Computer Graphics & Multimedia

PROJECT REPORT

GROUP MEMBERS

Romy Savin Peter

Emma Mary Cyriac

Riya Rajesh

Sai Krishna Kotina

MIT License

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION

1.1 Problem Statement

Games have always been an integral part of a person's early childhood development. It is a source of entertainment, encourages creativity and provides a means to take a break from the pressures of life. Board games were a popular genre in India. Kids used to play games such as Carrom, Ludo, Snakes & Ladders with their parents or family members. With the advent of video games, such classics are rarely given any attention. Therefore, we decided to create a digital version of one such classic game: Snakes & Ladders, which can be enjoyed in the way it used to be: with family and friends!

However, creating a video game is not so easy because it needs to have a proper balance of design, gameplay and immersion. It should have proper mechanics and passable graphic design. Hence, the general objective of this project was to develop a 2D game that has replay value, a good foundation about game mechanics and gameplay immersion. As such, we have made sure that our game meets the following baseline requirements:

- Contains a main menu to access the game’s interface.
- The game must be able to store and display stored data (such as player number, dice thrown in this case).
- The game must accept at-least one method of input (Say, mouse input for actions like throwing the dice, stopping it and so on).

1.2 Abstract

Snakes & Ladders is an ancient Indian board game regarded today as a worldwide classic. It is played between two or more players on a gameboard having numbered, gridded squares. A number of "ladders" and "snakes" are pictured on the board, each connecting two specific board squares. The objective of the game is to navigate one's game piece (also called a coin), according to die roll values, from the start (bottom square) to the finish (top square), helped or hindered by ladders and snakes respectively.

The game is a simple race contest based on sheer luck, and is popular with young children. The historic version had root in morality lessons, where a player's progression up the board represented a life journey complicated by virtues (ladders) and vices (snakes).

1.3 Introduction to the Project

The small project we designed and implemented here is ‘Snake and Ladders’, a very old Indian board game. It is usually played between two or more players on a gameboard having numbered, gridded squares. A number of "ladders" and "snakes" are pictured on the board, each connecting two specific board squares. The object of the game is to navigate one's game piece, according to die rolls, from the start (bottom square) to the finish (top square), helped or hindered by ladders and snakes respectively.

Here, we have developed meshes as base of our game board. We the map images to ensure proper graphics for enhanced gaming experience. User can exit game at any moment by pressing the 'Q' key. We have kept multiple frames in order to change the user’s perspective of the game flow eventually.

The project shows the order of events in three windows:

1. First window shows homepage and player selection.
2. Second window shows rules and instructions to play the game.
3. Third window shows the game board.
4. Fourth window shows the winner information.

2.0 SYSTEM REQUIREMENTS

2.1 Hardware & Software Requirements

For the successful, efficient and error-free functioning of any program that involves graphics, the system should meet some requirements. In this section, the various requirements that are necessary for this program are specified.

- Hardware requirements

    - CPU: Any x64-based processor
    - RAM: 1 GB or more
    - Graphics memory: 64 MB or more
    - Storage: 40 GB or more

- Software requirements

    - OS: Windows / Linux
    - Platform / Language: C++ with OpenGL as API
    - Libraries: LodePNG (included with program)
    - Software(s): Any text editor, GCC compiler, GLUT libraries

2.2 Installation Instructions

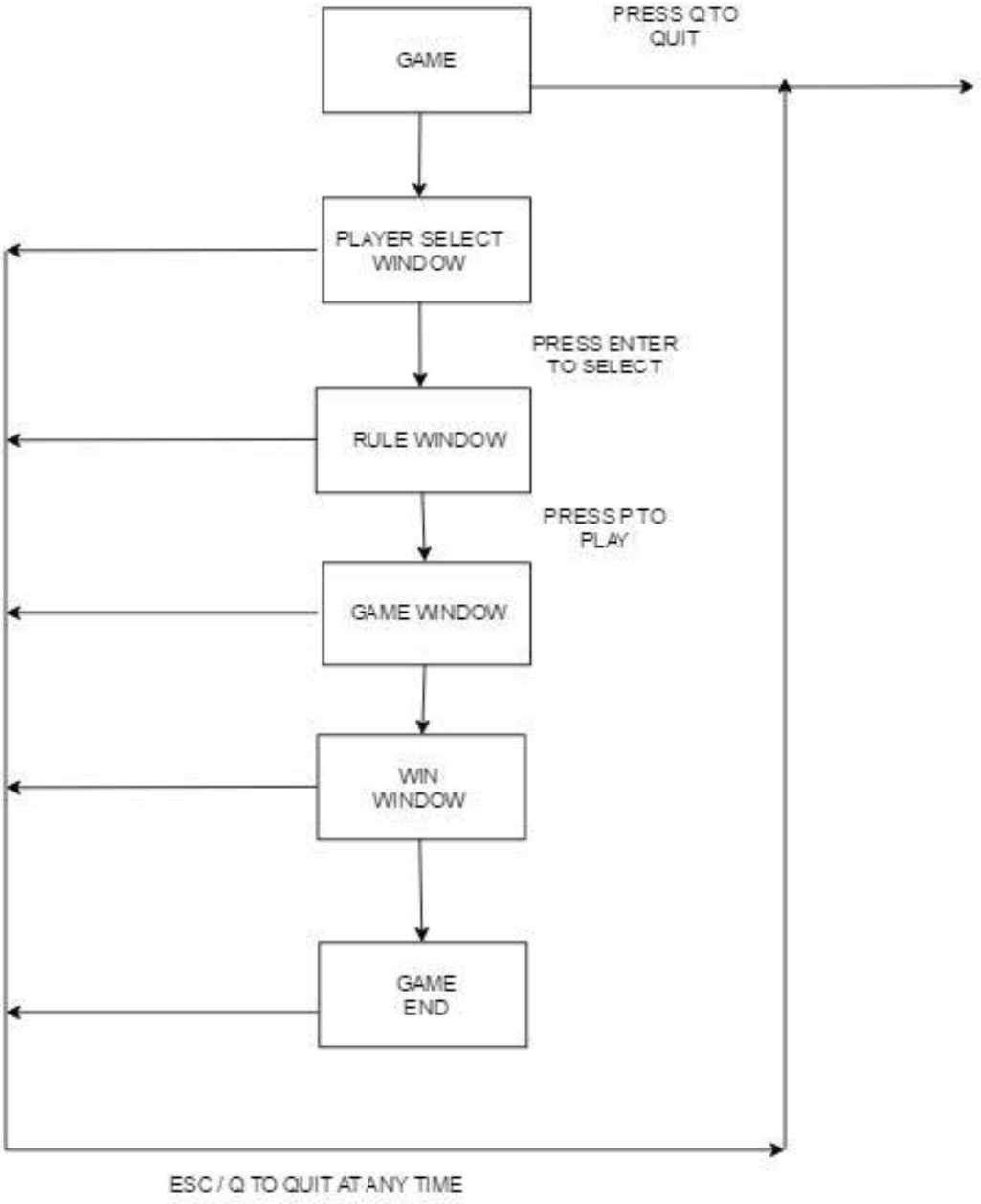
The program is pretty straightforward to compile and run. All required external libraries are already supplied with the code and a makefile is included for easy compilation.

- To compile the program, cd to the directory it is stored in and type "`make`"
- Subsequently, to run the program, type "`. /snake`"
- The rules and further instructions are mentioned in the game for reference.

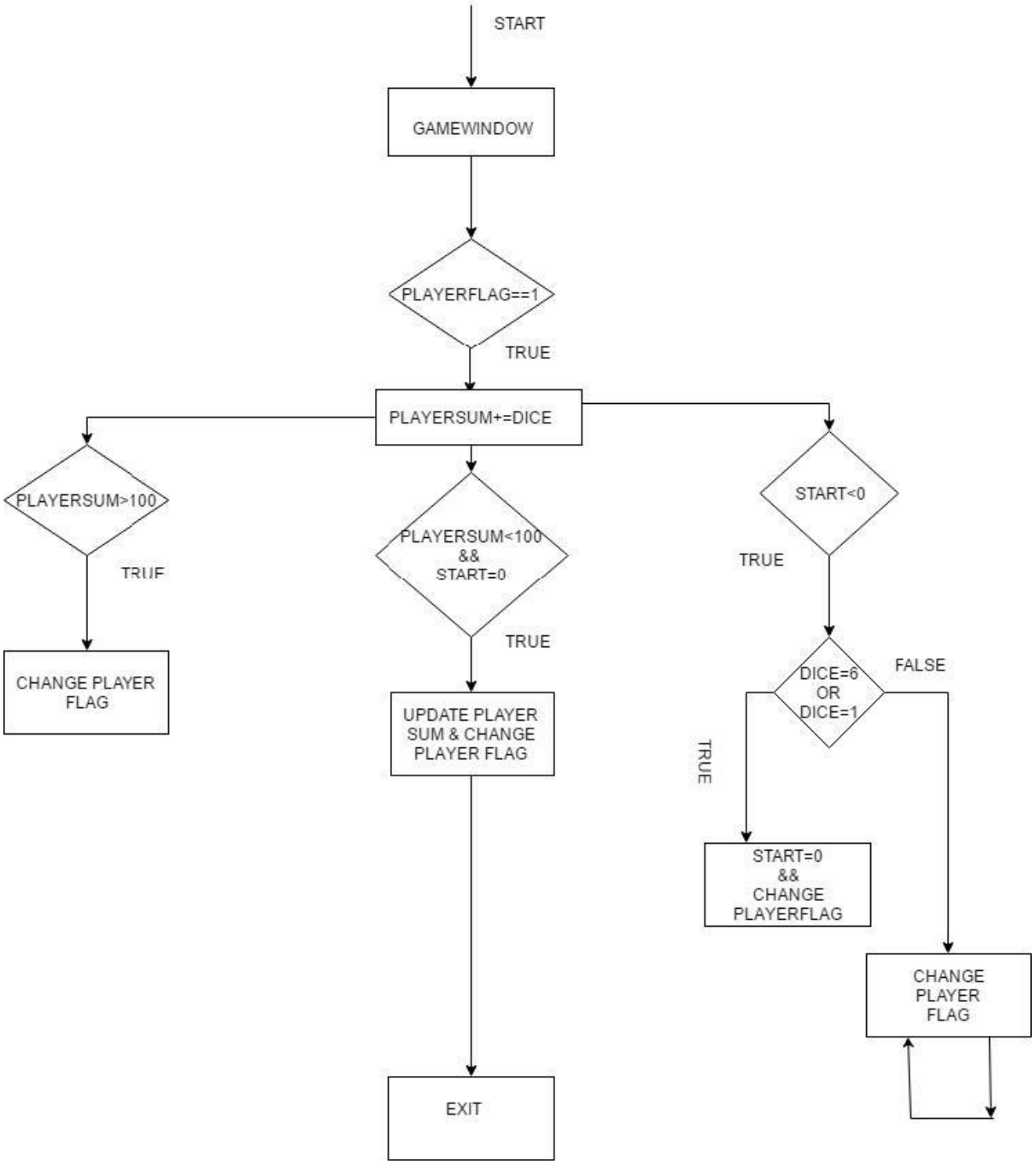
3.0 PROGRAM DESIGN

3.1 Implementation

- The below given flow chart gives a pictorial representation and basic overview about the data flow control from one block to another.



- A more detailed view of the gameplay and it's functioning is shown in the below given flowchart.



3.2 Source code

- The header files defined here are for the standard library, the graphics library and the user defined header file in which the primitives and other built-in functions related to image handling (PNG) are defined. This is followed by the variables required for smooth functioning of the game.

```
/***** All header files needed by the program are defined below  *****/

//Required OpenGL header files
#include <GL/glut.h>
#include <GL/glext.h>

//Required standard C/C++ header files
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream>
#include <vector>

//Custom user defined header file required to load images (PNG)
#include "lodepng.h"
```

```
//The below defined variables are used for window transitions and rendering
int windowWidth;
int windowHeight;
bool window1 = false;
bool window2 = false;
bool window3 = false;
bool window4 = false;

//The below defined variables are used for gameplay
int flag;»                                  //Observes if the mouse input is a right click or not
int n = 0;»                                 //Stores the image loading flag
float spin;»                                //Stores the spinning factor of the cube
int winner;»                                //Stores the winning player number
int dice[4];»                               //Stores the dice values of each player
int dicenum;»                               //Stores the dice throw value
int numplayers = 0;»                        //Stores the number of players
int pc_counter = 1;»                        //Stores chances condition factor
void *currentfont;»                         //Stores the address of the font
int set_pointer = 0;»                       //Set program counter
int select_flag = 0;»                       //Stores the number of places as specified by the user
int snake_pos[101];»                        //Stores the count of snake heads in the mesh
int stair_pos[101];»                        //Stores bottom of the ladders in the mesh
int currentplayer = 1;»                     //Stores the value of current player flag
int dice_position = -1;»                    //Stores the value of dice movement
int player_sum[4] = { 0 };»                  //Stores the current dice sum for every player
float dice_dimension = 50;»                 //Stores the dice dimension
int player_flag[4] = { 1, 0, 0, 0 };»       //Stores the player which has current chance
float start[4] = { -70, -70, -70, -70 };»   //Start positions of the players
float right_movement[4] = { 0 };»           //Monitors the player's horizontal position
float up_movement[4] = { 0 };»              //Monitors player's vertical position
```

- **Crucial functions used in the program:**

```
//The below defined functions are responsible for loading images
void setTexture(vector < unsigned char > img, unsigned width, unsigned height);
void invert(vector < unsigned char > &img, const unsigned width, const unsigned height);
void loadImage(const char *name, int n);

//The below defined functions are responsible for stroke drawing
void drawStrokeText(const char str[250], int x, int y, int z, float p1, float p2);
void setFont(void *font);
void drawstring(float x, float y, char *str);
```

```
//The below defined functions are required by the first output window (player selection menu)
void windowOne();
void drawoptions();
void selectoptions();

//The below defined function is required by the second output window (rules)
void windowTwo();

//The below defined functions are required by the third output window (gameplay area)
void windowThree();
void drawMesh();
void drawplayer();
void drawdice();
void spinDice();
void gameplay();
void diceimages();
void diceposition();
void check_ladder();
void check_snake();

//The below defined function is required by the fourth output window (displaying the winner)
void windowFour();
```

```
/***** All GLUT functions with altered definitions are defined below *****/

static void init(void);
static void idle(void);
static void display(void);
static void key(unsigned char key, int x, int y);
static void specialkeys(int key, int x, int y);
void mouse(int button, int state, int x, int y);
```

- These are some crucial functions, which are required for successful compilation and working of the program.
- The 'main' function handles control flow as the starting function.
- There are 4 windows that facilitates the proper flow of the game from player selection menu to displaying the winner.

- The **main( )** function contains the GLUT graphics engine initialization calls and other actions graphics engine related calls.

```
int main(int argc, char *argv[])
{
    //Responsible for loading the image into memory
    loadImage("logo.png", n);
    n = 1;
    loadImage("board.png", n);

    //Responsible for generating textures
    glGenTextures(1, &texname);

    //GLUT initialization call
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);

    //Defines the window attributes
    glutInitWindowSize(WIDTH, HEIGHT);
    glutInitWindowPosition(10, 10);
    glutCreateWindow("Snake and Ladders");
    windowWidth = glutGet(GLUT_WINDOW_WIDTH);
    windowHeight = glutGet(GLUT_WINDOW_HEIGHT);

    //Other initialization calls
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_BLEND);

    //Function calls
    init();
    glutFullScreen();
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutSpecialFunc(specialkeys);
    glutIdleFunc(idle);
    glutMouseFunc(mouse);

    //Responsible for continuously rendering the buffer
    glutMainLoop();

    return EXIT_SUCCESS;
}
```

```
static void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glViewport(0, 0, WIDTH, HEIGHT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 1000, 0, 1000, 0, 1000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

- **init( )** is used to set the viewport and camera projection along with the coordinate system.

```
static void display(void)
{
    if (!window2)
        windowOne();
    else if (!window3)
        windowTwo();
    else if (!window4)
        windowThree();
    else
        windowFour();
}
```

- **display( )** is used to output desired frame/window.

- **key( )** is used to define keyword to perform specific functions and changing frames on pressing the defined key.

```
static void key(unsigned char key, int x, int y)
{
    if (key == 'q' || key == 'Q' || key == 27) //This is the key to quit the game, which can be modified (default: Q)
    {
        exit(1);
    }
    else if (key == 13)//This key is used to select the number of players, which can be modified (default: Enter)
    {
        window2 = true;
    }
    else if (key == 'p' || key == 'P') //This is the key to start the game, which can be modified (default: P)
    {
        window3 = true;
    }
}
```

- **specialkeys()** is used to select the number of players by using the left & right keys.

```
static void specialkeys(int key, int x, int y)
{
    if (key == GLUT_KEY_RIGHT)
    {
        select_flag = (select_flag + 1) % 3;
    }
    else if (key == GLUT_KEY_LEFT)
    {
        select_flag--;
        if (select_flag < 0)
            select_flag = 2;
    }

    printf("\nselect_flag: %d\n", select_flag);
}
```

- **idle( )** is used to re-display the window or (calling of concurrent display function, simply put)

- **mouse( )** enable us to take user input from mouse. Left click is used to spin the dice & right click is used to display the dice output and end the current player's turn.

- **setTexture( )** is used to generate textures from images used.

```
void setTexture(vector < unsigned char > img, unsigned width, unsigned height)
{
    glBindTexture(GL_TEXTURE_2D, texname);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

    // This ensures textures have correct brightness. Else, they tend to appear dark
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height,
        0, GL_RGBA, GL_UNSIGNED_BYTE, &img[0]);
}
```

- OpenGL loads image in an inverted way but we require it to be upright so we invert it using **Invert()**.

```
void invert(vector < unsigned char > &img, const unsigned width, const unsigned height)
{
    unsigned char *imageptr = &img[0];
    unsigned char *first = NULL;
    unsigned char *last = NULL;
    unsigned char temp = 0;
    for (int h = 0; h < (int) height / 2; ++h)
    {
        first = imageptr + h *width * 4;
        last = imageptr + (height - h - 1) *width * 4;
        for (int i = 0; i < (int) width * 4; ++i)
        {
            temp = *first;
            *first = *last;
            *last = temp;
            ++first;
            ++last;
        }
    }
}
```

- **Loadimage( )** function is used to load the PNG images using LodePNG library.

```
void loadImage(const char *name, int n)
{
    //LodePNG Decode is used to decode input PNG images
    int error;
    if (n == 0)
    {
        if ((error = lodepng::decode(image_logo, logowidth, logoheight, name)))
        {
            cout << name << ":" << lodepng_error_text(error) << endl;
        }
        else
        {
            invert(image_logo, logowidth, logoheight);
            cout << "\nLogo was loaded successfully.\n";
        }
    }
```

- We have used Bitmap text from GLUT for displaying the text in the window.

- For placing the cursor and drawing bitmap character we use **glutBitmapCharacter**

- **glutBitmapCharacter** can print a single character at a time, thus a function **drawstring** is made which loops over the character array to print the whole passed string.

```
//Function responsible for setting the font
void setFont(void *font)
{
    currentfont = font;
}

//Function responsible for drawing strings
void drawstring(float x, float y, char *str)
{
    char *c;
    glRasterPos2f(x, y);
    for (c = str; *c != '\0'; c++)
        glutBitmapCharacter(currentfont, *c);

}
```

- **drawStrokeText( )** is used to render text on the display window.

```
void drawStrokeText(const char str[250], int x, int y, int z, float p1, float p2)
{
    int i;
    glPushMatrix();
    glTranslatef(x, y, z);
    glScalef(p1, p2, z);

    for (i = 0; str[i] != '\0'; i++)
    {
        glutStrokeCharacter(GLUT_STROKE_ROMAN, str[i]);
    }

    glPopMatrix();
}
```

- **drawoptions( ) and selectoptions( )** are used to input the number of players from the user(s). Here we use LEFT and RIGHT key to toggle the boxes which contain the number of players.

- **WindowOne( )** is used to showcase the content in the first window. This window is a welcome screen to the program. Button selection is done using the left & right keys. Enter key can then be used to make the selection.

```
void windowOne()
{
    glClear(GL_COLOR_BUFFER_BIT);
    float scale = 0.70;
    drawoptions();
    selectoptions();
    glPushMatrix();
    //Image begins here
    glEnable(GL_TEXTURE_2D);
    setTexture(image_logo, logowidth, logoheight);
    glPushMatrix();
    glTranslatef(300, 500, 0);
    glScalef(scale, scale, 1);
    glBegin(GL_POLYGON);
    glTexCoord2d(0, 0);
    glVertex2f(0, 0);
    glTexCoord2d(0, 1);
    glVertex2f(0, logoheight);
    glTexCoord2d(1, 1);
    glVertex2f(logowidth, logoheight);
    glTexCoord2d(1, 0);
    glVertex2f(logowidth, 0);
    glEnd();
    glDisable(GL_TEXTURE_2D);
    //Image ends here
    glPopMatrix();
    glutSwapBuffers();
}
```

- **WindowTwo( )** is used as an information window which conveys the instructions and rules for playing the game to the user. The instructions and rules are rendered using stroke functions.

```
void windowTwo()
{
    glClear(GL_COLOR_BUFFER_BIT);
    float xpos = windowWidth / 5;
    float ypos = windowHeight *3 / 4;
    float xtrans = windowWidth / 6;
    float ytrans = windowHeight;
    float fontsize = 0.13;

    glPushMatrix();
    glTranslatef(xtrans, ytrans, 0);
    glLineWidth(3.0);
    glColor3f(0.0, 1.0, 0.0);
    drawStrokeText("Snakes & Ladders - The Game of Chance", xpos, ypos, 0, 0.210, 0.210);
    glBegin(GL_LINES);
    glVertex2f(xpos, ypos - 15);
    glVertex2f(xpos + 620, ypos - 15);
    glEnd();
    glPopMatrix();
```

- **WindowThree( )** is the main gameplay window which shows the board and the dice. We update flags based on the mouse key press events. This enables player movement on the board.

- **drawMesh( )** is used to map the board of the game to facilitate player position and movement along the board.

```
void drawMesh()
{
    glLoadIdentity();
    glPushMatrix();
    glTranslatef(31, 81, -50.0);
    glPointSize(10.0);
    glScalef(0.9, 1, 1);
    for (int i = 0; i < pixelwidth; i += 70)
        for (int j = 0; j < pixelheight; j += 85)
        {
            glPointSize(4.0);
            glColor3f(1.0, 0.0, 0.0);
            glBegin(GL_LINE_LOOP);
            glVertex3f(i, j, 50);
            glVertex3f(i, j + 85, 50);
            glVertex3f(i + 70, j + 85, 50);
            glVertex3f(i + 70, j, 50);
            glEnd();
        }
}
```

- **drawPlayer( )** is used to draw the player's coin on the board.

```
void drawplayer()
{
    int pi = 3.14;
    float theta = 0, radius = 25;
    glPointSize(200.0);

    //Player 1
    glColor3f(1.0, 0.0, 1.0);
    glBegin(GL_POLYGON);
```

- **drawDice( )** is used to render the dice cube on the frame. Here we use OpenGL inbuilt functions to make the faces of the cube and connect them accordingly in a 3D plane

```
void drawdice()
{
    glColor3f(1, 0, 1);

    glBegin(GL_QUADS);
    //Top face
    glColor3f(1, 0, 1);
    glVertex3f(-dice_dimension, dice_dimension, +dice_dimension);
    glVertex3f(dice_dimension, dice_dimension, +dice_dimension);
    glVertex3f(dice_dimension, dice_dimension, -dice_dimension);
    glVertex3f(-dice_dimension, dice_dimension, -dice_dimension);
```

- **generate_num( )** is a function which outputs the dice values . It generates the dice numbers and return the same for proper gameplay. We use random function to generate numbers.

```
int generate_num()
{
    int chancenum;
    srand(time(NULL));
    chancenum = ((rand() % 6) + 1);

    if (chancenum == 0)
        dicenum = generate_num();
    else
        dicenum = chancenum;
    printf("\nDice number: %d\n", dicenum);
    return dicenum;
}
```

- **spinDice( )** is used to spin the cube in random directions in the 3D space according to programmer specified spin.

```
void spinDice()
{
    spin = spin + 50.0;
    if (spin > 360)
        spin -= 359;
    glutPostRedisplay();
}
```

- **gameplay( )** contains the main logic of the game. It enables us to:

  a. Move the players
  b. Flag the turns of the respective players
  c. Monitor encounters of snakes and ladders with the player
  d. Monitor the winner of the game.

- We have put multiple conditions to ensure the movement or turns of the respective players isn't missed upon failure. This is how the game works:

- The gameplay function uses array index data to monitor snake and ladders to and from their space positions on the board.

```
stair_pos[1] = 38;
stair_pos[4] = 14;
stair_pos[9] = 31;
stair_pos[21] = 42;
stair_pos[28] = 84;
stair_pos[36] = 44;
stair_pos[51] = 67;
stair_pos[71] = 91;
stair_pos[80] = 100;
```

- The game engine ensures proper management of turns among players by using player flag and monitoring their respective sum along the whole tenure of the gameplay.

- **diceImages( )** is used to display dice value after the number is generated for the respective player. We then use LodePNG here to load these images accordingly using **diceposition( )** function.

```
void diceimages( )
{
    if (dicenum == 1)
    {
        n = 11;
        loadImage("dice1.png", n);
    }
```

```
void diceposition( )
{
    spin = 0;

    //Player 1
    if ((pc_counter % numplayers) == 0)
    {
        if (dice[0] == 1)
        {
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity( );
            glEnable(GL_TEXTURE_2D);
            setTexture(image_dice1, dice1width, dice1height);
            glPushMatrix( );
            glTranslatef(850, 200, 0);
            glScalef(0.9, 1, 1);
            glBegin(GL_POLYGON);
            glTexCoord2d(0, 0);
            glVertex2f(60, -60);
            glTexCoord2d(0, 1);
            glVertex2f(60, 60);
            glTexCoord2d(1, 1);
            glVertex2f(-60, 60);
            glTexCoord2d(1, 0);
            glVertex2f(-60, -60);
            glEnd( );
            glPopMatrix( );
            glDisable(GL_TEXTURE_2D);
        }
```

- **WindowThree( )** function is used to render different images and figures whether 2D or 3D on the third window. This function holds all the related function calls and other required conditions to run the game engine in an optimized manner.


```
void windowThree()
{
    currentplayer = ((pc_counter + 1) % numplayers) + 1;
    int prev_player = currentplayer - 1;
    if (prev_player == 0)
    {
        if (numplayers == 2)
        {
            prev_player = 2;
        }
        else if (numplayers == 3)
        {
            prev_player = 3;
        }
        else
        {
            prev_player = 4;
        }
    }

    glPushMatrix();
    glTranslatef(900.0, 400.0, 0.0);
    glRotatef(spin, 1.0, 0.5, 1.0);
    if (dice_position < 0)
        drawdice();
    if (dice_position > 0)
        diceposition();

    glPopMatrix();

    glutSwapBuffers();
}
```

- After every frame change we use **glutSwapBuffer( )** to change the buffer used for rendering the frames in the game.

- **WindowFour( )** is used to display the winner information. This displays the congratulatory message and the player who reached the destination first and hence became the winner.

```
void windowFour()
{
    int num = 0;
    float cn = 500;
    num = (winner + 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glutIdleFunc(idle);

    glColor3f(1.0 *((rand() % 100) / 100.0), 1.0 *((rand() % 100) / 100.0), 1.0 *((rand() % 100) / 100.0));
    setFont(GLUT_BITMAP_HELVETICA_18);
    char name[100] = { "CONGRATULATIONS! THE WINNER IS PLAYER -->   " };
    char buffer[10] = { '\0' };
    drawstring(cn - 165, 500, name);
```

4.0 CONCLUSION

4.1 Takeaways

An attempt has been made to develop a 2D game with help of OpenGL, which meets the necessary expectations of the player as well as the outcomes of our course. It enabled us to learn about many of the concepts in OpenGL and given us an insight into the wide variety of uses involving Computer Graphics in general. We had to use considerable user defined functions along with a healthy variety of built-in functions too. This has given us a certain degree of familiarity with these functions and have now understood the utility of the same. We were able to comprehend the true nature of the best tools in OpenGL and have understood the reason why graphics is so crucial for providing an enjoyable experience in a game.

We can now converse with a certain degree of confidence regarding Computer Graphics topics along with the satisfaction of creating a game that is fun to play game with parents, friends & family members. We would like to end by saying that it has been a memorable experience over the course of which we have learned quite a bit and worked with group members who were previously unknown to us.

4.2 Future Work

Although we have tried our best to provide an all-rounded experience for players by using the knowledge that was available to us, there were things that could not be implemented due to paucity of time. As such, these can be improved in future iterations of the game. Some scopes of improvements are mentioned below:

- Various lighting effects can be implemented to show shadows.
- The game can be reworked in 3D with support for material properties.
- Better dice and movement animations can be implemented.
- Various user interactions can be enhanced by better graphics implementation.

END OF PROJECT REPORT