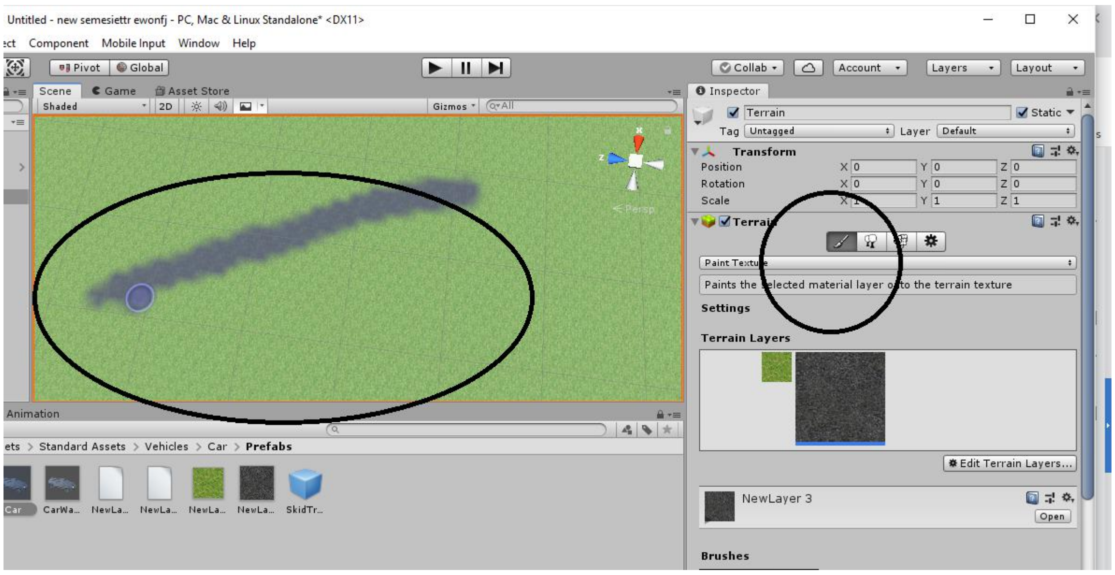
In the 3D Menu. A dialog window will appear where you can name the new environment object. The new object will be added to the hierarchy panel after clicking the Create Object button



Creating the environment object



Brush for drawing our environment



drawing the asphalt and the grass for the car game

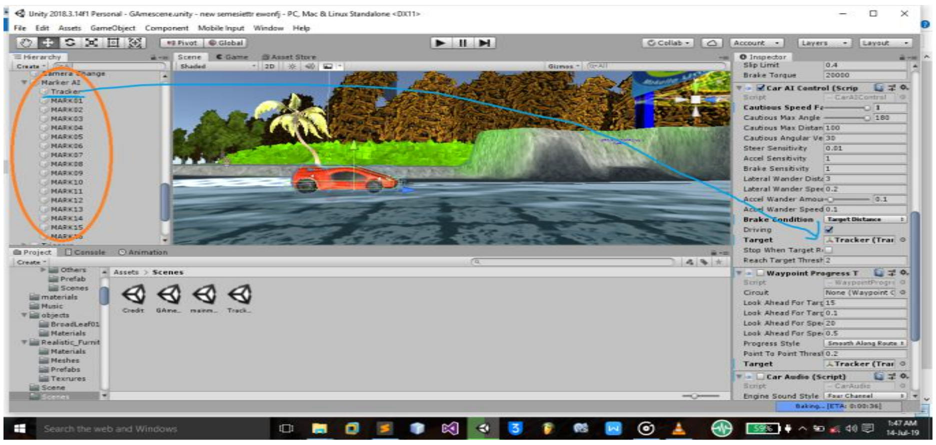
3.6.The AI in our game

3.6.1 The Other Cars and AI:

This section is where the fun begins. Up until this point we've been focusing on getting a car under the control of the player and also to set up a simple camera that will smoothly follow the car along a race track. But this is a "racing" game – we certainly need some other cars on the track that will race around the course trying to beat us. It will be the most difficult to date as the scripts in this section are more advanced than anything we've looked at so far, mainly because we need to create a method for opponent cars to drive themselves around the track. To begin the section we'll start by creating a car that drives itself around the race track. This car will be duplicated in the scene to create multiple opponent cars. Many different types of pathfinding problems exist. Unfortunately, no one solution is appropriate to every type of pathfinding problem. The solution depends on the specifics of the pathfinding requirements for any given game. For most racing game, the artificial intelligence (AI) for opponent characters is needed to find there path . In our paper, we focus on the car racing game, which can be seen as a kind of pathfinding problems.

In a car racing game, pathfinding is one of the most important problems. Poor pathfinding can make game characters seem very headless and artificial. Handling the problem of pathfinding effectively can go a long

way toward making a game more enjoyable and immersive for the player. The A* algorithm provides an effective solution to the problem of pathfinding and it also be one of the most popular algorithm used for the game's development . Assuming a path exists between the starting point and the ending point; then the A* algorithm guarantees to find the best path. Although the A* algorithm is efficient, it still can consume considerable CPU cycles, especially if you want to simulate pathfinding for a large number of game characters. The chief shortcoming of the A* algorithm in a racing game is that it can not solve the problem of random dynamics obstacles avoidance.



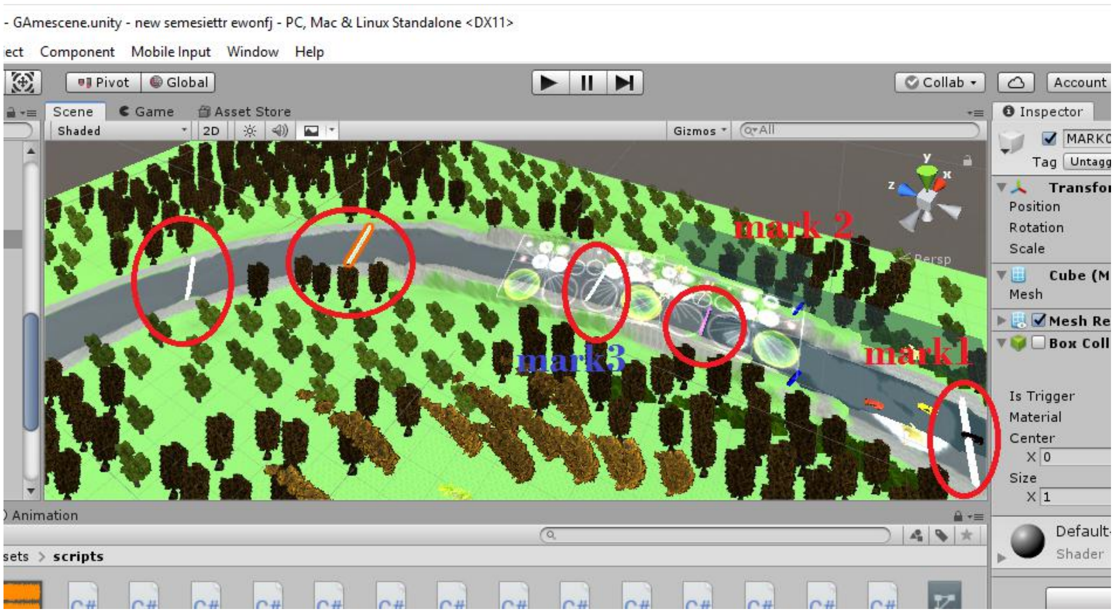
```
Camera Change
Marker AI
  Tracker
  MARK01
  MARK02
  MARK03
  MARK04
  MARK05
  MARK06
  MARK07
  MARK08
  MARK09
  MARK10
  MARK11
  MARK12
  MARK13
  MARK14
  MARK15
  MARK16
```

```
Slip Limit                0.4
Brake Torque              20000
Car AI Control (Script)
Script                    CarAIControl
Cautious Speed Fa         1
Cautious Max Angle        180
Cautious Max Distan       100
Cautious Angular Ve       30
Steer Sensitivity         0.01
Accel Sensitivity         1
Brake Sensitivity         1
Lateral Wander Dista      3
Lateral Wander Spee       0.2
Accel Wander Amou         0.1
Accel Wander Speed        0.1
Brake Condition           Target Distance
Driving                   ✓
Target                    Tracker (Tran
Stop When Target R
Reach Target Thresh       2
Waypoint Progress T
Script                    WaypointProgr
Circuit                   None (Waypoint C
Look Ahead For Targ       15
Look Ahead For Targ       0.1
Look Ahead For Spe        20
Look Ahead For Spe        0.5
Progress Style            Smooth Along Route
Point To Point Thresl     0.2
Target                    Tracker (Tran
Car Audio (Script)
Script                    CarAudio
Engine Sound Style        Four Channel
Baking... [ETA: 0:00:36]
```

3.7 Materials and Methods:

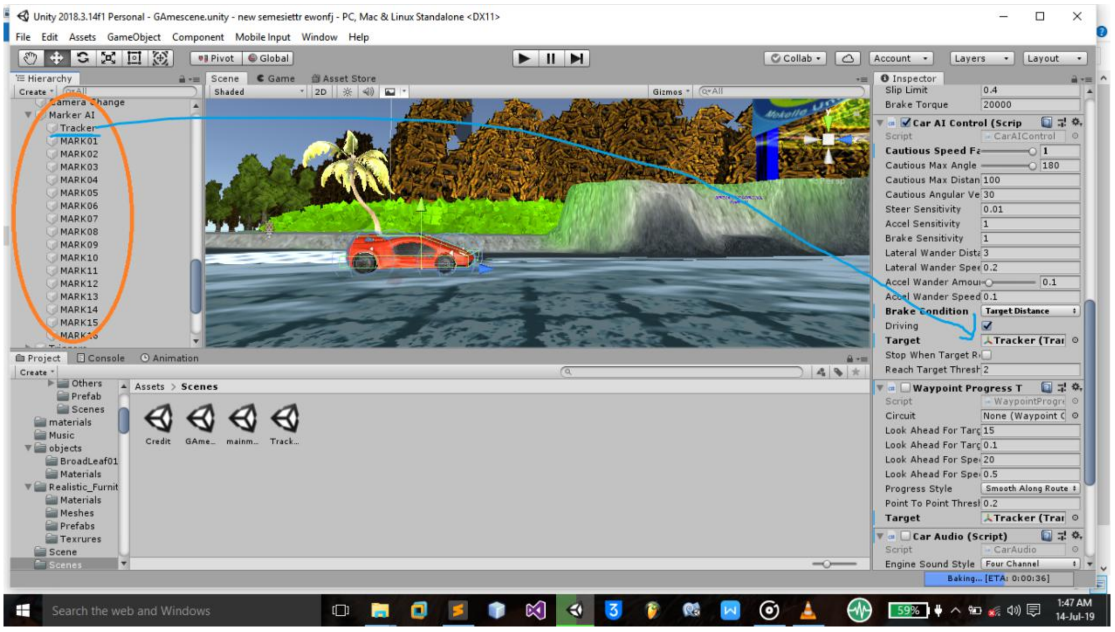
3.7.1 materials and methods used in AI racing game

For a car racing game, the most common artificial intelligence is waypoint navigation by carefully placing points (nodes) in the game environment to move the game- controlled characters between each point. The major drawback of this method is that these waypoints need to be manually setup, and it is a time consuming work. Meanwhile, these waypoints will depend upon the speedway track; different speedway track requires different configuration waypoints. In addition, the number of waypoints and the location of waypoints are also different due to human factors.modern game developing now days use A* algorithm since its very efficient and used for Dynamic path finding algorithm good way for random obstacles avoidance.we found it very hard to implement we use the waypoint algorithm.


- GAmescene.unity - new semesiettr ewonfj - PC, Mac & Linux Standalone <DX11>
ect Component Mobile Input Window Help
Pivot Global
Collab
Account
Scene Game Asset Store
Shaded 2D
Gizmos
Inspector
mark 2
mark3
mark1
Persp
Animation
sets > scripts

MARKO	
Tag	Untagg
Transfor	
Position	
Rotation	
Scale	
Cube (M	
Mesh	
Mesh Re	
Box Coll	
Is Trigger	
Material	
Center	X 0
Size	X 1
Default	
Shader	

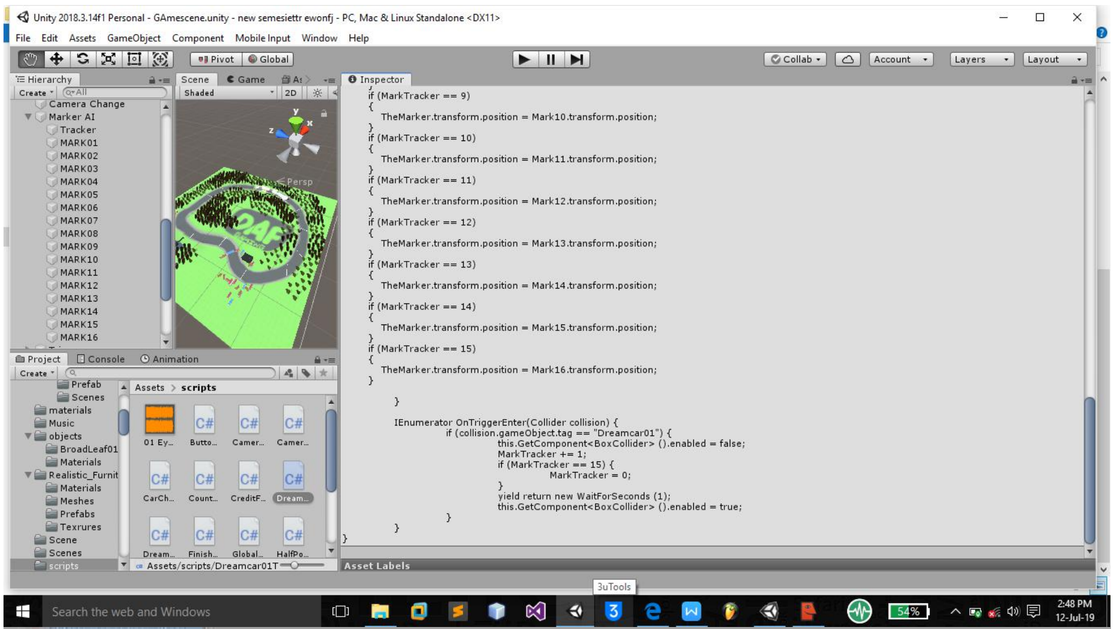
Fig 3 waypoint path for the AI car



Slip Limit	0.4
Brake Torque	20000
Car AI Control (Scrip	
Script	CarAIControl
Cautious Speed Fa	1
Cautious Max Angle	180
Cautious Max Distan	100
Cautious Angular Ve	30
Steer Sensitivity	0.01
Accel Sensitivity	1
Brake Sensitivity	1
Lateral Wander Dista	3
Lateral Wander Spee	0.2
Accel Wander Amou	0.1
Accel Wander Speed	0.1
Brake Condition	Target Distance
Driving	✓
Target	Tracker (Trai
Stop When Target R	☐
Reach Target Thresh	2
Waypoint Progress T	
Script	WaypointProgre
Circuit	None (Waypoint C
Look Ahead For Targ	15
Look Ahead For Targ	0.1
Look Ahead For Spe	20
Look Ahead For Spe	0.5
Progress Style	Smooth Along Route
Point To Point Thresh	0.2
Target	Tracker (Trai
Car Audio (Script)	
Script	CarAudio
Engine Sound Style	Four Channel

Baking... [ETA: 0:00:36]

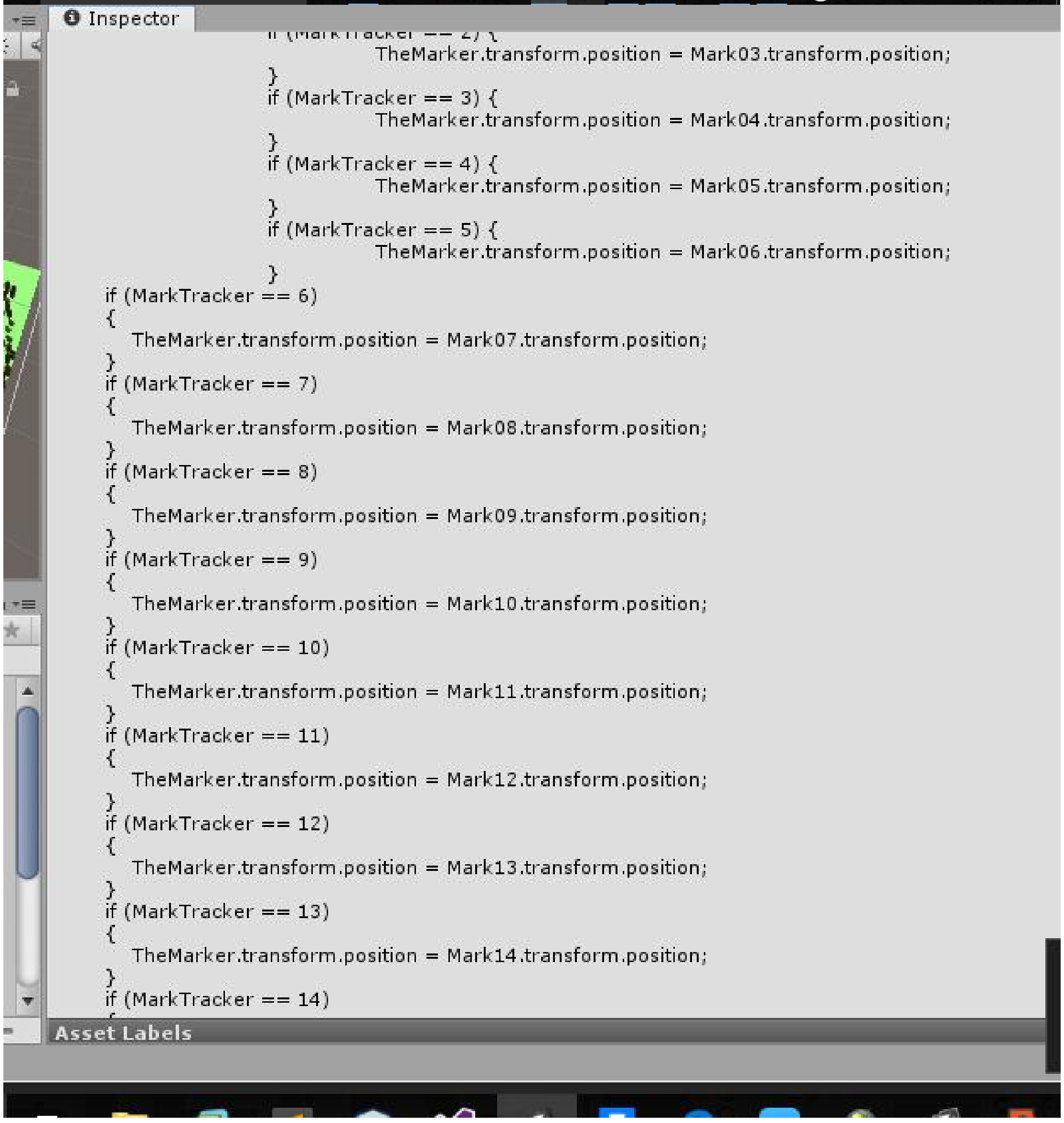
Attaching the path to the car target object


Unity 2018.3.14f1 Personal - GAmescene.unity - new semesiettr ewonfj - PC, Mac & Linux Standalone <DX11>

```
if (MarkTracker == 9)
{
  TheMarker.transform.position = Mark10.transform.position;
}
if (MarkTracker == 10)
{
  TheMarker.transform.position = Mark11.transform.position;
}
if (MarkTracker == 11)
{
  TheMarker.transform.position = Mark12.transform.position;
}
if (MarkTracker == 12)
{
  TheMarker.transform.position = Mark13.transform.position;
}
if (MarkTracker == 13)
{
  TheMarker.transform.position = Mark14.transform.position;
}
if (MarkTracker == 14)
{
  TheMarker.transform.position = Mark15.transform.position;
}
if (MarkTracker == 15)
{
  TheMarker.transform.position = Mark16.transform.position;
}

    }

    IEnumerator OnTriggerEnter(Collider collision) {
            if (collision.gameObject.tag == "Dreamcar01") {
                    this.GetComponent<BoxCollider> ().enabled = false;
                    MarkTracker += 1;
                    if (MarkTracker == 15) {
                            MarkTracker = 0;
                    }
                    yield return new WaitForSeconds (1);
                    this.GetComponent<BoxCollider> ().enabled = true;
            }
    }
}
```



```
                if (MarkTracker == 2) {
                        TheMarker.transform.position = Mark03.transform.position;
                }
                if (MarkTracker == 3) {
                        TheMarker.transform.position = Mark04.transform.position;
                }
                if (MarkTracker == 4) {
                        TheMarker.transform.position = Mark05.transform.position;
                }
                if (MarkTracker == 5) {
                        TheMarker.transform.position = Mark06.transform.position;
                }
if (MarkTracker == 6)
{
  TheMarker.transform.position = Mark07.transform.position;
}
if (MarkTracker == 7)
{
  TheMarker.transform.position = Mark08.transform.position;
}
if (MarkTracker == 8)
{
  TheMarker.transform.position = Mark09.transform.position;
}
if (MarkTracker == 9)
{
  TheMarker.transform.position = Mark10.transform.position;
}
if (MarkTracker == 10)
{
  TheMarker.transform.position = Mark11.transform.position;
}
if (MarkTracker == 11)
{
  TheMarker.transform.position = Mark12.transform.position;
}
if (MarkTracker == 12)
{
  TheMarker.transform.position = Mark13.transform.position;
}
if (MarkTracker == 13)
{
  TheMarker.transform.position = Mark14.transform.position;
}
if (MarkTracker == 14)
```

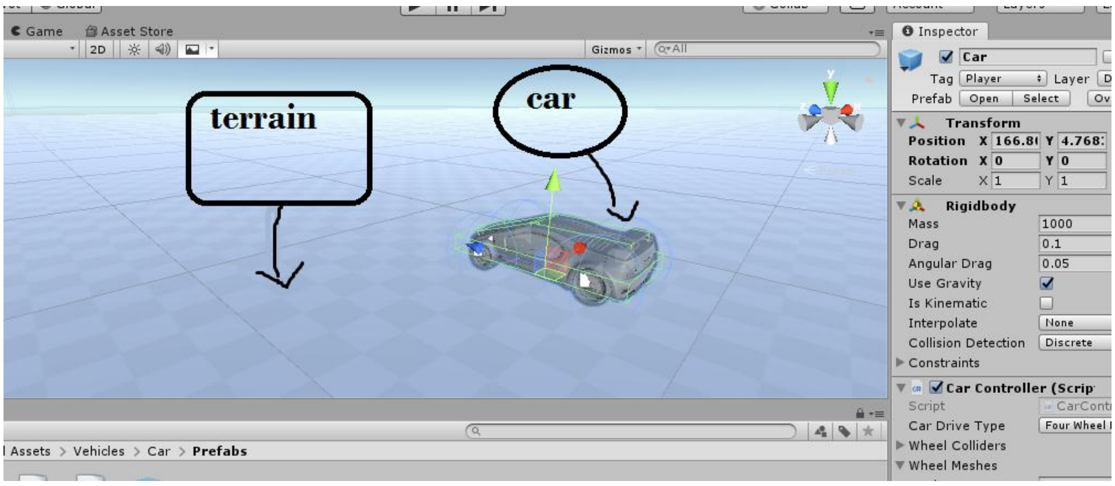
Fig3.7.1&2 this script show how the path change

3.8 Designing the Car Model:

For designing the car we can sketch out a blueprint which would fulfill all the requirements for the game.

First we need to get hold of some blueprints of the car that we want to model. But there is already built in assets in the unity full package so no need to sketch out for blue print.

Once we import the car from the package we drop it to our terrain environment so that it can be drived.



```
Car
Tag Player    Layer
Prefab  Open  Select
Transform
Position   X 166.8   Y 4.768
Rotation   X 0       Y 0
Scale      X 1       Y 1
Rigidbody
Mass                 1000
Drag                 0.1
Angular Drag         0.05
Use Gravity          ✓
Is Kinematic         ☐
Interpolate          None
Collision Detection  Discrete
Constraints
Car Controller (Scrip
Script               CarCont
Car Drive Type       Four Wheel
Wheel Colliders
Wheel Meshes
```

3.9The Game Camera:

The goal of this shorter section is to create and manage a camera to follow the player’s car as it races around the track. In a racing game, one of the most common camera views is from a third-person perspective; that is, behind and slightly above the player. This project will show you how to create one of these cameras and then to move it along in a fluid manner with the player car. In more advanced games you might wish to set up numerous cameras to capture the action from different angles.

This more complex camera work is done by enabling and disabling various cameras based on user input.

3.9.1. Basic Camera:

Let’s start by first simply getting the main camera positioned above and behind the player car. Open our previous scene’s Physics.unity scene. Camera Control Scripts
-> 1 Basic Follow Camera and name the file Camera. This scene contains all of the elements and scripts from the previous section of the tutorial. Now attach the Camera.cs script to the main camera. The Camera.cs script has a distance and height variable that we will be able to modify from the Inspector, but also a "target" variable that we will need to assign to the player’s car. Connect this variable to the player car object the same way we connected the wheel control variables to the wheel objects of the car object.

The Camerastable.cs script is very simple:

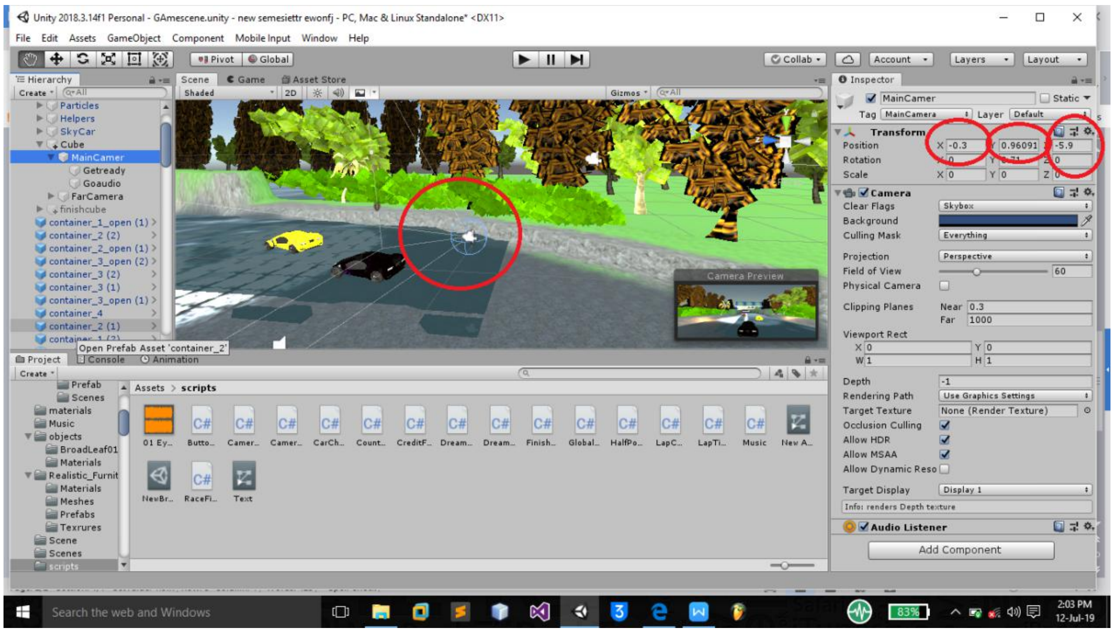
1:- We set up variables for height, distance and target.

2:- We use the function LateUpdate in this script to make sure that this particular script runs after any other scripts that are run during a frame.

3:- It gets the position and direction of the car to position itself.

4:- It places itself a specific distance and height behind the target using variables. In this instance we’ve specified in the script that behind means along the negative z-axis of the target. Then we move the camera upwards by height, so the camera looks down on the car.

5:- We rotate the camera to always look at the target. Run the scene and drive the player car to see the camera in action. Neat! But this is a very simple solution. What we’d really like to see is the camera reacting better to the motion of the player car. We need to create and use a camera script that smoothes out the motion of the camera as it follows the car.


Unity 2018.3.14f1 Personal - GAmescene.unity - new semesiettr ewonfj - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Mobile Input Window Help
Hierarchy
Scene Game Asset Store
Inspector
Project Console Animation
Assets > scripts
Add Component

MainCamer
Tag MainCamera Layer Default
Transform
Position X -0.3 Y 0.96091 Z -5.9
Rotation X 0 Z 0
Scale X 0 Y 0 Z 0
Camera
Clear Flags Skybox
Culling Mask Everything
Projection Perspective
Field of View 60
Clipping Planes Near 0.3 Far 1000
Viewport Rect X 0 Y 0 W 1 H 1
Depth -1
Rendering Path Use Graphics Settings
Target Texture None (Render Texture)
Target Display Display 1
Info: renders Depth texture
Audio Listener

Fig 1 implementing main camera in the car racing game with **x,y,z** angles

3.10 Coding

3.10.1basic codes

The codes are listed below

THE ENEMY CAR

```
//Daf Gaming inc
// This Script will track AI car

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Dreamcar01Track : MonoBehaviour {

    public GameObject TheMarker;
    public GameObject Mark01;
    public GameObject Mark02;
    public GameObject Mark03;
    public GameObject Mark04;
    public GameObject Mark05;
    public GameObject Mark06;
    public GameObject Mark07;
    public GameObject Mark08;
    public GameObject Mark09;
    public GameObject Mark10;
    public GameObject Mark11;
    public GameObject Mark12;
    public GameObject Mark13;
    public GameObject Mark14;
    public GameObject Mark15;
    public GameObject Mark16;

   public int MarkTracker;

   void Update () {
        if (MarkTracker == 0) {
             TheMarker.transform.position = Mark01.transform.position;
        }
        if (MarkTracker == 1) {
             TheMarker.transform.position = Mark02.transform.position;
        }
        if (MarkTracker == 2) {
             TheMarker.transform.position = Mark03.transform.position;
        }
        if (MarkTracker == 3) {
             TheMarker.transform.position = Mark04.transform.position;
}
        if (MarkTracker == 4) {
             TheMarker.transform.position = Mark05.transform.position;
        }
        if (MarkTracker == 5) {
             TheMarker.transform.position = Mark06.transform.position;
        }
            if (MarkTracker == 6)
            {
                  TheMarker.transform.position =
Mark07.transform.position;
            }
            if (MarkTracker == 7)
            {
```

```
                        TheMarker.transform.position =
Mark08.transform.position;
                }
                if (MarkTracker == 8)
                {
                        TheMarker.transform.position =
Mark09.transform.position;
                }
                if (MarkTracker == 9)
                {
                        TheMarker.transform.position =
Mark10.transform.position;
                }
                if (MarkTracker == 10)
                {
                        TheMarker.transform.position =
Mark11.transform.position;
                }
                if (MarkTracker == 11)
                {
                        TheMarker.transform.position =
Mark12.transform.position;
                }
                if (MarkTracker == 12)
                {
                        TheMarker.transform.position =
Mark13.transform.position;
                }
                if (MarkTracker == 13)
                {
                        TheMarker.transform.position =
Mark14.transform.position;
                }
                if (MarkTracker == 14)
                {
                        TheMarker.transform.position =
Mark15.transform.position;
                }
                if (MarkTracker == 15)
                {
                    TheMarker.transform.position = 
Mark16.transform.position;
            }

    }

    IEnumerator OnTriggerEnter(Collider collision) {
        if (collision.gameObject.tag == "Dreamcar01") {
            this.GetComponent<BoxCollider> ().enabled = false;
            MarkTracker += 1;
            if (MarkTracker == 15) {
                MarkTracker = 0;
            }
            yield return new WaitForSeconds (1);
            this.GetComponent<BoxCollider> ().enabled = true;
        }
    }
}
```

The camera stable

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

public class CameraStable : MonoBehaviour {

    public GameObject TheCar;
    public float CarX;
    public float CarY;
    public float CarZ;

    void Update () {
        CarX = TheCar.transform.eulerAngles.x;
        CarY = TheCar.transform.eulerAngles.y;
        CarZ = TheCar.transform.eulerAngles.z;
```

```
		transform.eulerAngles = new Vector3 (CarX - CarX, CarY, CarZ -
CarZ);

	}
}
```

The count down timer

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

public class Countdown : MonoBehaviour {

	public GameObject CountDown;
	public AudioSource GetReady;
	public AudioSource GoAudio;
	public GameObject LapTimer;
	public GameObject CarControl;
	public GameObject CarControl1;
	public GameObject CarControl2;
	public GameObject CarControl3;


	void Start () {
		StartCoroutine (CountStart ());
	}


	IEnumerator CountStart () {

		CarControl.SetActive (false);
		CarControl2.SetActive (false);
		CarControl1.SetActive (false);
```

```
        CarControl3.SetActive (false);
        yield return new WaitForSeconds (0.5f);
        CountDown.GetComponent<Text> ().text = "3";
        GetReady.Play ();
        CountDown.SetActive (true);
        yield return new WaitForSeconds (1);
        CountDown.SetActive (false);
        CountDown.GetComponent<Text> ().text = "2";
        GetReady.Play ();
        CountDown.SetActive (true);
        yield return new WaitForSeconds (1);
        CountDown.SetActive (false);
        CountDown.GetComponent<Text> ().text = "1";
        GetReady.Play ();
        CountDown.SetActive (true);
        yield return new WaitForSeconds (1);
        CountDown.SetActive (false);
        GoAudio.Play ();
        LapTimer.SetActive (true);
        CarControl.SetActive (true);
        CarControl2.SetActive (true);
        CarControl1.SetActive (true);
        CarControl3.SetActive (true);
    }

}
```

Lap time manager

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

public class LapTimeManager : MonoBehaviour {

    public static int MinuteCount;
    public static int SecondCount;
    public static float MilliCount;
    public static string MilliDisplay;

    public GameObject MinuteBox;
    public GameObject SecondBox;
    public GameObject MilliBox;

    void Update () {
        MilliCount += Time.deltaTime * 10;

        MilliDisplay = MilliCount.ToString ("F0");
        MilliBox.GetComponent<Text> ().text = "" + MilliDisplay;

        if (MilliCount >= 10) {
            MilliCount = 0;
            SecondCount += 1;
        }

        if (SecondCount <= 9) {
            SecondBox.GetComponent<Text> ().text = "0" + SecondCount
+ ".";
        } else {
            SecondBox.GetComponent<Text> ().text = "" + SecondCount
+ ".";
        }

        if (SecondCount >= 60) {
            SecondCount = 0;
            MinuteCount += 1;
        }

        if (MinuteCount <= 9) {
            MinuteBox.GetComponent<Text> ().text = "0" + MinuteCount
+ ":";
        } else {
            MinuteBox.GetComponent<Text> ().text = "" + MinuteCount
+ ":";
        }

    }
```

```
}


//LAP COMPLETE SCRIPT
```

The button

```
//DAF gaming
//This script will create your buttons
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ButtonOption : MonoBehaviour {

    public void PlayGame () {
        SceneManager.LoadScene (1);
    }

    public void TrackSelect () {
SceneManager.LoadScene (1);
    }

    public void MainMenu () {
        SceneManager.LoadScene (0);
    }

    //Below here are track selection buttons

    public void Track01 () {
        SceneManager.LoadScene (2);
    }

    public void credit () {
        SceneManager.LoadScene (3);
    }
}
```

CHAPTER 4.Result and Discussion

After we go through these steps we were very intersted to see the result infact we were seeing the results as we go from steps and steps. This game development stages are passed through many steps and results are observed in each step . we want the players to control his car and protect his car from being collide with other objects   the also the users is able to control the cars using keys on the key-board.And we are also able to control the AI cars by using nodes and we succeded .

CHAPTER 5. Recommendation and Future work

Any game developers that want to develop a game need to study programming languages,game engines,And others software tools

Make sure they choose appropriate developing methodology and they can lead to the end.

As ***Future work*** we will create a website like google play and we will upload games on them an then users are can download and play games on their pc.

Reference

1. Bethke 2003.p-4
2. Bethke 2003.p-7
3. Bethke 2003.p-14
4. Bates 2004.p-239
5. Bethke 2003.p-12
6. Bethke 2003.p-17
7. Bethke 2003.p-18-19
8. Bethke 2003, p. 3.
9. **Irwin, Mary Jane** (November 20, 2008). "Indie Game Developers Rise Up". *Forbes*. Retrieved January 10, 2011.
10. Bethke 2003.p-17-18
11. Bethke 2003.p-18
12. Jump up to:[a] [b] Moore & Novak 2010.p-19
13. Jump up to:[a] [b] [c] Moore & Novak 2010.p-
14. Jump up to:[a] [b] [c] Moore & Novak 2010.p-37
15. Moore & Novak 2010.p-18
16. **Crossley, Rob** (January 11, 2010). "Study: Average dev costs as high as $28m". Archived from the original on January 13, 2010. Retrieved October 17, 2010.
17. Adams & Rollings 2006.p-13
18. Chandler 2009.p-xxi
19. Reimer, Jeremy (November 7, 2005). "Cross-platform game development and the next generation of consoles — Introduction". Retrieved October 17, 2010.
20. Moore & Novak 2010.p-5