MINI PROJECT REPORT

On

<LEVIATOR>

Submitted by :

1.Abhinav Chaudhary(F - 181500010)

4. Kushagra Dubey (F - 181500345)

2.Tejasv Singhal (F - 181500757)

5.Vivek Kumar (I – 181500818)

3.Shobhit Chaturvedi(G - 181500685)

Department of Computer Engineering & Applications

Institute of Engineering & Technology

GLA University

Mathura- 281406, INDIA

2020

SYNOPSIS :PROJECT TOPIC: Hover 3D Racing

Project Group Members:

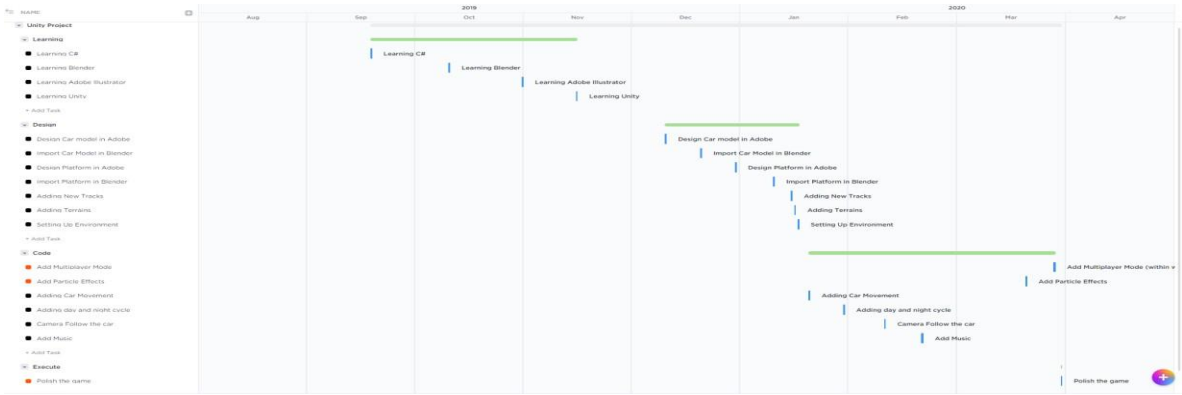
1 Abhinav Chaudhary(F - 181500010)
2.Kushagra Dubey(F-181500345)
3.Tejasv Singhal (F- 181500757)
4. Vivek Kumar (I -181500818)
5. ShobhitChaturvedi(G - 181500685)

Project Supervisor: Ms. Priya Agrawal

About the Project: The objective of "Hover 3D racing" is to create an entertaining-competitive environment for the players as well as giving them new challenges as they progress into the game further. Objective of the game is to get to the finish line as fast as possible. The software used to create the game is Unity engine combined with C#'s object oriented programming. Adobe illustrator and Adobe photoshop were used to create the model of the car and the platform. First we created the model of the car in illustrator and after that it was imported into Blender software to create the 3-D model of the car and the platform. For the surroundings, Unity provides inbuilt terrains which have been modified using blender in order to be used in the project. On the other hand, we also used Unity's component: Box collider and Rigidbody component. The objective of these components was to provide physics in our game such as adding force, gravity and collision with other objects. We also used Visual studio for adding scripts into the project for the movement of the car and also added a time cycle in the game which shifts between day and night.

Motivation: The motivation behind the project is to create a competitive multiplayer gaming platform for eSports because there has been a drastic rise in gaming in India. Since most of the competitive games are not from India it mostly benefits the other country so we wanted to create a game for our country.

Project Planning:



Tools required:

- **HardwareRequirements:**

  Requires a 64-bit processor and operating system

  OS: Windows 10 64bitProcessor: Intel Core i5 or AMD equivalent
  Memory: 8 GB RAM
  Graphics: NVIDIA GTX 660 or AMD Radeon HD 7950 or Intel HD 630 Network: Broadband Internet connection
  Storage: 2 GB

- **Software Requirements:**

  Microsoft Visual Studio
  Adobe Illustrator
  Adobe photoshop
  Blender
  Unity game development tool

Signature of ProjectGuide: _____

CERTIFICATE:

ACKNOWLEDGEMENT

This project is an acknowledgement to the intensity drive and technical competence of many persons who have contributed to it.

We express our heartiest gratitude and deepest thanks to Carl Callewaert(@CarlUnity) and other instructors who explained the basic mechanics and designing of a hover racing game .

We would also like to acknowledge Unity game store to provide us various assets related to this game. Without unity game store it would not be easy to give this game graphic appearance which is currently has.

We are very grateful to the Staff and Faculty Members of our college.

ABSTRACT

The proposed game will be a computer game version of a 3D basic racing terrain based game . Players will be able to play the Leviator game in a single player style known as hot seat, where users take turns at the same computer. The game will allow multiple players to play but the turn will have to be individual . The game will not deal with in artificial intelligence and will solely be intended for competitive single player use in order to break high scores .The game has been designed in 3D format based on the famous racing game Antigraviator .

The main menu will be the user's guide to the game where users will be able to change the in-game settings as per there comfort .

Users will also be able to toggle full screen and will be able to change graphics and resolution of the game.

TABLE OF CONTENTS

INTRODUCTION

1.1 Motivation And Overview

The motivation behind the project is to create a competitive multiplayer gaming platform for E-Sports because there has been a drastic rise in gaming in India. Since most of the competitive games are not from India , it mostly benefits other countries because of lack of involvement in this field in our country . This project focuses on making the involvement of people in the field of gaming to increase

The game starts with a basic start menu in which the user can change the graphics setting and along with graphics ,binded keys can also be changed. It's a racing game composed of three laps with a timer. The game also has speedometer that will be shown at the back of the Jet .The game is based on the day night terrain and the track is composed of basic blender development materials . Player should finish the game in least time in order to attain high score.The game is playable for all age groups.

1.2 OBJECTIVE

The objective of this project is to develop a game that can be published on online platforms such as Google Playstore and to be represented in Game Jams that are held all around the world . Moreover it can also be used to check decision making skills of the players indulged in this project . Collecting maximum points , trying to avoid obstacles which in this case are the track borders , when a player moves forward , it checks the thinking process of a player and many other games based on this objective have been used by scientists and psychologists all around the globe . Some games and projects like these are even a part of college and school curriculums in countries like U.K.

SOFTWARE REQUIREMENT ANALYSIS

2.1 DEFINE THE PROBLEM

As our country is developing at a rapid rate in the field of technology , lots of foreign companies are investing in different tech fields in our country . One of these foreign fields , Esports is also making its way in India through many games , like Player Unknown Battle Grounds , Fortnite and Counter Strike Global Offensive . This has open wide opportunity for local game developers in India. But since in Indian colleges game development is not taught in a general degree there are very few players in this field. Recently a game named Raji which is made by few Indian developers is launched on steam and it is getting huge hit just because it has been created by an Indian game development team , and under “Make in India” people actually support local work thus it’s a great opportunity to step in this field.

SOFTWARE REQUIREMENT ANALYSIS

2.2 MODULES AND THEIR FUNCTIONALITIES(SRS)

- **OVERVIEW**

  This document contains an overall description of the Leviator Game and specific constraints. There will be more specific details and information about the project content, general requirements and environment.

- **PRODUCT PERSPECTIVE**

  Leviator is aimed toward game players who like reflex free runner games .The product is independent and mostly self contained . This project is made by using Unity game engine where the whole designing process is done and scripting/coding part is done in Visual Studio 19 which are attached to the objects in the game to perform their roles .

- **SYSTEM INTERFACE**

  Leviator is developed in Unity game engine , Visual Studio 19 , Adobe Photoshop . All the components and scripts are executed on Windows using visual studio.

- **HARWARE INTERFACE**

  There is no constraint on which kind of hardware must be used. There are common hardware devices that are enough to interact with the game. These are :

  • monitor screen: Screen provides visual information to user.
  • keyboard: Keyboard provides user to control the game object.
  • mouse: Mouse is the main tool for navigating in game.
  • speaker: Speaker allows user to hear the background music.

SOFTWARE REQUIREMENT ANALYSIS

- **SOFTWARE INTERFACE**

  The required software products for Leviator are:

  •Unity game engine

  •Visual Studio 19

  • Adobe Photoshop

- **PRODUCT FUNCTION**

  PLAY BUTTON : This button is displayed on the user interface when the game is started and by pressing it the game will begin and score will start counting.

  HIGHSCORE : This shows the score of current game player is in and operating and is displayed at the end of the game in blue color .

  QUIT : This button helps user to go out of the game and resume their work

- **USER CHARECTERSTICS**

  All game players from all over the world will be a potential user for the product. There is no age limit. The only constraint for the users is being familiar with the rules of the games.

- **CONSTRAINTS**

  Multiplayer is constraint for this game since itsan offline game and dose not have server so it can not be played with other players online or offline ,though you can always compete with your high scores.

SOFTWARE REQUIREMENT ANALYSIS

- **USABILITY**

  Leviator will be very easy to play and enjoyable .It will not waste the time for users as they will directly enter the game by pressing play . Game is very user friendly for beginners as it dose not requires pre training or any kind of experience before operating this game.

- **RELIABILITY**

  The game is very reliable and is patch free currently and if any error will occur it will solved with further updates to remove the patches and error .

  There will also be updates to add further levels and new maps and upgrade the current setup to provide user with new and new experience.

- **AVILABILITY**

  Since the game is offline , there are no chance of server failure so there wont be any such game crash as it can easily be run on basic systems as well .

- **SECURITY**

  Leviator game security will be provided to all the users and the game doesn't ask for any personal information and neither has any server to store any confidential information.

- **MAINTAINABILITY**

  Our design is flexible. Whenever any functionality is needed for the application, it can be added with provided upgrades.

SOFTWARE REQUIREMENT ANALYSIS

- **PORTABILITY**

  This game is currently only available for Windows and soon will be available for MAC and LINUX as well.

- **SCRIPTS**

  - FINISH LINE :

```
using UnityEngine;

public class FinishLine : MonoBehaviour
{
    [HideInInspector]       public bool isReady;

    public bool debugMode;



    void OnTriggerEnter(Collider other)
    {

        if ((isReady || debugMode) &&
    other.gameObject.CompareTag("PlayerSensor"))
        {

                GameManager.instance.PlayerCompletedLap();
                isReady = false;
        }
    }
```

```
    }
```

- VEHICLE MOVEMENT

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


public class GameManager : MonoBehaviour
{
    public static GameManager instance;

    [Header("Race Settings")]
    public int numberOfLaps = 3;
    public VehicleMovement vehicleMovement;

    [Header("UI References")]
    public ShipUI shipUI;
    public LapTimeUI lapTimeUI;

    float[] lapTimes;
    int currentLap = 0;
    bool isGameOver;
    bool raceHasBegun;


    void Awake()
    {
        if (instance == null)
                instance = this;

        else if (instance != this)
```

```
            Destroy(gameObject);
}

void OnEnable()
{
    StartCoroutine(Init());
}

IEnumerator Init()
{
    UpdateUI_LapNumber();

    yield return new WaitForSeconds(.1f);

    lapTimes = new float[numberOfLaps];
    raceHasBegun = true;
}

void Update()
{

    UpdateUI_Speed ();



    if (IsActiveGame())
    {

            lapTimes[currentLap] += Time.deltaTime;
            UpdateUI_LapTime();
    }
}



public void PlayerCompletedLap()
```

```
{
    if (isGameOver)
            return;

    currentLap++;
    UpdateUI_LapNumber ();


    if (currentLap >= numberOfLaps)
    {
            isGameOver = true;
            UpdateUI_FinalTime();
            gameOverUI.SetActive(true);
    }
}

void UpdateUI_LapTime()
{
    if (lapTimeUI != null)
            lapTimeUI.SetLapTime(currentLap, lapTimes[currentLap]);
}

void UpdateUI_FinalTime()
{
    if (lapTimeUI != null)
    {
            float total = 0f;

            for (int i = 0; i < lapTimes.Length; i++)
                    total += lapTimes[i];

            lapTimeUI.SetFinalTime(total);
    }
}
```

```
void UpdateUI_LapNumber()
{
    if (shipUI != null)
            shipUI.SetLapDisplay (currentLap + 1, numberOfLaps);
}

void UpdateUI_Speed()
{
    if (vehicleMovement != null && shipUI != null)
            shipUI.SetSpeedDisplay (Mathf.Abs(vehicleMovement.speed));
}

public bool IsActiveGame()
{
    return raceHasBegun && !isGameOver;
}

public void Restart()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
}
```

SOFTWARE REQUIREMENT ANALYSIS

- LAP CHECKER

```
using UnityEngine;

public class LapChecker : MonoBehaviour
{
    public FinishLine finishLine;

    void OnTriggerEnter(Collider other)
    {

        if (other.gameObject.CompareTag("PlayerSensor"))
        {
                finishLine.isReady = true;
        }
    }
    }
```

- LAP TIME

```
using UnityEngine;

using TMPro;

public class LapTimeUI : MonoBehaviour
{
    public TextMeshProUGUI[] lapTimeLabels;
    public TextMeshProUGUI finalTimeLabel;

    void Awake()
    {
        for (int i = 0; i < lapTimeLabels.Length; i++)
                lapTimeLabels[i].text = "";
```

```
    finalTimeLabel.text = "";
}

public void SetLapTime(int lapNumber, float lapTime)
{


    if (lapNumber >= lapTimeLabels.Length)
            return;

    lapTimeLabels[lapNumber].text = ConvertTimeToString(lapTime);
}

public void SetFinalTime(float lapTime)
{
    finalTimeLabel.text = ConvertTimeToString(lapTime);
}

string ConvertTimeToString(float time)
{
    int minutes = (int)(time / 60);
    float seconds = time % 60f;

    string output = minutes.ToString("00") + ":" + seconds.ToString("00.000");
    return output;
}
}
```

SOFTWARE REQUIREMENT ANALYSIS

- PLAYER INPUT

```
using UnityEngine;

public class PlayerInput : MonoBehaviour
{
    public string verticalAxisName = "Vertical";
    public string horizontalAxisName = "Horizontal";
    public string brakingKey = "Brake";



    [HideInInspector] public float thruster;
    [HideInInspector] public float rudder;
    [HideInInspector] public bool isBraking;

    void Update()
    {

        if (Input.GetButtonDown("Cancel") && !Application.isEditor)
                Application.Quit();

        if (GameManager.instance != null &&
    !GameManager.instance.IsActiveGame())
        {
                thruster = rudder = 0f;
                isBraking = false;
                return;
        }

        thruster = Input.GetAxis(verticalAxisName);
        rudder = Input.GetAxis(horizontalAxisName);
        isBraking = Input.GetButton(brakingKey);
    }
    }
```

SOFTWARE REQUIREMENT ANALYSIS

- SHIP UI

```
using UnityEngine;
using TMPro;

public class ShipUI : MonoBehaviour
{
    [Header("UI Text References")]
    public TextMeshProUGUI currentSpeedText;
    public TextMeshProUGUI currentLapText;

    public void SetLapDisplay(int currentLap, int numberOfLaps)
    {
        if (currentLap > numberOfLaps)
                return;

        currentLapText.text = currentLap + "/" + numberOfLaps;
    }

    public void SetSpeedDisplay(float currentSpeed)
    {
        int speed = (int)currentSpeed;
        currentSpeedText.text = speed.ToString();
    }
    }
```

SOFTWARE REQUIREMENT ANALYSIS

- **VEHICLE MOVEMENTS**

```
using UnityEngine;

public class VehicleMovement : MonoBehaviour
{
    public float speed;

    [Header("Drive Settings")]
    public float driveForce = 17f;
    public float slowingVelFactor = .99f;
    public float brakingVelFactor = .95f;
    public float angleOfRoll = 30f;

    [Header("Hover Settings")]
    public float hoverHeight = 1.5f;
    public float maxGroundDist = 5f;
    public float hoverForce = 300f;
    public LayerMask whatIsGround;
    public PIDController hoverPID;
```

```
[Header("Physics Settings")]

public Transform shipBody;

public float terminalVelocity = 100f;

public float hoverGravity = 20f;

public float fallGravity = 80f;


Rigidbody rigidBody;

PlayerInput input;


float drag;

bool isOnGround;




void Start()

{

    rigidBody = GetComponent<Rigidbody>();

    input = GetComponent<PlayerInput>();




    drag = driveForce / terminalVelocity;

}
```

```
void FixedUpdate()
{
    speed = Vector3.Dot(rigidBody.velocity, transform.forward);

    CalculatHover();
    CalculatePropulsion();
}

void CalculatHover()
{
    Vector3 groundNormal;

    Ray ray = new Ray(transform.position, -transform.up);
    RaycastHit hitInfo;
    isOnGround = Physics.Raycast(ray, out hitInfo, maxGroundDist,
whatIsGround);

    if (isOnGround)
    {
        float height = hitInfo.distance;
        groundNormal = hitInfo.normal.normalized;
```

```
            float forcePercent = hoverPID.Seek(hoverHeight, height);
            Vector3 force = groundNormal * hoverForce * forcePercent;
            Vector3 gravity = -groundNormal * hoverGravity * height;
            rigidBody.AddForce(force, ForceMode.Acceleration);
            rigidBody.AddForce(gravity, ForceMode.Acceleration);
        }
        else
        {
            groundNormal = Vector3.up;
            Vector3 gravity = -groundNormal * fallGravity;
            rigidBody.AddForce(gravity, ForceMode.Acceleration);
        }
        Vector3 projection = Vector3.ProjectOnPlane(transform.forward,
groundNormal);
        Quaternion rotation = Quaternion.LookRotation(projection, groundNormal);
        rigidBody.MoveRotation(Quaternion.Lerp(rigidBody.rotation, rotation,
Time.deltaTime * 10f));
        float angle = angleOfRoll * -input.rudder;
        Quaternion bodyRotation = transform.rotation * Quaternion.Euler(0f, 0f,
angle);
        shipBody.rotation = Quaternion.Lerp(shipBody.rotation, bodyRotation,
Time.deltaTime * 10f);
}
```

```
void CalculatePropulsion()

{

    float rotationTorque = input.rudder - rigidBody.angularVelocity.y;

    rigidBody.AddRelativeTorque(0f, rotationTorque, 0f,
ForceMode.VelocityChange);

    float sidewaysSpeed = Vector3.Dot(rigidBody.velocity, transform.right);

    Vector3 sideFriction = -transform.right * (sidewaysSpeed /
Time.fixedDeltaTime);

    rigidBody.AddForce(sideFriction, ForceMode.Acceleration);



    if (input.thruster <= 0f)

        rigidBody.velocity *= slowingVelFactor;



    if (!isOnGround)

        return;




    if (input.isBraking)

        rigidBody.velocity *= brakingVelFactor;

    float propulsion = driveForce * input.thruster - drag * Mathf.Clamp(speed, 0f,
terminalVelocity);

    rigidBody.AddForce(transform.forward * propulsion,
```

```
ForceMode.Acceleration);

}


void OnCollisionStay(Collision collision)

{

    if (collision.gameObject.layer == LayerMask.NameToLayer("Wall"))

    {

        Vector3 upwardForceFromCollision = Vector3.Dot(collision.impulse,
transform.up) * transform.up;

        rigidBody.AddForce(-upwardForceFromCollision,
ForceMode.Impulse);

    }

}


public float GetSpeedPercentage()

{

    return rigidBody.velocity.magnitude / terminalVelocity;

}

}
```

SOFTWARE DESIGN

3.1 DATA MODEL AND DESCRIPTION

- **DATA DESCRIPTION**

  The project will include main menu ,a jet , a race track and a desert terrain.

- **DATA OBJECTS AND RELATIONSHIP**

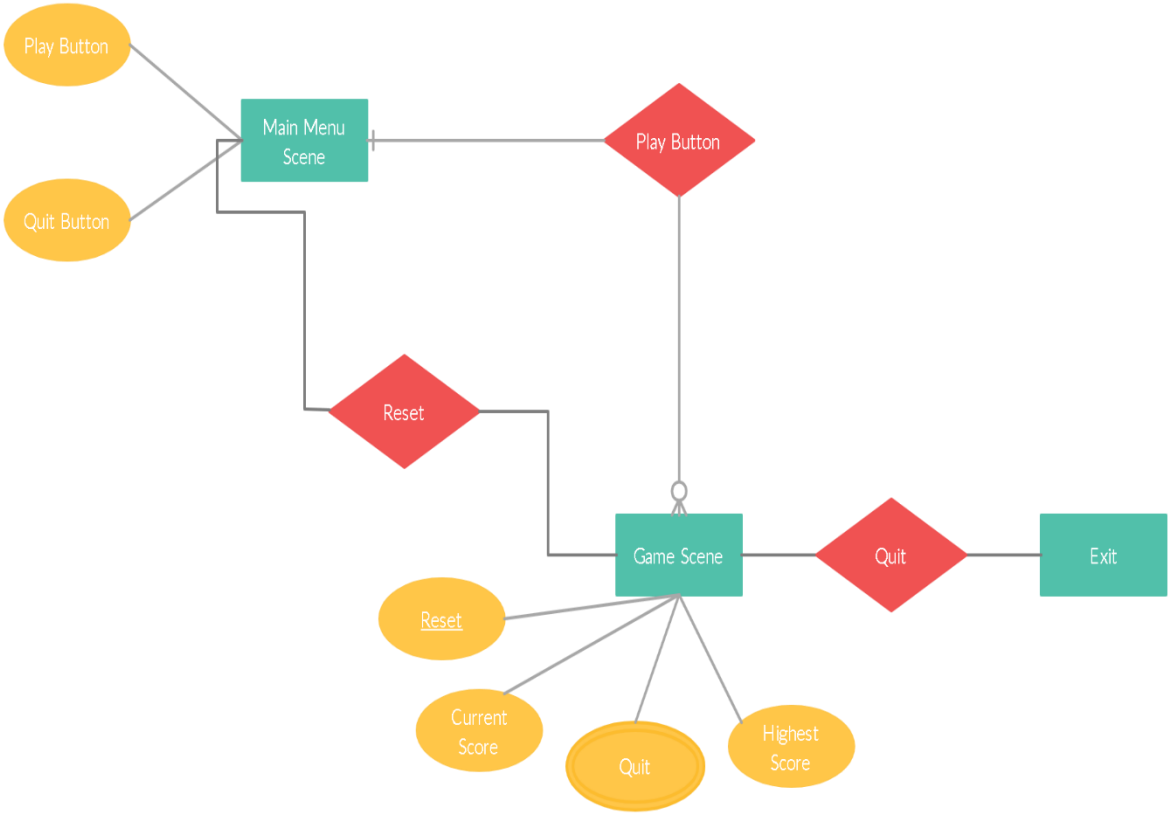
  The game revolves around a jet car which moves on a frictionless surface . Friction less surface is randomly generated with the help of the script attached to it .The jet moves forward and backward as per the user wants to control it . The jet, if collides with the borders of the track will cause it to slow down which will cause the player to take more time in order to complete the race .There in total are 3 laps at the end of which time taken will be displayed and at the end of the laps the total time taken will be displayed .

3.2 MODEL DAIGRAM

- **USE CASE DAIGRAM**

SOFTWARE DESIGN

- **ER DAIGRAM**



TESTING

4.1 IMPORTANCE

The importance of software testing and its implications with respect to software quality cannot be overemphasized. Software testing is a crucial element of software quality assurance and represents the ultimate review of specifications, design, and coding.

The increasing visibility of software as a system element and the attendant “cost” associated with a software failure are forces for well planned, through testing. It is not to expand 40% of total project effort on testing.

Detect can be caused by a flow in the application software or by a flow in the application specification. A structured approach to testing should use both dynamic and static testing rules:-

TESTING RULES :

1. Test early and test often.
2. Integrate the application development and testing life cycles, you will get better results.
3. Develop a comprehensive test plan; it forms the basic for the testing methodology.
4. Use both static and dynamic testing.
5. Define your expected results.
6. Understands the business reason behind the application. You will get a better application.
7. Use multiple levels and types of testing.

IMPLEMENTATION AND USER INTERFACE

4.2 TEST CASES

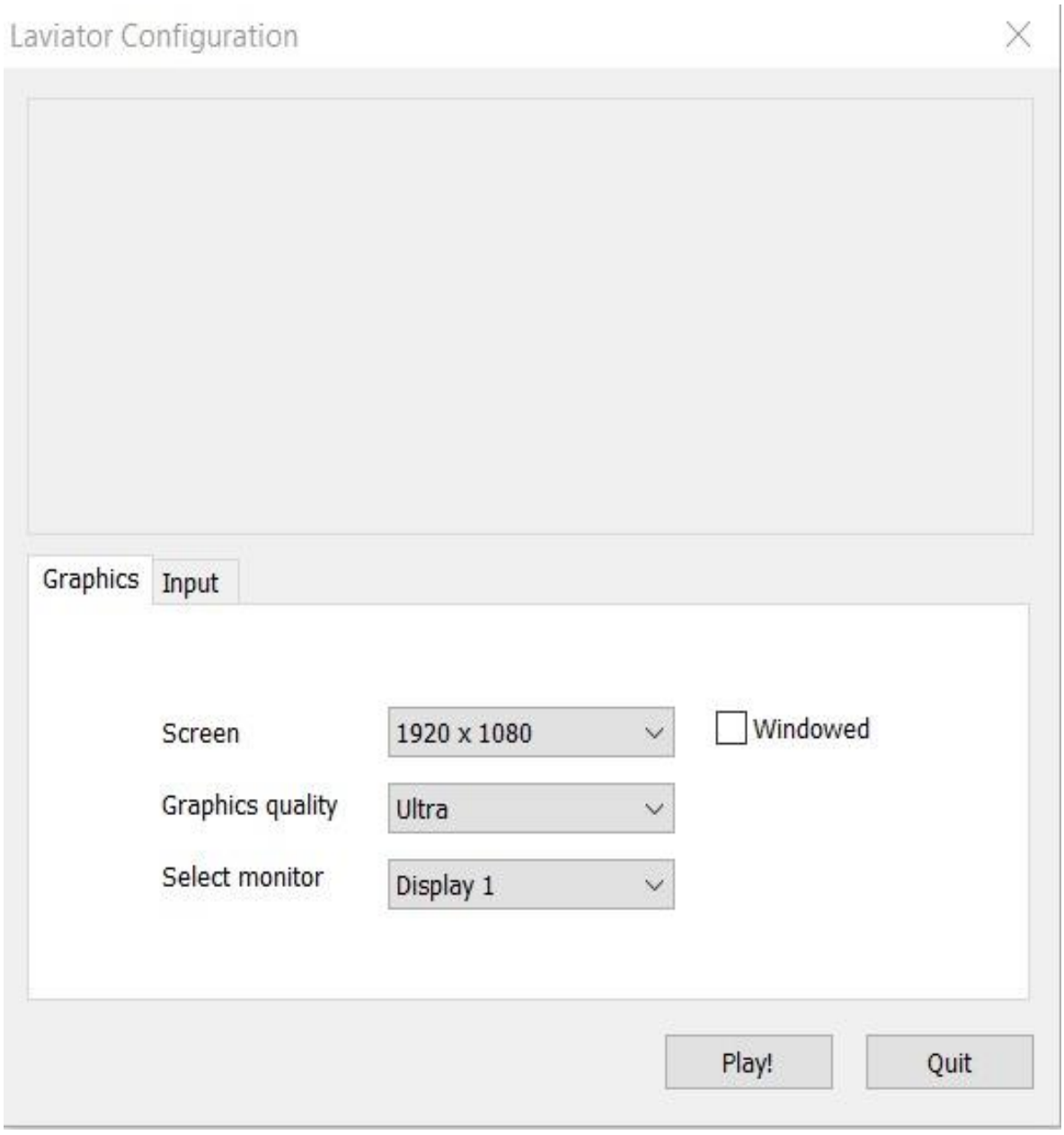
S.NO	DISCRIPTION	STATUS
1	**Resolution changing** Users cannot use the custom resolution.The game will run in the provided resolution only	**Success**
2	**Graphics alteration** Users cannot change the graphic settings of the game. It will run on its predefined graphics settings only.	**Success**
3	**Game pause** Users cannot pause the game once the ball has started to roll . Quite and Reset will be there only option available.	**Failed**
4	**Volume change** Users can change the volume of the game with there system volume keys to adjust the background music	**Success**
5	**Controls change** Users can not change the controls of the game according to there will.	**Success**
6	**Changing of game boundries** Users cannot change the boundries of game objects and enviornments	**Failed**

7	**Quiting the Game** The users can exit the game by pressing the quit button available in main menu	**Success**
8	**Changing the Main Menu** The users can not change the main menu according to there wish	**Failed**
9	**Play** The users can play the game by pressing play button available in main menu.	**Success**

IMPLEMENTATION AND USER INTERFACE

5.1 SAMPLE OUTPUT

- **START MENU**



IMPLEMENTATION AND USER INTERFACE

- **FIRST IMPRESSION**

- **VISUAL OVERVIEW**

IMPLEMENTATION AND USER INTERFACE

5.2 USER INTERFACE

- **TAP TO START :** This button is used to start the game . When user will press this button the game will start imidiately.
- **HIGH SCORE :** This menu shows the highest score which was scored by player till now.
- **SCORE :** This menu shows the current score scored by player in the following match.
- **QUIT :** :This button is used to exit the game and for this the user can bined a key according to his convenience.
- **RESTART :** This button is used to restart the game and this button is displayed only If theplayer is able to complete all the three laps .
- **Up and down arrow keys :** Initially the up and down arrow keys and binded to move the jet forward and backward . Using the Up arrow user will be able to move forward and using the down arrow , brakes can be applied .

REFRENCES/BIBLIOGRAPHY

6.1 APPENDIX 1 : Video Game And Console History Chart

Major Video Game and Handheld Consoles

Console Name	Manufacturer	U.S. Release	System Highlights*	Popular Games
Atari 2600	Atari	1977	First home game console to experience extraordinary success, selling more than 30 million units.*	*Pac-Man; Pitfall!*
NES	Nintendo	1985	The NES revived the floundering video game market and has since attained iconic status, selling nearly 62 million units worldwide.	*Super Mario Bros. series; Legend of Zelda; Tetris*
Genesis	Sega	1988	When Sega premiered its 16-bit system, the world fell in love with a blue hedgehog named Sonic. Thirteen million units were sold in the U.S. in just five years.	*Sonic the Hedgehog series; Mortal Kombat 2*
Game Boy	Nintendo	1989	Nintendo's first portable game player, and its later color version, is one of the most successful systems ever, selling more than 118 million units worldwide.	*Tetris; Pokemon; Super Mario Land*
Super Nintendo (SNES)	Nintendo	1991	Nintendo's momentum continued with its 16-bit system, outselling the competition at nearly 50 million units worldwide.	*Donkey Kong Country; Super Mario Kart; Street Fighter II*
PlayStation	Sony	1995	Sony proved the power of a brand with their 32-bit system, which used CDs rather than game cartridges, shipping more than 102 million units through 2006.	*Gran Turismo; Final Fantasy VII; Tomb Raider series*
PS2	Sony	2000	Sony's second system proved even more successful than the first, with more power and a DVD player selling over 100 million units.	*Grand Theft Auto series; Gran Turismo series*
Xbox	Microsoft	2001	Microsoft's first venture into the hardware industry had many computer-like attributes and has sold more than 24 million units.	*Halo series; Tom Clancy's Splinter Cell; Madden NFL*
DS	Nintendo	2004	Nintendo's latest handheld offered players two screens, one which is touch sensitive. The DS and DS Lite have sold more than 21 million units since launch.	*Nintendogs; Pokemon; Brain Age*
PSP	Sony	2005	Sony's response to the DS focused more on power than innovation, and sales have held strong at more than 20 million units.	*Grand Theft Auto: Vice City Stories; Monster Hunter*
Xbox360	Microsoft	2005	Xbox was the first to strike in the newest generation of gaming consoles. The 360 offered online gaming through Xbox Live and has sold 19 million units worldwide.	*Halo 3; Gears of War; Call of Duty 4*
PS3	Sony	2006	The most advanced and highest priced of the newest batch of consoles, the PS3 allows you to play Blu-Ray movies. Thus far, PS3 sales globally have passed 12 million.	*MotorStorm ; Resistance: Fall of Man; Grand Theft Auto IV*
Wii	Nintendo	2006	Offering innovation over sheer power, the Wii gives players a motion-sensitive controller and focuses on games that are fun, social and active, rather than intense. The Wii has sold more than 25 million units globally.	*Wii Play; Super Mario Galaxy; Super Smash Bros. Brawl*

** Data regarding the number of consoles sold (with the exception of the Xbox 360, PS3 and Wii) comes from an online Business Week slideshow, "A Brief History of Game Console Warfare." Available online at http://images.businessweek.com/ss/06/10/game_consoles/source/1.htm . Data on the three most recent systems comes from a May 22, 2008 article in DailyTech, available online at http://www.dailytech.com/Microsoft+First+to+100+Million+Consoles+Sold+Wins+War/article11792.htm .*

6.2 APPENDIX 2 :Regresion Analysis

The findings regarding the relationship between frequency, social context and civic qualities of gaming experiences and life civic outcomes were derived using regression analysis. This statistical technique allows us to pinpoint whether a relationship between different gaming experiences and civic and political outcomes exists after controlling for factors such as income, race, gender and parent involvement—all individual characteristics that have been previously found to be important predictors of civic and political engagement.

Table 1: Relationship between frequency of game play and civic and political outcomes

	Civic and Political Outcomes					
	Get info. about Politics	**Volunteer**	**Charity**	**Stay Informed**	**Protest**	**Political Interest**
	Exp(B)	**Exp(B)**	**Exp(B)**	**Exp(B)**	**Exp(B)**	**Exp(B)**
Demographic Variables						
Income	1.062	1.084	.977	1.104*	.920	1.080
Parent Hispanic	1.631	.619*	.892	.835	1.366	.771
Parent African American	1.152	.682	.802	1.117	1.120	1.208
Parent Other	2.582*	1.630	1.010	1.990*	.529	1.207
Child age (older)	1.426*	1.361*	1.091	1.982***	1.027	1.705***
Child sex (female)	1.013	1.213	1.305	1.180	1.585	1.090
Parent Involvement						
Parent volunteered	--	2.208***	--	--	--	--
Parent charity	--	--	2.047***	--	--	--
Parent protested	--	--	--	--	4.901***	2.277*
Parent stays informed	1.156	--	--	2.575***	--	.935
Frequency of Game Play						
Some Games (vs. little/none)	1.044	.982	1.187	1.078	2.545*	1.684**
Frequent Games (vs. little/none)	.677	.698	.939	.781	1.878	1.265
R^2	.046**	.102***	.051***	.119***	.065**	.054**

Note: For two of the civic and political outcomes measured, persuading others how to vote in an election and commitment to civic participation, the omnibus test was non-significant. Those outcomes are excluded from the table.

THE END