Department of Mathematics and Computing

Project Title : Design a Sudoku Game using Opengl

Subject : Computer Graphics

Submitted to : Dr Badam Singh Kushvah

Submitted by : < Group No. 4 >

Anshu Chaurasia (16JE002212)

Puneet Garg (16JE002217)

Sunny Kumar (16JE002246)

Achal Prakash Pandey (16JE002252)

Utkarsh Maddhesiya (16JE002255)

Nishant Bhaskar (16JE002260)

Akash Nirwan (16JE002297)

Objective:

The aim of this project is to design a 2-D Sudoku Solver Game using OpenGL

Rules to solve Sudoku:

Every row, column, and the square box should have all numbers between 1 to 9.

No number should be repeated in any row, column, and square.

Contribution:

- Anshu Chaurasia (16JE002212) – Sudoku Controller
- Puneet Garg (16JE002217) – Sudoku Model
- Sunny Kumar (16JE002246) – Sudoku Model

- Achal Prakash Pandey (16JE002252) – Sudoku View
- Utkarsh Maddhesiya (16JE002255) - Sudoku Model
- Nishant Bhaskar (16JE002260) – Sudoku Contoller
- Akash Nirwan (16JE002297) – Sudoku View

Methodology:

A. Create a struct named ‘Cell’ which will represent the cells in the Sudoku grid.

B. **Sudoku Model**

1. Declare an array of struct ‘Cell` of size 81 (9 x 9)
2. Create a valid board by filling in all the cells in such a way that all the rows, columns and boxes have digits from 1 to 9 without repetition.
3. Now, create the Sudoku puzzle by hiding some cell numbers such that a unique solution always exists from the new configuration formed after hiding some digits.
4. Check_validity function:

Checks if a number is present in the corresponding row, column and box. Used to fill in while solving.

5. NoUniqueSolution function:

Recursive function to check if current puzzle has a unique solution or not.

6. Print_board function:

Used to print the board on screen.

7. Find_unassigned_cell function:

Used to find any empty cell in the board

C. **Sudoku Controller**

This class contains functions for controlling the different stages of Sudoku solving and maintaining error lists as well as checking errors while solving.

1. ErrorList:

A member variable of type vector<int> to store list of cell positions which contain incorrect numbers (either row-wise, column-wise or box-wise)

2. CheckforErrors:

A function which is used to find out errors in the board such as duplicates in row, column and/or box.

3. CheckRowColBoxErrors:

A function which contains the logic for checking duplicates in row, column and box.

4. CheckExistingErrorCells:

A function to check if the existing errored cells are cleared or not.

5. SingleCellErrors:

D. **Sudoku View**

This class contains methods which control the view of the Sudoku grid and input features

1. reshape:

Function to resize the viewport and projection matrix according to input width and height. Called by GLUT.

2. display:

Displays the Sudoku grid as well as timer along with filled in cell numbers. Called by GLUT.

3. keyboard:

Sets number in selection box according to the input given by user.

4. specialkeys:

Let's user move the selection box using arrow keys.

5. mouseclick:

Let's user select a particular cell using mouse. And deselect any cell if clicked outside sudoku.

Sudoku Code :

```
/*
SUDOKU GAME USING OPENGL and MVC Design Pattern
To run this game, type make and then type ./Sudoku
A window will be launched. You can use the mouse or
the keyboard to select the box and enter the number.
*/
#include <iostream>
#include <time.h>
#include <stdlib.h>
```

```
#include <string>
#include <sstream>
#include <vector>
#include <map>
#include <algorithm>
#include <math.h>
#include <GL/glut.h>
using namespace std;

struct Cell{
   int number;
   bool given;
   bool mistake;
};

bool operator<(Cell c1,Cell c2){
   return c1.number<c2.number;
}

class SudokuModel{
public:
  Cell board[81];
  vector<int> position_list;      //list of positions (0-80) - to be shuffled later
  int number_of_filled_cells;     //used to keep a tab of how many cells user has filled (including filled cells given at the beginning)
```

```
  int selected_box;               //box that is selected by mouse click or keyboard keys (basically the cell in which the user can enter the number)


  //used to display and track time
  double Seconds_in_game;
  int Minutes_in_game;
  int Hours_in_game;


  bool NoMistake;                 //flag to check if all numbers entered by user is right


  SudokuModel();
        void Create_valid_board();      //used to create a valid board from which numbers are "hidden" later
        void Create_sudoku_puzzle();    //empty cells from valid board (for user to fill)


  bool Check_validity(int pos,int num);   //checks if num is there in corresponding row,col or box
        bool NoUniqueSolution(int *number_of_solutions);   //recursive function to find if current puzzle has a unique solution or not


        void Print_board();             //Prints board
  bool Find_unassigned_cell(int &pos); //checks if any cell in board is still unassigned


};


class SudokuController{
public:
```

```
  SudokuModel* model;             //Pointer to model object
  vector<int> ErrorList;          //List of cell positions that are blatantly incorrect (if there are same numbers in rows,cols or boxes)
  SudokuController(SudokuModel*);

  void CheckforErrors(int);       //Checks board for errors (Same numbers in rows,cols or boxes)
  void CheckRowColBoxErrors(int);
  void CheckExistingErrorCells(int);
  bool SingleCellErrors(int,int);
};

class SudokuView{
public:
  SudokuView(int argc, char** argv, SudokuModel* m, SudokuController* c);
  static void reshape(int, int); // Called by GLUT
  static void display(); // Called by GLUT
  static void keyboard(unsigned char key, int x, int y);
  static void specialkeys(int key, int x, int y);
  static void mouseclick(int button,int state, int x, int y);
  static SudokuModel* model;
  static SudokuController* controller;
};
time_t timer;
bool Success = false;

SudokuModel::SudokuModel(){
```

```
  number_of_filled_cells = 81;
  selected_box = -1;
  for(int i=0;i<81;i++){
   position_list.push_back(i);
  }
  random_shuffle(position_list.begin(),position_list.end());  //Shuffle the positions
          Create_valid_board();
          Create_sudoku_puzzle();
}

void SudokuModel::Create_valid_board(){
  srand(time(NULL));

  //Start with a valid board, and shuffle it
  //Swap 2 numbers, swap 2 rows in 0-2,3-5,6-8, swap 2 cols in 0-2,3-5,6-8
  int temp_sudoku[81]={3,2,9, 6,5,7, 8,4,1,
                       7,4,5, 8,3,1, 2,9,6,
                       6,1,8, 2,4,9, 3,7,5,
                       1,9,3, 4,6,8, 5,2,7,
                       2,7,6, 1,9,5, 4,8,3,
                       8,5,4, 3,7,2, 6,1,9,
                       4,3,2, 7,1,6, 9,5,8,
                       5,8,7, 9,2,3, 1,6,4,
                       9,6,1, 5,8,4, 7,3,2};
  for(int i=0;i<9;i++){
```

```
  for(int j=0;j<9;j++){
    board[i*9+j].number = temp_sudoku[i*9+j];
  }
}

int n1,n2;
int p1,p2;Cell temp;
int column_grid,row_grid,r1,r2,c1,c2;

for(int i=0;i<10000;i++)
{
  n1=rand()%9+1;
  do{
  n2=rand()%9+1;}while(n1==n2);
  for(int j=0;j<9;j++)
  {  for(int k=0;k<9;k++)
     {
       if(board[j*9+k].number==n1){p1=k;}
       if(board[j*9+k].number==n2){p2=k;}
     }
    temp=board[j*9+p1];
    board[j*9+p1]=board[j*9+p2];
    board[j*9+p2]=temp;
  }
}
```

```
  for(int i=0;i<10000;i++){
    row_grid=rand()%3;
    r1 = row_grid*3 + rand()%3;
    do{r2=row_grid*3+rand()%3;}while(r1==r2);
    for(int j=0;j<9;j++){
      temp=board[r1*9+j];
      board[r1*9+j]=board[r2*9+j];
      board[r2*9+j]=temp;
    }
  }

  for(int i=0;i<10000;i++){
    column_grid=rand()%3;
    c1 = column_grid*3 + rand()%3;
    do{c2=column_grid*3+rand()%3;}while(c1==c2);
    for(int j=0;j<9;j++){
      temp=board[j*9+c1];
      board[j*9+c1]=board[j*9+c2];
      board[j*9+c2]=temp;
    }
  }
   //Print_board();
}
```

```
void SudokuModel::Create_sudoku_puzzle(){
  // Erase cells whose positions are there in position_list, and check if it results in a unique solution
  // If no unique solution, then undo erase and move onto next cell.
  int original_number; int nSols=0;

  for(int i=0;i<position_list.size();i++){
    original_number = board[position_list[i]].number;
    board[position_list[i]].number = 0;
    board[position_list[i]].given = false;
    number_of_filled_cells-=1;
    nSols=0;
    if(NoUniqueSolution(&nSols)){
      board[position_list[i]].number = original_number;
      board[position_list[i]].given = true;
      number_of_filled_cells+=1;
    }
  }
}

bool SudokuModel::Find_unassigned_cell(int &pos)
{
  for(pos=0;pos<81;pos++){
    if(board[pos].number==0)
      return true;
  }
```

```
  return false;
}

bool SudokuModel::Check_validity(int pos, int num){
  int row = pos/9;
  int col = pos%9;
  for(int c=0;c<9;c++){
  if(board[row*9+c].number == num){
    return false;
  }          }

  for(int r=0;r<9;r++){
    if(board[r*9+col].number == num){
      return false;
    }
  }

  int startrow = row - row%3;
  int startcol = col - col%3;
  for(int r=startrow;r<startrow+3;r++){
    for(int c=startcol;c<startcol+3;c++){
      if(board[r*9+c].number == num){
        return false;
      }
    }
```

```
  }
  return true;
}

void SudokuModel::Print_board(){
for(int i=0;i<9;i++)
   {  for(int j=0;j<9;j++)
         {cout<<board[i*9+j].number<<" ";}
         cout<<endl;
   }
  cout<<endl;
}

bool SudokuModel::NoUniqueSolution(int *number_of_solutions){
  int pos;
  if (!Find_unassigned_cell(pos)){
   *number_of_solutions=*number_of_solutions+ 1;
   return true;}
  for (int num = 1; num <= 9; num++)
  {
    if (Check_validity(pos, num))
    {
       board[pos].number = num;
       if (NoUniqueSolution(number_of_solutions)){
          if((*number_of_solutions)>1){
```

```
            board[pos].number=0;
              return true;}
           }
       board[pos].number = 0;
    }
  }
  return false;
}

SudokuController::SudokuController(SudokuModel* m){
  model = m;
}

void SudokuController::CheckExistingErrorCells(int input_position){
  //Checks if the existing errors are cleared or not
  for(std::vector<int>::iterator it=ErrorList.begin();it!=ErrorList.end();++it){

    int position = *it;
    int row_number = position/9;
    int col_number = position%9;
    if(!SingleCellErrors(row_number,col_number)){
      model->board[*it].mistake = false;
      ErrorList.erase(it);
      --it;
    }
```

```
  }
}

bool SudokuController::SingleCellErrors(int row_number,int col_number){
  int box_start_row = (row_number/3)*3;
  int box_start_col = (col_number/3)*3;

  for(int j=0;j<9;j++){
    if(j==col_number){continue;}
    if(model->board[row_number*9+j].number ==
model->board[row_number*9+col_number].number ) {
      return true;
    }
  }

  for(int j=0;j<9;j++){
    if(j==row_number){continue;}
    if(model->board[j*9+col_number].number ==
model->board[row_number*9+col_number].number){
      return true;
    }
  }
  for(int i=box_start_row;i<box_start_row+3;i++){
    for(int j=box_start_col;j<box_start_col+3;j++){
      if(i==row_number || j==col_number){continue;}
      if(model->board[i*9+j].number == model->board[row_number*9+col_number].number){
```

```
        return true;
      }
    }
  }
  return false;
}

void SudokuController::CheckRowColBoxErrors(int position){
  int row_number = position/9;
  int col_number = position%9;
  int box_start_row = (row_number/3)*3;
  int box_start_col = (col_number/3)*3;
  model->board[position].mistake = false;
  for(int j=0;j<9;j++){
    if(j==col_number){continue;}
    if(model->board[row_number*9+j].number ==
model->board[row_number*9+col_number].number){
      if(!model->board[row_number*9+j].given && !model->board[row_number*9+j].mistake){
        model->board[row_number*9+j].mistake = true;
        ErrorList.push_back(row_number*9+j);
      }
      if(!model->board[row_number*9+col_number].mistake){
        model->board[row_number*9+col_number].mistake = true;
        ErrorList.push_back(row_number*9+col_number);
      }
    }
```

```
  }


  for(int j=0;j<9;j++){
    if(j==row_number){continue;}
    if(model->board[j*9+col_number].number ==
model->board[row_number*9+col_number].number){
      if(!model->board[j*9+col_number].given && !model->board[j*9+col_number].mistake){
        model->board[j*9+col_number].mistake = true;
        ErrorList.push_back(j*9+col_number);
      }
      if(!model->board[row_number*9+col_number].mistake){
        model->board[row_number*9+col_number].mistake = true;
        ErrorList.push_back(row_number*9+col_number);
      }
    }
  }


  for(int i=box_start_row;i<box_start_row+3;i++){
    for(int j=box_start_col;j<box_start_col+3;j++){
      if(i==row_number || j==col_number){continue;}
      if(model->board[i*9+j].number == model->board[row_number*9+col_number].number){
        if(!model->board[i*9+j].given && !model->board[i*9+j].mistake){
          model->board[i*9+j].mistake = true;
          ErrorList.push_back(i*9+j);
        }
        if(!model->board[row_number*9+col_number].mistake){
```

```
        model->board[row_number*9+col_number].mistake = true;
        ErrorList.push_back(row_number*9+col_number);
       }
     }
   }
  }
}

void SudokuController::CheckforErrors(int position){
  model->NoMistake = true;
  CheckExistingErrorCells(position);
  CheckRowColBoxErrors(position);
  if(!ErrorList.empty()){
    model->NoMistake = false;
  }
}
SudokuModel* SudokuView::model=0;
SudokuController* SudokuView::controller=0;

SudokuView::SudokuView(int argc, char** argv, SudokuModel* m, SudokuController* c){
  model=m;
  controller = c;
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
  glutInitWindowSize(700, 700);
```

```
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Sudoku Game");
  glClearColor(1.0, 1.0, 1.0, 0.0);   // white background
  glutDisplayFunc(display);
  glutIdleFunc(display);
  glutKeyboardFunc(keyboard);
  glutSpecialFunc(specialkeys);
  glutMouseFunc(mouseclick);
  glutReshapeFunc(reshape);
  glutMainLoop();
}

void SudokuView::reshape(int w,int h){
  glViewport(0,0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0.0, (GLdouble)w, 0.0, (GLdouble)h, (GLdouble)-w, (GLdouble)w);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void DrawText(const char* text,int length,double x,double y){
  glRasterPos2f(x,y);
  for(int i=0;i<length;i++){
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,(int)text[i]);//Character Display
```

```
  }
}

void SudokuView::display(){
   // clear all
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  // Clear the matrix
  glLoadIdentity();
  //Track time
  double Seconds_in_game;
  time_t current_time;
  if(!Success){
    current_time = time(NULL);
    Seconds_in_game = difftime(current_time,timer);
    model->Seconds_in_game = fmod(Seconds_in_game,60.0);
    model->Minutes_in_game = int(Seconds_in_game)/60;
    if(model->Minutes_in_game >=60){model->Minutes_in_game %= 60;}
    model->Hours_in_game = int(Seconds_in_game)/3600;
    if(model->Hours_in_game >= 24){model->Hours_in_game %=24;}
  }
  std::ostringstream strm;
  strm <<model->Hours_in_game;std::string numStr = "TIME =
";if(strm.str().length()==1)numStr+='0'; numStr += strm.str(); strm.str(std::string());
  strm <<model->Minutes_in_game; numStr = numStr + ":";
if(strm.str().length()==1)numStr+='0'; numStr += strm.str();strm.str(std::string());
```

```
  strm <<model->Seconds_in_game; numStr = numStr +
":";if(strm.str().length()==1)numStr+='0'; numStr += strm.str();strm.str(std::string());
  glColor3f(0.0,0.0,1.0);
  DrawText(numStr.data(),numStr.length(),300.0,650.0); //Display Time
  int width_board=400;int height_board=400;

  //Draw the outer box of the sudoku
  glColor3f(0.0,0.0,0.0);
  glLineWidth(4.0);
  glBegin(GL_LINE_LOOP);
  glVertex2f(150.0,150.0);
  glVertex2f(550.0,150.0);
  glVertex2f(550.0,550.0);
  glVertex2f(150.0,550.0);
  glEnd();

  double width_cell = 400.0/double(9);
  double height_cell = 400.0/double(9);

  //draw the lines in between to create rows and cols and cells
  for(int i=1;i<9;i++){
   glLineWidth(1.0);
   if(i%3==0){glLineWidth(4.0);}
   glBegin(GL_LINES);
   glColor3f(0.0,0.0,0.0);
   glVertex2f(150.0+i*width_cell,150.0);
```

```
  glVertex2f(150.0+i*width_cell,550.0);
  glEnd();

  glBegin(GL_LINES);
  glColor3f(0.0,0.0,0.0);
  glVertex2f(150.0,150.0+i*height_cell);
  glVertex2f(550.0,150.0+i*height_cell);
  glEnd();
 }

 int row_number,col_number;
//draw the selected box (in green)
 if(model->selected_box!=-1){

  row_number = model->selected_box/9;
  col_number = model->selected_box%9;

  double x_coord = 150.0 + (width_cell*col_number);
  double y_coord = 550.0 - (height_cell*(1+row_number));
  glColor4f(0.0,0.0,1.0,1.0);
  glLineWidth(5.5);
  glBegin(GL_LINE_LOOP);
  glVertex2f(x_coord,y_coord);
  glVertex2f(x_coord+width_cell,y_coord);
  glVertex2f(x_coord+width_cell,y_coord+height_cell);
```

```
  glVertex2f(x_coord,y_coord+height_cell);
  glEnd();
}



//display the numbers
for(int i=0;i<81;i++){
  if(model->board[i].number!=0){
    row_number = i/9;
    col_number = i%9;
    if(model->board[i].given){
      glColor4f(0.0,0.0,0.0,1.0);
    }
    else{
      if(model->board[i].mistake){
        glColor3f(1.0,0.0,0.0);
      }
      else{
        glColor4f(0.196078,0.8,0.196078,1.0);
      }
    }
    std::ostringstream strm;
    strm << model->board[i].number;
    std::string numStr = strm.str();
```

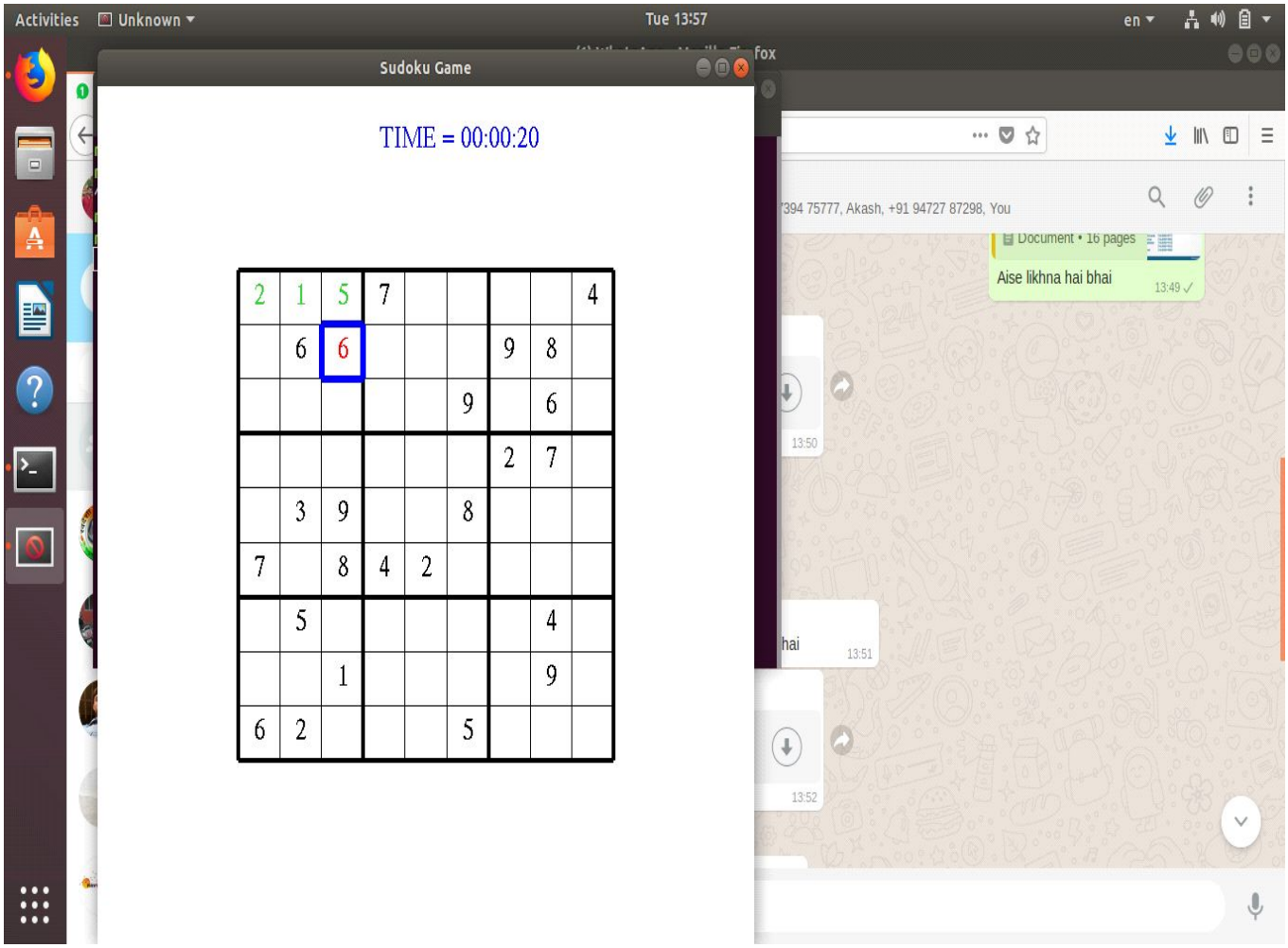
```
DrawText(numStr.c_str(),1,(150.0+(col_number*width_cell)+(width_cell/2.0)-5.0),(550.0-(row_
number*height_cell)-(height_cell/2.0)-5.0));
    }
  }
  if(model->number_of_filled_cells==81 && model->NoMistake){
    glColor3f(1.0,0.0,0.0);
    string text = "Congratulations!! You have solved the sudoku";
    Success = true;
    DrawText(text.data(),text.length(),175.0,625.0);
  }
  glutSwapBuffers();
}
void SudokuView::keyboard (unsigned char key, int x, int y)
{
  if(!Success){
    // Keystroke processing here
    switch (key-48){
      case 0:
      case 1:
      case 2:
      case 3:
      case 4:
      case 5:
      case 6:
      case 7:
```

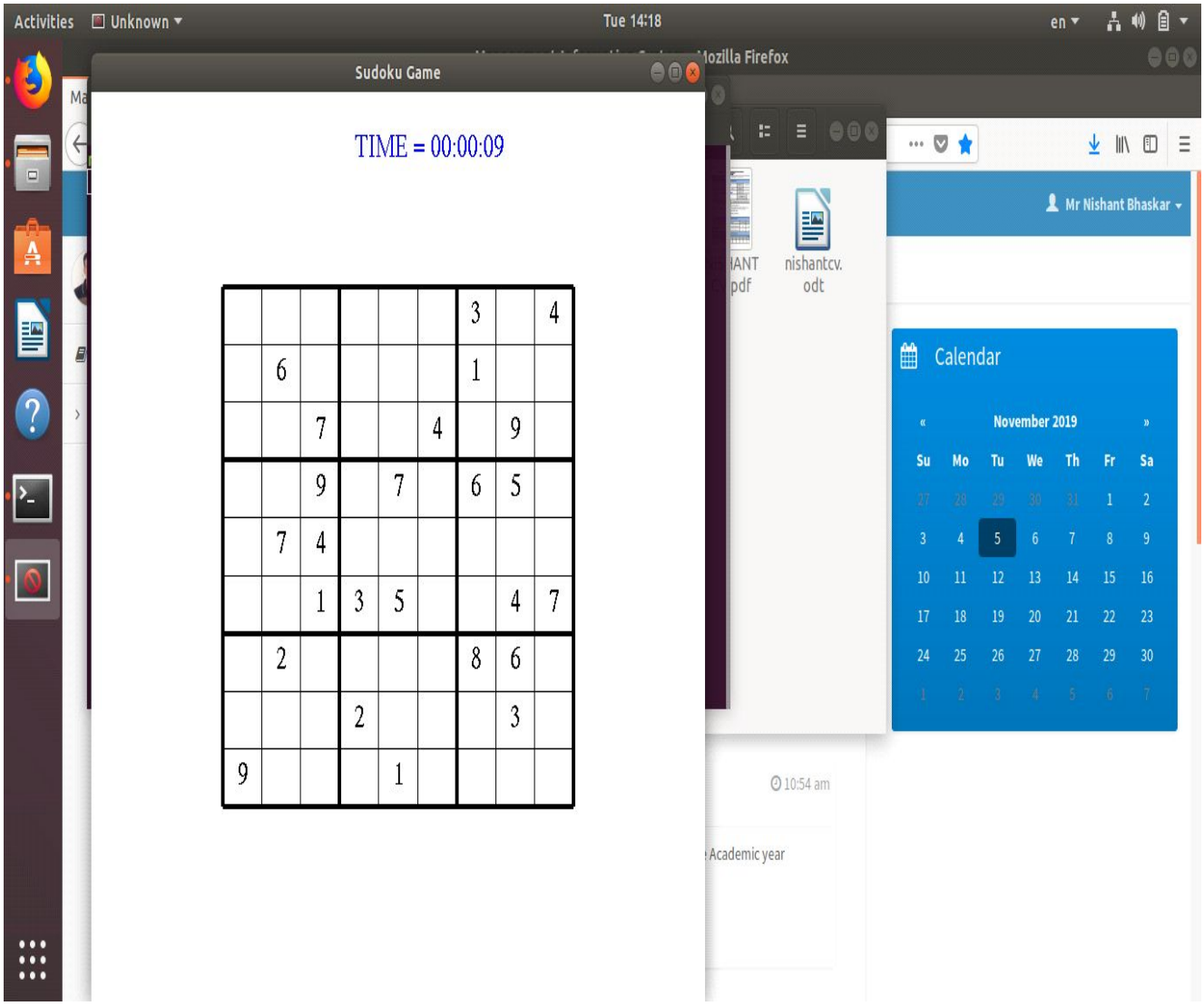
```
      case 8:
      case 9:
       if(model->selected_box!=-1)
       {
        if(!model->board[model->selected_box].given){

          if(model->board[model->selected_box].number==0){
            model->number_of_filled_cells+=1;
          }
          model->board[model->selected_box].number = key-48;
          //When user enters a number check if it eliminates any of the existing errors or creates a
new error
          controller->CheckforErrors(model->selected_box);
        }
       }
       break;
    }
  }
}

void SudokuView::specialkeys(int key,int x,int y){
  // If user presses any of the arrow keys move the box
  switch(key){
    case GLUT_KEY_RIGHT:
      if(model->selected_box==-1){model->selected_box+=1;}
      else if(model->selected_box%9==8){model->selected_box-=8;}
```

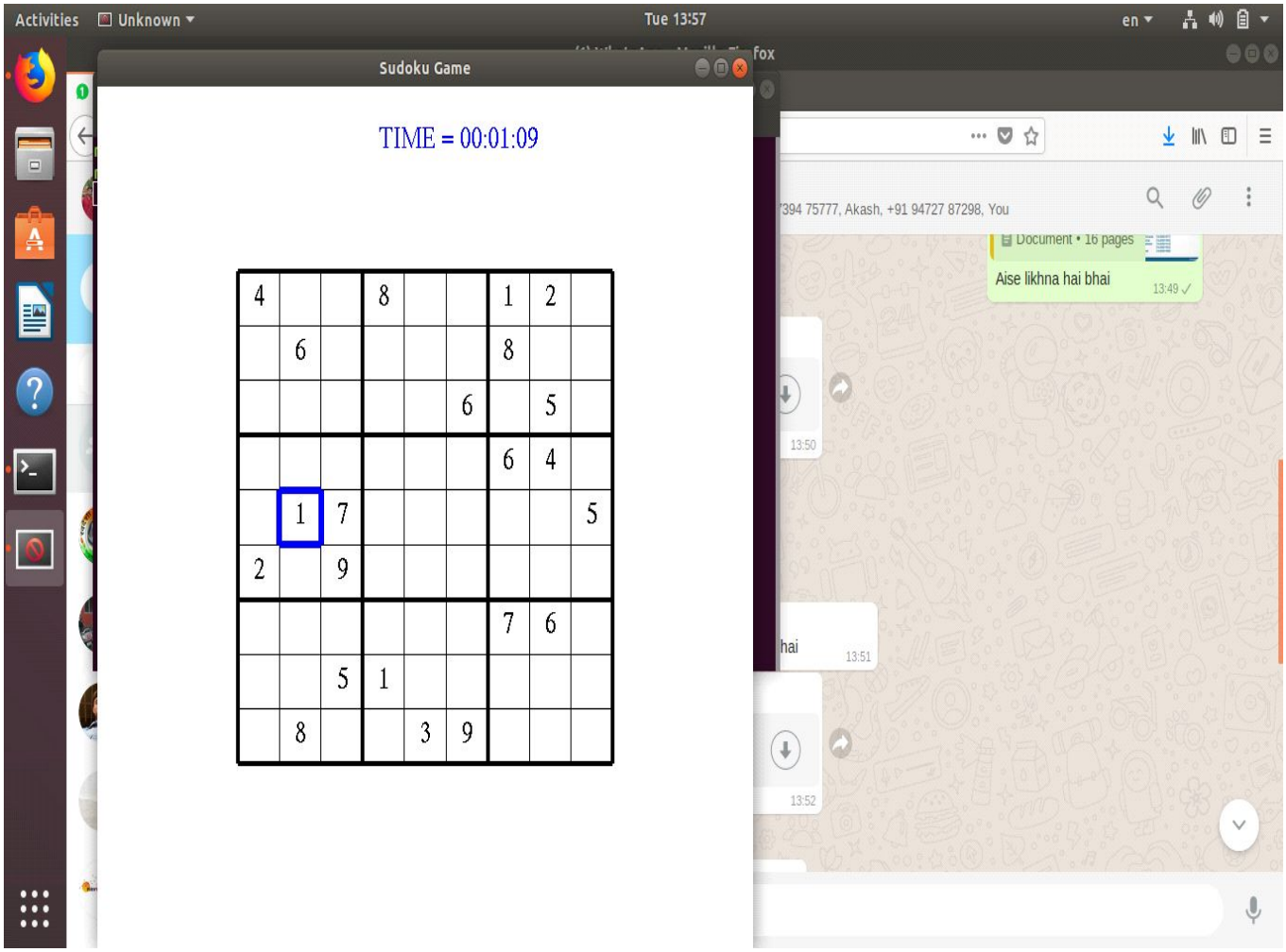
```
      else {model->selected_box+=1;}
      break;
    case GLUT_KEY_LEFT:
      if(model->selected_box==-1){model->selected_box+=1;}
      else if(model->selected_box%9==0){model->selected_box+=8;}
      else {model->selected_box-=1;}
      break;
    case GLUT_KEY_UP:
      if(model->selected_box==-1){model->selected_box+=1;}
      else if(model->selected_box>=0&&model->selected_box<=8){model->selected_box+=72;}
      else {model->selected_box-=9;}
      break;
    case GLUT_KEY_DOWN:
      if(model->selected_box==-1){model->selected_box+=1;}
      else
if(model->selected_box>=72&&model->selected_box<=80){model->selected_box-=72;}
      else {model->selected_box+=9;}
      break;
  }
}



void SudokuView::mouseclick(int button,int state, int x, int y)
{
  if(x<150.0 || y<150.0 || x>550.0 || y>550.0){
```

```
    model->selected_box = -1;
  }
  else{
    int row_number = ((y-150.0)*9)/400;
    int col_number = ((x-150.0)*9)/400;
    model->selected_box = (row_number*9)+col_number;
  }
}

int main(int argc, char** argv){
  timer = time(NULL);
  SudokuModel* m = new SudokuModel();
  SudokuController* c = new SudokuController(m);
  SudokuView* v = new SudokuView(argc,argv,m,c);
  glutMainLoop();
  return 0;
}
```

Output :


Activities
Unknown
Tue 13:57
en
Sudoku Game
TIME = 00:00:20
2 1 5 7 4
6 6 9 8
9 6
2 7
3 9 8
7 8 4 2
5 4
1 9
6 2 5
Document • 16 pages
Aise likhna hai bhai
13:49
13:50
hai
13:51
13:52


Activities
Unknown
Tue 14:18
en
Mozilla Firefox
Sudoku Game
TIME = 00:00:09
Mr Nishant Bhaskar
nishantcv.
odt
Calendar
November 2019
Su Mo Tu We Th Fr Sa
10:54 am
Academic year



Conclusion:

In this project, we have used the OpenGL library to design a 2-D grid and write numbers in the cell. Then we wrote an efficient algorithm to check the correctness of solved sudoku.