University of Dhaka

Department of Computer Science and Engineering

Project Report:
Fundamentals of Programming Lab(CSE-1211)

Project Name:
EPIC SNAKE GAME

Team Members
1. *AE-10 Shamik Shafkat*
2. *RK-09 Annoor Sharara Akhand*

1. Introduction:

This project focuses on the development of a version of the arcade game generally known as “Snake Game”. In a classic snake game, the objective is to attain the highest score by eating the maximum number of fruit that appears on the screen. The length of the snake increases each time it consumes a fruit. The game ends when the snake bites its own body and dies. It is a single-player game with the difficulty escalating in each round. Our version of the game introduces some new obstacles to make the game interesting. Each fruit eaten corresponds to points in the game. The main goal of this game is to keep the snake in the arena alive for the longest possible time and achieve the highest score.

2. Objectives:

The main purpose of this project is to provide the user a nostalgic experience of playing a classic video game with some exciting new features. As snake is considered a classic game, our project can be an entertaining game for people of all ages. If a player wants to pass time, then they can play ‘Classic’ mode with difficulty set to Easy. Then as an average user, the player can play ‘Arcade’ mode with different difficulty at different levels. At last, with the ‘Campaign’ mode the user can experience a journey which gives the player a goal, to defeat the enemy. The game also has high scoring opportunities which can create a competitive and experience for users.

Aside from this, the technical objectives are:

- Making a simple graphic game project using SDL/SDL2 library
- Learning the use of C/C++ programming language thoroughly
- Learning the implementation of Simple DirectMedia Layer (SDL) in application building

3. Project Features:

The user interface (UI) is the point of human-application interaction and communication in the game. This includes display screens, pages, and visual elements like buttons and icons that enable a person to interact with the game. Now we will discuss the individual aspects of various features in our game.

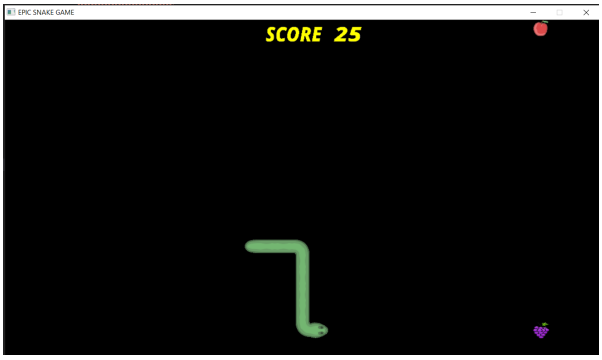
Main Menu Page:

The user will come in contact with this page first every time they interact with the application. This page will show the game's name and provide the user with multiple options to choose from to proceed in the game. The options will present themselves as clickable text on the screen. They are: New game, High Score, Difficulty, Help, Exit Game.

- **New Game:**
  By choosing the 'New Game' button, the user will reach another page that will show the options for Mode. The player has to click on the 'New Game' to start. There are three modes in our game, the names will be displayed on the screen as clickable text. The modes are: CLASSIC, ARCADE, and CAMPAIGN.

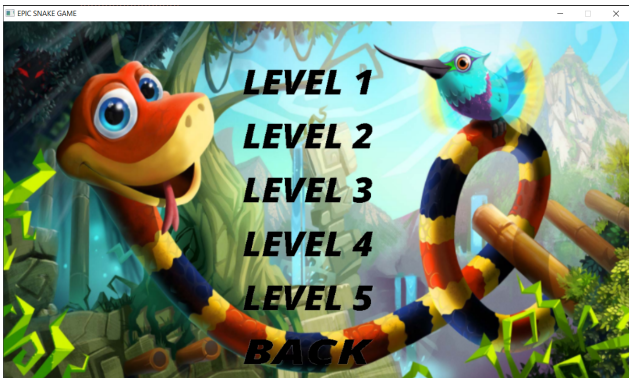
➔ *CLASSIC Mode:*

This mode will let the player play the snake game in the simplest way. The arena in CLASSIC mode has no outward boundary and there are no obstacles. The snake gets bigger every time it eats an apple and the player receives 5 points. After the appearance of five apples, a special fruit appears. The score value of fruit decreases as time passes. The point for each special fruit is:(20 + time remain * 2) points. It stays on screen for a specific time. The game ends when the snake bites its own body.

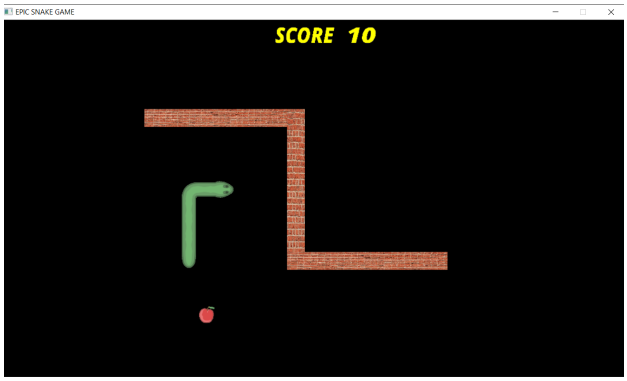


➔ *ARCADE Mode:*

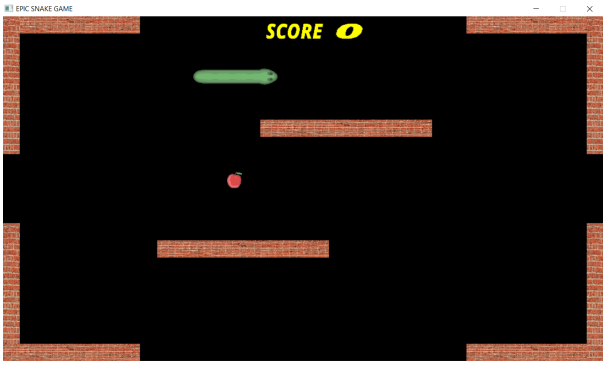
By clicking on ARCADE from the Mode page, the player will visit a page showing options for five levels.



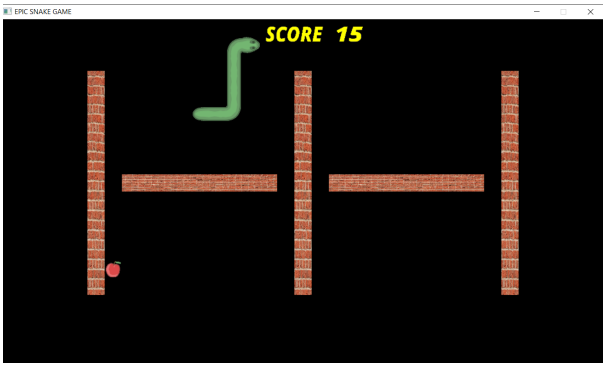
Each level will take the player to an arena with different types of obstacles made of bricks. When the snake comes in contact with bricks or bites itself, the snake dies and the game ends. The point distribution is the same as the CLASSIC mode.



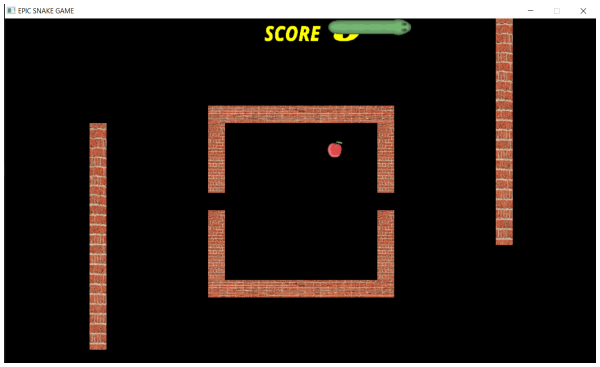
Level 1



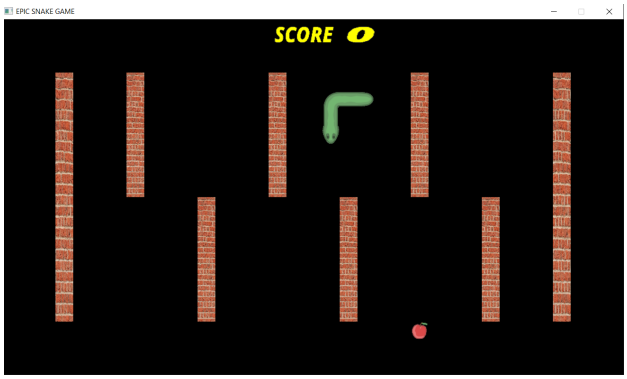
Level 2



Level 3



Level 4

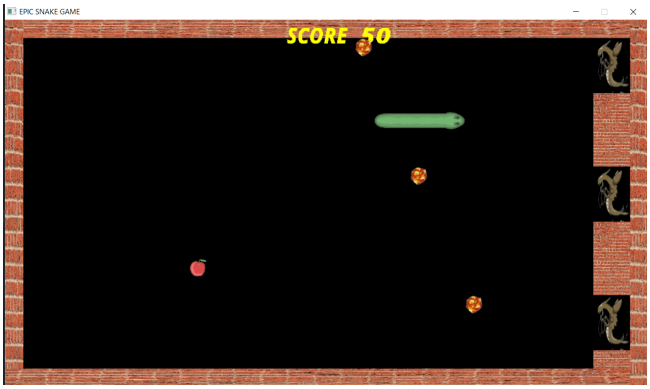


Level 5

➔ *CAMPAIGN MODE:*

This mode integrates all the different gameplay arenas of the ARCADE mode. The player starts the game in the first level of arcade mode and when they eat a specific apple, they enter the arena of the next level by pressing the spacebar. After playing all five levels, the player will enter a special level, the Dragon's Den.

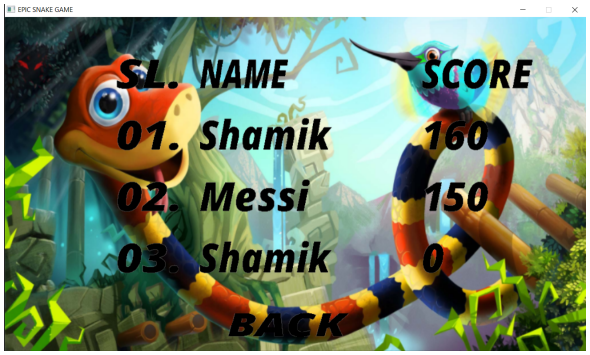
This is the final level of the 'Campaign' mode and this ends when all the dragons are dead. The way to kill the dragon is to hit their head with poison which is available when the snake collects the poison bottle. Poison bottle arrives at the screen after consumption of 30 apples.



Dragon's Den

- **High Score:**

This option will show the top 3 highest scores of previous games in descending order for each Mode and Difficulty level. To view the scores, the player has to select the Mode and then the corresponding Difficulty level from the menu that will appear on the screen.

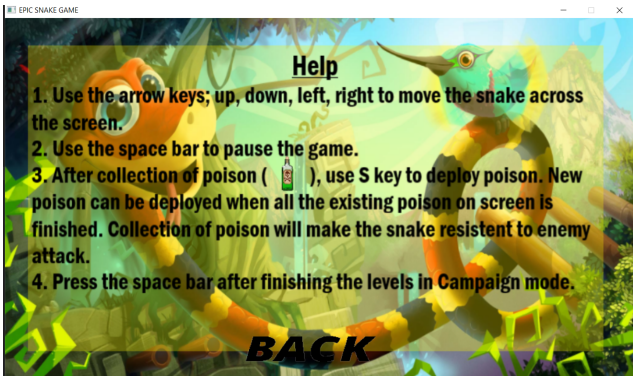


- **Difficulty:**

This option will let the players choose the speed of the snake from three options, Easy, Medium, Hard. By choosing a level and clicking on the text, the player can pick the speed. The back button will take the player to the main menu.

- **Help:**

This button will take the players to a screen where the basics of how to play the game will be explained.



- **Exit Game:**

This option will take the player out of the game and close the game window. The player can quit the game by going to the menu and clicking on the exit button.
The player can also quit the game anytime by clicking the cross on the right corner of the game window.

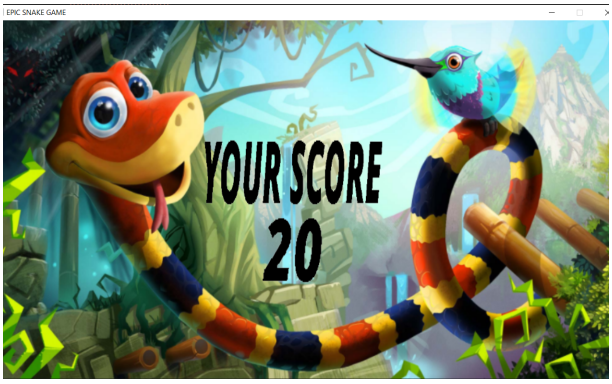
Game Over Feature:

This feature comes into action when the snake bites its own body or comes into contact with any obstacle. The game ends then and the application asks the player for their name to store it if they score a high score. The player needs to type their name in 08 characters. After typing the player name, the player has to press the "Enter" button. Then it takes the player to the main menu page.

Score Feature:

The player gains points by eating fruit in every mode. But in the campaign mode, the player achieves 50 additional points after finishing a campaign successfully. In Dragon's Den level, if the snake is hit by dragon fire, the player loses 10 points. For killing each dragon, the player receives 100 points.

While playing the game, the score achieved by the player will be constantly displayed over the top of the game screen. The score will be updated every time it changes. After the game ends, the score will be displayed on the screen.



Score storage feature:

The top three highest scores of each mode and difficulty are stored. If the score achieved by the player does not count as a high score, it is not stored. Otherwise, the player is asked to input his/her name and the player has to press the 'Enter' button after doing so. The name has to be within 8 characters and must not contain space in between.

Sound Effects:

There are multiple sound effects used throughout the game. Each time the snake eats a fruit, when the snake dies, a sound effect is heard by the player. In the Dragon's Den level, sound effects are heard when the dragon shoots fire, when the snake ejects poison, when the dragon is hit by poison, when the dragon dies, when a level is completed. The menu page also plays ambient music.

4. Project Module

In our project, we have implemented 13 custom header libraries. Here are detailed instructions about those libraries:

I. **libdeclare.h :** In this header file, we have defined all our preprocessors(except the other custom header files),enums, structure types and global variables with extern. We have used 8 types of enum here. They are :

A. **SNAKE:** Used for marking different textures of images.

B. **GAME_STATES:** Used for flagging different game stages and the part of the game is running.

C. **TRUE_TYPE_FONT:** Used for marking ttf images.

D. **MODES:** Used for marking 'Modes' array of SDL_Rect which are used for different modes of the game.

E. **LEVEL:** Used for marking different 'Arcade' and 'Campaign' levels.

F. **MOVEMENT:** Used for flagging the snake movement; up, down, left, and right.

G. **PASS:** Used for marking snake body parts whether it's passing through a wall or not.

H. **SOUND_EFFECTS:** Used for flagging different sound effects in different places.

We have used 5 different types of structure variables in our project. They are mentioned below :

A. **My_Snake:** This structure is used for every rect of the snake. These structure variables determine the movement, rotation, flip, condition for passing the walls of the snake.

B. **Game_Elements:** This structure contains all the necessary game elements.

C. **enemy:** This structure maintains the things of dragons of 'campaign' last level.

D. **VENOM:** This structure contains the venom of the snake which is available in the last level of 'campaign' mode after consumption of a poison bottle.

E. **Wall:** This structure maintains the position and movement of bricks.

So, the inclusion of this header file will allow us to use all the above-mentioned things.

II. libinitialize.h: This header is used for initializing SDL, SDL_image, SDL_ttf, SDL_music and game materials. And also for loading images, TTF and music. There are 6 function prototypes in this header. They are specified below :

A. **create_window:** This function takes no parameter. This function initializes all the SDL things and creates a window, a renderer and a surface. Its return type is bool. When it initializes all the things successfully it returns true, unless false.

B. **load_media:** This function takes no parameter. This function loads all the images, ttfs and sounds used in the project. Its return type is bool. If it cannot load an image or ttf or sound it returns false, else it returns true.

C. **load_texture:** This function takes two parameters as input, one is a string for the path of the image to be textured and the other is an integer. That integer determines whether the black colour of the image has to be transparent or not. If '1' is passed then the black colour gets transparent, unless nothing is done.

D. **loadttf:** This function has no parameter. This function loads all the texts that are needed to be used as a texture. Its return type is bool. Returns true when it successfully loads all the texts as textures, false in the other case.

E. **MAKETTF:** This function takes input three parameters, one is the text/string that has to be made a texture. The second one is an integer that determines the colour of the text. If '0' is passed then the colour is black, unless it's yellow. The third one is a bool variable which determines whether the texture made before has to be made free or not as the TTF_RenderText_Solid function allocates memory dynamically. The

function returns the pointer of the texture made of the text.

F. **INITIALIZER:** This is a void function with no parameter. This function initializes different SDL_Rects with which part of the image has to be rendered.

III. snake_properties.h : This header is used for maintaining the mechanism of the snake body like movement, wall pass, flip, rotations. There are 5 function prototypes in this header. Their functionality is given below :

A. **SETTING_THE_DIRECTION:** This function has no parameter. This function sets the directions of the different snake body parts, more specifically the SDL_Rects of the snake. Its return type is void.

B. **SETTING_THE_WALL_PASSING_STATES:** This is a void function with no parameter. This function sets the wall passing states of the snake body.

C. **SETTING_THE_MOVEMENTS:** This is a void function with no parameter for setting the movements of the snake body.

D. **SETTING_THE_ROTATION_ANGLES:** This is also a void function with no parameter for setting the rotation angles of the image that has to be rendered.

E. **SETTING_THE_FLIP_STATES:** A void function without a parameter for flip states of the snake.

IV. lib_brick_modes.h: This header is used for functions that determine the placement of bricks in different levels, movement of them and also set up the rects for different modes. There are 8 function prototypes in this header. They are specified below :

A. **BRICKS_SETUP_LEVEL_1, BRICKS_SETUP_LEVEL_2, BRICKS_SETUP_LEVEL3, BRICKS_SETUP_LEVEL_4, BRICKS_SETUP_LEVEL_5,BRICKS_SETUP_LEVEL_BOSS :** These 6 functions are all void functions with no parameters. They are used for setting up bricks in different places at different levels.

B. **MOVE_THE_BRICKS:** This is a void function with three parameters. The first two parameters are integers which determine the number of bricks that have to be moved. The third integer determines which level of bricks it is. For example, if the three parameters are 8,10,1(LEVEL 2) then 8-9 bricks will move according to the algorithm of level 2.

C. **MODES_SETUP:** This void function setup SDL_rects for different modes of the game. It has no parameters.

V. food.h: This header file is used for creating and placing food. It has two function prototypes. These are :

A. **FOOD_QUADRANT:** This function returns an SDL_Rect for food, bonus food, poison bottle. It has no parameters.

B. **CREATING_FOOD:** This function returns an SDL_Rect which determines the place of food, bonus food and poison bottle in free space; not inside bricks, inside snake body or outside the window.

VI. librender.h: This function is used for rendering on the screen. Functions related to rendering are here. There are 4 of them: **RENDER_BRICKS, RENDER_SNAKE, RENDER_SCORE, RENDERING_SCREEN**. All of them are void functions with no parameter. They render the parts as their names are.

VII. libUI.h: This header is used for the user interface part. It has two function prototypes. They are :

A. **MOUSE_HANDLING:** This void function with no parameter does all the work of the mouse in the project.

B. **GAME_LOOP:** This void function takes the input of the keyboard from the user and does work accordingly to that. It has no parameters.

VIII. lib_HS_DIF.h: This header file is used for high scores and difficulty in the game. It has 5 function prototypes in it. They are specified below :

A. **HIGHSCORE:** This void function has three parameters. First is a string which is the path of the text file which we want to use. The next two are two SDL_rect for determining the rendering place of text “HIGH SCORE!” and the score of the user respectively. This function is called after gameover and if the user has scored a score better than the top three then it rewrites the file with the proper order of the score according to difficulty.

B. **INPUT_NAME:** This is a void function and it takes a character array as a parameter that will contain the user's name after the end of the function's work. This is done by SDL’s feature of taking text input. The name must have at least a character and it cannot have more than 8 characters and space between it. This function also does the working of rendering the name while taking input.

C. **SHOW_HIGHSCORE:** This void function works by showing the current high score of different modes and difficulty. It has three parameters, first one is the path of the text file which determines the mode whose high score needs to be shown, the next two are two integers for giving the difficulty.0 & 2,3 & 5,6 & 8 in the second and third parameter for the easy, medium and hard mode of difficulty.

D. **READING_DIFFICULTY:** This void function sets up the difficulty for the game.

E. **SETTING_DIFFICULTY:** This void function takes input an integer and sets the difficulty according to that.120 for easy,90 for medium and 70 for

hard as defined in libdeclare.h.

IX. lib_dragon_poison.h: This header is for the dragon and poison feature of the game. It has 4 function prototypes in it. They are :

A. **dragon_setup:** This void function sets up the position of the dragons. It has no parameters.

B. **dragon_mechanics:** This void function does the work of animation of the dragons, the fire positioning of the dragons and rendering images according to that if the dragons are alive. It has no parameters.

C. **venom_mechanics:** This void function with no parameter moves the venom to the direction it has been used and also calls the function ‘collision_of_venom’.

D. **collision_of_venom:** This void function determines the collision of the venom with bricks, dragon, dragon fire and with the snake's own body itself. Thus renders and plays music according to that.

X. libNew.h: The functions in this header does the work of setting the game in the starting position. There are 4 functions in this header.

A. **MEMSET_ALL:** This void function with no parameter sets the ‘flag’ and ‘Python’ array with 0.

B. **GAME_STARTER:** This void function with no parameter sets all the variables to the game starting state and does other works regarding that.

C. **SET_THE_LEVEL:** This function sets the levels in ‘campaign’ mode one after another. It’s a void function with no parameter.

D. **LEVEL_COMPLETED:** This void function with no parameter does the work of showing the text “LEVEL COMPLETED!” and play music after level completion.

XI. liblife.h: The functions in this header contains the code for determining the living state of the snake. It has two function prototypes in it.

A. **COLLISION_DETECTION:** This is a bool function that takes two SDL_Rects as its parameter. Returns false when a collision is detected. True when there is no collision between them.

B. **ALIVE_OR_DEAD:** This void function determines if the snake has collided with its own body, with a brick, with a dragon fire or with a dragon and sets things according to that.

XII. libend.h: This header is for the things after the snake is dead. It has two function prototypes in it.

A. **END_OF_GAME:** This void function with no parameter does the work after the snake is dead or ‘campaign’ mode is completed. And also calls the ‘HIGHSCORE’ function with the appropriate path.

B. **close:** This void function destroys all the textures, windows, renderers, closes TTF font, frees music and quits all the SDL materials like image, TTF, music.

XIII. libstring.h: This is a self-made string library. It has two functions inside it.

A. **number_to_string:** This void function takes an integer and a character array as its two parameters. The function turns the integer to a string and stores it in the array. The number is stored in reverse order of the integer. For

example, 150 is stored as “051” is the array.

B. **string_reverse:** This void function takes a character array and makes it a reverse string.

The code is divided into 13 .cpp files. They are named like their corresponding header files and contain the code related to that.

5. Team Member Responsibilities :

Shamik Shafkat, Roll-10 (Team Leader)

- Initialization of SDL, SDL_image, SDL_TTF, SDL_mixer.
- Animating the snake
- Mechanisms related to the snake: movement of the snake, method of wall passing of snake
- Score related works like converting the score into ttf and projecting it on the screen and constantly updating it
- Work related to the user input via keyboard
- Work related to inputting the name of high scorer after the end of the game
- Arena design in Classic and Campaign mode
- Level synchronization in Campaign mode
- Implementation of 'Dragon's Den' level in Campaign mode, including the rendering of dragons, poison, collision method of dragon, poison, and fire
- Work related to rendering images

Annoor Sharara Akhand, Roll-09

- Converting text-based information to ttf and implementing ttf feature
- Works related to the user input via mouse throughout the project
- Setting the various speed for the snake to increase the difficulty of the game
- Detection of collision of the snake with fruit, snake, obstacle and itself
- Converting the collision data of snake with fruit and special fruit into scores and calculating total score

- Keeping track of state of snake-- dead or alive
- Designing and rendering the arenas of five levels of Arcade mode; setting all the bricks and visualizing/rendering them.
- Work related to the end of the game, including calculating the total score, exhibiting score, determining the high score
- Highscore option; displaying the top three listed high scores of different modes according to their difficulty.
- Rendering the help page containing basic rules and tips for the player
- Adding sound effects to the game

6. Platform, Library & Tools

Platform:

The platform that was used to build the game is Windows 10. The game can be played on both Windows 10 and Linux platforms

Library:

The game was built using C/C++ programming language and SDL2 library

Tools :

The tools Mp3 cutter, Online music converter, Online image resizer, Windows Paint 3D were used.

7. Limitations:

- The control of the snake is done by the up, down, left, right keys and the control cannot be changed to any other keys.
- The name entered by the player must contain less than 8 characters.
- The name entered by the player cannot contain any space.
- The music in the game cannot be stopped from within the game.
- Only the top three scores in each mode and difficulty are stored.
- In the campaign mode, the player can only play through one campaign of a specific set of arenas.
- When a new game starts, no record of previous games aside from high scores exists.

8. Conclusions:

As this was the first time either of us tried to develop a complete application using the SDL library, we faced some problems but gained a lot of experience. We wanted to build a special mode where each level would present an arena with randomly generated obstacles. But we could not achieve that goal. We had no prior knowledge of animation or application development. Besides, we failed to find proper images and technical help regarding various things from the internet.

Despite having these setbacks, this project helped us gain a lot of practical knowledge about the C/C++ language and the development of projects. We learnt to use the SDL library and to write our own custom header files. We learnt basic game development. Moreover, while facing problems, we learnt to find the issue and solve it ourselves. This practice massively helped our problem-solving skills. As we had to work together closely, our communication and coordination skills improved. This project developed our coding skills massively and helped us learn the work that goes behind every part of a game. It taught us the basics of animation. The experience we gained from this project will surely impact all our future endeavours in coding.

9. Future Plan

As the graphics in this game are very basic, we plan to make a better version with more advanced graphics along with adjustable skins.

We want to make the animation in the game more advanced to create a better experience for the player. We want to add more levels of varying difficulty in the Arcade and Campaign mode of the game. We want to create more versions of the Campaign mode with random playing courses. We want to create a better version of our game using the knowledge we are further learning. We wish to increase the experience we achieved while building this game.

Repositories :

- **Github Repository :** https://github.com/ShamikShafkat/EpicSnakeGame.git
- **Youtube Video :** https://youtu.be/Bd8woqt1sVQ

References :

- https://lazyfoo.net/tutorials/SDL/index.php
- https://wiki.libsdl.org/
- https://youtu.be/nK_sT12h22s