PROJECT TITLE

A PROJECT

Submitted to

DEPARTMENT OF COMPUTER ENGINEERING

For the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Under the supervision of

Ms Savita Yadav

By

Vinay Kumar – 199/EC/15

Vivek Kumar – 204/EC/15

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NETAJI SUBHAS INSTITUTE OF TECHNOLOGY

2018

ACKNOWLEDGEMENT

First and foremost, we would like to thank to our supervisor of this project, Ms. Savita Yadav for her valuable guidance and advice. She inspired us greatly to work in this project. Her willingness to motivate us contributed tremendously to our project. Besides that, we would like to thank Netaji Subhas Institute of Technology (NSIT) for providing us with a good environment and facilities to complete this project.

INTRODUCTION

Snake game is a simple game where the player has to collect the fruits to score high. Eating the fruits also increases the length of the snake. The player has to collect as many fruits as possible without letting the snake head touch its body. The game is played in a 2D space with a boundary. The concept of snake originated in the 1976 arcade game “Blockade” , and the ease of implementing the snake game has led to hundreds of versions for many platforms. Its popularity gained traction when it was packaged with the Nokia mobile phones in early 2000s. Since then, hundreds of variants are made for different platforms.

Whenever the snake eats the fruit it increases its length progressively, making it increasingly harder to maneuver and increasing the score.

The first draft was made in a console window, but we decided to improve the visuals, so a graphics library was used. This library is pre-packaged with MS-DOS OS and hence comes packaged with TURBO-C compiler. But to run the executable file outside the compiler, a virtual machine like DOSBOX is needed. However it is not necessary as you can just run the application in the compiler itself.

LOGIC OVERVIEW

As already stated, the bulk of the code is written using graphics.h library, which is packaged with Borland compilers, and hence comes pre-packaged with TURBO-C++ compiler. The application has to be run inside the aforementioned compiler or by using a virtual machine like DOSBOX.

TITLE SCREEN:

The player is greeted with a Title Screen, where you have select from the various options to play the game. The keys ‘w’ and ‘s’ move the selection pointer, and the key ‘k’ has to be pressed to select the pointed option. After the selection, the player is taken to the game screen.

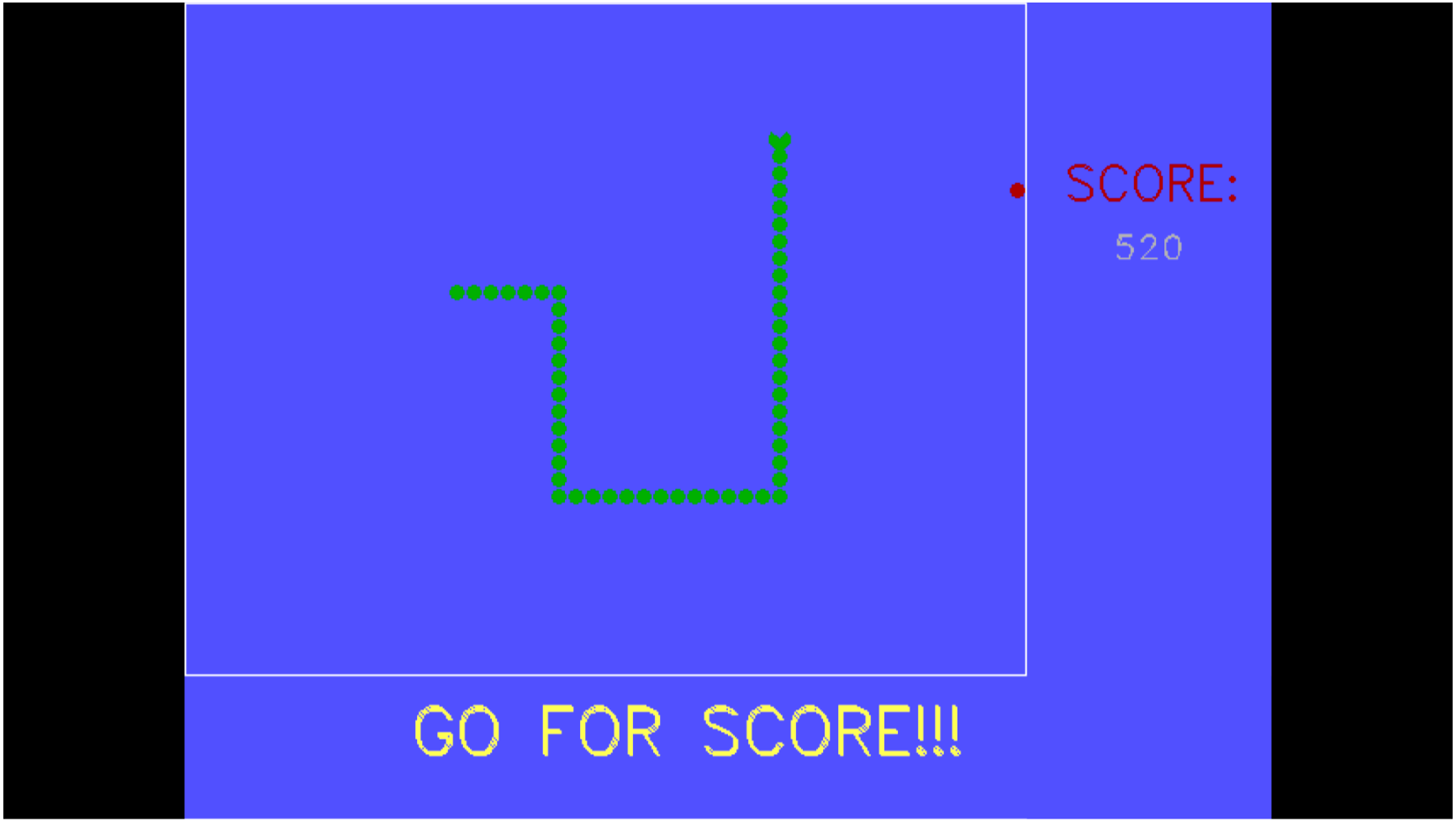
GAME SCREEN:

The movement controls for the game screen are ‘w’, ‘s’, ‘a’ and ‘d’, for upward, downward, leftward and rightward movement of the Snake head respectively. Initially, the Snake head is at rest. The snake head is positioned at the centre of the rectangular box, which is the boundary of the movement of the snake. When a key is pressed, it starts moving in the direction of the corresponding key. The objective of the game is collect as many fruits as possible, demonstrated as the red dots, and hence attaining as high a score as possible, before losing. As the snake eats the fruit, its length keeps increasing correspondingly. If the snake head comes in contact with its body, you lose. As the body of the snake keeps increasing as the snake eats more fruits, the game progressively gets harder and harder.

When you lose, the game overs, if the snake moves across the boundary, the snake teleports to opposite side of the boundary.

For each fruit you eat, your score increases by 10, and is reflected in the score counter. When you lose, you get the option to retry. You have to choose yes or no, by pressing the ‘y’ key or the ‘n’ key, respectively. If you press the ‘n’ key, the application shuts down. If you press ‘y’, the score counter resets to zero, the length of the snake also resets to zero, and the snake head is positioned to the centre of the screen.

Title Screen



Game Screen

FUNCTIONS USED IN THE PROJECT CODING

Getch() -
Reads a single byte character from input. getch**()** is a way to get a user inputted character. It can be used to hold program execution, but the "holding" is simply a side-effect of its primary purpose, which is to wait until the user enters a character.

Rand() -
Returns a pseudo-random integral number in the range between 0 and RAND_MAX.

Cleardevice() -
Cleardevice erases (that is, fills with the current background color) the entire graphics screen and moves the CP (current position) to home (0,0).

Setcolor() -
setcolor sets the current drawing color to color, which can range from 0 to getmaxcolor. The current drawing color is the value to which pixels are set when lines, and so on are drawn. The drawing colors shown below are available for the CGA and EGA, respectively.

```
Syntax- setcolor(int color);
```

Settextjustify() -
Text output after a call to settextjustify is justified around the current position (CP) horizontally and vertically, as specified. The default justification settings are LEFT_TEXT (for horizontal) and TOP_TEXT (for vertical). The enumeration text_just in graphics.h provides names for the horiz and vert settings passed to settextjustify.

```
Syntax- settextjustify(int horiz, int vert);
```

Settextstyle() -
settextstyle sets the text font, the direction in which text is displayed, and the size of the characters. A call to settextstyle affects all text output by outtext and outtextxy.

```
Syntax- settextstyle(int font, int direction, int charsize);
```

Outtextxy() -
outtextxy displays a text string in the viewport at the given position (x, y), using the current justification settings and the current font, direction, and size. To maintain code compatibility when using several fonts, use textwidth and textheight to determine the dimensions of the string. If a string is printed with the default font using outtext or outtextxy, any part of the string that extends outside the current viewport is truncated. outtextxy is for use in graphics mode; it will not work in text mode.

```
Syntax- outtextxy(int x, int y, char *textstring);
```

Itoa() -
Converts an integer value to a null-terminated string using the specified base and stores the result in the array given by str parameter.

```
Syntax- itoa(int value, char * str, int base);
```

Kbhit() -

It is generally used by Borland's family of compilers. It returns a non-zero integer if a key is in the keyboard buffer. It will not wait for a key to be pressed.

Rectangle() -
rectangle draws a rectangle in the current line style, thickness, and drawing color.

```
Syntax- rectangle(int left, int top, int right, int bottom);
```

Setfillstyle() -
setfillstyle sets the current fill pattern and fill color. To set a user-defined fill pattern, do not give a pattern of 12 (USER_FILL) to setfillstyle; instead, call setfillpattern.

```
Syntax- setfillstyle(int pattern, int color);
```

Pieslice() -
pieslice draws and fills a pie slice centered at (x,y) with a radius given by radius. The slice travels from stangle to endangle. The slice is outlined in the current drawing color and then filled using the current fill pattern and fill color.

```
Syntax- pieslice(int x, int y, int stangle, int endangle, int radius);
```

Getmaxx() -
getmaxx returns the maximum (screen-relative) x value for the current graphics driver and mode.

Getmaxy() -
getmaxy returns the maximum (screen-relative) y value for the current graphics driver and mode.

CONCLUSION

The snake game has been running smoothly, and we learned a great deal about implementation of graphics, as well as improving our programming skills. Designing the logic part of the code was a very good exercise in programming.

The first draft was made solely in console, but we felt the need to improve it, and hence the decision to implement graphics was made, and in hindsight, we believe it was the right choice as it helped us explore and increment our programming knowledge.