

“SALIM HABIB UNIVERSITY”

**Project Report**

**DSA LAB**

**Submitted to:**

**Sir Shahid Munir**

**Miss Dua Baig**

**Sir Owais**

PickUp Snoopy Game

**June 16, 2021**

Table of Contents

[Overview 1](#_Toc75016390)

[Goals 1](#_Toc75016391)

[Specifications 2](#_Toc75016392)

[Group Members 2](#_Toc75016393)

# Overview:

We will be creating a 2D snake game by using SFML library in C++. SFML stands for simple and fast multimedia library which provides a simple interface to the various components of your PC, to ease the development of games and multimedia applications. We will be using doubly linked list as the fundamental block of this 2D snake game. The game will work on simple insertion and deletion of nodes displayed as a snake and creating animations like snake is moving also it will detect collisions when touched with the snoopy and simultaneously it will increase score and display it on the corner of the screen, if the snake will touch it self so the game will be exit.

# Goals

1. **Insertion In Doubly Linked List**.:
2. **Deletion in Doubly Linked List**
3. **SFML Library**

# Specifications

This game will be run flawlessly with the help of SFML, This game totally work on the insertion and the deletion concept of a doubly linked list which will be helping us to create a smooth animation by continuously deleting and inserting new nodes as the size of snake increases.

# Group Members

1. Altamash Bin Masroor
2. Ali Salman
3. Bilal Amjad

# Source Code:

#include<SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#define side 15

#define size 208.5

using namespace sf;

class node

{

public:

int posx, posy;

int direction;

Sprite\* s;

node\* llink;

node\* rlink;

};

using NODE = node\*;

enum direction

{

up, down, left1, right1

};

class Game

{

private:

NODE head, tail, cur;

int speed = 150;

int count = 0;

int score = -1;

Vector2f snoopyPos;

RenderWindow window;

Sprite TrainSprite;

Texture spriteSheet;

Sprite snoopy;

Event event;

Text Score;

Font font;

Texture Background;

Sprite bg;

void Update()

{

if (head->posx > 810)

head->posx = 0;

if (head->posx < 0)

head->posx = 810;

if (head->posy > 600)

head->posy = 0;

if (head->posy < 0)

head->posy = 600;

HandleSnoopyTouched();

head = insertHead(head, &TrainSprite);

tail = deleteTail(tail);

checkIntersection(head, &window);

count = 0;

}

void Render()

{

window.clear();

window.draw(bg);

window.draw(Score);

setHeadSprite(head, &TrainSprite);

cur = head;

while (cur != NULL)

{

if (cur != head)

setTrailingSprite(cur, &TrainSprite);

(\*cur->s).setPosition(cur->posx, cur->posy);

window.draw(\*cur->s);

cur = cur->rlink;

}

window.draw(snoopy);

window.display();

}

void Start()

{

LoadSprites();

head = new node;

head->posx = 0;

head->posy = 0;

head->direction = right1;

head->s = &TrainSprite;

head->llink = head->rlink = NULL;

tail = head;

snoopy.setPosition(head->posx, head->posy);

Score.setPosition(700, 10);

Score.setString(String(std::to\_string(score + 1)));

window.create(VideoMode(800, 600), "Pick-Up Snoopy", Style::Titlebar | Style::Close);

}

void LoadSprites()

{

this->spriteSheet.loadFromFile("resources/trainsprite.png");

TrainSprite.setTexture(spriteSheet);

TrainSprite.setScale(Vector2f(0.25, 0.25));

TrainSprite.setOrigin(104.5, 104.5);

snoopy.setTexture(spriteSheet);

snoopy.setTextureRect(IntRect(size \* 3, size \* 1, size, size));

snoopy.setScale(0.2, 0.2);

snoopy.setOrigin(104.5, 104.5);

font.loadFromFile("resources/new.otf");

Score.setFont(font);

Score.setFillColor(Color::White);

Score.setStyle(Text::Bold);

Score.setCharacterSize(50);

this->Background.loadFromFile("resources/background1.jpg");

bg.setTexture(Background);

bg.setScale(2, 2);

}

NODE insertHead(NODE head, Sprite\* sprite)

{

NODE temp;

temp = new node;

switch (head->direction)

{

case direction(up):

temp->posy = head->posy - side \* 2;

temp->posx = head->posx; break;

case direction(down):

temp->posy = head->posy + side \* 2;

temp->posx = head->posx; break;

case direction(left1):

temp->posx = head->posx - side \* 2;

temp->posy = head->posy; break;

case direction(right1):

temp->posx = head->posx + side \* 2;

temp->posy = head->posy; break;

}

temp->direction = head->direction;

temp->s = sprite;

temp->llink = NULL;

temp->rlink = head;

head->llink = temp;

return temp;

}

NODE deleteTail(NODE tail)

{

NODE temp;

temp = tail->llink;

temp->rlink = NULL;

free(tail);

tail = NULL;

return temp;

}

void processEvents()

{

if (window.pollEvent(event))

{

switch (event.type)

{

case Event::Closed: window.close();

break;

case Event::KeyPressed:

if ((event.key.code == Keyboard::W || event.key.code == Keyboard::Up) && head->direction != down)

{

head->direction = up;

}

else if ((event.key.code == Keyboard::S || event.key.code == Keyboard::Down) && head->direction != up)

{

head->direction = down;

}

else if ((event.key.code == Keyboard::A || event.key.code == Keyboard::Left) && head->direction != right1)

{

head->direction = left1;

}

else if ((event.key.code == Keyboard::D || event.key.code == Keyboard::Right) && head->direction != left1)

{

head->direction = right1;

}

}

}

}

void setTrailingSprite(NODE node, Sprite\* sp)

{

switch (node->direction)

{

case up:

sp->setTextureRect(IntRect(size \* 1, size \* 1, size, size));

node->s = sp;

break;

case down:

sp->setTextureRect(IntRect(size \* 2, size \* 1, size, size));

node->s = sp;

break;

case left1:

sp->setTextureRect(IntRect(size \* 0, size \* 1, size, size));

node->s = sp;

break;

case right1:

sp->setTextureRect(IntRect(size \* 0, size \* 1, size, size));

node->s = sp;

break;

}

}

void setHeadSprite(NODE node, Sprite\* sp)

{

switch (head->direction)

{

case up:

sp->setTextureRect(IntRect(size \* 1, size \* 0, size, size));

head->s = sp;

break;

case down:

sp->setTextureRect(IntRect(size \* 0, size \* 0, size, size));

head->s = sp;

break;

case left1:

sp->setTextureRect(IntRect(size \* 3, size \* 0, size, size));

head->s = sp;

break;

case right1:

sp->setTextureRect(IntRect(size \* 2, size \* 0, size, size));

head->s = sp;

break;

}

}

Vector2f getSnoopyCoords()

{

return Vector2f((rand() % 26 + 1) \* side \* 2, (rand() % 19 + 1) \* side \* 2);

}

void HandleSnoopyTouched()

{

if (isSnoopyTouched(head, &snoopy))

{

head = insertHead(head, &TrainSprite);

speed--;

while (1)

{

snoopyPos = getSnoopyCoords();

cur = head;

while (cur != NULL)

{

if (cur->posx == snoopyPos.x && cur->posy == snoopyPos.y)

{

break;

}

cur = cur->rlink;

}

if (cur == NULL)

break;

}

snoopy.setPosition(getSnoopyCoords());

score++;

Score.setString(String(std::to\_string(score)));

}

}

bool isSnoopyTouched(NODE head, Sprite\* snoopy)

{

return Vector2f(head->posx, head->posy) == snoopy->getPosition();

}

void checkIntersection(NODE head, RenderWindow\* window)

{

NODE cur = head->rlink;

while (cur != NULL)

{

if (head->posx == cur->posx && head->posy == cur->posy)

{

window->close();

RenderWindow window(VideoMode(800, 600), "Pick-Up Snoopy", Style::Titlebar | Style::Close);

CircleShape shape(800);

shape.setFillColor(Color::Red);

shape.setOrigin(300,300);

while (window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

}

Text t;

t.setFont(font);

t.setString("Score");

t.setCharacterSize(100);

t.setOrigin(-290,-120);

window.draw(shape);

window.draw(t);

Score.setOrigin(340,-175);

Score.setCharacterSize(180);

window.draw(Score);

window.display();

}

}

cur = cur->rlink;

}

}

public:

void Run()

{

Start();

while (window.isOpen())

{

Render();

processEvents();

if (count == speed)

{

Update();

}

count++;

}

}

void sound()

{

Music music;

music.openFromFile("resources/sound1.wav");

music.setVolume(10);

music.setLoop(true);

music.play();

}

};

int main()

{

RenderWindow window(VideoMode(800, 600), "Pick-Up Snoopy", Style::Titlebar | Style::Close);

CircleShape s(500);

while (window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

if (event.type == Event::Closed)

window.close();

}

Text t;

Font f;

s.setFillColor(Color::Red);

s.setOrigin(80,60);

f.loadFromFile("resources/new.otf");

t.setFont(f);

t.setString("Start");

t.setCharacterSize(100);

t.setOrigin(-290,-120);

window.draw(s);

window.draw(t);

window.display();

Event evt;

if (window.pollEvent(evt))

{

switch (evt.type)

{

case Event::Closed: window.close();

break;

case Event::KeyPressed:

if (evt.key.code == Keyboard::Enter)

{

Game game;

Music music;

music.openFromFile("resources/sound1.wav");

music.setVolume(30);

music.setLoop(true);

music.play();

window.close();

game.Run();

break;

}

else if(evt.key.code == Keyboard::Escape)

{

window.close();

break;

}

}

}

}}

# **Output**:

