1. Acknowledgement

Doing a project in any programming language is the only way to know the importance of programming, its essentials and knowing the features of the language. We are glad that university has provided us the platform to do this in object oriented programming using C++. We are very grateful to the Department of Electronics and Computer Engineering, Pulchowk Campus, IOE for this opportunity. We must thank our class teacher Mr. Basanta Joshi for driving us with the concepts of object oriented programming.

We must acknowledge the makers of 2048 game, and very peculiar ‘Prime numbers’ in the number system. We cannot remain silent about the help and support provided by our colleagues during the making of the project.

Arjun Dahal (071/BEX/405)

Biplab Gautam(071/BEX/411)

Bishal Khanal (071/BEX/412)

2. Abstract

Prime numbers are pretty strange in the number system and their pattern of occurrence has fascinated the mathematicians from ages. We as students, have always believed that mathematics and programming are always correlated. Primo is an approach to present prime numbers in a fun way. Not only of about developing an algorithm for a program, mathematics itself can be fun. This is the motive of game Primo. In Primo, a player plays according to game rules to make the prime number of the particular level.

Table of Contents:

3. Objectives:

The objectives of the game Primo are mainly listed below:

- Fun.
- To help user develop familiarity with prime and composite numbers.
- Mental exercise and refreshment for the user.
- Enhancement of mathematical skills and strategic mathematical thinking.
- To get oneself away from boredom.

4. Introduction

Primo is a number's game. In Spanish '*Numero Primo'* means Prime numbers and the name of the game Primo is simply an extraction of it. The game is basically a prime version adaptation of a popular game **2048.** It is a strategic and calculative game.

4.1 Basic game description

The game continues level by level and each level is assigned a particular prime number. The main objective of the player is to get that particular prime number by following the rules of the game. This game is little bit unconventional in the sense that the levels don't start from 1,2,3…. Instead, the level starts from the first two-digit prime number 11 and goes on as 11, 13, 17, 19, and so on for successive prime numbers.

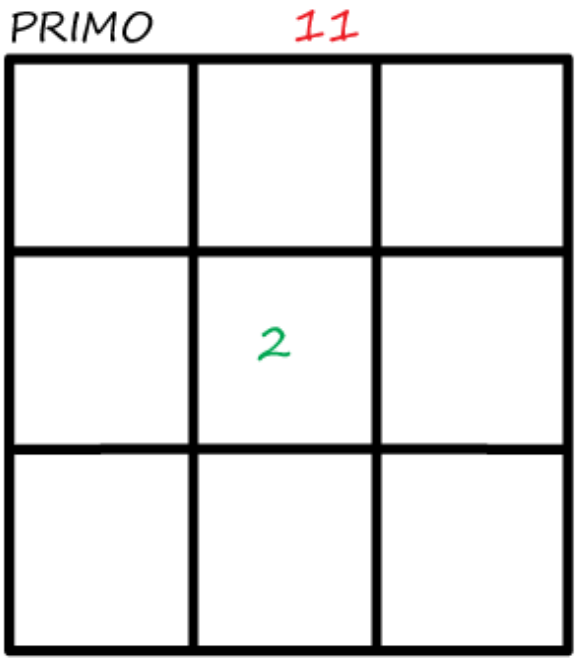


Figure 1:Primo Playing Board

The game starts from level 11 and the playing board consists number 2 at the centre of the board at the start of each level. Aim of each level is to make the prime number corresponding to that level (as indicated in the top of the playing board). The game is played by pressing the cursor control keys.

For example, if cursor key *down* is pressed then all the numbers try to add column wise from top to bottom but are added only if they are allowed by the game rule. Similar is the case if cursor key *up* is pressed but the numbers add from bottom to top column wise. If cursor key *right* is pressed then, the numbers add row wise from left to right and if cursor key *left* is pressed then, the numbers add row wise from right to left. After every step or key press, basic numbers (numbers from 1 to 5) pop out randomly from random boxes.

The **addition rule** of the game is:

a. Friendly numbers (term for the game) 1 and 2 add to any number prime or composite.
b. Composite numbers (numbers except prime) can add among themselves.
c. Prime numbers only add (to prime and composites) to make another prime number.

These statements will be more clear on illustration in the **Result** part of the report.

5. Application

Playing Primo not only proves Mathematics is interesting but also it can be fun to hang around with. The target audience of the game Primo includes school and college students and math enthusiasts. Game like Primo can be an effective way to help people learn about number system, prime numbers and composite numbers. User come up with a special strategy to play the game and this enhances creative thinking and logical decision making.

6. Existing System

It is already mentioned that the game is prime version adaptation of **2048** game. The 2048 game is also a number's game in which the user plays to make the numbers which are integral powers of 2 and reach ultimately to 2048; and scoring is done accordingly. However, it is true that it is a pioneer game itself involving the prime numbers.

7. Methodology

The game is coded in C++ programming language making use of the Object Oriented Concept. The concept of Objects and Classes, Polymorphism, Friend function, etc. are used in programming of the game. For displaying the board and the numbers, graphics library SDL (Simple Directmedia Layer) is used. It is a cross-platform development library designed to provide low level access to audio, keyboard, mouse and graphics hardware.

7.1 Features of Object Oriented Programming used in the Game

a. Objects and Classes

The game program consists of three classes each for specific purpose. They are listed below:

Class *NumberObjects* :

This class consists of only one integer data member *Num*. It consists of member functions *putvalue* to assign value to *Num* and *getvalue* to get the value contained in *Num* from outside. This class overloaded the '+' operator for implementing the addition rule of the game (this will be discussed in further elaboration in coming heading).

Nine objects in 2-D array form are declared. These objects correspond to the number value of each box in the game. The private data member *Num* holds the value. Computation part is primarily done through this class, as the actual value is stored in the objects of this class.

Class *MainTexture* :

This class is primarily for graphics related activities. It consists of data members viz.

- Array of 11 *SDL_Texture pTexture* which is required for displaying 11 boxes 9 for board numbers and two for program title and level number.
- *pWidth* and *pHeight* for the board height and width.
- Array of 11 *SDL_Rect Boxes* required for drawing and displaying the boxes and numbers in the playing board.
- Two arrays of 11 integers *mWidth and mHeight* for the dimensions of image that is obtained from the text.

One object of this class is declared.

The member functions of the class are:

- No argument Constructor is used to initialize the data members and it calls the *SetBoxPos* function of the class itself to set the positions and dimensions of the boxes.
- Destructor frees the allocated memory and calls the *free* function of the class itself which destroys the textures.
- *DrawBoard* function draws the basic board using SDL library functions.

- *EasyLoader* function loads numbers of the playing board from the argument passed by calling another function *LoadFromRenderedText* function in the class itself which converts string to image so that it can be displayed.
- *FinalRender* function puts the loaded images of from the text to appropriate boxes.

Class *LoadingImages* :

This class is used to display the initial welcome window, instructions window, level completion window and game over window. Four objects of this class are declared for each type of window.

- It consists of one data member of type SDL_Texture *newTexture.*
- Function *Texture_Loader* which loads an image from path argument apart from constructor which initialises the data member to Null and destructor which destroys the SDL_Texture on program termination.

b. Friend Functions

The class *NumberObjects* has declared the functions *CheckLevel, NewLevel* and *RandomGenerator* functions as friend so that these functions can access the private data member *Num* of the object instantiated of the class.

c. Polymorphism (Operator Overloading)

The '+' operator is overloaded in the class *NumberObjects* as per the addition rule of the game. This has made coding of this game immensely easier and faster. The objects are simply added directly using '+' operator.

Pass by reference with pointer is used so that it became possible to alter the value of objects, just by using + operator between them and nothing more.

7.2 Global Functions (User-Defined) used in the game

- *Initialiser* – It initialises the SDL subsystem, SDL window, TTF and IMG subsystems.
- *Closer* – Called at the end of the program to shut down subsystems.
- *CheckPrime* – Checks whether the number is prime or not.

- *CheckLevel* – Checks whether the level is completed or not.
- *NewLevel* – Initialises all variable to zero at the start of new level.
- *DownAdder, UpAdder, LeftAdder, RightAdder* – Adds as per the game rule called by cursor key presses.
- *RandomGenerator*– Generates random number at random places.
- *LoadFont* – Loads the true type font file in the program.

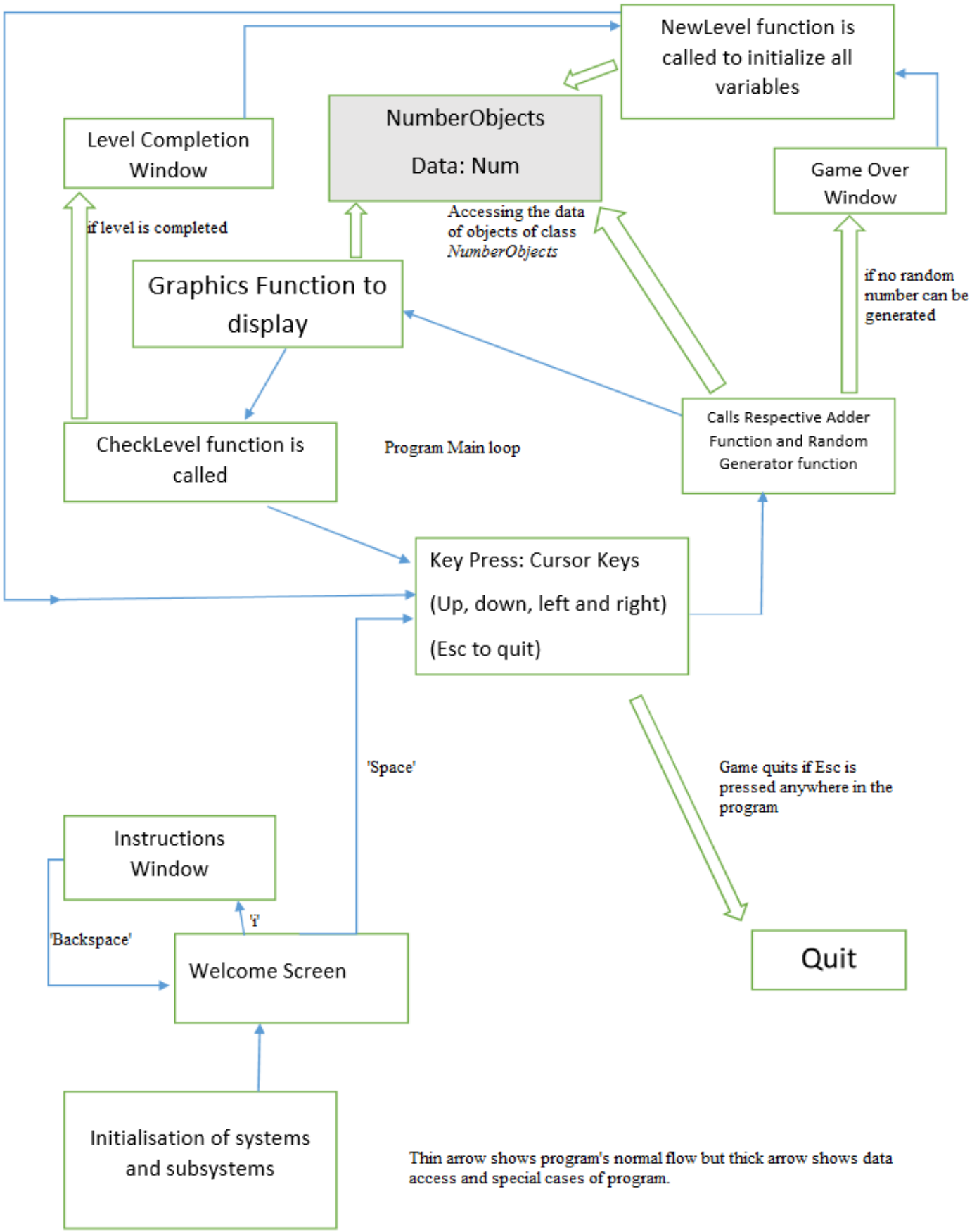
7.3 Header Files used in the game

- *iostream* – For basic input/output stream operations.
- *SDL.h* – For graphics window, surface.
- *SDL_image.h* – For loading image texture and handling images
- *SDL_ttf.h* – For loading image from font file.
- *string.h* – For handling string related operations.
- *vector* – For using vector class and its functions to put a number in random place.
- *algorithm* – For using function *random_shuffle* to put a number in random place.
- *Ctime* – To access the time of the computer.
- *Sstream* – To access the string based stream functions.

8. Implementation

The implementation part of the project includes the block diagram of the working of game and the source code of the game.

8.1 Block Diagram:



8.2 Source Code:

```
#include <iostream>
#include <SDL.h>
#include <SDL_image.h>
#include <SDL_ttf.h>
#include <cstring>
#include <vector>
#include <algorithm>
#include <ctime>
#include <sstream>

using namespace std;

enum
{
   Box0=0,Box1,Box2,Box3,Box4,Box5,Box6,Box7,Box8,Box9,Box10,Box11
};

//Screen Dimensions Constants
const int SCREEN_WIDTH = 360;
const int SCREEN_HEIGHT = 480;

SDL_Window* PrimoWindow =NULL ;
SDL_Renderer* PrimoRend =NULL ;
TTF_Font* Pfont =NULL;
bool quit=false;

SDL_Color ColorPrime={0xFF,0x25,0x00};
SDL_Color ColorComposite={0xFF,0xFF,0xFF};
SDL_Color ColorFriendly={0x25,0xCD,0x00};
SDL_Color ColorTitle={0xFF,0x00,0x00};

bool Initialiser();//It initialises SDL window, SDLimage, SDLttf and other required elements.
```

```
void Closer();//It shuts down SDL subsystems on closing of program.
bool LoadFont();//It loads the true type font file.
bool CheckPrime(int);//Checks the number is prime or not
void CheckLevel();
void NewLevel();
//SDL_Texture* Texture_Loader( string );

int levels[]={11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,87,89,91,97,101};
int LEVEL,LevelCounter=0;

class MainTexture
{
   public:
                //Initializes variables
                MainTexture();

                //Deallocates memory
                ~MainTexture();

                //Deallocates texture
                void free();

                //Renders texture at given point
                void FinalRender(int );

                 //Set positions of the boxes
                 void SetBoxPos();

                 // It draws the main playing board
                 void DrawBoard();

                 void LoadFromRenderedText(std::string,SDL_Color,int , TTF_Font*);

                 void EasyLoader(int,int);
```

```
                //friend class NumberObjects;

        private:
                //The actual hardware texture
                SDL_Texture* pTexture[11];

                //Image dimensions
                int pWidth;
                int pHeight;
                SDL_Rect Boxes[11];
                //for the dimensions of image from text
      int mWidth[11];
                int mHeight[11];
};
MainTexture :: MainTexture()
{
   for(int i=0;i<11;i++)
   pTexture[i]=NULL;
   pWidth=SCREEN_WIDTH;
   pHeight=SCREEN_HEIGHT;
   SetBoxPos();
}

MainTexture :: ~MainTexture()
{
   free();
}

void MainTexture :: free()
{
   //Free texture if it exists
   for(int i=0;i<11;i++)
   {
```

```
        if( pTexture[i] != NULL )
        {
            SDL_DestroyTexture( pTexture[i] );
            pTexture[i] = NULL;
        }
        pWidth = 0;
        pHeight = 0;
    }
}

void MainTexture :: FinalRender(int Boxnum )
{
    //Set rendering space and render to screen
            SDL_Rect tempRect;
            tempRect.h=Boxes[Boxnum].h-80;
            tempRect.w=Boxes[Boxnum].w-80;
            tempRect.x=Boxes[Boxnum].x+30;
            tempRect.y=Boxes[Boxnum].y+30;
            //Render to screen
            SDL_Texture* TempTexture = pTexture[Boxnum];
            SDL_RenderCopy( PrimoRend,TempTexture, NULL, &tempRect );
            SDL_DestroyTexture(TempTexture);
}

void MainTexture :: SetBoxPos()
{
    Boxes[0] = {0,0,240,pHeight/4};
    Boxes[1] = {240,0,pWidth/3,pHeight/4};
    Boxes[2] = {0,pHeight/4,pWidth/3,pHeight/4};
    Boxes[3] = {pWidth/3,pHeight/4,pWidth/3,pHeight/4};
    Boxes[4] = {240,pHeight/4,pWidth/3,pHeight/4};
    Boxes[5] = {0,pHeight/2,pWidth/3,pHeight/4};
    Boxes[6] = {pWidth/3,pHeight/2,pWidth/3,pHeight/4};
    Boxes[7] = {240,pHeight/2,pWidth/3,pHeight/4};
```

```
    Boxes[8] = {0,360,pWidth/3,pHeight/4};
    Boxes[9] = {pWidth/3,360,pWidth/3,pHeight/4};
    Boxes[10] = {240,360,pWidth/3,pHeight/4};
}

void MainTexture :: DrawBoard()
{
    SDL_SetRenderDrawColor( PrimoRend, 0xE0, 0xE0, 0xE0, 0xFF );
    for(int i=2;i<11;i++)
        SDL_RenderDrawRect(PrimoRend,&Boxes[i]);

        SDL_RenderDrawLine(PrimoRend,0,119,360,119);//Upper line
        SDL_RenderDrawLine(PrimoRend,1,120,1,480);//Leftmost line
        SDL_RenderDrawLine(PrimoRend,0,478,360,478);//Bottom line
        SDL_RenderDrawLine(PrimoRend,358,120,358,480);//Rightmost line
}
//Function to load elements of playing board easily
void MainTexture ::EasyLoader(int ObjectValue,int Boxnumber)
{
    stringstream ss;
    ss << ObjectValue;
    string ValueString = ss.str(); //To convert Integer to string

    SDL_Color SendingColor;
    if (ObjectValue==0){ValueString=" ";}
    else if(ObjectValue==1||ObjectValue==2)
    {
        SendingColor = ColorFriendly;
    }
   else if(CheckPrime(ObjectValue))
    {
       SendingColor = ColorPrime;
    }
    else SendingColor = ColorComposite;
```

```
   LoadFromRenderedText(ValueString,SendingColor,Boxnumber,Pfont);
}
void MainTexture::LoadFromRenderedText(string Boxstring, SDL_Color NumColor, int
Boxnum, TTF_Font* Tempfont)
{
   //Free all textures.
   free();
   //Render Text Surface
   SDL_Surface* TextSurface = TTF_RenderText_Solid(Tempfont, Boxstring.c_str(),
NumColor);
   if(TextSurface==NULL)
   {
      cout<<"Unable to render text surface! SDL_ttf Error:"<<TTF_GetError()<<endl;
   }
   else
      //Create Texture from the loaded surface
   {
      pTexture[Boxnum]= SDL_CreateTextureFromSurface(PrimoRend,TextSurface);
      if(pTexture[Boxnum]==NULL)
      {
         cout<<"Unable to create texture from rendered text! SDL
Error:"<<SDL_GetError()<<endl;
      }
      else
                  {
                        //Get image dimensions
                        mWidth[Boxnum] = TextSurface->w;
                        mHeight[Boxnum] = TextSurface->h;
                  }

                  //Get rid of old surface
                  SDL_FreeSurface( TextSurface );
   }
```

```
}

class NumberObjects
{
private:
   int Num;
public:
   void putvalue(int);
   int Givevalue()
   {
      return Num;
   }

   bool operator +(NumberObjects *);
   friend void CheckLevel();
   friend void NewLevel();
   friend int RandomGenerator();

};

void NumberObjects:: putvalue(int a)
{
   Num=a;
}
//Defines How the objects are added as per the game rule.
bool NumberObjects::operator+(NumberObjects* Temp)
{
   bool addable = true;
   int tempSum;
   if(Num==0||Num==1||Num==2||Temp->Num==0||Temp->Num==1||Temp->Num==2)
   {
      tempSum=Num+Temp->Num;
      Temp->putvalue(tempSum);
      Num=0;
```

```
    }
    else if(!(CheckPrime(Num)||CheckPrime(Temp->Num)))
    {
        tempSum = Num+Temp->Num;
        Temp->putvalue(tempSum);
        Num=0;
    }
    else if(CheckPrime(Num+Temp->Num))
    {
        tempSum = Num+Temp->Num;
        Temp->putvalue(tempSum);
        Num=0;
    }
    else addable = false;
    return addable;
}

class LoadingImages
{
private:
    SDL_Texture* newtexture;
public:
    LoadingImages()
    {
        newtexture = NULL;
    }
    ~LoadingImages()
    {
        SDL_DestroyTexture(newtexture);
    }

    void Texture_Loader( string path)
    {
    SDL_Surface* newsurface=IMG_Load(path.c_str());
```

```
    if(newsurface==NULL)
    {
        cout<<"Surface could not be created.";
    }
    else newtexture=SDL_CreateTextureFromSurface(PrimoRend,newsurface);
    if (newtexture==NULL)
        cout<<"Texture could not be loaded";
    SDL_FreeSurface(newsurface);
    }

    SDL_Texture* giveTexture()
    {
        return newtexture;
    }

};


NumberObjects Objects[3][3]; //Numbers of each boxes
MainTexture main1;
LoadingImages Welcome, LevelCompletion, GameOver, Instructions;


void DownAdder()
{
    for(int j=0;j<3;j++)
    {
        int k=0;
        for(int i=0;i<2;i++)
        {
            Objects[i][j] + &Objects[i+1][j];
        }
        Objects[k][j] + &Objects[k+1][j];
    }
```

```
}

void UpAdder()
{
   for(int j=0;j<3;j++)
   {
      int k=2;
      for(int i=2;i>0;i--)
      {
         Objects[i][j] + &Objects[i-1][j];
      }
      Objects[k][j]+&Objects[k-1][j];
   }
}

void RightAdder()
{
   for(int i=0;i<3;i++)
   {
      int k=0;
      for(int j=0;j<2;j++)
      {
         Objects[i][j] + &Objects[i][j+1];
      }
      Objects[i][k]+ &Objects[i][k+1];
   }
}

void LeftAdder()
{
   for(int i=0;i<3;i++)
   {
      int k=2;
      for(int j=2;j>0;j--)
```

```
        {
            Objects[i][j] + &Objects[i][j-1];
        }
        Objects[i][k] + &Objects[i][k-1];
    }
}


bool CheckPrime(int num)
{
    int checker;
    bool Prime= true;
    for(int i=2;i<num;i++)
    {
        checker=num%i;
        if(checker==0)
        {
            Prime = false;
            break;
        }
    }
    return Prime;
}

void CheckLevel()
{
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
          if (Objects[i][j].Num==LEVEL)
          {
              SDL_Delay(2000);
              LevelCounter++;
```

```
LEVEL=levels[LevelCounter];
NewLevel();
LevelCompletion.Texture_Loader("LevelCompleted.png");
SDL_Event Q;

bool KeepDisplaying=true;
while(KeepDisplaying)
{
   //cout<<"Level Completed 1";
   SDL_RenderClear(PrimoRend);
   SDL_SetRenderDrawColor(PrimoRend,0xFF,0xFF,0xFF,0xFF);
   SDL_RenderCopy(PrimoRend,LevelCompletion.giveTexture(),NULL,NULL);
   SDL_RenderPresent(PrimoRend);
   while(SDL_PollEvent(&Q)!=0)
   {
      if(Q.type==SDL_QUIT)
      {
         KeepDisplaying=false;
         quit = true;
         break;
      }
      else if(Q.type==SDL_KEYDOWN)
      {
         switch (Q.key.keysym.sym)
         {
         case SDLK_ESCAPE:
          {
             quit = true;
             KeepDisplaying=false;
             break;
          }
          case SDLK_SPACE:
             {
             KeepDisplaying=false;
```

```
                        break;
                        }
                    }
                }
            }
        }


    }

  }
}
//Initialise Objects value at new level to zero.
void NewLevel()
{
  for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    {
      Objects[i][j].Num=0;
    }
  }
  Objects[1][1].Num=2;
}

struct PosDeterminer
{
  int Row,Col;
};

int RandomGenerator()
{
 //put the random number(1-5) on board by detecting the void space
```

```
        vector <PosDeterminer> RndList;
        PosDeterminer P[9];
        int check=0;
        int k=0;

for(int i=0; i<3; i++)
{
   for(int j = 0;j<3;j++)
   {
      P[k].Row=i;
      P[k].Col=j;
      if(Objects[i][j].Num==0)
      {
         RndList.push_back(P[k]);
         check++;
      }
      k++;
   }
}
      if(check!=0)
        {
                        int RandIndex, One_Five=0;

                        random_shuffle ( RndList.begin(), RndList.end() );

                        srand (time(NULL));

                        RandIndex = rand() % RndList.size();

                        while(One_Five==0)
                        {
                                One_Five=rand()%6;
                        }
```

```
                PosDeterminer Ptemp = RndList[RandIndex];
                Objects[Ptemp.Row][Ptemp.Col].putvalue(One_Five);
 }
else
 {
   SDL_Delay(1000);
   GameOver.Texture_Loader("Gameover.png");
   bool KeepDisplay=true;
   SDL_Event Q;
  while(KeepDisplay)
  {
     SDL_RenderClear(PrimoRend);
     SDL_SetRenderDrawColor(PrimoRend,0xFF,0xFF,0xFF,0xFF);
     SDL_RenderCopy(PrimoRend,GameOver.giveTexture(),NULL,NULL);
     SDL_RenderPresent(PrimoRend);
     while(SDL_PollEvent(&Q)!=0)
     {
        if(Q.type==SDL_QUIT)
        {
           quit = true;
           KeepDisplay=false;
           break;
        }
       else if(Q.type==SDL_KEYDOWN)
        {
           switch (Q.key.keysym.sym)
           {
           case SDLK_ESCAPE:
            {
               quit = true;
               KeepDisplay=false;
               break;
            }
            case SDLK_SPACE:
```

```
                    {
                        NewLevel();
                        KeepDisplay=false;
                        break;
                    }
                }
            }
        }


    }
}

//Function Definitions
bool Initialiser()
{
   // Initialization Flag
   bool success = true;
   //Initialise SDL subsystem
   if( SDL_Init( SDL_INIT_VIDEO ) < 0 )
        {
            cout<< "SDL could not initialize! SDL Error: %s"<<SDL_GetError()<<endl;
            success = false;
        }
        else
   {
      if( !SDL_SetHint( SDL_HINT_RENDER_SCALE_QUALITY, "1" ) )
            {
                cout<< "Warning: Linear texture filtering not enabled!"<<endl ;
            }
            //Create SDL window
```

```
            PrimoWindow = SDL_CreateWindow("Primo" ,
SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, SCREEN_WIDTH,
SCREEN_HEIGHT, SDL_WINDOW_SHOWN);

            if(PrimoWindow == NULL )
    {
       cout<<"Window could not be initialized."<<endl<<SDL_GetError()<<endl;
       success=false;
    }
    else
    {
       //Create the renderer
       PrimoRend = SDL_CreateRenderer(PrimoWindow,-
1,SDL_RENDERER_PRESENTVSYNC);
       if( PrimoRend == NULL )
                        {
                                cout<<"Renderer could not be created! SDL Error:\n"<<
SDL_GetError()<<endl;
                                success = false;
                        }
                        else
       {
          SDL_SetRenderDrawColor(PrimoRend,0xE0,0xFF,0xFF,0xFF);
          int imgFlags= IMG_INIT_PNG;
          if(!(IMG_Init(imgFlags) &imgFlags ))
          {
             cout<<"SDL image could not initialize."<<IMG_GetError();
             success=false;
          }
          //Initialize SDL_ttf
                                if( TTF_Init() == -1 )
                                {
                                        cout<<"SDL_ttf could not initialize! SDL_ttf
Error:"<<TTF_GetError()<<endl;
```

```
                                        success = false;
                                    }
                                    LoadFont();
            }
        }
        return success;
    }
}

void Closer()
{
        //Destroy window
        SDL_DestroyRenderer( PrimoRend );
        SDL_DestroyWindow( PrimoWindow );
        PrimoWindow = NULL;
        PrimoRend = NULL;

        //Quit SDL subsystems
        IMG_Quit();
        SDL_Quit();
        TTF_Quit();
}

bool LoadFont()
{
    bool success = true;
    Pfont = TTF_OpenFont("Calibri.ttf",1000);
    if(Pfont==NULL)
    {
        cout<<"Failed to load the font file."<<TTF_GetError()<<endl;
        success=false;
    }
    return success;
}
```

```
int main(int argc , char* argv[])
{
   LEVEL=levels[LevelCounter];
   NewLevel();

   if(!Initialiser())
   {
      cout<<"Failed to initialize.";
   }
   else
   {

      SDL_Event e;
      Welcome.Texture_Loader("Welcome.png");
      Instructions.Texture_Loader("Instructions.png");
      bool Displaying=true;
      while(Displaying)
      {
         SDL_RenderClear(PrimoRend);
         SDL_SetRenderDrawColor(PrimoRend,0xFF,0xFF,0xFF,0xFF);
         SDL_RenderCopy(PrimoRend,Welcome.giveTexture(),NULL,NULL);
         SDL_RenderPresent(PrimoRend);
         bool Displayinginst=true;
         while(SDL_PollEvent(&e)!=0)
         {
            if(e.type==SDL_QUIT)
            {
               quit= true;
               Displaying=false;
               break;
            }
```

```
else if( e.type==SDL_KEYDOWN)
{
   switch(e.key.keysym.sym)
   {
   case SDLK_SPACE:
   {
      Displaying=false;
      break;
   }
   case SDLK_ESCAPE:
      {
         Displaying=false;
         quit=true;
         break;
      }
   case SDLK_i:
      {
         while(Displayinginst)
         {
            SDL_RenderClear(PrimoRend);
            SDL_SetRenderDrawColor(PrimoRend,0xFF,0xFF,0xFF,0xFF);
            SDL_RenderCopy(PrimoRend,Instructions.giveTexture(),NULL,NULL);
            SDL_RenderPresent(PrimoRend);
            while(SDL_PollEvent(&e)!=0)
               {
                  if(e.type==SDL_QUIT)
                  {
                     quit= true;
                     Displayinginst=false;
                     break;
                  }
                  else if (e.type==SDL_KEYDOWN)
                  {
                     switch(e.key.keysym.sym)
```

```
                    {
                    case SDLK_SPACE:
                        {
                            Displayinginst=false;
                            Displaying=false;
                            break;
                        }
                    case SDLK_ESCAPE:
                        {
                            Displayinginst=false;
                            Displaying=false;
                            quit=true;
                            break;
                        }
                    case SDLK_BACKSPACE:
                        {
                            Displayinginst=false;
                            break;
                        }
                    }
                }
            }
        }

    }
  }
}

//Game loop starts
    do
    {
      //Handle events on queue
```

```
while( SDL_PollEvent( &e ) != 0 )
{
        //User requests quit
        if( e.type == SDL_QUIT )
        {
                quit = true;
        }

         if( e.type == SDL_KEYDOWN )
        {
                //Select functions based on key press
                switch( e.key.keysym.sym )
                {
                   case SDLK_UP:
                   {
                      UpAdder();
                      RandomGenerator();
                      break;
                   }
                   case SDLK_DOWN:
                   {
                      DownAdder();
                      RandomGenerator();
                      break;
                   }
                   case SDLK_RIGHT:
                   {
                      RightAdder();
                      RandomGenerator();
                      break;
                   }
                   case SDLK_LEFT:
                   {
                      LeftAdder();
```

```
                                        RandomGenerator();
                                        break;
                                    }
                                    case SDLK_ESCAPE:
                                    {
                                        quit=true;
                                        break;
                                    }
                                }
                            }
                        }
                        SDL_SetRenderDrawColor(PrimoRend,0x00,0x80,0xFF, 0xFF
);
                        SDL_RenderClear( PrimoRend );
                        main1.DrawBoard();

                        for(int k_Load=2,i_Load=0;i_Load<3;i_Load++)
        {
            for(int j_Load=0;j_Load<3;j_Load++)
            {
                main1.EasyLoader(Objects[i_Load][j_Load].Givevalue(),k_Load);
                 main1.FinalRender(k_Load);
                k_Load++;
            }
                        }

        char Nepal[]="Primo";
                        main1.LoadFromRenderedText(Nepal,ColorTitle,Box0,Pfont);
                        main1.FinalRender(Box0);
                        main1.EasyLoader(LEVEL,Box1);
                        main1.FinalRender(Box1);
                        SDL_RenderPresent(PrimoRend); //To update the screen

        CheckLevel();
```

```
                    //SDL_RenderPresent( PrimoRend );
                    SDL_Delay(250);

        }while(!quit);
    }

    Closer();
    return 0;
}
```

9. Result: A sample run of the program is shown below:

Figure 2:Welcome Window

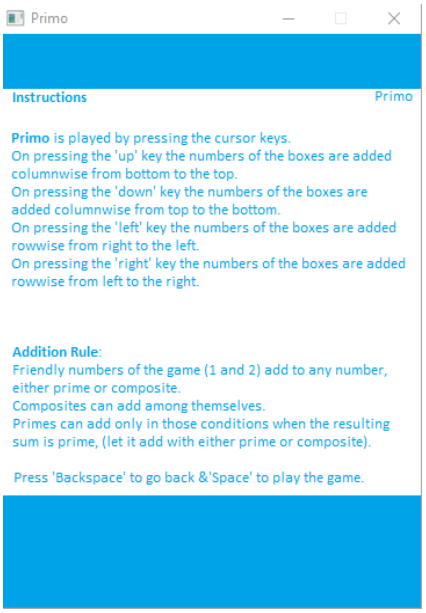


Figure 2:Instructions Window

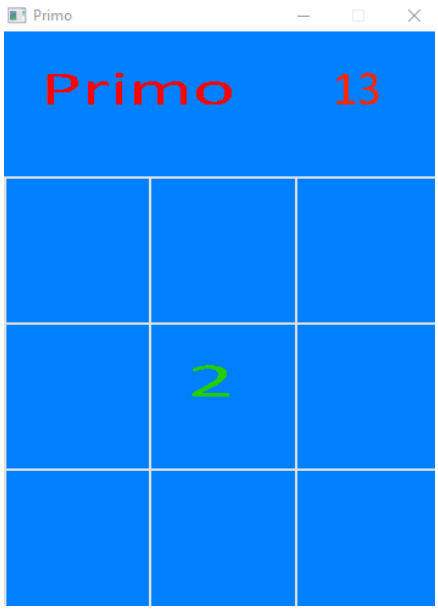


Figure 3: Initial Beginning of Level 13

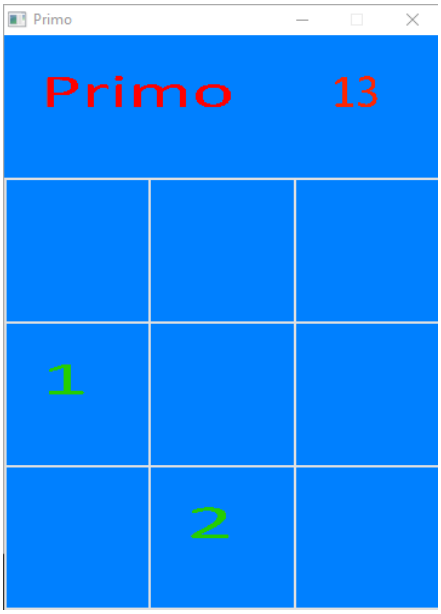


Figure 4: Pressing Down Key

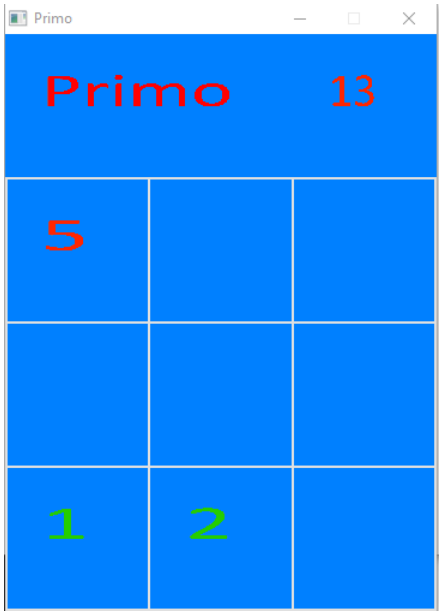


Figure 5: Pressing Down Key

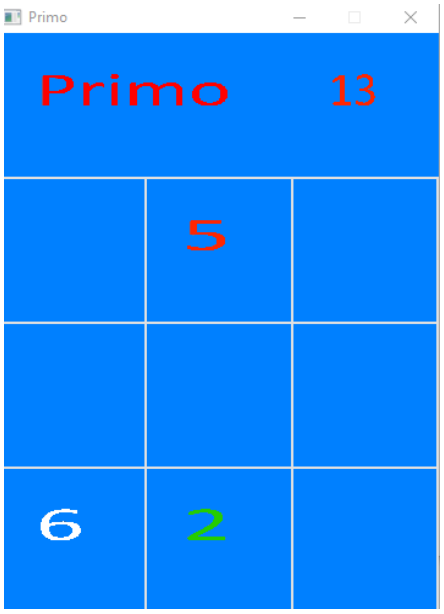


Figure 6: Pressing Down Key

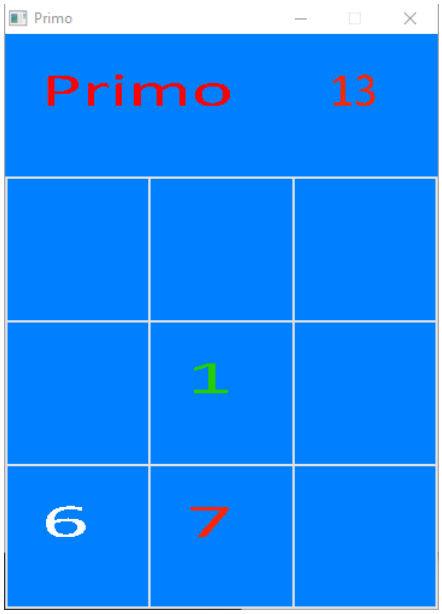


Figure 7: Pressing Down Key

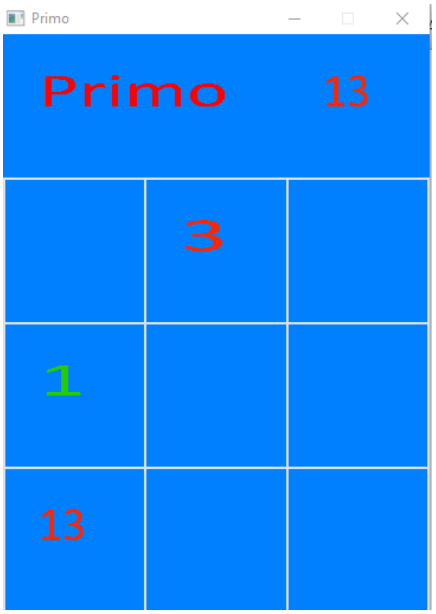


Figure 8: Pressing Left Key

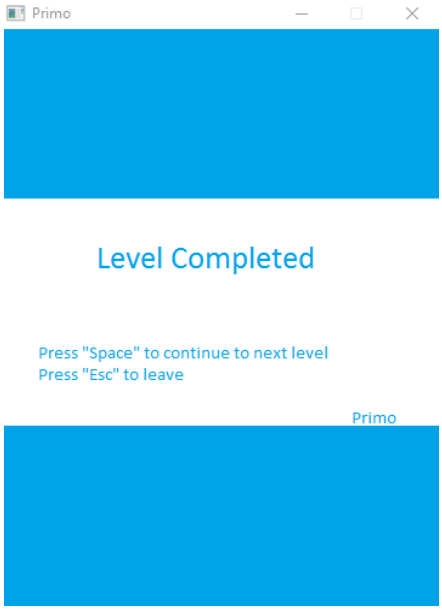


Figure 9: Level Completion

10. Problems Faced and Solutions

During the addition of two numbers as per the game rule, the values of both the numbers should change, one should take the sum whereas another a zero value. Simply overloading by passing value could have made the program more complex and error-prone, so the argument was passed by reference using pointer. So, two objects were given new values just by using + operator between them.

During the development of the playing board of the game, nine rectangles were drawn, one for each box. But, the edges of the boxes overlapped and made a little bit darker lines inside the board but thinner lines at the outer edges of the board. So, to remove this irregularity, a line just beside each outer edge was drawn to make the board uniform.

In the game, the welcome window, instruction window, game over and level completion window just were images to be loaded in the program. So instead of making a global function and different textures, a class *LoadingImages* was defined and those images were loaded instantiating four objects of the class.

In the program it was needed to convert the integer to string because our function which converted the font to image used string as argument. So, to convert integer to string, a stringstream variable was declared and was used as in the code.

11. Limitations and Future Enhancements

This game is in the form of 3x3 table consisting of nine boxes to play. However, higher the primes the game becomes difficult to play in the small board. So, this limitation is to be broken and the game will be expanded to 4x4 and 5x5 versions. While playing the game, the menu buttons and sound is not available so these features will be added.

The program will be made more user friendly, attractive and more logical; meantime more efficient as well. Our future plan is to make this game for **Android** operating system and put it in the play store.

12. Conclusion and Recommendations

Primo is a way to express how mathematics can be fun. We are happy that we tried something that related prime numbers in to a game. The object oriented approach programming i.e. using objects and classes helped to model the game in programming easier and the features provided by this approach such as Polymorphism (operator overloading), friend functions etc. proved very useful and efficient during the game programming.

We recommend our game to be played not only by geeks and math enthusiasts but anybody who wants some refreshments. We would be overwhelmed, if our program by any means would contribute to something new in mathematics and something new experience for user.

13. References

Following references were used for developing the program:

a. Robert Lafore, “Object Oriented Programming in C++”, 4th Edition 2002, Sams Publication
b. Daya Sagar Baral and Diwakar Baral, “Secrets of Object Oriented Programming in C++”, 1st Edition 2010, Bhundipuran Prakasan
c. www.lazyfoo.net
d. www.wikipedia.org