University of Dhaka

Department of Computer Science and Engineering

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

Project Name:
DINO RUNNER

Team Members:
Raisa Islam
Nur Jannat Meherin
Mahbuba Khanam Shifa

Table of Contents

Table of Figure

Introduction:

The objective of this project is to review the students' ability to implement the theoretical knowledge they have acquired throughout the semester. In this semester students have been taught C programming language and problem-solving using C. Through this project, students are to use the C language and SDL library to solve real-life problems.

Project Introduction:

This project is a 2D platform game, where the main character is a Dinosaur. It has to pass cactuses, tree logs and kill the snakes avoiding balls and rabbits. The name of the game is "DINO RUNNER" because the dinosaur has to face all the challenges throughout the four levels while running.

Objectives:

This game is nature-based where the player needs to control a 2D character which is a Dinosaur and the player has to ensure to pass all the challenges in each level without any kind of collision.

The opponent includes cactuses, tree logs, snakes, balls, and rabbits. The balls and rabbits can move at different constant speeds and the snakes can rotate. The cactuses and tree logs are immobilized. Here the cactuses and tree logs can't be destroyed. Rather the dinosaur has to pass them by jumping, ensuring no collision. The snake has to be destroyed by emitting fire. Moreover, it has to avoid rabbits and balls by moving forward or backward or by jumping. It has to collect all the diamonds to increase the scores.

Project Features:

- Menu:

The menu page will emerge immediately after the game is run. The player can select different options according to his/her choice.

Figure 1: Manu page

- New Game:

If the player chooses this option the following pages will appear. The first image will appear as the first page and here level 1 will be unlocked by default. After the completion of the first level, if the player chose the New Game option again the second image will emerge. Here, the second level will be unlocked. Each level will be unlocked in this way.

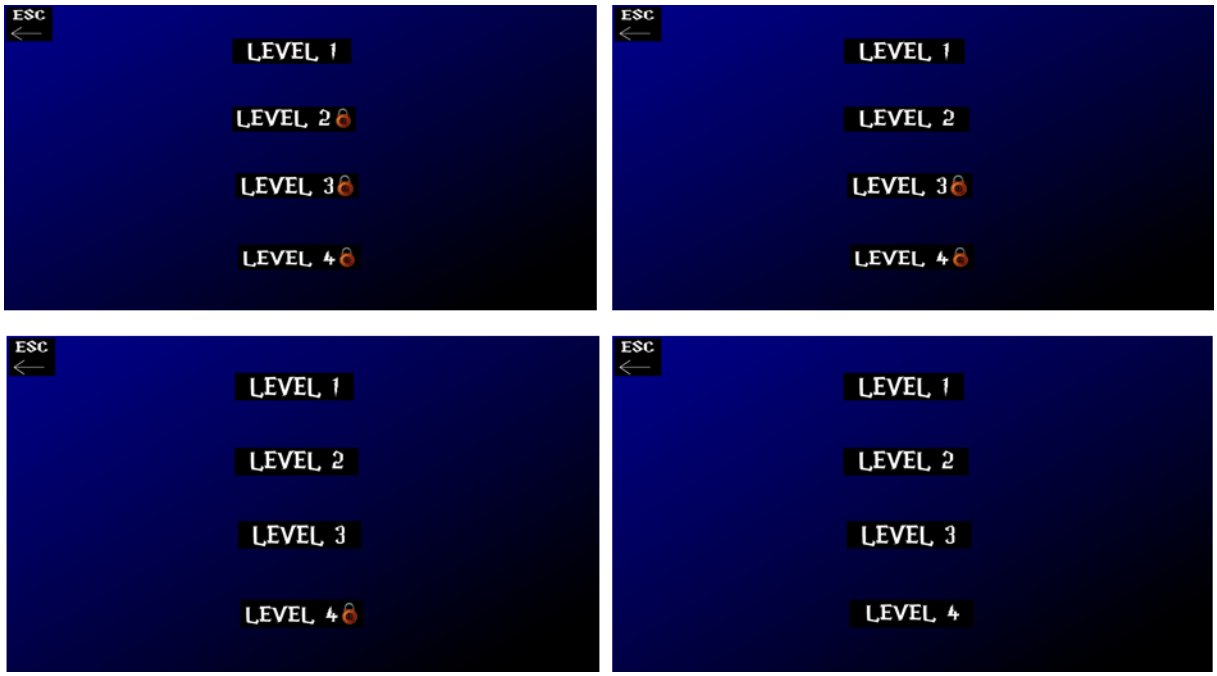

Figure 2: Levels of Dino Runner

- Highest Score:

In this option, the highest score will be shown. It will be updated continuously.

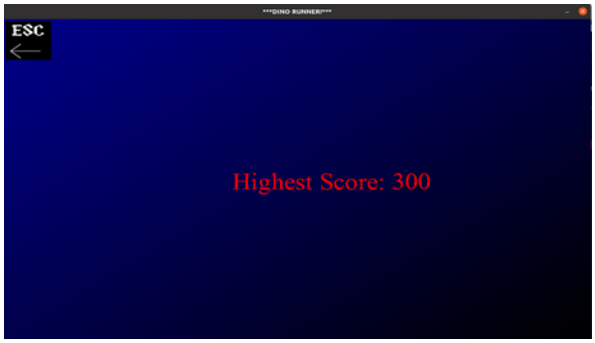

Figure 3: Scoring Page

- Instruction:

This section discusses the way the game is played, the motives of the game and the scoring system.

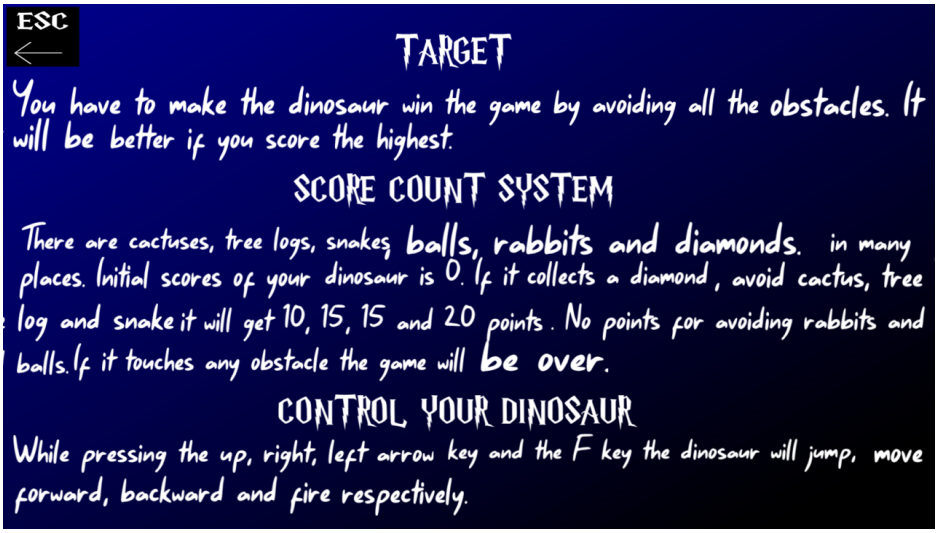

Figure 4: Instruction Page

- Exit:

Without clicking the x button, there's another way to leave the game and that is clicking the Exit option. The player will be asked if he/she is sure to leave the game.

Figure 5: Exit Page

❖ Game:

- Levels:

The game is made with four levels. If one level is played, the next level will be unlocked. As the level increases, the difficulty to face the challenges will also increase.

Figure 6: Different Levels

- Character:

The width of the dinosaur is 484 pixels and the height is 280 pixels. The character moves forward while pressing the right arrow key, moves backward while pressing the left arrow key, jumps when the up arrow key is being pressed. It emits fire when the F key is being pressed.

Figure 7: Dinosaur

- Opponents:

Five opponents can kill the dinosaur. They are cactuses, tree logs, snakes, balls, and rabbits. The first two opponents are still. The snakes can rotate. Balls move in 2D and rabbits move in 1D. They have their own constant velocity.

Figure 8: Opponents

Game Structure:

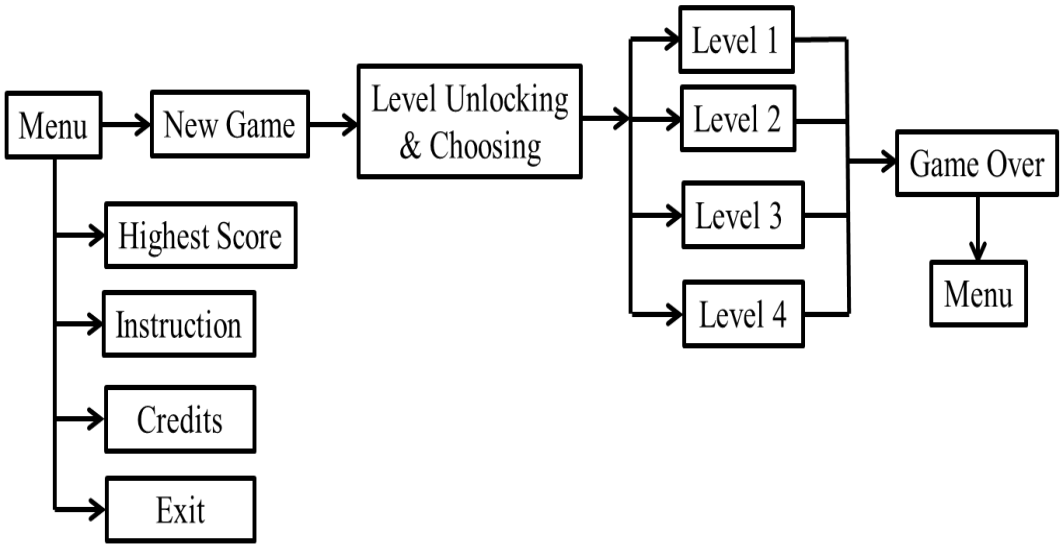

Figure 9: Game Structure

Game Logic Flow Diagram:

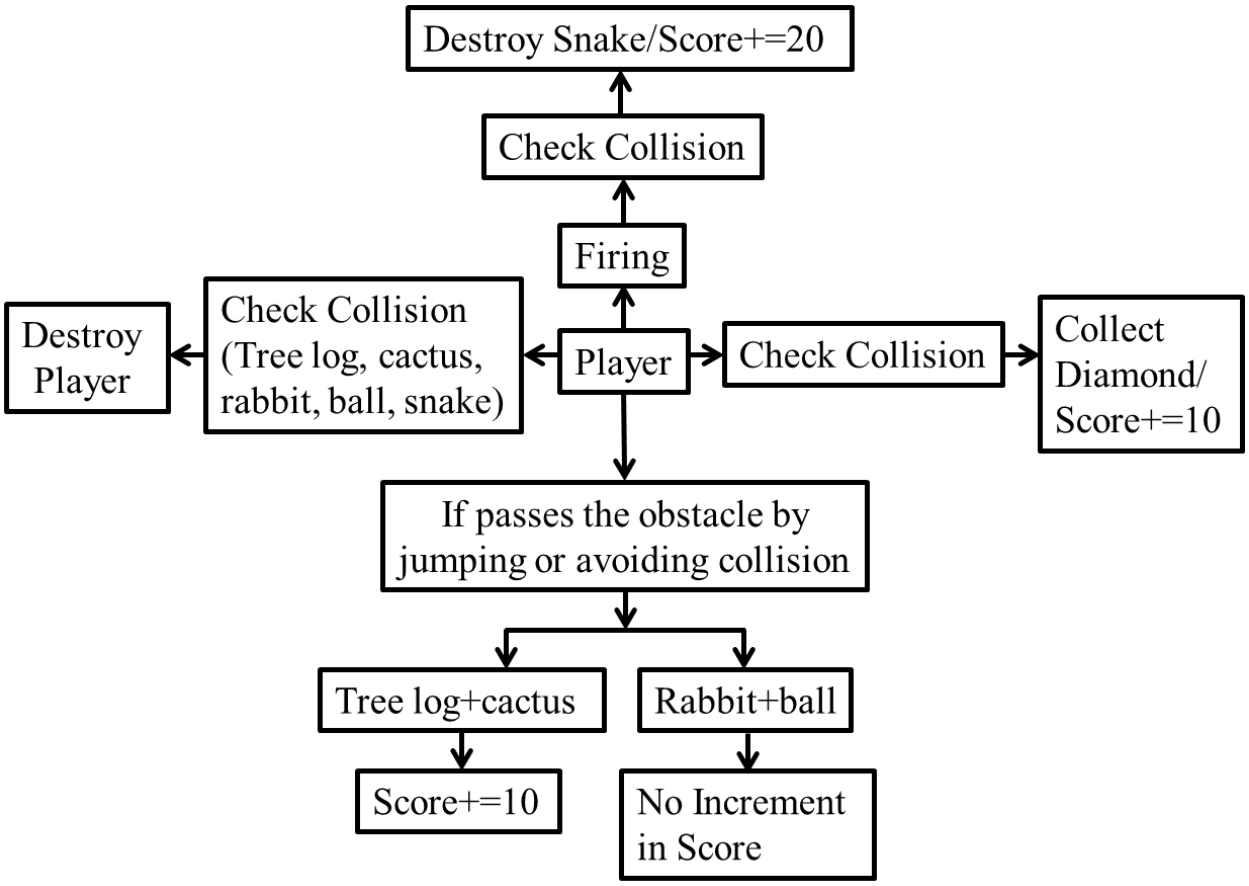

Figure 10: Game Logic Flow Diagram

Project Modules:

In this C programming-based project, media, music, font is included using SDL(Simple Direct Media Layer). As there are many objects in the game, we used the oop concept, especially the class concept. Here, along with C and SDL2 related header files, we used some user-defined header files to make the game flexible and the code readable. The header file used in C and SDL2 is SDL2/SDL2_image.h, SDL2/SDL2_mixer.h, SDL2/SDL2_ttf.h, SDL2/SDL2.h. The header files which were created for the game are Ball.h, Cactus.h, Tree.h, Snake.h, LTexture.h, Rabbit.h, Diamond.h, Dino.h.

- *class Dino*

```
{
        public:
                static const int Dino_WIDTH = 484;
                static const int Dino_HEIGHT = 280;
                static const int Dino_VEL = 10;
                Dino();
                void handleEvent(SDL_Event & e);
                void scrolling(SDL_Rect & screen);
                void move();
                int flipflag = 0;
                int firing = 0;
                void render(int camX, int camY);
                int Yi;
                int jumpState = 0;
                int getPosX();
                int getPosY();

        private:
                int mPosX, mPosY;
                float mVelX, mVelY;
};
```

This is the most important class of this game. This class defines the characteristics of the main character such as movement, jump, firing. This class defines relations with other classes.

- *class Cactus*

```
{
        public:
                Cactus();
                void render(int camX, int camY);
                void setPosition(int x, int y);
                int checkColision(int dot_x, int dot_y, int dot_width, int dot_height);
                int getPosX();
                int getPosY();
        private:
                int cPosX, cPosY;
                const int height = 100;
                const int width = 70;
```

```
};
```

Cactus is one of the immobilized opponents. This class has defined the characteristics of the cactus and used it for positioning and rendering the cactus. It also checks the interaction with the main character.

- *class Tree*

```
{
        public:
                Tree();
                void render(int camX, int camY);
                void setPosition(int x, int y);
                int checkColision(int dot_x, int dot_y, int dot_width, int dot_height);
                int getPosX();
                int getPosY();

        private:
                int tPosX, tPosY;
                const int height = 70;
                const int width = 100;
};
```

Tree log is the other immobilized opponent. This class has defined the characteristics of the tree log and used it for positioning and rendering the tree log. It also checks the interaction with the main character.

- *class Snake*

```
{
        public:
                Snake();
                void render(int n, int camX, int camY);
                void setPosition(int n, int x, int y);
                int checkColision(int n, int dot_x, int dot_y, int dot_width, int dot_height,
int firing, int flipflag);
                int killed[50] = { 0 };
                int getPosX();
                int getPosY();
```

```
        private:
                int sPosX, sPosY;
                const int height = 194;
                const int width = 75;
};
```

Snake is one of the moving opponents. But it's also the only rotating opponent. This class is used to animate the snake, positioning and rendering on the screen. It also defines the interaction with the dinosaur. The dinosaur has to kill it by firing and here it has to collide with the snake at the same time. But if it collides with the snake without the firing moment the game will be over.

- ```
  class Ball
  {
          public:

                  Ball();
                  void render(int n, int camX, int camY );
                  void setPosition(int n, int x, int y );
                  int checkColision(int i, int dot_x, int dot_y, int dot_width, int dot_height);
                  int killed[20]={0};
                  void move();
                  int r_velx = -5;
                  int r_vely = -5;
                  int getPosX();
                  int getPosY();

          private:
                  int rPosX, rPosY;
                  const int height = 60;
                  const int width = 60;
  };
  ```

The ball is one of the moving opponents. It moves in 2D. This class is used to animate the ball, positioning and rendering on the screen. It also defines the interaction with the dinosaur. The dinosaur has to avoid it by jumping, moving in a forward or backward direction. But if it collides with the ball, the game will be over.
  ```

- *class Rabbit*

```
{
        public:

                Rabbit();
                void render(int camX, int camY );
                void setPosition(int x, int y );
                int checkColision(int dot_x, int dot_y, int dot_width, int dot_height);
                void move();
                int r_velx = -5;
                int getPosX();
                int getPosY();

        private:
                int rPosX, rPosY;
                const int height = 70;
                const int width = 42;
};
```

Rabbit is the last moving opponent. It moves in 1D. This class is used to animate the rabbit, positioning, and rendering on the screen. It also defines the interaction with the dinosaur. The dinosaur has to avoid it by jumping, moving in a forward or backward direction. But if it collides with the rabbit the game will be over.

- *class Diamond*

```
{
        public:
                Diamond();
                void render(int n, int camX, int camY);
                void setPosition(int n, int x, int y);
                int checkColision(int i, int dot_x, int dot_y, int dot_width, int dot_height);
                int collected[50] = { 0 };
                int getPosX();
                int getPosY();

        private:
```

```
        int dPosX, dPosY;
        const int height = 50;
        const int width = 50;
};
```

Diamond is a bonus object in this game. This class is used for positioning and rendering the diamond. If the dinosaur touches a diamond the store will increase by 10 points.

- *class LTexture*

```
{
    public:
        LTexture();
        ~LTexture();
        bool loadFromFile(std::string path);
        #if defined(SDL_TTF_MAJOR_VERSION)
        bool loadFromRenderedText(std::string textureText, SDL_Color textColor);
        #endif

        void free();
        void setColor(Uint8 red, Uint8 green, Uint8 blue);
        void setBlendMode(SDL_BlendMode blending);
        void setAlpha(Uint8 alpha);
        void render(int x, int y, SDL_Rect *clip = NULL, double angle = 0.0, SDL_Point *center = NULL, SDL_RendererFlip flip = SDL_FLIP_NONE);
        int getWidth();
        int getHeight();

    private:
        SDL_Texture * mTexture;
        int mWidth;
        int mHeight;
};
```

This class is derived from the Lazy Foo website to load and use textures and fonts to make them appear in the game. This class is used to combine all the classes, and music.

Game Theory:

- Movement and Jump:

The movement of the character was defined in the class. It was implemented by manipulating the x and y positions of the dinosaur. As long as a corresponding key was pressed, and movement along a direction was valid, the position of the character would change by a certain defined value. The movement would stop immediately as the key was released. To make the jump, a realistic one, we had to use a process similar to real-life gravity. If a corresponding keypress was found, the character would be given an initial velocity that would reduce by a predetermined amount after every tick.

- Collision Detection:

Collision detection is the most used and important part of the game. It is used for the player to collect the diamonds, passing the challenges like cactus and tree log, avoiding rabbits and balls. This was done using rectangular collision detection concepts. We have learned the concepts from youtube and lazy foo. Because of having a large size player and different kinds of challenges with different properties, we had to change the collision box. That's why we had to use the detection function in the opponents' class. The rectangular collision detects if the areas of the two rectangles given are overlapping or not. If they are overlapping that means there is a collision, this was done by the function checkColision function. This function was used every time there was a need to check the collision between two rectangles.

- Level System:

This game is a level_based game having four levels. At first, except for the first level, others will be locked. After winning a level, the next will be unlocked. To maintain the sequence, we had to use many constants. Giving all the levels an integer value, we have reduced the level change complexity. Moreover, we had used a flag and a file to maintain the unlocking process. All the levels were declared as a function, and in the main function, we run our levels by calling these functions.

- Highest Score:

We have implemented the score updating and showing process using the knowledge of files that we learned in our class this semester. When a player finishes the game or dies, it simply compares the current score of the player with the already saved high scores in the file and then replaces the lowest high score with the current score. If the current score is greater, then delete the previously saved high scores and update the file.

Team Member Responsibilities:

The Menu part is designed by Raisa Islam, The graphics part is designed by Nur jannat Meherin. In the main part of the game, all the classes are designed by Raisa Islam and Nur Jannat Meherin. The collision, level mapping, and setting, audio addition part was designed by Raisa Islam. Level termination, level unlocking, scoring, showing highest score, creating header files- these are done by Raisa Islam and Nur Jannat Meherin together. The rest of the part is done by Mahbuba Khanam Shifa.

Platform, Libraries, and Tools:

The platform of the project is 2D. It is a C language-based project and it uses the SDL2 library for developing the game. The additional header files are SDL2 is SDL2/SDL2_image.h, SDL2/SDL2_mixer.h, SDL2/SDL2_ttf.h, SDL2/SDL2.h.

Limitations:

Though the project was interesting and we tried so hard to make the game a perfect one, there remain some limitations. We could show the highest score but we couldn't show the individual score with the player name. We couldn't add the resume feature in the game.

Conclusions:

This project is our first project at the University. Creating the game from the beginner level is very hard but possible. We've learned how to use the SDL2 library with our code to make the game. The project was interesting, fun, and exciting. There was always something new to learn and implement. Finding the bugs was exhausting. But when we could fix the bugs we were relieved. We learned how to work in a team and how to divide our work among the members. Dividing the workload and understanding each other's code was tough at times but all's well that ends well. In short, the project was very hard though it seemed easy. It was a lifetime experience and we can use this experience in the future.

Future Plan:

Though we couldn't make the game a perfect one, we could use more features if we had more time. If there are more attractive features, good graphics, more quality music the game will have the ability to entertain people properly. As a result, the game will have the possibility to take a good place in the market.

Repositories:

Github Repository: https://github.com/Raisa-islam/dino_runner.git

Youtube Videos: https://youtu.be/5VWidyO7AWY

References:

- *Lazy Foo' Productions - Beginning Game Programming v2.0*
- *https://www.libsdl.org/*
- https://dev.to/noah11012/using-sdl2-opening-a-window-79c?fbclid=IwAR3hmza3SeAMvssibjwVrrb4JYP0VqqpyKfkijNCrtyiQQoe41uCPnitUmo
- https://www.parallelrealities.co.uk/tutorials/?fbclid=IwAR3hmza3SeAMvssibjwVrrb4JYP0VqqpyKfkijNCrtyiQQoe41uCPnitUmo