Department of Computer Science & Engineering

SOUTHEAST UNIVERSITY

CSE400: Research Methodology

Mad Dash Car Crash

A dissertation submitted to the Southeast University in partial fulfillment of the requirements for the degree of B. Sc. in Computer Science & Engineering

Submitted by

Md Abdul Kader
ID: 2009100000014

Supervised by

Monirul Hasan
Lecturer and Coordinator,
Department of CSE
Southeast University

August, 2015

Letter of Transmittal

September 5, 2015

The Chairman,
Department of Computer Science & Engineering,
Southeast University,
Banani, Dhaka.

Through: Supervisor, Monirul Hasan

Subject: Submission of Research Paper

Sir,
I am pleased to submit here my Research Paper of Mad Dash Car Crash. It was a great pleasure to work under your guideline. I have tried to work hard and accomplish your desire. I am grateful to my supervisor for continuous guideline to complete the work.

Thank you.

Sincerely yours,	Supervisor:
________________	______________
Md Abdul Kader	Monirul Hasan
ID: 2009100000014	Lecturer and Coordinator,
Batch: 22 Program: CSE	Department of CSE
	Southeast University

Certificate

This is to certify that the research paper titled 'Mad Dash Car Crash' is the bona-fide record of research work done by Md. Abdul Kader for the partial fulfillment of the requirements for B.Sc. in Computer Science & Engineering (CSE) from Southeast University.

This paper was carried out under my supervision and is record of the bona-fide work carried out successfully.

Author:	**Approved by the Supervisor:**
Md Abdul Kader	Monirul Hasan
ID: 2009100000014	Lecturer and Coordinator,
Batch: 22 Program: CSE	Department of CSE
	Southeast University

Abstract

Mad Dash Car Crash is a 3D car race fighting video games build on C++ platform. It is a cross platform video games with a low graphics card requirement. The game has modern car racing features with additional car fighting features. Player can choose and upgrade their car in a attractive car choosing menu. No extra game engine has been used in the development of the game. The game is fully built in scratch and help with some fine tools like OpenGL libraries and Blender Model creator. Most attractive part of the game is the applied physics in the car that bring the car like real simulation.

Acknowledgements

I would like to express my special thanks of gratitude to my supervisor Monirul Hasan, Lecturer and Coordinator, Dept. of CSE for guiding me throughout the research CSE: 400. Research itself helped me quite a lot in building up the fundamental knowledge of the field. I would like to thank the Department of CSE for allowing me to work on this program. I would also like to thank my parents who helped me a lot in finishing this research within the limited time.

Contents

List of Figures

Chapter 1

Introduction

Mad Dash Car Crash is a 3D genre car racing tournament video games. With having traditional car racing tournament feature, the game has its own car fighting mode that will give the gamer flavour of a fighting games. The game has also on playing point collection features like power-ups games. We can choose or buy different cars on the base of power ups points gathered. Also we can upgrade a car with different amination and filled with grenades that can be used during the race to demolish other racers car. Artificial intelligence applied on enemy car give real racing environment on the game. The game is build on C++ platform with no additional game engines included. It support cross-platform, or multi-platform systems and run on as many as almost all existing platforms. It is a third person perspective game, but user can switch to first person mode while playing the game. Having both perspective features gamer will able to feel the real car simulation flavour from first person perspective mode and fighting effect with the third person perspective mode. The game plot has been created on a story. This story will help the readers better understand of the game rules and features.

Near Sunaristo there is an Island name 'MadLand'. Its also named as 'Prison Of MadLand'. One side of the Island is surrounded by the lake full of large deadly crocodiles. On the other side of the Island is covered by a dark forest with fiercest *creatures beasts.*

The beasts of the forest are dominated by the evil forces. Elder citizens of the Island are all long gone. The beasts tooks the weakest first. They need to sacrifice those body for the evil force. There is only gate to get rid of this Island. The gate is guarded with five evil force. No one has ever returned alive those who tried to get out off this Island.

A Mad King name 'Rafa' is the ruler of this land. But he is well known as the 'Mad King'. He worship the Evil God 'Arikonus' for black magic and control a force of evil race. He has also control over the natural force of water and fire. He sacrifice his only son to the Evil God to gain this power.

The Mad King has a racing car factory. The peoples of MadLand forcibly work in his car factory. He build this car factory to feed his evil force and get resource for his worship and make strength to become a Dark Lord. The Mad King is so much ruthless that those who go against his command, he put them in one of his testing car from factory without any breaks and force them in a race and lose the evil forces behind them. Somehow those who managed to save from car, the evil force rips them apart and take their sole.

Once he decided to arrange a car race competition every year in the MadLand. The one who will win the race can get out of this MadLand with huge amount of precious jewelries and golds. But the path of the race is deadly by itself. There is only one rule to survive in the race and win freedom is to kill all other racers. For the last 10 years no one has ever succeeded alive till the end. But the race coverage become popular as planned all over the world. King named the race 'Mad Dash Car Crash'.

The race begins with a minimum number of 8 racers. Every racers can choose and decorate his racing cars with the rules and availability of equipment with level and experience. The race will held constantly every day till the last racers alive. Every single day racers have to finish up 5 laps through the tracks. Everyday tracks changed and racers get new levels with new armours. Points will be distributed amongst the racers according to their health and position every time after single day race finished. Racers can acquire points in many other ways from the race. Damaging other cars will gather most of the points. By destroying a racers car will double his armour in the car. A desperate scramble attempt or Mad Dash is the key to win the race. Every the race will begin with the last day left alive racers. Each level up or day change will make much more harder to survive. Those who catch the first opportunity to kill the first blood will be strongest first. But there is always will be less strong for the three racers combination. No matter how stronger or multiplied stronger a racers is three racers combination will always took him down.

By following in the next chapter we will learn some of the racing car games and car fighting games description as Background Study. Specifically, I try to introduce the reader if they don‘t know about racing games or car fighting games, what is racing games how many types and varieties in racing games. Gives some understand of real car racing tournaments all over the world and kind or playing rules of them. On following I discussed about some famous 3D car racing games. Gives some list of car racing games and some short description about them. After on some car fighting famous games and their playing way has been discussed in this section. Also need of game engine in the games and physics for developing has been discussed in this chapter. Overall I tried to discuss about what sort of knowledge need to develop and understand for this games has been discussed in this background study chapter. In Chapter 3 we will learn about the game features and front end description. Arising problems for developing the game and the possible solution of the problem has been discussed in this chapter. The code implementation and tools needed for this games has been discussed in the Chapter 4 Implementation. The basic IDE and framework for developing the game or use to code is discussed. Following what sort of libraries or resources for implementing the code further and what their properties has been discussed in this chapter. In the last result Chapter 5 will talk about the outcomes and visual outputs of the game. We will possibly find more information useful after reading this following chapters.

Chapter 2

Background Study

From the beginning of time there always remains a racing intention in every living things. Some race to live some race for life and some race for hobby. Centuries to centuries the word made the racing words on its way to various sports.

The history of racing is always remains unique amongst all sports. The basic concept of racing is always- get from one point to another point before any competitor does. Unlike other sports it provides one with so many opportunities to enjoy the sport.

2.1 Car Racing/ Motor Racing

Car racing/ motor racing is also known as auto racing/ automobile racing in the racing world. In the car racing/ automobile competitions the main aim is to set out the fastest time in a set number of laps or provided time limit. Finishing with the fastest time lapse determined to be win as in the first place, second-fastest in second place and so on. There are numerous types of auto racing could be founded where each has its own different adjusted rules, for all cars and drivers to comply.

Car racing can be differentiates by car and location. Some are listed here [7]:

- Dirt track racing
- Kart racing
- Midget car racing
- Monster trucks
- Open wheel racing
- Rally racing
- Solar car racing
- Touring car racing
- Tractor pulling

According to my research, I categorized the car racing according to the popularity and category of cars, their performance, and the tracks where the race is conducted. Some of the various types of races are listed below [8]:

- Formula 1 Racing
- Touring Car Racing
- Sport Car Racing
- Production Car Racing
- One Make or One Model Racing
- Stock Car Racing
- Rallies
- Off Road Racing
- Drag Racing

Figure 2.1: Formula One Racing Car [1]

2.1.1 Formula 1 Racing

Formula One race or F1 race or Grand Prix is determined as the highest class of single-seat auto racing sanctioned according by the Federation Internationale de l'Automobile (FIA). It is mainly part of the formula racing, which has almost 300 different category and F1 or Formula one comes first on them. The race consists a series of races conducted by Grand Prix throughout the world on their purpose-built F1 circuits and public roads [7]. The race named after formula because of its all sets of rules that must perform by the racers. The cars of Formula One race follow at speeds of up to 360 km/h (220 mph) with their engines currently is limited in performance approximate to a maximum of 15,000 RPM. Although the cars are a lot capable of lateral acceleration. The performance of the cars is very much dependent on electronics. The formula has radically evolved and changed through the history of the sport.

2.1.2 Touring Car Racing

Touring Car Racing is another popular form of racing that conducted among different car companies with heavily modified road-going cars. This types of

Figure 2.2: Touring Racing Car [2]

forms are mostly popular in in Argentina, Australia, Brazil, Britain, Germany, Sweden and Norway. Touring car racing competition rules vary from country to country. Some touring car competitions names given here [9],

- World Touring Car Championship (Worldwide)
- British Touring Car Championship (United Kingdom)
- DTM (Germany/Europe)
- Nurburgring VLN Endurance racing Series (Germany)
- Scandinavian Touring Car Championship (Sweden/Denmark)
- V8 Supercars (Australia and New Zealand)

2.1.3 Sport Car Racing

Sport car racing are two seats and enclosed wheels racing which have the familiarity with touring car racing that may propose to go road going cars. This car racing style often defined as hybrid of touring car racing and the annual Le Mans 24 Hours endurance race. It is one of the oldest form of motor car racing competition include the Italian classics, the Targa Florio (1906 1977) and Mille Miglia (1927 1957), and the Mexican Carrera Panamericana (1950 1954). This type of

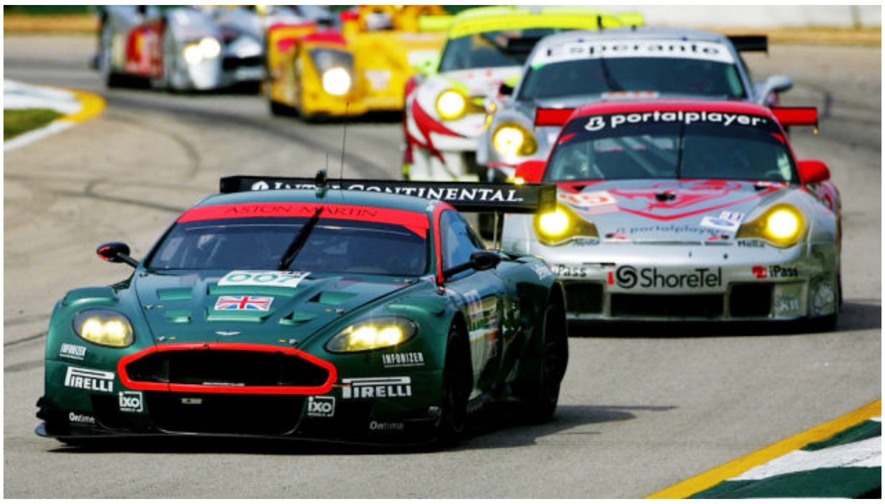


Figure 2.3: Sport Racing Car [3]

races usually emphasize endurance, pure speed, reliability, complex pit strategy and regular driver changes [10].

2.1.4 Production Car Racing

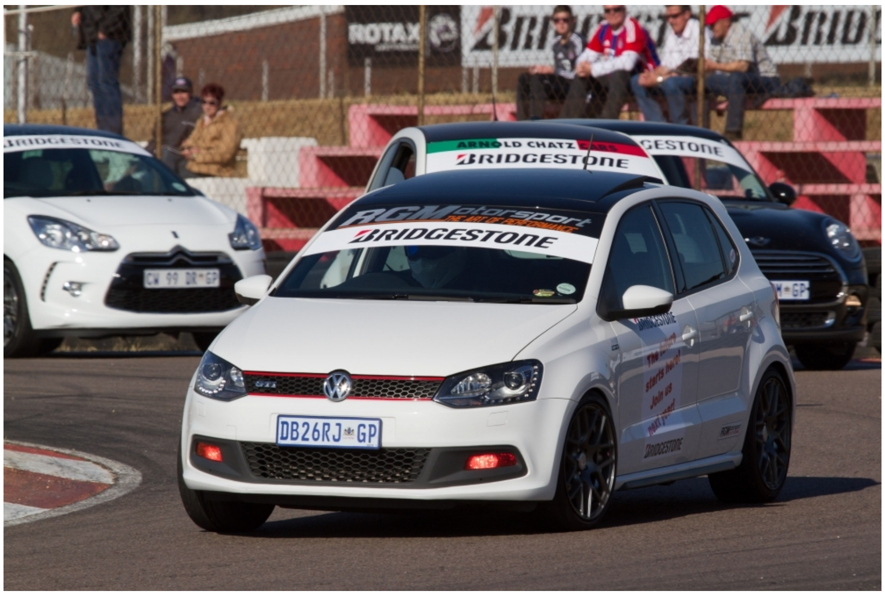


Figure 2.4: Production Racing Car [3]

Production car racing is another one of the form of racing where all cars are unmodified (or very lightly modified) cars race each other in consequence, outright and also in categorized in classes. This form of racing is also identified as one of the economical forms of racing to increase the popularity of the production cars amongst the speed car lovers. Production car racing also known differently in the US where they named as showroom stock racing. This form of racing occurs as an economical racing thus its rules are in some of restricted version of touring car racing, mainly to restrict costs.

2.1.5 One Make or One Model Racing

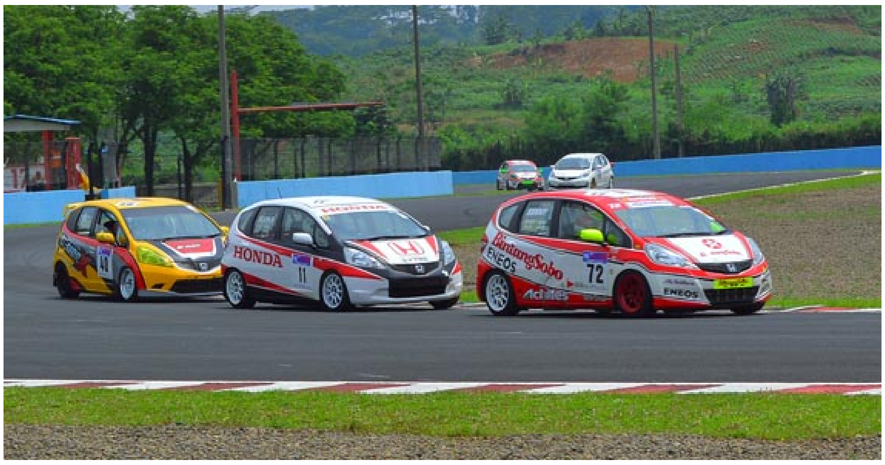


Figure 2.5: One Make Racing Car [4]

Thomas Middleton of the Shankill Corinthian Club first came up with the idea of one-make design idea at south of Dublin, Ireland in the year 1887. One make racing form are conducted between number of same type of cars which are manufactured by the same make or company. Hence this form of racing is known as one make racing because of its adopted in sports which use complex equipment, whereby all vehicles, gliders or boats have identical or very similar designs or models. There races are also conducted in separate tracks made for them. The important factors being measured in one make racing help to equalize the vehicles heavily used in sailboat racing and put more emphasis on the skill of the competitors. One-Design racing also refers two primary methods of competition in sailboat racing. The first boat to finish wins the race is the only rules of One-Design racing for sailboat racing. There was some attempts of bringing the sport of competitive glider racing with the advantages of one-design. Paul A Schweizer principal of Schweizer Aircraft on One-Design concept he wrote:

'The true measure of pilot ability and experience is usually shown by his final standing in a contest. What could be more indicative of this when pilots are flying identical sailplanes with identical performance. One-design competition is the sure test of soaring skill[11].'

2.1.6 Stock Car Racing

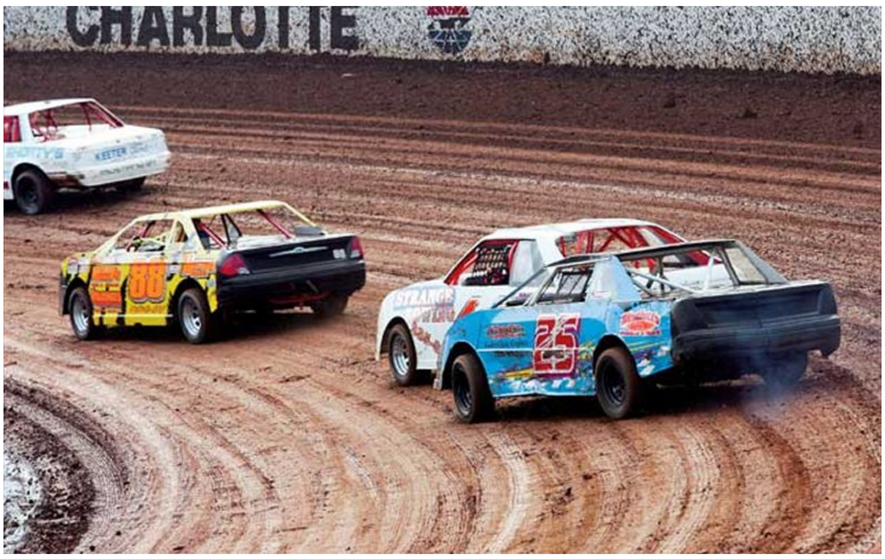


Figure 2.6: Stock Racing Car [1]

Stock car racing is a type of racing which involves the usage of purposely build race cars either they may wrecked or crashed cars and old cars are run on oval tracks measuring approximately 0.25 to 2.66 miles (0.4 to 4.3 kilometers) with average speeds in the top classes are usually 70—80% at the same tracks. These cars are thus altered and then used for racing. Hence this type of racing is known as stock car racing. Stock car racing found mainly in the United States, Canada, New Zealand, Australia, United Kingdom, Mexico, Brazil and Argentina. The idea of stock car racing firstly come from production car racing with factory build cars, but later it has been differentiate in different ideas and configurations. Stock car racing has been differentiates in some class each with slightly different rules but with near-identical specifications that look like production cars. Some of the classes are,

- Street Stock / Pure Stock
- Super Stock
- Late Model
- Crossover drivers

2.1.7 Rallies

Figure 2.7: Rallies Racing Car [2]

Rally is another one of the most important form of racing of cars where there are races which consist of usage of off road tracks and on road tracks. The cars used in this type of racing are production based cars. This type races are held more than two days. This motorsport is distinguished by running not on a circuit, but instead in a point-to-point format in which participants and their co-drivers drive between set control points, leaving at regular intervals from one or more start points.

Figure 2.8: Off Road Racing Car [4]

2.1.8 Off Road Racing

Off road racing is one of the form of racing in cars where the name itself indicates that this type of racing is conducted in off road tracks. Where the cars used in this type of racing may be either production cars or modified cars such as specially modified vehicles (including cars, trucks, motorcycles, and buggies).

- Desert Racing, North America (Mexican desert)
- Short Course Racing, North America (Midwestern United States)

2.1.9 Drag Racing

Drags racing of cars are the most predominant type of racing. Where this type of racing involves the performance of unique type of stunt called dragging. This type racing is performed only for shorter distance. Cars used in this type of racing are purposely built cars. Hence these are some of the types of racing involved in cars. Some popular drags racing organization are,

- National Hot Rod Association (NHRA), North America
- Australian National Drag Racing Association (ANDRA), Australia
- New Zealand Hot Rod Association (NZHRA), New Zealand
- Curacao Autosport Foundation (FAC), Curacao

Figure 2.9: Drag Racing Car [3]

- ADRL, QATAR MILE, NATIONAL STREET DRAG CHAMPIONSHIP, QATAR DRIFT CHAMPIONSHIP, FREESTYLE DRIFT and SEALINE SAND DRAGS, Middle East
- Tarlton International Raceway and ODI Raceway, South Africa

2.2 Racing Video Game

Racing video game are one of most popular form of genre video games [12] in the gaming world. Most of the racing video games are either built in the first-person or third-person perspective mode, in which the player takes part in a racing competition with any type of environment like land, air, or sea vehicles. It may also be categorised under the the category of sports games due to its popularity and simulation skill.

At the beginning of racing video games they were usually only played as an arcade games. Arcade games are sort of like coined games, mainly a coin-operated entertainment machine is played as arcade game that installed in public businesses, such as restaurants, bars, and particularly amusement arcades. With the revolution of changing from 2D video games to 3D video games this arcade games were played a role amongst all around the gamers of the world.

According to the google trend Need for Speed is the most searched and played video games around the world which is also known by its initials NFS published by Electronic Arts (EA) and has been developed by several studios around the world including the Canadian and the British companies. Need for Speed games has been developed by Distinctive Software video game studio based in Vancouver, Canada. According to gamers world website Need for Speed is the most successful racing video game series in the world, and one of the most successful video game franchises of all time. In the game player controls a race car in a variety of races to win the race. Player can change or choose a vehicles and tracks and can play as either in the mode of tournament or career mode.

Figure 2.10: Need for Speed: No Limits [5]

List of Need For Speed Video Games: [10]

- The Need for Speed (1996)
- Need for Speed II (1997)
- Need for Speed III: Hot Pursuit (1998)
- Need for Speed: High Stakes (1999)
- Need for Speed: Porsche Unleashed (2000)

- Need for Speed: Hot Pursuit 2 (2002)
- Need for Speed: Underground (2003)
- Need for Speed: Underground 2 (2004)
- Need for Speed: Underground Rivals (2005)
- Need for Speed: Most Wanted (2005)
- Need for Speed: Carbon (2006)
- Need for Speed: ProStreet (2007)
- Need for Speed: Undercover (2008)
- Need for Speed: Shift (2009)
- Need for Speed: Nitro (2009)
- Need for Speed: World (2010)
- Need for Speed: Nitro-X (2010)
- Need for Speed: Hot Pursuit (2010)
- Shift 2: Unleashed (2011)
- Need for Speed: The Run (2011)
- Need for Speed: Most Wanted (2012)
- Need for Speed Rivals (2013)
- Need for Speed: No Limits (2015)
- Need for Speed (2015)

2.3 Car Fighting Video Games

Car fighting video games are mostly played as video games where the primary objectives of the players gameplay includes cars, usually armed with various machine guns, missiles, molotov cocktails, pipe bombs, hand grenades, and other prepared weapons, attempting to destroy the other vehicles in the game controlled by the CPU or by opposing another user played in multiplayer mode. Each of them

Figure 2.11: Mad Max Video Games[6]

can differently choose cars with its own strengths, damage modes, and special attacking features. There is also some features where the players may also able to unlock the hidden cars by finishing certain levels and in-game tasks. Most of the car fighting games follow certain common rule of patterns. The player must defeat increasing numbers enemies with increasingly skilled. Often have to defeat in increasingly complex battlefields, before facing off against a final, super-powerful, boss character. Some car fighting games are listed here,

- Mad Max (2015)
- Auto Assault (2006)
- Armageddon Riders (2011)
- Blood Drive (2010)
- Carmageddon series (1997, 1998, 2014)
- DiRT: Showdown (2012)
- FlatOut series (2004, 2006, 2007)
- Full Auto (2006)
- Knight Rider (1989, 1990)
- Need for Speed: Hot Pursuit (2010)
- Rage (2011)

- San Francisco Rush 2049 (1999, 2000)
- Vigilante 8 (1998)
- Zombie Driver (2009)

2.4 Game Engines

Game engines are usually build to make easier everything in the process of game development. Every game engine have their own unique features of developing in gaming process. But generally they have only main purpose in building game engine is to provide easy abstraction layers for graphics, audio, input, scene management, collision detection, maths and general useful utilities for developing a game. Some game engines provide with wrappers and some plugins for physics APIs. Some even have some AI support (mostly limited to algorithms like as FSMs, pathfinding and with the current trend behaviour of trees).Most of the game engines support and pride themselves, on cross-platform functionality.

Usually game engine is a framework which provides the facilitates and the kinds of tasks which helps in the process at the time of writing a game. So the question is, what are the kinds of tasks we want to do? We want to display images on the screen in game terminology which is to be displayed on the screen is called a sprite. We always may need to draw some of menus or text on the screen of game window. Thus we want to organise all of our sprites so that some are not left behind. Some games use realistic physic maths that so objects collide or fall with gravity, which creates real simulation.

Purpose of a game engine's is to make the life of game developer a lot easier to create a game, without having any shorts of hazards in creating a game from scratch in a small amount games. It also helps from installing a whole bunch of libraries and writing on development of our own wrappers for them to suit in our game.

A lot of game engines are out there already in handy, but beware of some significant difference between a graphics engine and a games engine (For as example, Unity 3D is a games engine, whereas Ogre is a graphics engine).

2.4.1 Components of the Modern Game Engine

It is true that game engines are interconnecting parts or elements sets of components which provide a lot of useful features in the process of making games. Unlike any other general purpose development frameworks, like Cocoa Touch or .NET games, game engines are made specifically for developing games and give all of other components organized to the detriment of other forms of applications. To compensate for the lacking it is easy tools for building menu bars and widgets, game engines have graphics engines optimized it so that to be as fast as possible and instead of using default popup windows and system sounds they contain sound engines which place sounds in 3D space.

Input

One of the most important aspects of a game is the means to play it, so game engines are usually support an array of input types like, keyboard, mouse, gamepad and touch with less-common input methods like, joystick, steering wheel, rollerball, multi-touch for being subsets thereof. There are many different ways to handle this type of input events, but there are two common means: events and polling.

Input events only work by the computer input type like, mouse button pressed, keyboard key released, joystick axis changed, touch pressed events for triggering our custom code. It combined with a mapping table, which will connect keyboard, controller or mouse buttons to named as actions, such as to jump or shoot, so that we can develop our code without having worry about the user layout than the one we build our game around.

Polling is called position values which usually as the x/y coordinates of the mouse or the amount of tilt of a gamepad's analog stick. Game engine provides the options and help to retrieve these values whenever the developer wants to react to changes in these values.

Graphics

As explained above, game engines provide assistance in the rendering sprites to the screen. It's not efficient to just load images into our game one by one, it requires more effect on it. The idea is to put all our images onto a big image sheet and then load it as one image. When we need one of the images it can be cropped from the sheet. For developing complex games it is difficult to implement from scratch but in a good game engine like Cocos2D or LibGDX[13] it's simplifies the work for us. They also allow us to create frame by frame animations from these sprite sheets. If we want to make our character walk we can provide images of the character in all the stages of walking. The game engine can then loop through these images to create a walking animation.

Game engines also provide more complex features as described below [6]:
LibGDX game engine for Android, provide completely shields the knowledge how the graphics are implemented. It helps in using it during development of our game run it as a native Windows or OSX program on our computer for avoiding the slow buggy Android emulator.

Game engines also provide help with animations like Cocos2D which proves a number actions like CCMoveTo and CCFadeOut. We can create an action which will send our sprite from one location to a different location in a specified amount of time or to fade out our image. We can also create a sequence of actions to be executed one after the other. This saves us having to write our own classes to update the object‘s position every frame.

Game engines also help we organise our game. We can create scenes which contain layers. Each level can be represented by one scene and the background can be on a different layer to our player. This helps to keep our game organised and reduced the complexity of the code.

Sound

Another game engine features of the audio engine which consists of any algorithms related to sound. It can calculate process things quickly on the CPU, or on dedicated ASIC. SOme of the APIs, such as e.g. OpenAL, SDL audio, XAudio 2, etc. are available with the sound engines features.

Games sound effects don't usually just come out of our speakers. They were recorded, but most game engines have the opportunities to place sounds inside the 3D world of the game which will modify the volume depending on where our character is relative to the sound. There are also a lot of ways to make sound's realistic by adding pitch or modulation and reverberation to make it seems like the sound is bouncing off the walls of its surroundings. As example, the sound of clashing swords in a open ground versus down in the depths of a dungeon and how it reflect the world around them makes some difference in the sound quality. This type of things are now handled gently by the sound engines.

Physics

All good game engines come with physics built in. Physic is a term which covers a wide variety of functions including: detecting collisions, applying forces and velocity to objects. By using some of a physics engine we can define our shapes and wait watch them respond like real world shapes would. Game angry birds most of the features relies almost entirely on physics for it's game play. Physics engine will help us to know when a grenade or bullet hits our players car. It can also create some of a wobbly bridge or a water effect in the game. After sorting these things out we are going to face very difficult and would require a minimum of a degree in Maths of Physics. But by using a pre-made, free, physics engine we can get a straight way into creating our game. In my research the best physics engine I have found is Box2D. It's very powerful and has implementations for iPhone and Android both and is included in all the best game engines. The benefit of using a very popular cross platform engine like Box2D from Android to iOS we don't need to re-learn how to use the physics engine!

Some basic ideas of related works needs to be understood for that physics isn't all over an integral part of rendering a 3D gaming worlds, despite in the modern game engines combine this two with the end-user where developer doesn't have to be aware of the distinction. When we try to render a straight cube in a game, sometime it's just only a simple visual effect, perhaps it is combined with light refraction and bumped maps to give it a good sense that it is really very much existed in the world, but there is nothing to be inherent in the cube which says that it has to be maintained according to physical laws. For games, physics needs to be developed to the cube for as it to react to gravity sense or being pushed or shot effect understanding by the user. While giving the cube a physical shapes, which may not be exact the same as the visual shape creates, as well as mass, friction, bounciness and other properties that is been used to create an object which can interact with the world around it.

It is too much costly in handling the physics process, but also only adding some of physical bodies to the objects is not that need to react, as well as making of their physical shape will be less complicated than the visual one. Thus its a simple way to make the games run more faster then ever. This is the reason why in the game we can slide along a row of trees without any hazard like real simulation, even as if they were a flat wall, because of the computation that is needed to determine all our collision with every single count of tree trunk is a waste of time for the game when we should just going to be following the corridor rather than exploring the whole anyway.

User Interface

If we not really being capable of building the typical user-interface of a Windows program, with a menu bar and floating windows, game engines gives the feature with this GUI capabilities. Every games user interface have their own custom GUI to fit with the style of the game. So providing a standard UI isn't really as important as giving developers the means to build their own custom buttons, drop-downs, sliders and such by combining textures, colors and events.

Many other different game engines handle the problem of GUI differently. Some game engines just ignore the issue altogether as they require the developers to build the functions manually. This isn't exactly as that hardest thing to do, as a GUI is pretty much just a list of text or images which can be identified for clicked on or selected using the keyboard or gamepad.

Chapter 3

Problem

Mad Dash Car Crash is mainly based on genre car racing third person 3D video games. The game is leveled based racing like tournament racing but different from its basic idea. Every level has to be completed with a fixed leveled track and fixed lap based on level and enemy car left. Mad Dash Car Crash is also fighting games but the fighting mode is only based on car and amination integrated with in the car. But player can choose and change the car, upgrade and repair its damages and add different type of animation based on level, points and position on the race.

Gamer will have the flexibility of playing either in full screen mode or window mode. Simple car choosing mode and settings mode will give player simple understanding game environment within it. In this basic version single user can play one time at a time with a fixed number of computer controlled enemy car. After choosing car from start screen mode player will screen displaying either player's wants to play in tutorial mode or tournament mode. The basic version of the game will give simple racing control features in tutorial mode. There will be no enemy car or destruction in the tutorial mode. By choosing tournament player can't get back to the tutorial mode after starting. If gamer exits in the middle of the tournament he or she will lose the game. After starting the game player must complete all the level in a consecutive sequence. Any short of termination in this mean time will cause lose of his full afford and begin his game from the beginning.

Game environment is set on a base of island. Players have to complete the fixed amount of lap around the island alive. During race players have to acquire certain points from tracks during race to upgrade cars. Most of the points will come from demolishing enemy cars. Also players only goal in the games is to demolish all the enemy cars. If gamers unable to demolish all the enemy cars before the end of a level, the enemy cars and environmental enemies will grew stronger in every upper level. So players have to acquire points to gather amination to demolish enemy cars and in the mean time gamers have to destroy enemy cars too.

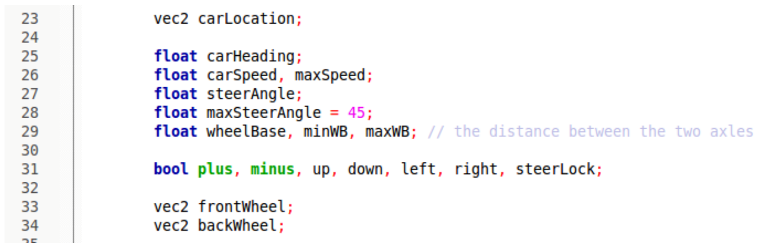
Mad Dash Car Crash is the competition of the MadLand declared by Mad King Rafa that has been discussed in the Introduction. As for the cruel rule of the Mad King, no one has ever able to win the competition alive. Because there is a black minded form in the race that was always hidden from the racers. The rule that is always unware of every racers. After destroying or killing all other racers, the winning racer have to face the last lap race with the dark force. In this lap after destroying all enemy car they will reveal and will try to catch us. They will not only try to catch but also will through fire and dark magic towards the racers. The racers have to dos those powers Racers can't destroy the evil force. The only way to win is to avoid those powers and cross the finish line before the evil force destroy the racers car.

The game has a hero who will change the rules after winning the game. He will get rid of this prison for the first time, and with the winning of the game for the first time the evil force will be declined from getting feed. For the reason they will get scrambled and get out of controlled of the Mad King. Mad King will get angry and will try to imprisoned the evil force. Which will make more scrambled amongst the evil force unity. The evil force will begin to attack the

Mad King. A great dark magic war will took place in the Island of MadLand. In this situation the hero will began to imprison all the prisoners of the island and get them safely out of the Mad Island. When all the villagers of the Island will safely reach the shore they will see the Kingdom is falling down and blust altogether taking with the Mad King and his Evil Force.

The starting and ending part story is not included with the basic version of the game. It requires some animation which is been kept for the future work. When the animation will be complete it will be included with the new version with more extra features.

To track in which state of our car in the game we are discussing probably have something like:



```
vec2 carLocation;

float carHeading;
float carSpeed, maxSpeed;
float steerAngle;
float maxSteerAngle = 45;
float wheelBase, minWB, maxWB; // the distance between the two axles

bool plus, minus, up, down, left, right, steerLock;

vec2 frontWheel;
vec2 backWheel;
```

Figure 3.1: Physics algorithm Code

Physics algorithm in this game will update the carLocation and carHeading in each frame according to the above assumptions shown in the figure, and the input part of our game can update carSpeed and steerAngle based on the user input hardcoded input. So to calculate the next position and heading of the car I have broke down process in basic three steps:

- Position finding in the game world of a imaginary front and back wheels
- Wheel movement according to the current pointing direction of the wheel
- Find car next location according to wheel current location and next heading to.

3.1 Position Finding

In the game I use geometry calculations to find the actual positions of the wheels (which are at the centre of each axle). The gap between the axles is given by the wheelBase, so each wheel is half that distance from the car centre, in the direction that the car is facing. The following diagram shows the geometry: Applied code

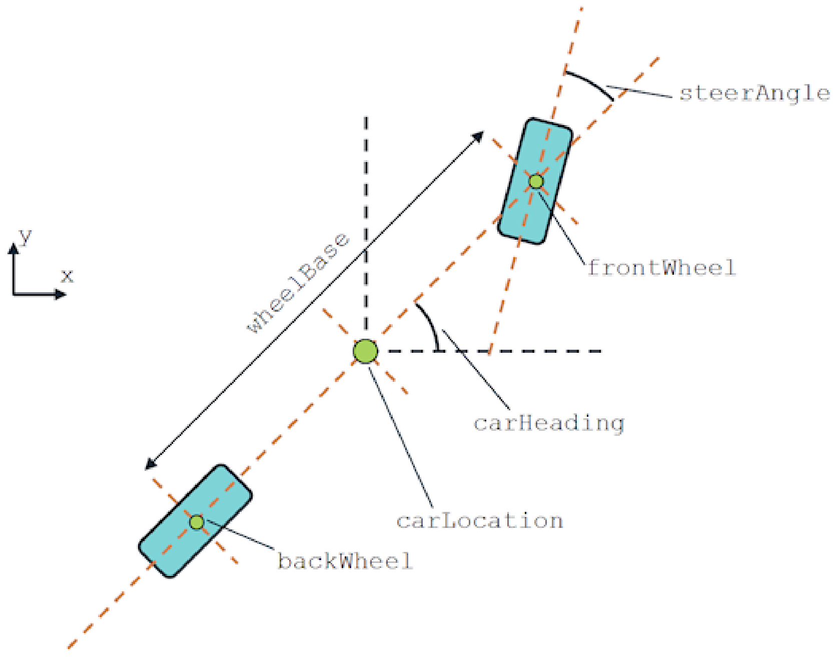
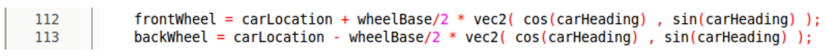


Figure 3.2: Position Finding

for front wheel and back wheel as shown in the picture given below:



```
frontWheel = carLocation + wheelBase/2 * vec2( cos(carHeading) , sin(carHeading) );
backWheel = carLocation - wheelBase/2 * vec2( cos(carHeading) , sin(carHeading) );
```

Figure 3.3: Physics algorithm Code

3.2 Wheel Movement

Each wheel should move forward by a certain amount in the direction it is pointing. The distance it needs to move depends on the car speed, and the time between frames (I'll call it dt here). The rear wheel is easy, it moves in the same direction the car is heading. For the front wheel, we have to add the steer angle to the car heading as the diagram shows: Applied code for front wheel and back

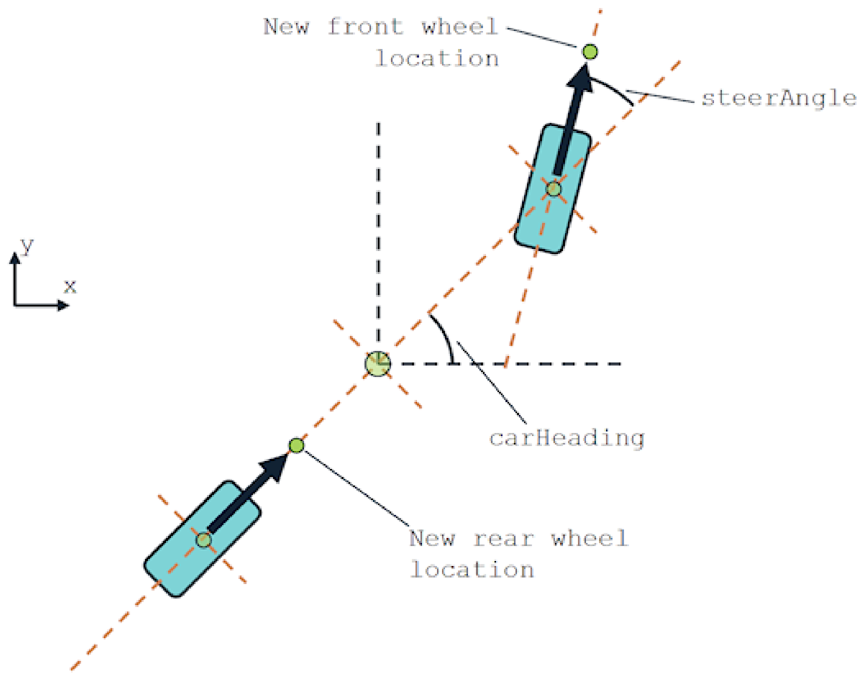
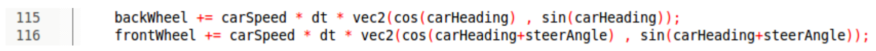


Figure 3.4: Wheel Movement

wheel motion with directional movement as shown in the picture given below:



```
backWheel += carSpeed * dt * vec2(cos(carHeading) , sin(carHeading));
frontWheel += carSpeed * dt * vec2(cos(carHeading+steerAngle) , sin(carHeading+steerAngle));
```

Figure 3.5: Physics algorithm Code

3.3 Car Next Location

For finding car next location we need to find the wheels new position. For this we calculated by averaging the two new wheel positions. Car new location need to find where the car heading to by the steer angle. The new car heading can be found by calculating the angle of the line between the two new wheel positions: Applied code for car current location and where the car will next heading to shown in the picture given below:

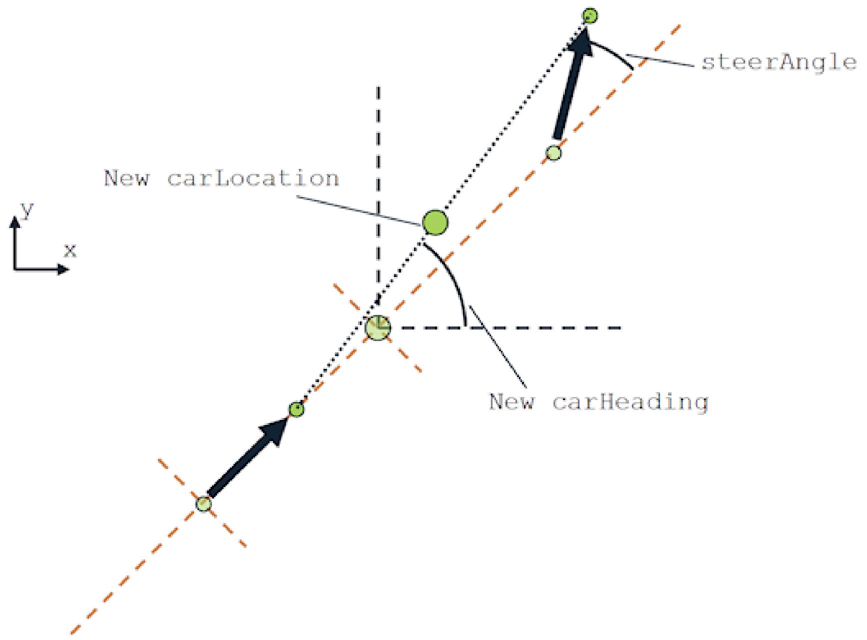
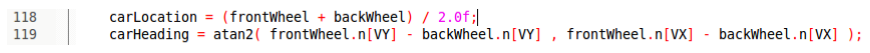


Figure 3.6: Car Next Location



```
carLocation = (frontWheel + backWheel) / 2.0f;
carHeading = atan2( frontWheel.n[VY] - backWheel.n[VY] , frontWheel.n[VX] - backWheel.n[VX] );
```

Figure 3.7: Physics algorithm Code

3.4 Car Next Location

A problem with this algorithm. It doesn't conserve the baseline, that is, the distance between the front and rear tire. Over time, if we move forward while turning, the baseline will shrink down to carSpeed * dt, and the motion will become completely unrealistic. Testing with an initial baseline of 50, carSpeed = 50, dt = 1/60, and steerAngle = 20 degrees. By frame 500 the baseline had shrunk down to 26.2.

3.5 Pathfinding Algorithm

For road map and moving car with that map following, I use A* path finding algorithm in the game. Its a searching algorithm that find the coordinates according to the nodes placed for the map int the visual world. We will learn more about that in the result section of Chapter 5.

Chapter 4

Implementation

Mad Dash Car Crash games has not been developed with any sort of game engine. The game is based on C++ platform with basic C libraries. C++ provides object-oriented and generic programming features It also provides facilities for low-level memory manipulation, which helps for running the game fast. It used GNU Compiler Collection or GNU C Compiler (GCC) compiler to compile the game.

To develop the games it used.some tools and libraries that has been discussed below.

Libraries:

- OpenGL
- GLEW
- SDL
- Assimp

IDE‘s:

- Codeblocks
- Blender
- Photoshop

4.1 OpenGL (Open Graphics Library)

Open Graphics Library or OpenGL is a software interface that allow multi-platform, cross-language application programming interface (API) to communicate with graphics hardware for rendering 2D and 3D vector graphics. Purpose of the API is to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. Since OpenGL is merely a graphics library, window and context creation must be handled by an external library, usually provided by the operating system. But since we could use a different operating system by using a cross-platform library and FreeGLUT is an Open Source library that does exactly that. Modeled after the long abandoned but still popular GLUT library (OpenGL Utility Toolkit), FreeGLUT provides a modern Open Sourced alternative that is easy to use, cross platform compatible, and suitable for creating demonstrative programs such as the ones in this book. FreeGLUT is licensed under the X-Consortium license. The basic graphics control and model rendering of Mad Dash Car Crash has been done by OpenGL API. OpenGL GLUT and GLU libraries has been used for perspective rendering, lighting and camera controlling.

4.2 GLEW (OpenGL Extension Wrangler)

For loading extensions it can be create a platform-dependency, so the next library will need is the GLEW library (OpenGL Extension Wrangler), which makes it easier to use OpenGL extensions in programs.We're skipping an important part of OpenGL by using this 3rd party library, We can found the GLEW library at glew.sf.net for free, as it is also Open Source and licensed under several unrestricted licenses that allow us to use GLEW in any code base, similar to the license that FreeGLUT uses. Please make sure to get version 1.5.4 or higher for OpenGL 4.0 support.

4.3 SDL (Simple DirectMedia Layer)

Simple DirectMedia Layer is a cross-platform multimedia development library that has been used countless time in developing reputed games and well known commercial projects. The has been developed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. SDL works with a platform's underlying multimedia capabilities to provide a constant and open API across multiple operating system. Some of SDL operating systems, including:

- Windows
- Mac OS X
- OS 9
- Linux
- Google Android
- AmigaOS
- Haiku/BeOS
- Syllable
- WebOS

SDL is written in C, works natively with C++, and there are bindings available for several other languages, including C# and Python. It also provides bindings to many other languages, including (a partial listing):

- C#
- Ada
- Eiffel
- D
- Euphoria
- Erlang

- Haskell
- Guile
- Lisp
- Java
- ML

SDL can be used in combination with the OpenGL API or Direct3D API for 3D graphics. Basic Initialization and Shutdown, Configuration Variables, Error Handling, Log Handling could be done by SDL. Supports easy rotation, scaling and alpha blending, all accelerated using modern 3D APIs. Vast use in display and window management, surface functions, rendering acceleration, etc. Can create and manage multiple windows. Acceleration is supported in SDL using OpenGL and Direct3D, and there is also a software fallback in it. It is mostly been used for event handling features for Keyboard, Mouse, Joystick and Game controller. Events and API functions provided for:

- Application and window state changes
- Mouse input
- Keyboard input
- Joystick and game controller input
- Multitouch gestures

Some other features of SDL are:

- Force Feedback
- Audio
- Filesystem Paths, File I/O Abstraction
- Shared Object Support
- Threads
- Timers

- CPU Feature Detection
- Shared Object Loading and Function Lookup
- Platform and CPU Information
- Endian Independence
- Power Management Status
- Platform-specific functionality

Window initialization, event handling, audio control of Mad Dash Car Crash has been games are done by the SDL libraries. The game is not developed with any kind of game engines but in a hand SDL is also one kind of game engines. So in that circumstance it is using SDL game engine.

4.4 Assimp (Open Asset Import Library)

Open Asset Import Library or Assimp is a generic importer library to load and process geometric scenes from various data formats. Open Asset Import Library or Assimp can import and export various common 3D formats. The library is written in platform-independent, portable C++. Bindings for other languages and ecosystems are available. Assimp supported file formats are:

- Collada (*.dae;*.xml)
- Blender (*.blend) 3
- Biovision BVH (*.bvh)
- 3D Studio Max 3DS (*.3ds)
- 3D Studio Max ASE (*.ase)
- Wavefront Object (*.obj)
- Stanford Polygon Library (*.ply)
- AutoCAD DXF (*.dxf)
- IFC-STEP, Industry Foundation Classes (*.ifc)

- Neutral File Format (*.nff)
- Sense8 WorldToolkit (*.nff)
- Valve Model (*.smd,*.vta) 3
- Quake I (*.mdl)
- Quake II (*.md2)
- Quake III (*.md3)
- Quake 3 BSP (*.pk3) 1
- RtCW (*.mdc)
- Doom 3 (*.md5mesh;*.md5anim;*.md5camera)
- DirectX X (*.x).
- Quick3D (*.q3o;q3s).
- Raw Triangles (.raw).
- AC3D (*.ac).
- Stereolithography (*.stl).
- Autodesk DXF (*.dxf).
- Irrlicht Mesh (*.irrmesh;*.xml).
- Irrlicht Scene (*.irr;*.xml).
- Object File Format (*.off).
- Terragen Terrain (*.ter)
- 3D GameStudio Model (*.mdl)
- 3D GameStudio Terrain (*.hmp)
- Ogre (*.mesh.xml, *.skeleton.xml, *.material)3
- Milkshape 3D (*.ms3d)
- LightWave Model (*.lwo)
- LightWave Scene (*.lws)

- Modo Model (*.lxo)
- Character Studio Motion (*.csm)
- Stanford Ply (*.ply)
- TrueSpace (*.cob, *.scn)2
- XGL (*.xgl, *.zgl)

For Mad Dash Car Crash, Callada file formats has been used to integrate model file. Most of the models, objects, structures and all cars are loaded in the game with Open Asset Import Library. Models are being loaded with lighting and meterial within it.

4.5 IDE‘s

For code implementation and need of cross platform support, in the development of the game used a free open source IDE Code::Blocks. It has a custom build system and optional Make support. Any kind of functionality can be added by installing/coding a plugin. For instance, compiling and debugging functionality is already provided by plugins! Written in C++, it offers interfaces for both C and C++ and with also in Fortran. For its easy support of multiple compilers, including GCC and MinGW, it help linking the graphics library such as OpenGL, SDL, etc is simple to use and link up in this IDE.

Free and open-source 3D computer graphics software product used for creating animated f3D pipeline modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. It also has a game engine features in it. Its interface uses OpenGL to provide a consistent experience.Blender is free to use for any purpose, including commercially or for education. All car models for this game has been built with blender and export with COLLADA plugin to execute in the code with assimp library.

Photoshop is a digitally used image-editing software developed and manufactured by Adobe Systems Inc. Photoshop can be used for almost any kind of image editing, such as touching up photos, creating high-quality graphics, and much, much more. Photoshop is considered one of the leaders in photo editing software. The software allows users to manipulate, crop, resize, and correct color on digital photos. The software is particularly popular amongst professional photographers and graphic designers. Some of the images of the paper done with this software and some editing too.

4.6 Class Diagram

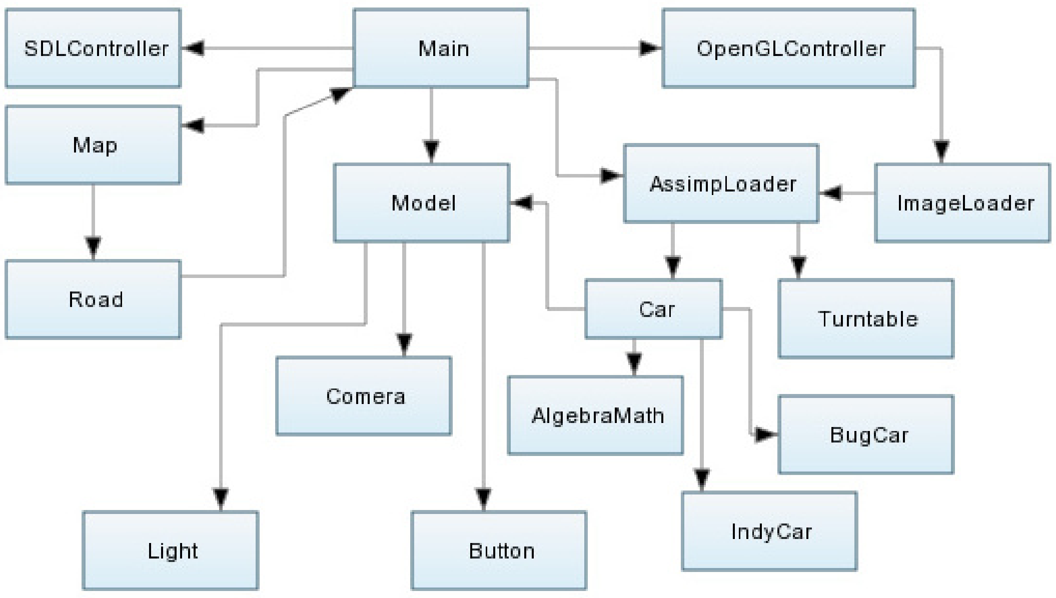


Figure 4.1: Games Full Class Diagram

4.6.1 Main

It is the main function that declared as a non-member function in the global namespace. It initialize the Controller class where it initialize all other classes to load the game. The initialized functions are listed here:

```
void roadSetup();

void carChoose();

void raceScreen();

bool display();

void classInitializer();
```

4.6.2 Controller

```
bool quit;

bool fullScreen = false;

bool gameStart = true;

int chooseCar = 0;
```

Functions

```
void fullPower();

void nullPower();

void carStatistics();

void controllScreen() ;

void keyPressed( SDL_Event event );

void keyReleased( SDL_Event event );

void windowEvent(SDL_Event event);

void mouseEvent(SDL_Event event);
```

4.6.3 SDLController

```
Uint32 Flag;

char* Game_Title;

int SCREEN_WIDTH = 800;

int SCREEN_HEIGHT = 600;

int SCREEN_FPS = 60;

SDL_Window* myWindow; /* Our window handle */
```

```
SDL_SysWMinfo info;
SDL_GLContext myContext; /* Our opengl context handle */
```

Functions

```
SDLController(char* title);
~SDLController();
bool init();
void setFlag(int a);
int getWidth();
int getHeight();
```

4.6.4 OpenGLController

```
bool init();
```

Functions

```
OpenGLController();
~OpenGLController();
```

4.6.5 Model

```
GLfloat x, y, z;
GLfloat speed;
GLfloat angle, Rx, Ry, Rz;
bool active;
```

Functions

```
Model();
~Model();
void setActive(bool s);
virtual void render();
```

4.6.6 Camera

```
float camX, camY, camZ;
float lookX, lookY, lookZ;
float upX, upY, upZ;
float camR;
static const int FIXED_MODE = 0;
static const int REVOLVE_MODE = 1;
static const int THIRD_PERSON_MODE = 2;
static const int FIRST_PERSON_MODE = 3;
```

Functions

```
Camera(Model *target);
void setCameraMode(const int mode);
void setCamera();
void setTarget(Model *target);
```

4.6.7 Light

```
int LIGHT;
float ambient[4];
float diffuse[4];
float specular[4];
float position[4];
float spot_direction[3];
float spot_exponent;
float spot_cutoff;
float constant_attenuation;
float linear_attenuation;
float quadratic_attenuation;
int on_off_state;
```

Functions

```
Light ( );
~Light ( );
void Init ( int l );
void SetLight ( int l );
void SetValues ( );
void TurnOn ( );
void TurnOff ( );
void Toggle ( );
void GetOnOffState ( );
void SetAmbientColor ( float r, float g, float b, float a );
void GetAmbientColor ( float *r, float *g, float *b, float *a );
void SetAmbientColor ( float c[4] );
void GetAmbientColor ( float *c[4] );
void SetDiffuseColor ( float r, float g, float b, float a );
void GetDiffuseColor ( float *r, float *g, float *b, float *a );
void SetDiffuseColor ( float c[4] );
void GetDiffuseColor ( float *c[4] );
void SetPosition ( float x, float y, float z, float w );
void GetPosition ( float *x, float *y, float *z, float *w );
void SetPosition ( float p[4] );
void GetPosition ( float *p[4] );
void SetSpecular ( float r, float g, float b, float a );
void GetSpecular ( float *r, float *g, float *b, float *a );
void SetSpecular ( float s[4] );
void GetSpecular ( float *s[4] );
void SetSpotDirection ( float x, float y, float z );
void GetSpotDirection ( float *x, float *y, float *z );
void SetSpotDirection ( float s[3] );
void GetSpotDirection ( float *s[3] );

void SetSpotExponent ( float exponent );

float GetSpotExponent ( );

void SetSpotCutoff ( float cutoff );

float GetSpotCutoff ( );

void SetConstantAtt ( float constant );

float GetConstantAtt ( );

void SetLinearAtt ( float linear );

float GetLinearAtt ( );

void SetQuadraticAtt ( float quadratic );

float GetQuadraticAtt ( );
```

4.6.8 Map

```
float MapX = 0;

float MapY = 450;

float MapW = 200;

float MapH = 150;

float CenterX = MapX+(MapW/2);

float CenterY = MapY+(MapH/2);

int mapCoordinates[10][10][2];

trackCoordinate * track = new trackCoordinate[10];

int gMap = 0;
```

Functions

```
Map();

~Map();

void setGMap(float m);

int getGMap();

void loadTrack();
void setMapLoc(float x, float y, float w, float h);
void setMapCoordinates(float x, float y, float w, float h);
```

4.6.9 ImageLoader

```
char* pixels;
int width;
int height;
```

Functions

```
Image(char* ps, int w, int h);
~Image();
GLuint loadTexture(Image* image);
```

4.6.10 Car

```
GLUquadricObj *FRWheel = gluNewQuadric();
GLUquadricObj *FLWheel = gluNewQuadric();
GLUquadricObj *BRWheel = gluNewQuadric();
GLUquadricObj *BLWheel = gluNewQuadric();
GLUquadricObj *FAxile = gluNewQuadric();
GLUquadricObj *BAxile = gluNewQuadric();
float dt;
Model *target;
vec2 carLocation;
float carHeading;
float carSpeed, maxSpeed;
float steerAngle;
float maxSteerAngle = 45;
float wheelBase, minWB, maxWB; // the distance between the two axles
bool plus, minus, up, down, left, right, steerLock;
```

```
vec2 frontWheel;

vec2 backWheel;
```

Functions

```
Car(float x, float y);

~Car();

void FrontRightWheel();

void FrontLeftWheel();

void BackRightWheel();

void BackLeftWheel();

void FrontAxile();

void BackAxile();

void CarBody();

virtual void render();

void setWheelBase(float wb);
```

Chapter 5

Result and Evaluation

The most challenging part this game developing was applying the car physics in the game. Most of the 3D car games apply almost similar car physics to move the car. Control car wheels while moving specifically the rear wheels are need to slide sideways whenever we turn. The very simple assumption that we will start with is that each wheel can only move in the direction it is pointing. This is actually a very good approximation for normal driving, and exactly appropriate for these kinds of parking games. The other simplification we will make is to use what is called a bicycle model, we imagine that the car has just two wheels; one at the front in the middle to steer, and one at the back in the middle that cannot steer. Of course we can draw a real car with four wheels, but the physics will only be considering two wheels at the centre of each axle.

5.1 Implementing the A* algorithm

The first step in finding the path algorithm is to explain the searching area where we need some way to represent our game world in a manner which that allows the searching algorithm to search for and find the best path. In our game, we use color as trick of to do the collision detection [13]. Using this trick us first need to make a collision detection map, as shown in Fig. 3. This can be done easily by using any image processing software (e.g., Photoshop), and then changes the track to the color which users want to set it as the collision detection color (e.g., block color). Everything else in the collision detection map, where the cars are not allowed to drive, we just need to paint them to white (or any color just do not use the track color black). Ultimately, the game world is simplified by placing 1280*782 nodes, throughout the game environment. White color nodes represent obstacles and other colors nodes represent the nodes which can be passed. After that, we have divided our search area into a 1280*782 square grid. This particular method reduces our search area to a simple two dimensional array. Each item in the array represents one of the squares on the grid, and its status is recorded as passable or impassable. The path is found by figuring out which squares we should take to get from node A to node B. Once the path is found, the game-controlled car moves from the center of one square to the center of the next until the target is reached.

5.2 Pathfinding algorithm for random obstacles avoidance

In order to generalize the pathfinding algorithm in a racing game solves the dynamic obstacles avoidance problem. We have recently proposed a dynamic pathfinding method. Two collision detection points are put in front of the car‘s right side and left side. Where the variable y is the half width of car, that is 6 pixels, and the collision detection distance x is an adjusted variable indicated the distance from the car center to the center of the two collision detection points.

To perform this algorithm we need to put some color detecting points around the moving car. For this purpose, if we found that the position of the car‘s color detection point its color is the same as the tracked color, then no collision occurs. In the other hand, if we found that the position of the color detection point is white, then it indicates the car is leaving the track, which means the car needs to turn a direction to keep the car inside the track. In my game for the implementation, we will calculate the car‘s position in advance. If in the next time frame the collision is detected. We will just simply turn on a default setting circle around for the car to avoid collision. In other words, when the detection point of left front touches the edge of track, the car will turn in a clockwise direction. On this, the car will turn in a counterclockwise direction, if the detection point of right front touches the edge of track. In order to make the game-controlled car look more natural and smooth, the car‘s rotation speed is set to 0.25 radians (14.3 degrees) in this study.

The most common artificial intelligence in a racing game is waypoint navigation by carefully placing waypoints (nodes) in the game environment to move the game-controlled characters between each point. This is a very time consuming and CPU intensive problem. Using the A* algorithm can effectively solve the pathfinding problem in a static racing game environment; therefore, we present two modified A* algorithm instead of putting waypoints by hand and minimum the lap time. Finally, we propose a more general dynamic algorithm which can solve the random obstacles avoidance problem in a racing game. All the three algorithms are able to find the path for a car racing game and can save the most import resource in game, CPU cycles.

Chapter 6

Conclusion

This dissertation set out to investigate the role that computation plays in various aspects of preference aggregation, and to use computation to improve the resulting outcomes. In this final chapter, we will review the research contributions of this dissertation, as well as discuss directions for future research.

6.1 Contributions

The following are the main research contributions of this dissertation. (Some minor contributions are omitted.)

In these research paper I tried to cover only the implementation I could have done in my small time of research as I covered in Chapter 1 about my game and story or plot of the game. Following on Chapter Background Study where I tried to discuss about my background studies what we need to understand before starting the brief inside of the game implementation of the game. In Problem and Result Chapter I covered up what problems I faced in the developing of this games and what possible solution I could come up with.

Many problems that I faced in developing the game is already solved, but some of them are still unshorted due to less time get in the research periods. One of them is enemy car racing AI. It is the most important part of my game that I couldn‘t cover up in my research time. AI of other racing cars or CPU controlled car possibly also be covered up with the pathfinding algorithm.

Fighting mode of the game which makes it a fighting car racing games is not covered up in this research.

References

[1] Wikipedia, "Auto racing," April 2013. [Online]. Available: https://en.wikipedia.org/wiki/Auto_racing

[2] H. H. C. CHAMPIONSHIPS, "Advance booking has now closed. tickets will be available to purchase at the gate." 2013. [Online]. Available: http://www.topracingcars.com/

[3] C. Chapple, "Smart phone wallpaper," April 2014. [Online]. Available: http://www.spwallpapers.com/Others-Resolution/Cool-racing-cars-wallpapers-854x480/Cool-racing-cars-wallpapers-854x480-31

[4] Manic, "Manic," April 2013. [Online]. Available: http://www.racing-cars.com/main.asp?sitepages=manic

[5] BlackPanthaa, "Need for speed no limits announced! - is underground 3 next?" Nov 2014. [Online]. Available: https://www.youtube.com/watch?v=5_PtCwH_-8w

[6] B. Bernstein, "'mad max' game: 5 fast facts you need to know," August 2014. [Online]. Available: http://heavy.com/games

[7] J. A. G. L. delaOssa and V. Lopez, *Improvement of a car racing controller by means of Ant Colony Optimization algorithms.* pp. 365-371: in IEEE Symposium on Computational Intelligence and Games, 2008.

[8] T. N. S. Fujii and H. Ishibuchi, *A study on constructing fuzzy systems for high-level decision making in a car racing game.* in IEEE Congress on Evolutionary Computation, 2008.

[9] P. B. J. Togelius and S. M. Lucas, *Multi-population competitive co-evolution of car racing controllers.* in IEEE Congress on Evolutionary Computation, 2007.

[10] J. Togelius and S. M. Lucas, *Evolving robust and specialized car racing skills.* in Proceedings of the IEEE Congress on Evolutionary Computation, 2006.

[11] J. Y. Wang and Y. B. Lin, *An Effective Method of Pathfinding in a Car Racing Game.* in The 2nd International Conference on Computer and Automation Engineering, 2010.

[12] T. Nakashima and S. Fujii, *Designing high-level decision making systems based on fuzzy if-then rules for a point-to-point car racing game.* Soft Computing.

[13] C. Chapple, "The top 16 game engines for 2015," April 2015. [Online]. Available: http://www.develop-online.net/tools-and-tech/the-top-16-game-engines-for-2014/0192302