THE SNAKE GAME

CO – 203 INNOVATIVE PROJECT

SIDDHANT KHANNA – 2K20/CO/441
VARIN THAKUR – 2K20/CO/475

TABLE OF CONTENTS

INTRODUCTION

- “SNAKE” is a Game where a Snake is deployed (controlled by a player) to eat fruits randomly placed over the area.
- The Logistics and Rules of the Game are as follows-

  => As the Snake eats the fruit, the length of the snake’s tail increases.

  => The Game ends when –

  The Snake has touched any of the walls OR

  The Snake has touched any part of its tail.

  => The Score of the Player increases by a factor of 10 as the snake eats the fruit.

STEP 1 – MAKING CLASSES AND MEMBER FUNCTIONS

- 4 Classes –
  - GOD – Just like God decides our fate, this decides the fate of our beloved snake!
    - Includes Game Over and Score Functions.
  - MAP – Handles the playing area.
    - Responsible for the width, height, walls and empty space.
  - SNAKE – Controls the snake.
    - The head and the tail are primarily its members.
  - FRUIT – Controls the fruit.
    - The X and Y coordinates of the fruit are its primary members.

GOD CLASS -

- Data members (Private)
  - bool Game_Over – the game runs while the value of this variable is false and terminates otherwise.
  - int score – records the score of the player.

- Member Functions
  - God() - Constructor to initialize Game_Over to false and score to 0.
  - cursorReset() - Resets the cursor position and makes it point to {0, 0}.
  - EndGame() - Ends the game by updating the value of Game_Over to true.
  - ScoreIncrease() - Increments the score when the snake eats the fruit by a factor of 10.
  - display_Score() - Displays the score of the player.
  - get_Game_Over() - returns the value of the Game_Over variable.

```
class God
{
    bool Game_Over;
    int score;

public:
    God()
    {
        Game_Over = false;
        score = 0;
    }

    void cursorReset();
    void EndGame();
    void ScoreIncrease();
    void display_SCORE();
    bool get_Game_Over();
};

void God ::cursorReset()
{
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), {0, 0});
}

void God ::EndGame()
{
    Game_Over = true;
}
```

```
void God ::ScoreIncrease()
{
    score += 10;
}

void God ::display_SCORE()
{
    Color(7);
    cout << "Score : " << score << endl;
}

bool God ::get_Game_Over()
{
    return Game_Over;
}
```

MAP CLASS -

- All members and members functions are kept public for Inheritance purposes.

- Data Members (Public)
  - int height – to store the height of the map.
  - int width – to store the width of the map.

- Member Functions
  - Map() – Constructor to initialise height and width.
  - void Draw_WALL() – Prints a wall character in “Yellow” colour.
  - void Draw_EMPTY_SPACE() – Prints a empty space character in “Green” colour.

```cpp
class Map
{
public:
    int width;
    int height;

    Map(int x = 50, int y = 20)
    {
        width = x;
        height = y;
    }
    void Draw_WALL();
    void Draw_EMPTY_SPACE();
};

void Map ::Draw_WALL()
{
    Color(6);
    char b = 219;
    cout << b;
}

void Map ::Draw_EMPTY_SPACE()
{
    Color(10);
    char a = 219;
    cout << a;
}
```

SNAKE CLASS-

- Inherits from the Map class publicly to access the height and width.

- Data Members (Private)
  - int head_x_coordinate - stores the x-coordinate of the snake's head.
  - int head_y_coordinate – stores the y-coordinate of the snake's head.
  - vector<int> tail_x_coordinate{100} – stores x-coordinates of the elements of snake's tail.
  - vector<int> tail_y_coordinate{100} - stores y-coordinates of the elements of snake's tail.
  - int length_tail – stores the length of snake's tail.
  - enum Direction – a user defined data type, a variable of type Direction stores the direction of snake's movements. (Public)
  - Direction dir – stores the direction of snake's movement.

```
class Snake : public Map
{

    int head_x_coordinate;
    int head_y_coordinate;

    vector <int> tail_x_coordinate = vector <int> (100);
    vector <int> tail_y_coordinate = vector <int> (100);
    int length_tail;

public:
    enum Direction
    {
        STOP = 0,
        LEFT,
        RIGHT,
        UP,
        DOWN
    } dir;

    Snake()
    {
        head_x_coordinate = width / 2;
        head_y_coordinate = height / 2;
        length_tail = 0;
        dir = STOP;
    }
```

```
    void Draw_HEAD();
    void Draw_TAIL();
    void logic_MOVE();
    void logic_TAIL();
    void increment_TAIL();
    int get_head_x_coordinate();
    int get_head_y_coordinate();
    int get_tail_length();
    vector <int>::iterator get_tailX();
    vector <int>::iterator get_tailY();
    void Input(God *&g);
};

void Snake::Draw_HEAD()
{
    Color(1);
    char a = 219;
    cout << a;
}

void Snake::Draw_TAIL()
{
    Color(5);
    char a = 219;
    cout << a;
}
```

- Member Functions –
  - Snake () – a constructor to initialize the data members of the class.
  - Draw_HEAD () – used to draw the head of the snake. Prints a head character in “BLUE” colour.
  - Draw_TAIL () – used to draw the tail of the snake. Prints a tail character in “PURPLE” colour.
  - logic_MOVE () - Controls the movement of the snake using Direction variable dir.
    dir is used as the switch case variable and the movement gets decided by its value.
    - dir = LEFT – head_x_coordinate gets decremented thus the snake moves left.
    - dir = RIGHT – head_x_coordinate gets incremented thus snake moves right.
    - dir = UP – head_y_coordinate gets decremented thus snake moves up.
    - dir = DOWN – head_y_coordinate gets incremented thus snake moves down.
  - logic_TAIL() - It is used to set the x and y coordinates of the elements of the tail to that of previous one using prev_x and prev_y which store the x and y coordinates of the previous elements of the snakes tail respectively.

```cpp
void Snake::logic_MOVE()
{
    switch (dir)
    {
    case LEFT:
        head_x_coordinate--;
        break;
    case RIGHT:
        head_x_coordinate++;
        break;
    case UP:
        head_y_coordinate--;
        break;
    case DOWN:
        head_y_coordinate++;
        break;
    default:
        break;
    }
}

void Snake::logic_TAIL()
{
    int prev_x = tail_x_coordinate[0];
    int prev_y = tail_y_coordinate[0];
    tail_x_coordinate[0] = head_x_coordinate;
    tail_y_coordinate[0] = head_y_coordinate;
    int prev_2x, prev_2y;

    if(length_tail > 100)
    {
        tail_x_coordinate.resize(length_tail);
        tail_y_coordinate.resize(length_tail);
    }

    for (int i = 1; i < length_tail; i++)
    {
        prev_2x = tail_x_coordinate[i];
        prev_2y = tail_y_coordinate[i];
        tail_x_coordinate[i] = prev_x;
        tail_y_coordinate[i] = prev_y;
        prev_x = prev_2x;
        prev_y = prev_2y;
    }
}
```

```cpp
void Snake::increment_TAIL()
{
    length_tail++;
}

int Snake::get_head_x_coordinate()
{
    return head_x_coordinate;
}

int Snake::get_head_y_coordinate()
{
    return head_y_coordinate;
}

int Snake::get_tail_length()
{
    return length_tail;
}

vector<int> :: iterator Snake::get_tailX()
{
    return tail_x_coordinate.begin();
}

vector<int> :: iterator Snake::get_tailY()
{
    return tail_y_coordinate.begin();
}
```

- Increment_TAIL() - Increments the length of snake’s tail when it encounters a fruit by incrementing the length_tail variable.
- get_head_x_coordinate() - returns x coordinate of snake’s head.
- get_head_y_coordinate() - returns y coordinate of snake’s head.
- tail_length() - returns the length of snake’s tail.
- get_tailX() - returns the base address of the tail_x_coordinate array.
- get_tailY() - returns the base address of the tail_y_coordinate array.
- Input(God* &g) - The function updates the value of ‘dir’ variable depending on the key pressed and if ‘x’ is pressed, it calls Endgame() function of God class to end the game.
    - kbhit() – return a Boolean value, true if a key is pressed and false otherwise.
    - _getch() – returns the ASCII value of the key pressed.

```
void Snake ::Input(God *&g)
{
    if (_kbhit())
    {
        switch (_getch())
        {
        case 'a':
            dir = LEFT;
            break;
        case 'd':
            dir = RIGHT;
            break;
        case 'w':
            dir = UP;
            break;
        case 's':
            dir = DOWN;
            break;
        case 'x':
            g->EndGame();
            break;
        }
    }
}
```

FRUIT CLASS -

- Inherits from the Map class publicly to access the height and width.

- Data Members (Private)
  - int fruit_x_coordinate – to store the X coordinate of the fruit.
  - int fruit_y_coordinate – to store the Y coordinate of the fruit.

- Member Functions
  - Fruit() – constructor to initialise coordinates of the fruit.
  - void logic_FRUIT() – assigns random coordinates to the fruit.
  - int get_fruit_x() – returns the X coordinate of the fruit.
  - int get_fruit_y() – returns the Y coordinate of the fruit.

```cpp
class Fruit : public Map
{

    int fruit_x_coordinate;
    int fruit_y_coordinate;

public:
    Fruit()
    {
        srand(time(0));
        fruit_x_coordinate = rand() % width;
        fruit_y_coordinate = rand() % height;
    }

    void Draw_FRUIT();
    void logic_FRUIT();
    int get_fruit_x();
    int get_fruit_y();
};

void Fruit ::Draw_FRUIT()
{
    Color(4);
    char a = 254;
    cout << a;
}

void Fruit ::logic_FRUIT()
{
    fruit_x_coordinate = rand() % width;
    fruit_y_coordinate = rand() % height;
}

int Fruit ::get_fruit_x()
{
    return fruit_x_coordinate;
}

int Fruit ::get_fruit_y()
{
    return fruit_y_coordinate;
}
```

STEP 2 – DEFINING CONTROL FUNCTIONS

SETUP(God** g, Snake** s, Map** m, Fruit** f)-

- Calls constructor for all objects.

- Game should be running with the initial conditions-
  - => SCORE should be 0.
  - => Position of the head should be known.
  - => Fruit should be at a random location.
  - => The Snake should not move.
  - => The Snake should be the smallest in its length .

```
void Setup(God **g, Snake **s, Map **m, Fruit **f)
{
    *g = new God();
    *s = new Snake();
    *m = new Map();
    *f = new Fruit();
}
```

DRAW(God* g, Snake* s, Map* m, Fruit* f)-

- Uses all the draw functions of individual objects upon arriving at certain conditions.
- Starts with resetting the cursor to (0,0). (Continuous Gameplay).
- Two nested for loops are used along with if-else conditional statements to draw the things according to the conditions.
- Also prints the Score at the end so that the user can feel good about how he plays.

```
void Draw(God *g, Snake *s, Map *m, Fruit *f)
{
    g->cursorReset();
    for (int i = 0; i < m->width + 2; i++)
        m->Draw_WALL();
    cout << endl;

    for (int i = 0; i < m->height; i++)
    {

        for (int j = 0; j < m->width+2; j++)
        {
            bool wall_print = false ;
            if (j == 0 || j == m->width + 1)
            {
                m->Draw_WALL();
                wall_print = true ;
            }
            if (i == s->get_head_y_coordinate() && j == s->get_head_x_coordinate())
                s->Draw_HEAD();
            else if (i == f->get_fruit_y() && j == f->get_fruit_x())
                f->Draw_FRUIT();
            else
            {
                bool print = false;
                for (int k = 0; k < s->get_tail_length(); k++)
                {

                    if (s->get_tailX()[k] == j && s->get_tailY()[k] == i)
                    {
                        s->Draw_TAIL();
                        print = true;
                    }
                }
                                    }
                                if (print == 0 && wall_print == 0)
                                    m->Draw_EMPTY_SPACE();
                            }

                        }
                        cout << endl;
                    }

    for (int i = 0; i < m->width + 2; i++)
        m->Draw_WALL();
    cout << endl;
    g->display_SCORE();
}
```

LOGIC(God* g, Snake* s, Map* m, Fruit* f)-

- Snake’s movement – Implemented using the logic_MOVE and logic_Tail function of snake class.
- Ending the game when Snake hits the wall or its own tail.
- Updating the score, Increasing snakes tail length and relocating the fruit when the snake eats the fruit.

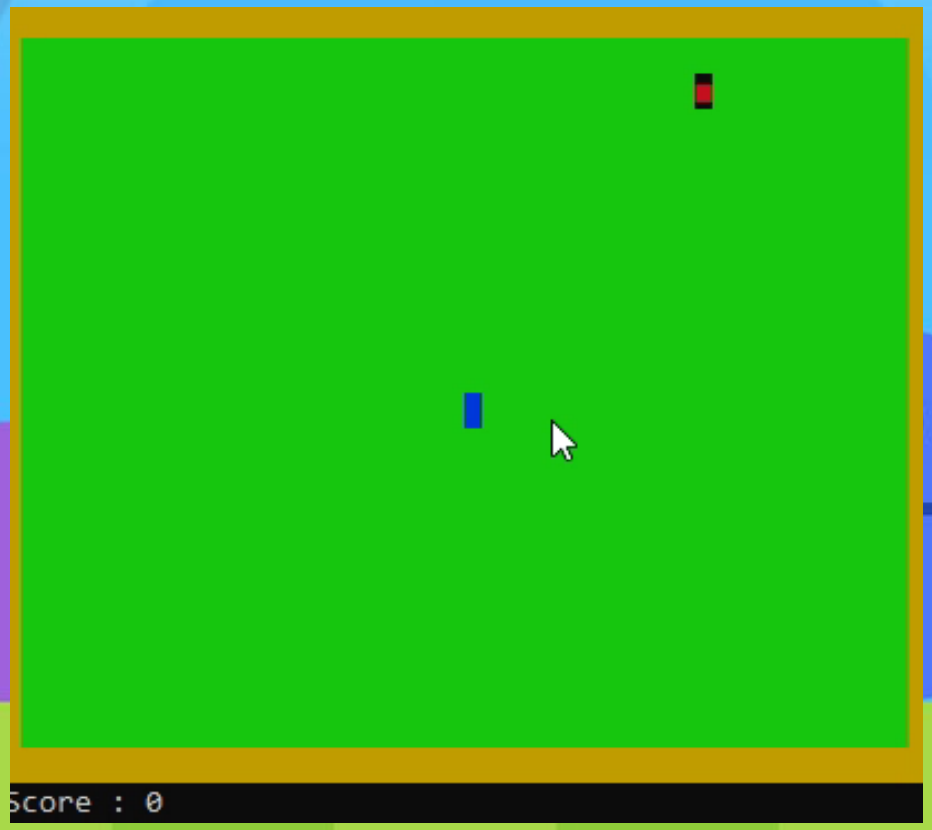
```
void Logic(God *g, Snake *s, Map *m, Fruit *f)
{
    if (s->get_head_x_coordinate() == f->get_fruit_x() && s->get_head_y_coordinate() == f->get_fruit_y())
    {
        g->ScoreIncrease();
        f->logic_FRUIT();
        s->increment_TAIL();
    }
    s->logic_TAIL();
    s->logic_MOVE();

    if (s->get_head_x_coordinate() > m->width  || s->get_head_x_coordinate() < 0 || s->get_head_y_coordinate() > m->height - 1 || s->get_head_y_coordinate() < 0)
        g->EndGame();

    for (int i = 0; i < s->get_tail_length(); i++)
        if (s->get_tailX()[i] == s->get_head_x_coordinate() && s->get_tailY()[i] == s->get_head_y_coordinate())
            g->EndGame();

}
```

MAIN()-

```
int main()
{
    God *g;
    Snake *s;
    Map *m;
    Fruit *f;
    loadingBar();
    Setup(&g, &s, &m, &f);
    while (!g->get_Game_Over())
    {
        Draw(g, s, m, f);
        Logic(g, s, m, f);
        s->Input(g);
        Sleep(50);
    }
    return 0;
}
```

- First pointers of classes are declared.
- Setup() is called to initialise.
- A while loop is used to repeatedly call functions and the output of this is the game.

STEP 3 - DEMO



With this demo of the game, we culminate our presentation.

THANK YOU. You've been a lovely audience!