Visvesvaraya Technological University
Belgaum - 590014

Mini-Project Entitled

ReMorse
REpresentative MORSe Educator

For the academic year 2016-2017
Submitted by:

Ravikiran R (1MV14CS085)
Raghava G Dhanya (1MV14CS077)

Project carried out at
Sir M. Visvesvaraya Institute of Technology
Bangalore-562157

Under the guidance of
Mrs. Monika Rani H G
Asst.Professor, Dept.of.CSE
Sir. MVIT, Bangalore

Department Of Computer Science And Engineering
Sir M. Visvesvaraya Institute of Technology
Hunasamaranahalli, Bangalore - 562157

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY
BANGALORE-562157.

(Affiliated to Visvesvaraya Technological University, Belagavi)
Bangalore-562157

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the mini-project work entitled **"ReMorse - REpresentative MORSe Educator"** is a bonafide work carried out by **Ravikiran R (1MV14CS085)** and **Raghava G Dhanya (1MV14CS077)** in partial fulfillment for the award of Degree of **Bachelor of Engineering** in **Computer Science & Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2016-2017 in **Computer Graphics and Visualization Laboratory**. The mini-project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the course of Bachelor of Engineering Degree.

Signature of the Guide

Mrs.Monika Rani H G
Asst. prof, Dept of CSE
Sir MVIT

Signature of the HOD

Prof. Dilip K Sen
HOD, Dept of CSE
Sir MVIT

External Examiner

Internal Examiner

Acknowledgement

We are truly grateful to our college **Sir M. Visvesvaraya Institute of Technology, Bangalore** for giving us an opportunity to work on this mini-project.

We are thankful to **Prof. Dilip K Sen, Head of Department, Computer Science and Engineering** for providing us with the required facilities. We wish to express our heartfelt gratitude to our teacher **Mrs. Monika Rani H G** for her valuable suggestions and for guiding us through this mini-project.

We thank **Samuel F. B. Morse**, **inventor** of the Telegraph and Morse Code, on which this mini-project is based.

Our immense gratitude to members of the faculty of the **department of CSE, Sir MVIT.** Lastly we wish to thank our friends for their help and support.

Ravikiran R **(1MV14CS085)**

Raghava G Dhanya **(1MV14CS077)**

Abstract

The mini-project is built as a 2D side scrolling game in openGL. It consists of a main character whose objective is to avoid collision with obstacles by jumping over them. The frequency of obstacles corresponds to the Morse representation of the letters shown on the screen.

There are two kinds of obstacles in the game. One type is represented by a series of three triangles, and the other by a single triangle. These two types of obstacles correspond to the two fundamental characters in a Morse code sequence, namely a dash and a dot (respectively).

When the game commences, a single letter is displayed on the screen. This letter is chosen randomly. As the player moves forward, he is faced by these triangular obstacles. To avoid collision, the player has to jump over them. Jumping over the single triangle requires a short jump (short tap on the jump key) while a long obstacle requires a long jump (long tap on the jump key).

The selection and occurrence of these obstacles is governed by the Morse Code representation of the letter displayed on the screen. As the player progresses in the game, he is faced by more such letters and obstacles. Since the person playing the game watches the letter on the screen and simultaneously taps out the Morse Code for that letter, he/she learns to associate the tap sequence with the letter.

Contents

1 Introduction **1**

1.1 Morse Code 1

1.2 History of Morse Code 2

1.3 Applications of Morse Code 2

1.3.1 Aviation 2

1.3.2 Amateur radio 2

1.3.3 General Public 3

1.3.4 Assistive Technology 3

1.4 Associative learning 3

1.5 Computer Graphics 3

1.6 Applications of Computer Graphics 4

2 ReMorse - The Game **5**

2.1 Problem Statement 5

2.2 Existing System 5

2.3 Proposed System 5

3 System Requirements **6**

3.1 Hardware Requirements 6

3.2 Software Requirements 6

3.3 User Requirements 6

4 Design and Implementation **7**

4.1 Design 7

4.2 Implementation 8

4.2.1 Structure 8

4.2.2 External Libraries 9

4.2.3 Source Code File Structure 10

4.3.4 Header Files 11

4.3.5 Tools Used 12

5 Code Snippets **13**

5.1 Physics 13

5.1.1 PhysicalObject 13

5.1.2 Player 13
5.1.3 Wall 14
5.1.4 Obstacle 14
5.1.5 ObstacleManager 15
5.1.6 ContactListener 16
5.1.7 LetterToMorse 16
5.2 Graphics 17
5.2.1 Antialiasing 17
5.2.2 Image inversion 17
5.2.3 Setting texture 18
5.2.4 Game Loop 18
6 Screenshots 20
7 Conclusion 23
Bibliography 24

List of Figures

Figure	Description	Page
1.1	International Morse Code	1
1.2	Applications of Computer Graphics	4
4.1	A state machine	7
4.2	Structure of ReMorse	8
6.1	Start screen	20
6.2	Gameplay - Player on the ground	21
6.3	Gameplay - Player jumping	21
6.4	Pause screen	22
6.5	Game-over screen	22

Chapter 1

Introduction

1.1 Morse Code

Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment. It is named for Samuel F. B. Morse, an inventor of the telegraph.

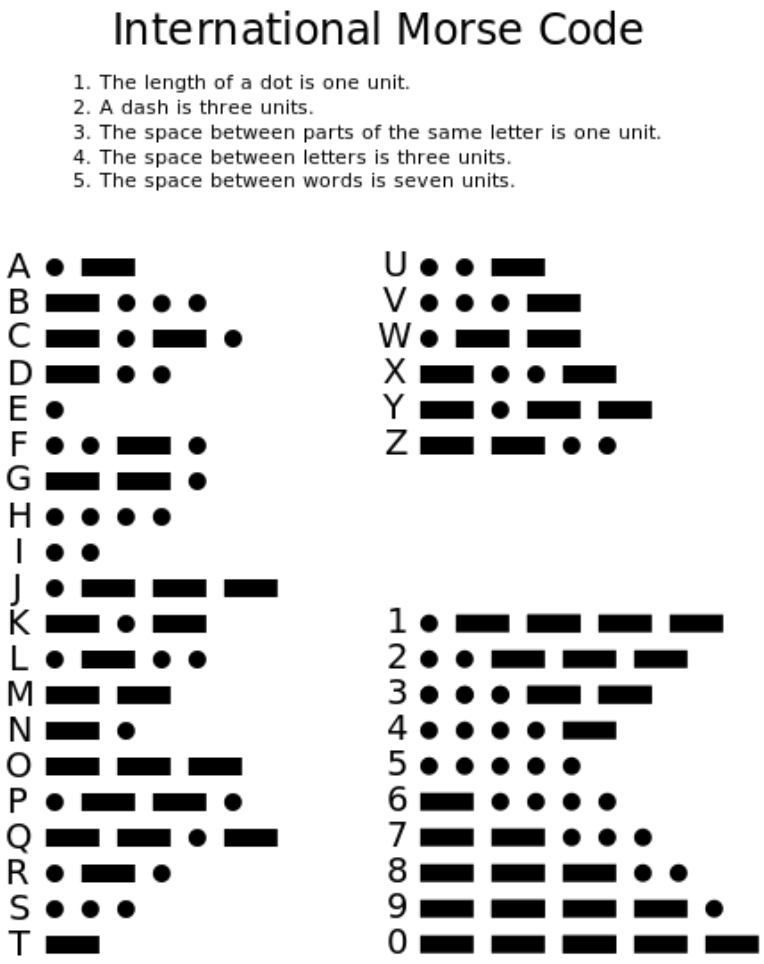

Fig 1.1 International Morse Code

1.2 History of Morse Code

In 1836, the American inventor Samuel F. B. Morse, American physicist Joseph Henry, and Alfred Vail developed an electrical telegraph system. This system sent pulses of electric current along wires which controlled an electromagnet that was located at the receiving end of the telegraph system. A code was needed to transmit natural language using only these pulses, and the silence between them. Therefore, around 1837, Morse developed an early forerunner to the modern International Morse code.

In the 1890s, Morse code began to be used extensively for early radio communication, before it was possible to transmit voice. In the late 19th and early 20th centuries, most high-speed international communication used Morse code on telegraph lines, undersea cables and radio circuits. In aviation, Morse code in radio systems started to be used on a regular basis in the 1920s.

1.3 Applications of Morse Code

1.3.1 Aviation

In aviation, instrument pilots use radio navigation aids. To ensure that the stations the pilots are using are serviceable, the stations all transmit a short set of identification letters (usually a two-to-five-letter version of the station name) in Morse code.

1.3.2 Amateur radio

International Morse code today is most popular among amateur radio operators, where it is used as the pattern to key a transmitter on and off in the radio communications mode commonly referred to as continuous wave.

1.3.3 General Public

An important application is signalling for help through SOS, (. . . _ _ _ . . .). This can be sent many ways: keying a radio on and off, flashing a mirror, toggling a flashlight, and similar methods.

1.3.4 Assistive Technology

Morse code has been employed as an assistive technology, helping people with a variety of disabilities to communicate.

1.4 Associative learning

Association in psychology refers to a mental connection between concepts, events, or mental states that usually stems from specific experiences. Associative learning is when a subject creates a relationship between stimuli or behavior and stimulus. This view of learning assumes that the acquisition of associations is the basis for learning.

1.5 Computer Graphics

The computer graphics phrase was coined in 1960 by researchers Verne Hudson and William Fetter. It's often abbreviated as CG. It was understood in the past, in a broad sense, to describe "anything on computers that are not text or sound". With a current definition, we can say that computer graphics is the field of computer science that has the objective of producing images by modeling, manipulating, storing geometric entities, and subsequent rendering, for which a computer generates the representation of a three-dimensional scene in the form of a two-dimensional image.

The development of computer graphics has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design in general. In the 1990s, other fields such as visualization of information were developed, and a more scientific viewpoint focused on representation of

three-dimensional phenomena (architecture, meteorology, medicine, biology, etc). Emphasis is placed on realistic rendering of volumes, surfaces, sources of illumination, and so on, perhaps with a dynamic component (time).

1.6 Applications of Computer Graphics

We can classify applications of computer graphics into four main areas:

1. Display of information
2. Design
3. User interfaces
4. Simulation

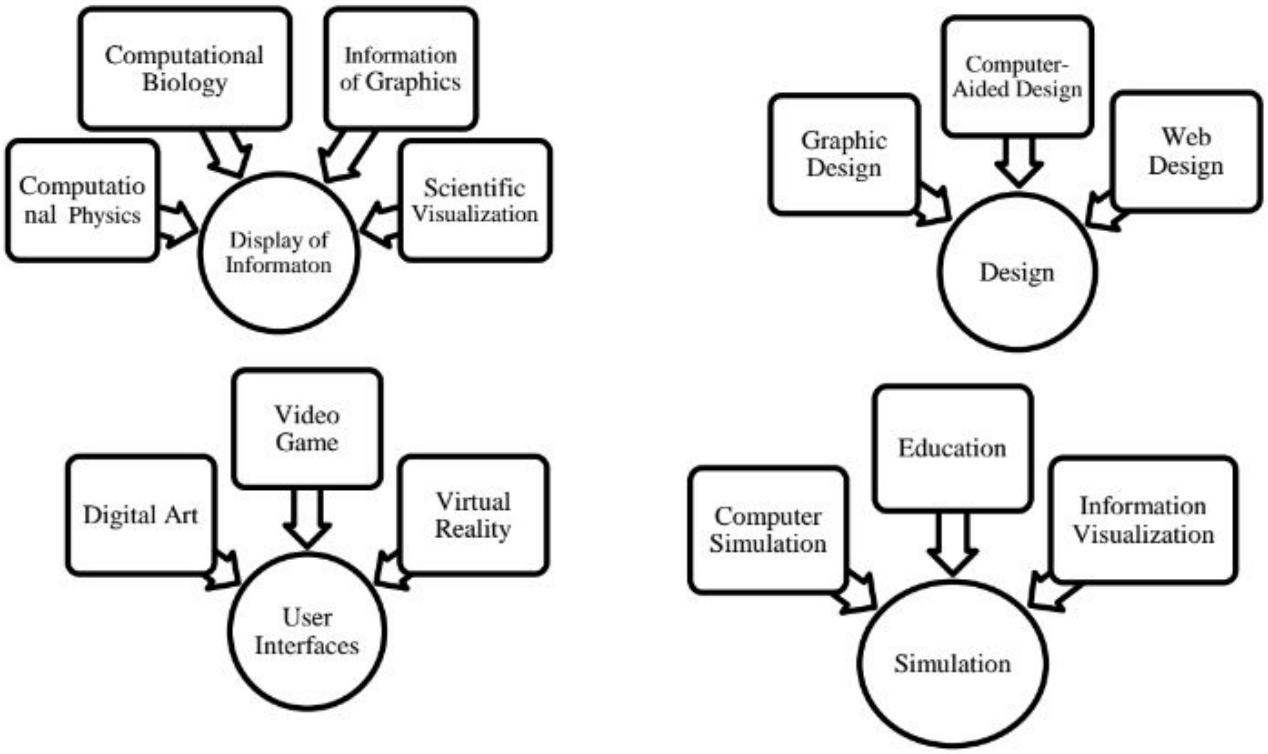

Fig 1.2 Applications of Computer Graphics

Chapter 2

ReMorse - The Game

2.1 Problem Statement

Development of an application to make learning of Morse Code fun and intuitive. The user must be able to subconsciously associate the letters of the alphabet with their corresponding Morse Code representation.

2.2 Existing System

People learning Morse code using the Farnsworth method are taught to send and receive letters and other symbols at their full target speed, that is with normal relative timing of the dots, dashes, and spaces within each symbol for that speed.

Another popular teaching method is the Koch method, which uses the full target speed from the outset but begins with just two characters. Once strings containing those two characters can be copied with 90% accuracy, an additional character is added, and so on until the full character set is mastered.

2.3 Proposed System

A game to make learning Morse Code a fun experience, while subconsciously training the player's mind to associate letters with the correct sequence. As the letters are displayed on the screen, the player must be able to tap the corresponding sequence correctly. Motivation to continue playing is with the help of a scoring system and displaying the high score.

Chapter 3

System Requirements

3.1 Hardware Requirements

- 100MB of free disk space
- 64MB of RAM
- Graphic card supporting OpenGL 1.2 or higher
- CPU of x86 or x64 architecture with at least 266MHz clock speed
- A display monitor
- Keyboard

3.2 Software Requirements

- Windows or Linux Operating System
- Display driver supporting OpenGL 1.2 or higher

3.3 User Requirements

- Good hand-eye coordination
- Willing to learn Morse Code

Chapter 4

Design and Implementation

4.1 Design

The entire lifecycle of the program can be represented as a state machine. Transitions between states occur on events such as key presses and collisions.

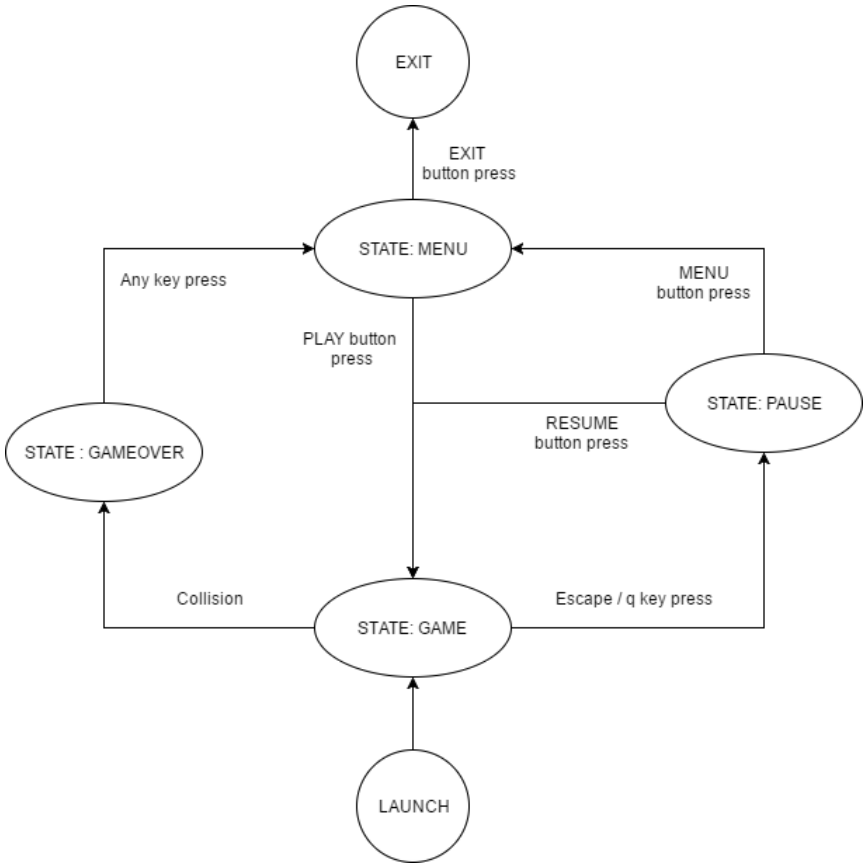

FIG 4.1 A state machine

4.2 Implementation

4.2.1 Structure

The code can be divided into two classes, frontend and backend. The backend consists of the game logic and interfaces with the physics library to compute quantities such as impulse and velocity. It also determines occurrence of events like collision. Frontend handles the display logic, including graphics and user input events.

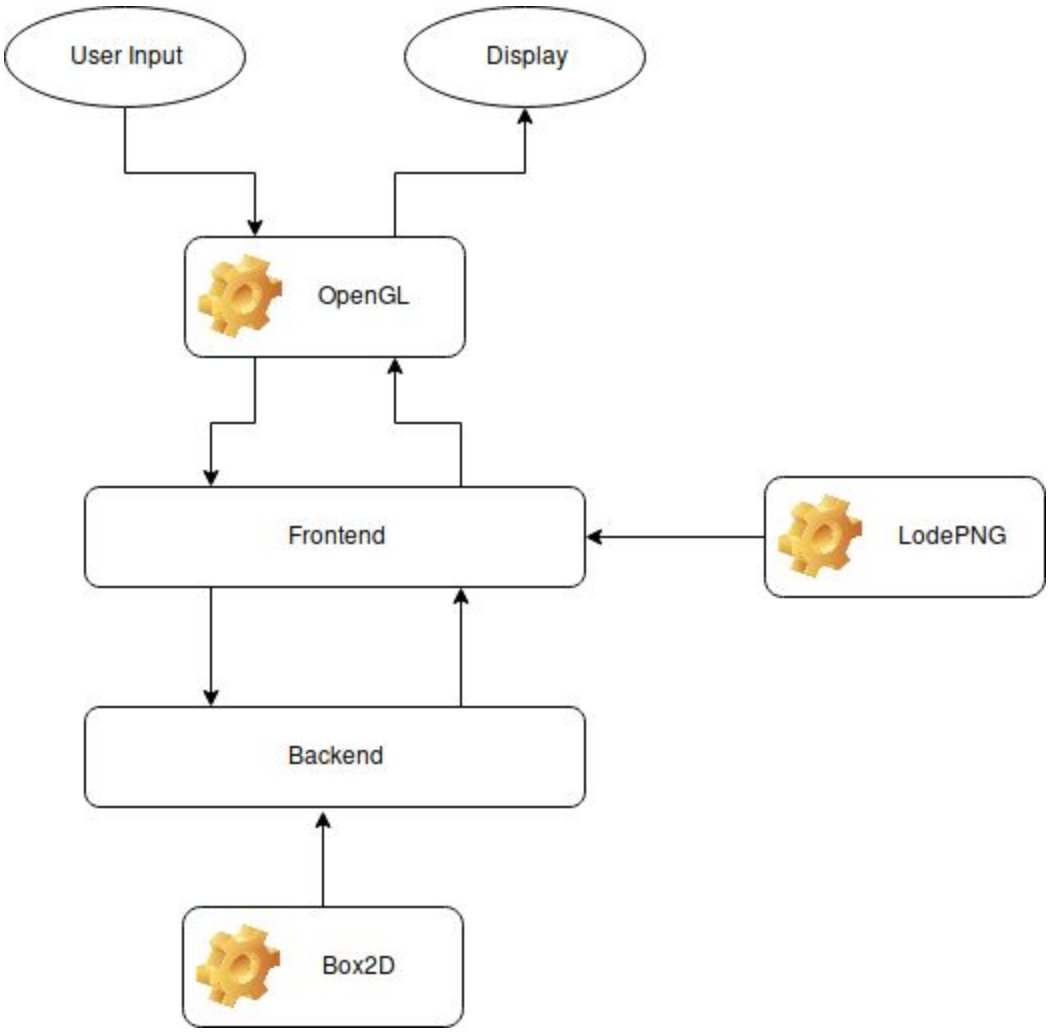

Fig 4.2 Structure of ReMorse

4.2.2 External Libraries

1. **OpenGL**

   OpenGL is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three dimensional scenes from simple primitives. We use this library to render the scenes to the screen as well as to handle user input events.

2. **GLUT**

   GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT provides a portable API, so OpenGL programs work across all PC and workstation OS platforms. It also has a C++ binding. We use this for providing a cross platform GUI with windowing and key event handling.

3. **LodePNG**

   LodePNG is a PNG image decoder and encoder with no dependency on zlib and libpng. It has a convenient C++ wrapper. We use this library to load images for both just to display and to use them as textures in OpenGL.

4. **Box2D**

   Box2D is an open source C++ engine for simulating rigid bodies in 2D with fairly accurate reaction forces/impulses and continuous collision detection. It has been used in many popular games including Angry Birds and Limbo. We use this library as an engine for computation of physical quantities and for collision detection.

4.2.3 Source Code File Structure

This file structure of the project is shown below.

```
ReMorse
├──── compile.sh
├──── keys.cpp
├──── keys.h
├──── lib
│     ├──── lodepng.cpp
│     └──── lodepng.h
├──── LICENSE
├──── main.cpp
├──── physics.cpp
├──── physics.h
├──── README.md
├──── remorse.h
├──── res
│     ├──── morse.png
│     ├──── sam01.png
│     ├──── sam02.png
│     ├──── sam03.png
│     └──── sam04.png
├──── states.cpp
├──── states.h
├──── timer.cpp
└──── timer.h
```

The file **main.cpp** contains glut initialisations, display loops for all the states of the game and also some custom functions to draw custom objects etc.
The files **states.*** contain state variables and values. This is used to set and get state of the game
The files **keys.*** contain functions to handle key events for all the game states.
The files **physics.*** contains game logic and collision detection
The files **timer.*** contains glut timer functions to maintain FPS and synchronise backend and frontend
The folder **res** contains image resources for the game
The folder **lib** contains external library headers

4.3.4 Header Files

- **iostream**
  Provides standard C++ input/output streams. Used in console logging for debugging.
- **string**
  The standard string class provides support for string objects. Used for handling and manipulating strings with ease.
- **sstream**
  Stream class to operate on strings. Used to format strings to C++ standard.
- **fstream**
  Input/output stream class to operate on files. Used to persist high score on the system.
- **ctime**
  Contains functions to get and manipulate date and time information. Used to seed the random number generator.
- **stdlib.h**
  Provides several general purpose functions including random number generation. Used to generate random letters for the obstacles.
- **glut.h**
  It implements a simple windowing application programming interface (API) for OpenGL. GLUT provides a portable API.
- **glext.h**
  A header file with all the opengl extensions and constants defined in it. We use it mainly for MultiSample AntiAliasing (MSAA)
- **keys.h**
  Custom header for interfacing key handling functions to other source files and also to access variables defined in keys.cpp.
- **physics.h**
  Abstracts the backend functionality for the frontend by providing position values and functions.
- **remorse.h**
  Custom header for including generic headers of GLUT and OpenGL.

- **states.h**
  Custom header to access and change game state variables from other source files.
- **timer.h**
  Custom header for interfacing timer functions to other source files and also to access variables defined in timer.cpp.

4.3.5 Tools Used

- **GCC**
  The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages and is the standard compiler for most Unix-like Operating Systems. We used it to compile, run and debug the project.
- **Code::Blocks**
  Code::Blocks is a free, open-source cross-platform IDE that supports multiple compilers including GCC. We used this IDE mainly in windows platform.
- **Sublime Text**
  Sublime Text is a proprietary cross-platform source code editor. We used this mainly in the Linux platform.
- **Git**
  Git is a free and open source distributed version control system designed to handle projects with speed and efficiency. We used Git to keep track of changes to the source code and coordinating work across development environments.
- **GitHub**
  GitHub is a web-based Git or version control repository and Internet hosting service. We made use of free private repositories for students to host our project and stay synchronized.
- **Travis CI**
  Travis CI is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub. We use this for Test-driven development (TDD), which is a software development process that relies on the repetition of a very short development cycle.

Chapter 5

Code Snippets

5.1 Physics

5.1.1 PhysicalObject

```
class PhysicalObject
{
    protected:
        b2Body *body;
        b2World *world;
        b2BodyDef bodyDef;
        b2PolygonShape shape;
        b2FixtureDef fixtureDef;

    public:
        PhysicalObject(b2World *world);
        ~PhysicalObject();

        Config getConfig();
};
```

Acts as the base class for other entities representing physical objects. Contains Box2D specific variables for defining shape, size and other properties of generic physical entities.

5.1.2 Player

```
class Player: public PhysicalObject
{
    static constexpr float WIDTH = 1.10/2.0;
    static constexpr float HEIGHT = 1.86/2.0;

    static const int MAX_JUMP = 4;
    static const int JUMP_IMPULSE = 30;

    int GROUND_POS = 5;

    bool inAir;     //To hover on long press
    public:
        Player(b2World *world, Config initConfig): PhysicalObject(world);

        ~Player() {}

        void setGroundPos(int pos);
        void setPos(int x, int y);

        /* When called with true, player will jump with initial impulse, and hover there.
           When called with false, player will fall if hovering */
        void setJump(bool jumpEnable);

        float getXPos();
        float getYPos();
};
```

Subclass of PhysicalObject representing the main character of the game. Provides functions to get/set player's position and for jumping.

5.1.3 Wall

```
class Wall: public PhysicalObject
{
    static constexpr float WIDTH = 200.0f;
    static constexpr float HEIGHT = 5.0f;

    public:
        Wall(b2World *world, Config initConfig): PhysicalObject(world);

        ~Wall() {}

        float getHeight();
};
```

Subclass of PhysicalObject for the static physical objects, in our case the ground. Configures the body and provides function to return its height.

5.1.4 Obstacle

```
class Obstacle
{
    //Max of 5 characters in a Morse letter
    //Worst case of three obstacles for each
    static const int MAX_BODIES = 15;

    //Width and height for dot obstacle
    static constexpr float DOT_WIDTH = 0.9;
    static constexpr float DOT_HEIGHT = 0.8;

    //Width and height for dash obstacle
    static constexpr float DASH_WIDTH = 0.6;
    static constexpr float DASH_HEIGHT = 0.5;

    static const int speed = 10;      //Speed of each obstacle
    static const int spacing = 7;    //Spacing between obstacles

    float curPos;                      //Starting position of first body
    char curLetter;                  //The letter represented by this obstacle
    char lastMorseChar;              //Last letter of Morse characters

    float groundLevel = 5.0;

    b2World *world;
    b2Body *bodies[MAX_BODIES];

    //Creates a triangular kinematic body in bodies[] array at the index
    void createBody(int index, float base, float height, Config initConfig);

    void destroyBody(int index);
```

```
    public:
        Obstacle() {}
        Obstacle(b2World *world, float startPos);

        ~Obstacle() {}

        /* Calling this function will destroy existing bodies, and create
        new ones to represent the new letter. Although this can be optimized
        by doing checks such as checking for same letter, retain similar
        bodies etc, it is not done for the sake of simplicity */
        void setLetter(char letter);

        //Destroys all bodies
        void reset();

        //Set curPos value. Change will reflect only when setLetter is called.
        void setCurPos(int pos);

        void setGroundLevel(int gLevel);

        //Returns right most coordinate of last body in the array(bottom right vertex)
        int getLastPos();

        float getBodyPos(int index);
        int getBodyType(int index);
        char getCurLetter()
};
```

Class representing a single letter. Holds a set of bodies corresponding to the Morse of the letter. A dot is a single triangular body and a dash is a set of three smaller triangles.

5.1.5 ObstacleManager

```
class ObstacleManager
{
    static const int LEFT_BOUNDARY = -2;
    static const int BUFFER_SIZE = 4;
    static const int letter_spacing = 15;

    int curIndex;                       //Index of left most obstacle
    int textIndex;                      //Index of current letter in letterQueue
    string letterQueue;                 //String of letters
    Obstacle buffer[BUFFER_SIZE];       //Obstacles that will be recycled

    int numReset;                       //To keep track of number of times reset() has been called
    int groundLevel = 5;

    public:
        //Important: text should have atleast one character
        ObstacleManager();

        ~ObstacleManager() {}

        //Call to reinitialize.
        //There is no reset method for ObstacleManager because the physics bodies are explicitly destroyed
        void init(b2World* world, string text);

        void update();
        void updateTriPos(float arr[85][2]);
        void setGroundLevel(int level);
        char getDisplayChar();
};
```

Class that manages the obstacles and the sequence in which they arrive. Updates the values returned to the display system periodically to draw the obstacles.

5.1.6 ContactListener

```
class ContactListener: public b2ContactListener
{
    bool collided;

    public:
        ContactListener();
        ~ContactListener() {}

        bool hasCollided();

        //Gets called on contact between two fixtures A and B
        void BeginContact(b2Contact* contact);

        void reset();
};
```

Subclass of Box2D provided ContactListener to receive callbacks on collision. Main function continuously polls for collision check using hasCollided function.

5.1.7 LetterToMorse

```
string letterToMorse(char let)
{
    //Reference: https://en.wikipedia.org/wiki/Morse_code#/media/File:International_Morse_Code.svg
    switch(let=toupper(let))
    {
        case 'A': return ".-";      case 'B': return "-...";
        case 'C': return "-.-.";    case 'D': return "-..";
        case 'E': return ".";       case 'F': return "..-.";
        case 'G': return "--.";     case 'H': return "....";
        case 'I': return "..";      case 'J': return ".---";
        case 'K': return "-.-";     case 'L': return ".-..";
        case 'M': return "--";      case 'N': return "-.";
        case 'O': return "---";     case 'P': return ".--.";
        case 'Q': return "--.-";    case 'R': return ".-.";
        case 'S': return "...";     case 'T': return "-";
        case 'U': return "..-";     case 'V': return "...-";
        case 'W': return ".--";     case 'X': return "-..-";
        case 'Y': return "-.--";    case 'Z': return "--..";
    }

    return "0";
}
```

Returns the Morse string of dots('.') and dashes('-') for given letter. Used by Obstacle class to set the array of physical bodies.

5.2 Graphics

5.2.1 Antialiasing

```
void antialias()
{
    if(R_settings::ANTIALIAS)
    {
        ////////////////////////Do anti alias//////////////////////////
        /* smoothen lines n points, doesn't seem to get affected by MULTISAMPLE.
        works only if called after the BlendFunc */
        glEnable(GL_POINT_SMOOTH);
        glEnable(GL_LINE_SMOOTH);
        // creates spaces (lines) bw polygon if multisample does not work
        glEnable(GL_POLYGON_SMOOTH);
        glEnable(GL_MULTISAMPLE);
        glutSetOption(GLUT_MULTISAMPLE, 8);
        ////////////////////////end of anti alias/////////////////////
    }
    else
    {
        glDisable(GL_MULTISAMPLE);
        printf("MSAA off\n");
    }
}
```

We use MultiSample AntiAliasing (MSAA).

5.2.2 Image inversion

```
void invert(vector<unsigned char> &img,const unsigned width,const unsigned height)
    {
        unsigned char *imagePtr = &img[0];
        unsigned char *top = NULL;
        unsigned char *bottom = NULL;
        unsigned char temp = 0;
        for( int h = 0; h <(int) height/2; ++h )
        {

            top = imagePtr + h * width * 4;
            bottom = imagePtr + (height - h - 1) * width*4;
            for( int w = 0; w < (int)width*4; ++w )
            {
                temp = *top;
                *top = *bottom;
                *bottom = temp;
                ++top;
                ++bottom;
            }
        }
}
```

OpenGL has coordinate system in which y-axis increases as we move up the screen but most image formats have the coordinate system in which y-axis increases as

move down the screen. So we need to convert the images to openGL coordinate system. We can do this on vector storing image as in the above code snippet

5.2.3 Setting texture

```
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);

    // without this texture darkens
    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]);
}
```

The above function sets the current texture texname to given image.

5.2.4 Game Loop

```
void gameLoop()
{
    glClear(GL_COLOR_BUFFER_BIT );
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    setLetter(R_physics::curLetter);

    //display score
    //why StringStream? to_string() doesn't work in mingw compiler
    glLineWidth(3);
    glColor3ub(0xff,0xff,0xff);
    glPushMatrix();
    glTranslatef(5,HEIGHT-40,0);
    glScalef(.3,.3,0);
    ostringstream stm;
    stm<<R_physics::SCORE;
    glutStrokeString(GLUT_STROKE_ROMAN,(unsigned char*)stm.str().c_str());
    stm.str("");
    glPopMatrix();

    glPushMatrix();
    stm<<"$"<<R_physics::HIGHSCORE;
    glTranslatef(WIDTH- getButtonWidth(stm.str().c_str()),HEIGHT-40,0);
    glScalef(.3,.3,0);
    glutStrokeString(GLUT_STROKE_ROMAN,(unsigned char*)stm.str().c_str());
    glPopMatrix();
    /* enable texture.
    !!!!!!!!Very dangerous!!!!!!!. might affect other objects. disable before drawing other objects */
    glEnable(GL_TEXTURE_2D);
    {
        //scoping so that these variables aren't accessible elsewhere
        int sel=rand()%2;
        int i=(R_physics::jumpForceOn?2:0);
        setTexture(R_images::sam[i+sel],R_images::samWidth[i+sel],R_images::samHeight[i+sel]);
    }
    glPushMatrix();
```

```
    glBegin(GL_POLYGON);
    {
        //scoping so that these variables aren't accessible elsewhere
        float p00x = getScaled(R_physics::getPlayerX(), true);
        float p00y = getScaled(R_physics::getPlayerY(), false);
        float p01x = p00x;
        float p01y = getScaled(R_physics::getPlayerY()+R_physics::playerHeight*2.0, false);
        float p11x = getScaled(R_physics::getPlayerX()+R_physics::playerWidth*2.0, true);
        float p11y = p01y;
        float p10x = p11x;
        float p10y = p00y;

        glTexCoord2d(0,0);  glVertex2f(p00x, p00y);
        glTexCoord2d(0,1);  glVertex2f(p01x, p01y);
        glTexCoord2d(1,1);  glVertex2f(p11x, p11y);
        glTexCoord2d(1,0);  glVertex2f(p10x, p10y);
    }
    glEnd();
    glPopMatrix();
    glDisable(GL_TEXTURE_2D);
    //draw ground...
    glPushMatrix();
    glBegin(GL_POLYGON);
        glColor3ub(0xF4,0x43,0x36);
        glVertex2f(0,0);
        glVertex2f(WIDTH,0);

        glColor3ub(0xC6,0x28,0x28);

        float g_height = getScaled(R_physics::groundHeight, false);
        glVertex2f(WIDTH,g_height);
        glVertex2f(0,g_height);
    glEnd();
    glPopMatrix();

    glPushMatrix();
    glColor3ub(0x00,0x00,0x00);
    glBegin(GL_TRIANGLES);
    for(int i=0;i<60;i++)
    {
        if(R_physics::triPos[i][0]!=-1)
        {
            float wid=R_physics::dotWidth;
            float hei=R_physics::dotHeight;
            if(R_physics::triPos[i][1]==0)
            {
                wid=R_physics::dashWidth;
                hei=R_physics::dashHeight;
            }

            float x0=(float)R_physics::triPos[i][0]-wid/2.0;
            float y0=(float)R_physics::groundHeight;
            float x1=x0+wid/2.0;
            float y1=y0+hei;
            float x2=x0+wid;
            float y2=y0;
            glVertex2f(getScaled(x0,true),getScaled(y0,false));
            glVertex2f(getScaled(x1,true),getScaled(y1,false));
            glVertex2f(getScaled(x2,true),getScaled(y2,false));
        }
    }
    glEnd();
    glPopMatrix();
    glFlush();
}
```

This is the display function used when game is in the state GAME

Chapter 6

Screenshots

Fig 6.1 Start screen

Fig 6.2 Gameplay - Player on the ground

Fig 6.3 Gameplay - Player jumping

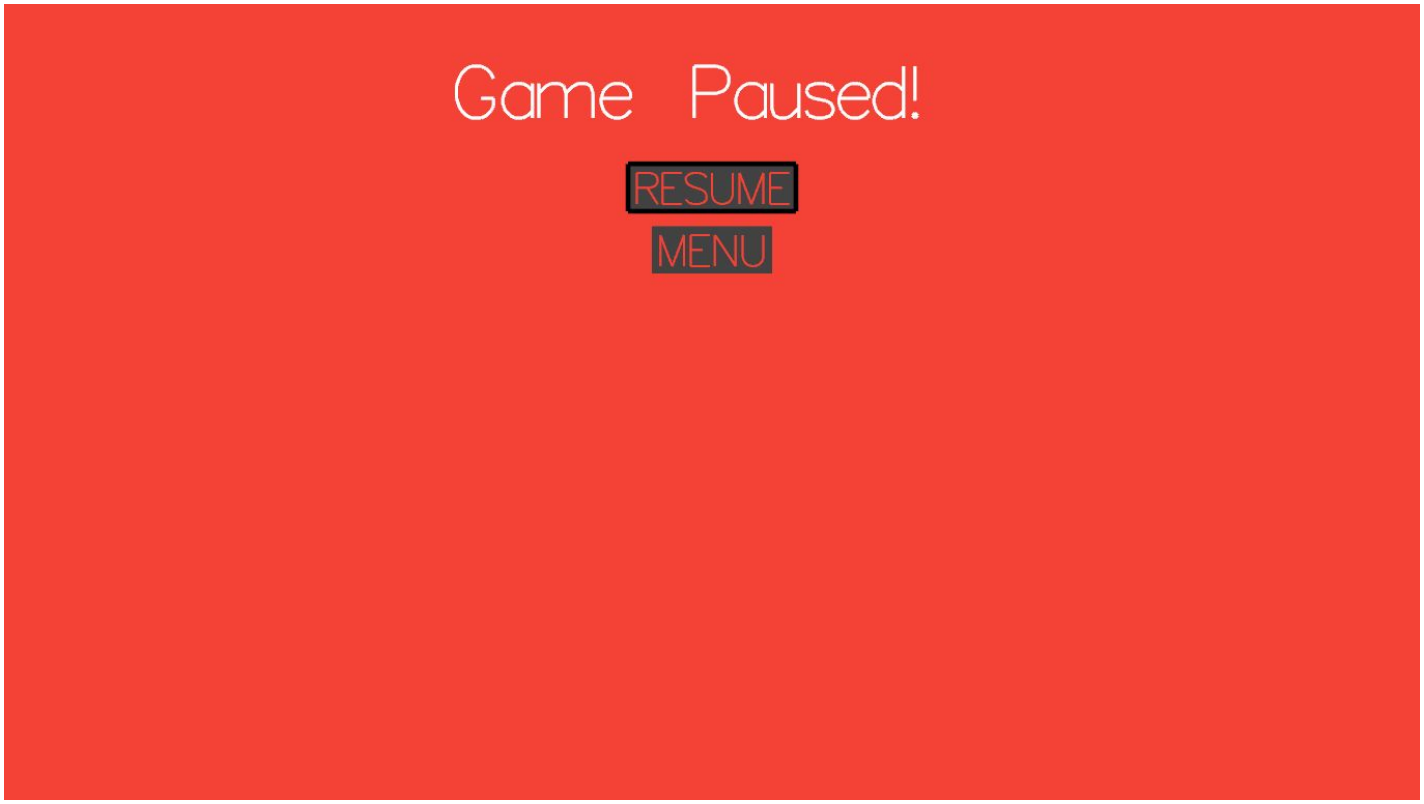

Fig 6.4 Pause screen

Fig 6.5 Game-over screen

Chapter 7

Conclusion

A desktop application (game) has been developed with which one can efficiently learn Morse code by practice. It makes the process a fun and enjoyable experience for the player. It can also be played as an arcade game, without needing to worry about Morse code.

With this mini-project, we learnt to make use of OpenGL for game development. We also learnt to effectively apply the concepts of object oriented programming to model real world scenarios in code. In this mini-project we made use of professional software development tools and practices. This includes version control and continuous integration tools, with agile and pair programming. It has been a great learning experience for us.

Bibliography

References

- Interactive Computer Graphics - Edward Angel
  5th edition, Addison Wesley, 2009
- OpenGL reference - https://www.khronos.org/opengl/
- C++ reference - http://www.cplusplus.com/
- Box2D - http://box2d.org/manual.pdf
- LodePNG - http://lodev.org/lodepng/
- Wikipedia - https://en.wikipedia.org/wiki/Morse_code
- Stack Overflow - http://stackoverflow.com/
- GitHub - http://github.com/
- Travis CI - https://travis-ci.com