# A blue circle with white text Description automatically generated**README for "Buzz Bombers" Game**

## **Overview**

Originally this game was developed in 1983 in Assembly language

This is my First sem solo programming fundamentals project.

"Buzz Bombers" is a game implemented using the SFML (Simple and Fast Multimedia Library) in C++ where the player controls a character that interacts with various in-game elements such as bees, honeycombs, flowers, and a hummingbird. The goal is to prevent bees from escaping while managing spray cans, firing bullets, and collecting items.

The game includes different states, such as the main menu, gameplay, high score viewing, and game-over scenarios. The core mechanics revolve around controlling the player and shooting bullets to interact with bees and honeycombs.

Made by: ***Abdul Ahad a proud fastian….:)***

## **game Contents**

1. **Constants & Initial Setup**
2. **Game States (Enums & Flow)**
3. **Core Gameplay Functions**
4. **Graphics & Rendering**
5. **Sound & Music**
6. **High Score Handling**
7. **4 Levels 3 normal ones the 4th is boss level**
8. **Detailed Function Explanation**

### **+Mechanism for Spray Cans, Lives, and Milestones Reward**

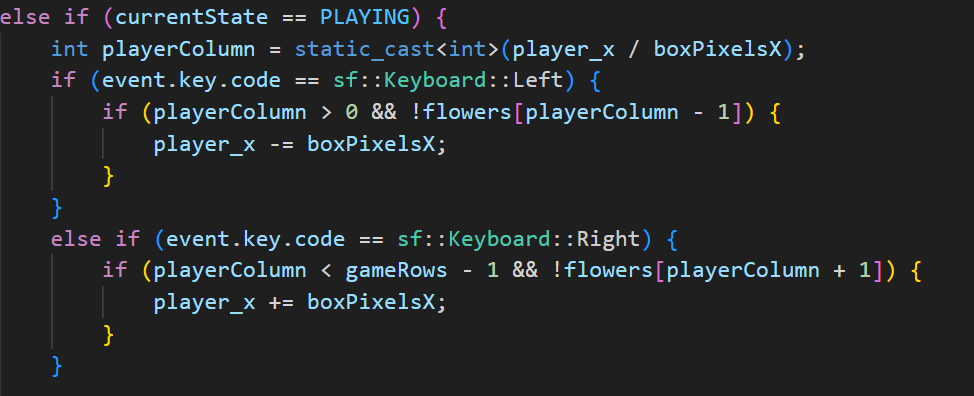
#### **Mechanism Overview:**

**In this game, the spray can serves as a resource for the player to fire shots (sprays). As the player progresses, they are rewarded with additional spray cans through milestones. The spray cans also have lives, meaning they can run out after a certain number of uses. The player can move left and right, fire sprays, and see the number of sprays remaining in each spray can.**

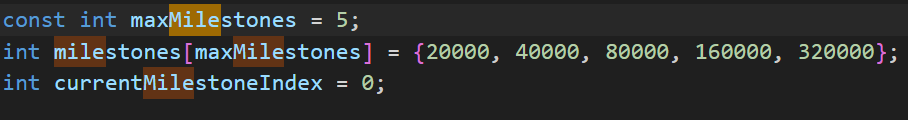
**The milestones reward the player with additional spray cans once specific score thresholds are reached, making the game easier as they progress. The lives mechanic ensures that once a spray can's sprays are exhausted, it is consumed, and the player will need to use a new one.**

**Code for Displaying Spray Cans, Sprays, and Lives**

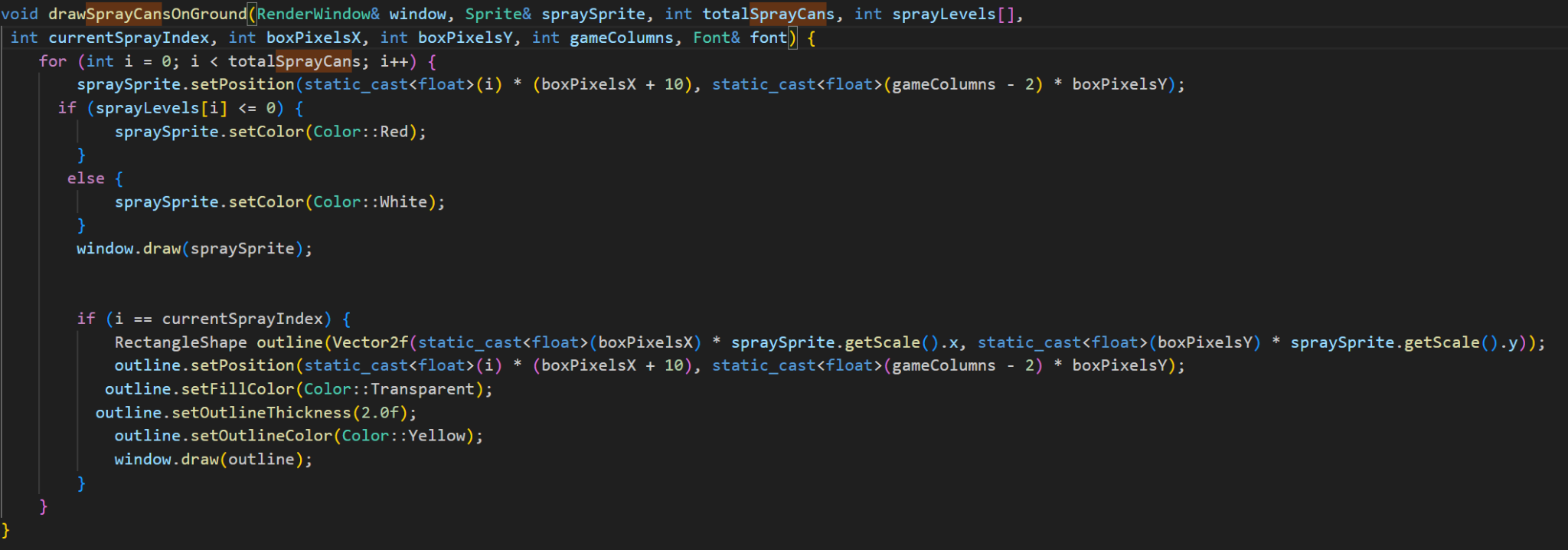
**Player ie spraycan movement on ground**

****

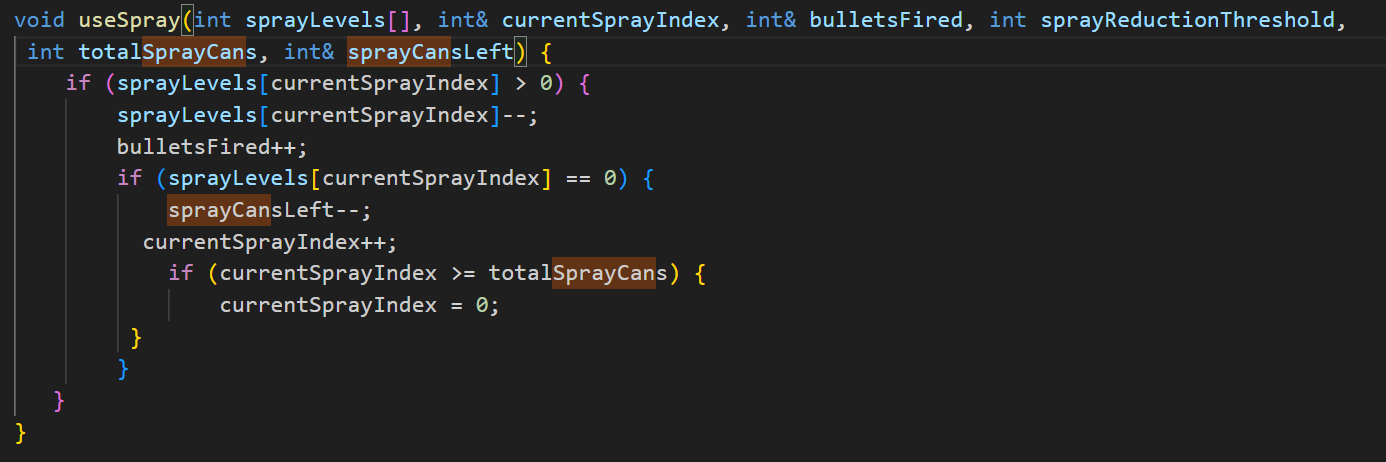
**Milestone score for unlocking additional can**

****

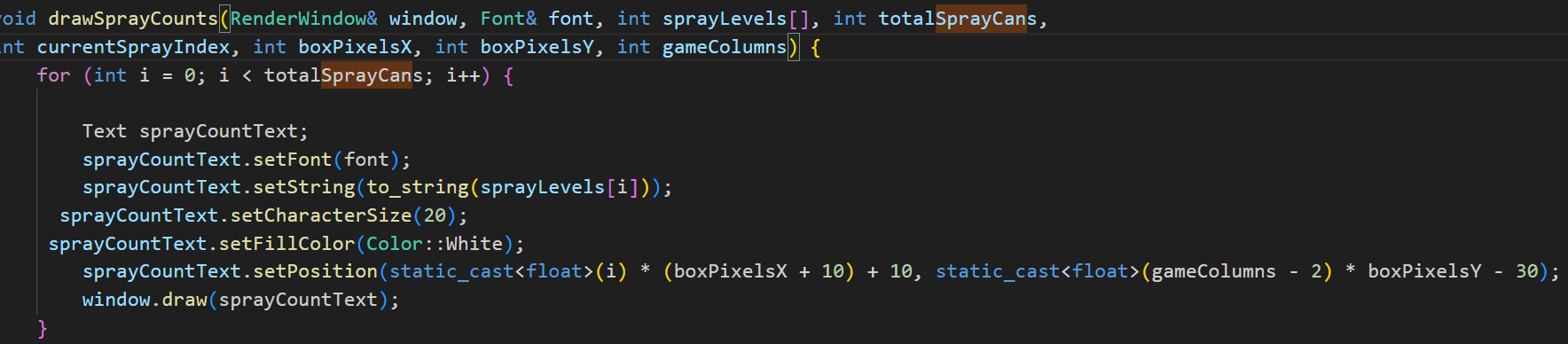
**Drawing spray can**

****

**One bullet per spray making sure of remaining sprays in the can**

****

**Spray counts above the spray cans left or in use :)**

****

**Now, let's see how these pieces work together in the context of the game:**

1. **ovement:**
   * **The player can move left or right across the screen, ensuring they don't run into flowers or other obstacles.**
2. **Sprays:**
   * **The player uses sprays from the spray cans. Each spray can has a fixed number of uses (maxSpraysPerCan), and once a can is empty, the player switches to the next available can.**

**3.Lives:**

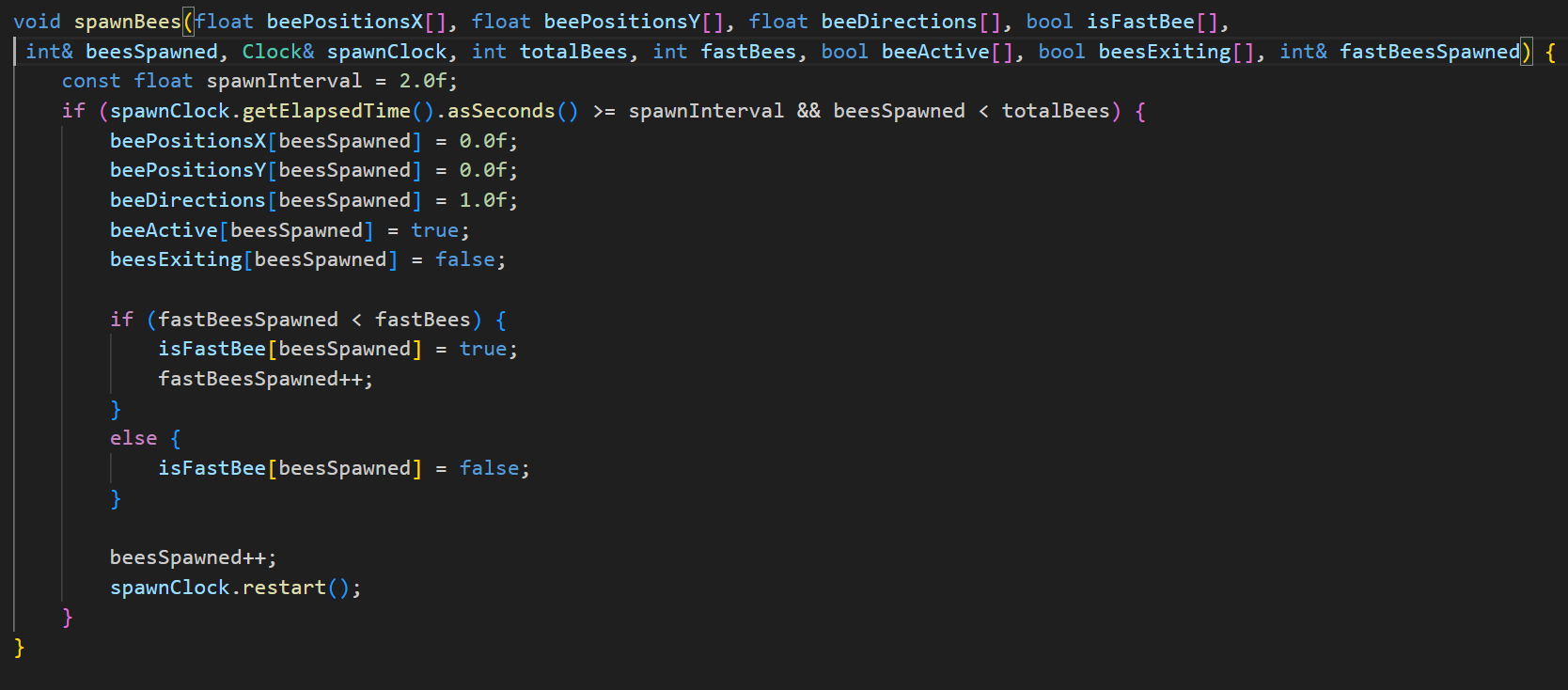
* + **The player has a set number of lives (lives). If they lose all their lives, the game ends, and the player must restart.**

**4. Milestones:**

* + **As the player achieves milestones by scoring points, they are rewarded with additional spray cans. These rewards are limited by the maximum number of spray cans the player can have (maxSprayCans).**

**The Bees 🐝**

### **1. The spawnBees FunctionThe spawnBees function handles the creation of bees at regular intervals. The logic in this function is responsible for controlling the number of bees to spawn, marking some bees as "fast," and updating relevant state variables**

****

### **Key Points in spawnBees:**

* **Intervals: Bees are spawned at regular intervals of 2 seconds.**
* **Position & Movement: Initially, bees are placed at the left (beePositionsX[beesSpawned] = 0.0f) and set to move right (beeDirections[beesSpawned] = 1.0f).**
* **Bee Type (Fast/Normal): Some bees are fast (isFastBee[beesSpawned] = true), and others are normal. The maximum number of fast bees is controlled by the fastBees parameter.**
* **Activation & State Tracking: The function tracks if the bees are active or exiting (beeActive[beesSpawned], beesExiting[beesSpawned]).**

### **2. The moveBees Function**

**The moveBees function is where most of the core mechanics for bee movement, collision detection, and interactions with honeycombs and flowers are handled.**

**Its a very long one which includes most of bee related mechanics here are few points**

### **Key Points in moveBees:**

* **Bee Movement:**
  + **The bees move based on their direction and speed, either normal or fast.**
  + **Bees change direction when they hit screen borders, and they move down to the next tier of flowers after each bounce (beePositionsY[i] += tierHeight).**
* **Honeycomb Collision:**
  + **Worker Bees: Collide with honeycombs (yellow honeycombs) and bounce off, moving to the next tier. They are the only type that interacts with honeycombs.**
  + **Hunter Bees: Do not collide with honeycombs (red honeycombs), allowing them to freely pass through.**
* **Pausing Worker Bees:**
  + **Worker bees occasionally stop for a short duration, as indicated by the random pause mechanism (rand() % 1000 < 5).**
* **Exiting the Screen:**
  + **Once a bee reaches the bottom (beePositionsY[i] >= tierAbovePlayer), it starts pollinating the flowers, alternating between placing flowers on the left and right sides.**
  + **After pollination, bees are marked as "exiting" and move off the screen.**

**Flower Creation 🌻**

**Its mechanism is incorporated in move bees which passes onto draw bees function**

**The First Bee: Creates two consecutive flowers on the left and right borders of the screen when it reaches the bottom.**

**Subsequent Bees: Each subsequent bee creates one flower, alternating between the left and right sides.**

**Middle Flower: A flower is placed in the middle of the screen every time a bee reaches the bottom.**

* **he first bee that reaches the bottom (beePositionsY[i] >= tierAbovePlayer) triggers the creation of two flowers: one on the left and one on the right.**
* **The firstBeeTriggered flag ensures that this only happens once.**

**Subsequent Bees:**

* **Each subsequent bee creates one flower, which alternates between the left and right sides. The variable placeLeftFlowerNext handles this alternation.**

**Middle Flower:**

* **In addition to the left and right flowers, a flower is placed in the middle. The index for the middle column is calculated as gameColumns / 2.**

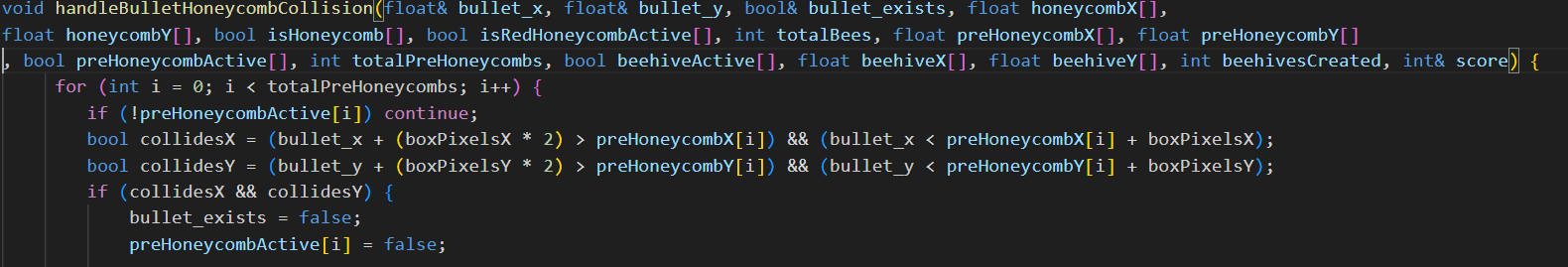
**Flower Index Tracking:**

* **leftFlowerIndex and rightFlowerIndex are used to track where flowers are placed on the left and right sides of the grid, respectively.**

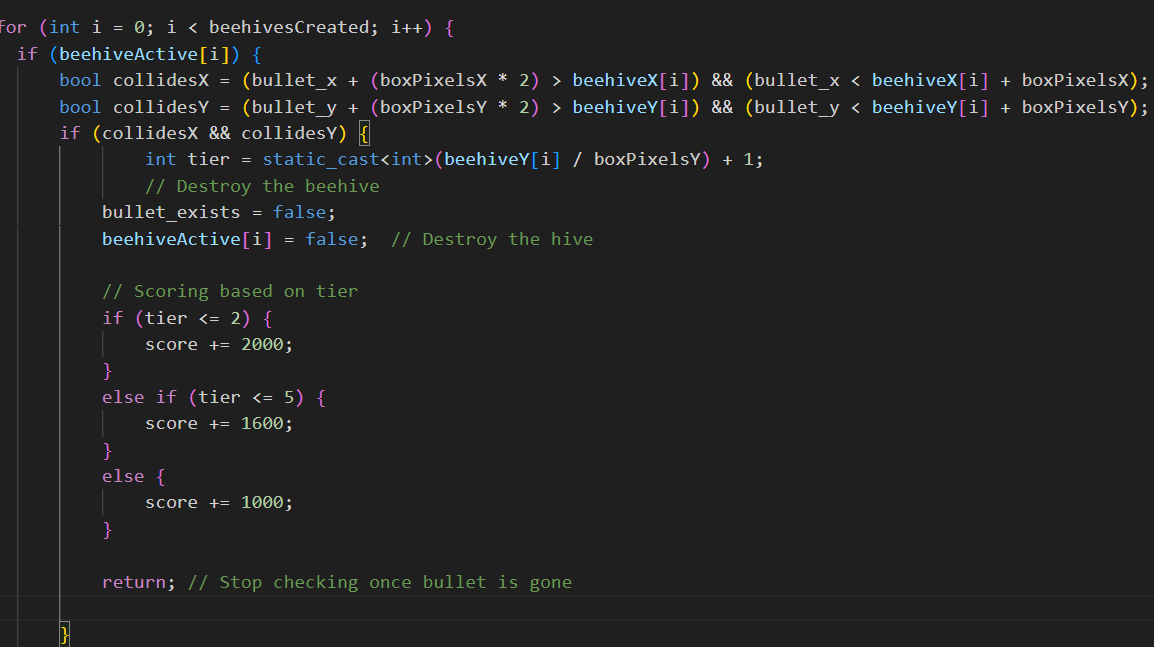
**Flower Placement in the Grid:**

* **Each flower is placed by setting the corresponding index in the flowers[] array to true.**

**Bullet honeycomb and beehive mechanics 🔫**

****

**For beehive:**

****

### **Key Points Summary:**

* **Collision Detection: The handleBulletHoneycombCollision function checks for collisions between the bullet and honeycombs or beehives, and updates the game state accordingly (e.g., deactivating the honeycomb or beehive, and updating the score).**
* **Scoring System: The scoring is based on the tier (vertical position) of the honeycomb or beehive, with higher tiers awarding lower scores. Red honeycombs have a higher score value compared to normal honeycombs.**
* **Beehive Creation: Beehives are created when a bee gets "stuck." The initBeeHives function initializes the beehive data structures.**
* **Drawing Beehives: The drawBeehives function draws active beehives on the screen.**

### **+Detailed Analysis of Hummingbird Mechanics 🕊️The core mechanics of the hummingbird are centered around three key functionalities: *1Movement Towards the Closest Honeycomb***

* **Pausing Mechanism**
* **Random Movement**
* **Eating the Honeycomb**

### **1. Movement Towards the Closest Honeycomb && Eating it away ,random motion of humming bird**

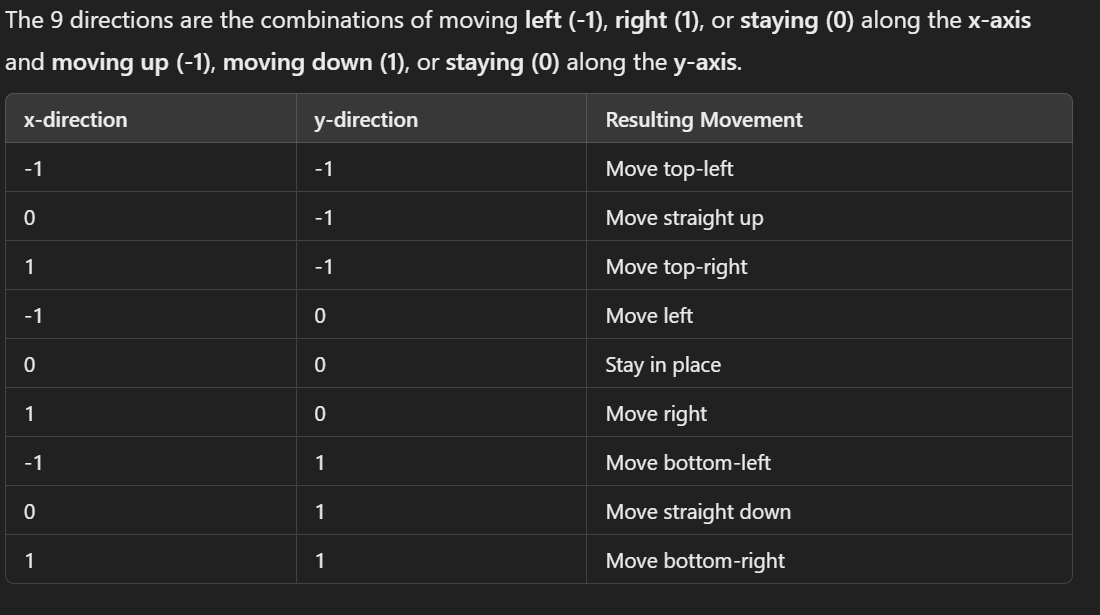
**Code:**



**Explanation:**

* **Purpose: This block of code is responsible for calculating the direction of the hummingbird towards the closest honeycomb. If the hummingbird is paused, it searches for the closest honeycomb to move toward.**
* **Steps:**
  + **Pause Time Handling:**
    - **When hummingbird\_paused is true, the hummingbird\_pauseTime is reduced by moveInterval (0.05 seconds). Once the hummingbird\_pauseTime becomes zero or negative, the hummingbird stops being paused and proceeds to look for the closest honeycomb.**
  + **Finding the Closest Honeycomb:**
    - **It loops over all honeycombs (totalBees) to check if each honeycomb is active (isHoneycomb[i]).**
    - **For each active honeycomb, it calculates the Euclidean distance (distance) between the hummingbird’s current position (hummingbird\_x, hummingbird\_y) and the honeycomb’s position (honeycombX[i], honeycombY[i]).**
    - **If this distance is smaller than the current minDistance, it updates minDistance, and the target coordinates (targetX, targetY) of the closest honeycomb.**
  + **Movement Calculation:**
    - **Once the closest honeycomb is found, the hummingbird’s movement direction is calculated by normalizing the vector from the hummingbird to the honeycomb. The difference in coordinates (dx, dy) is divided by the magnitude (magnitude) to get the unit direction vector (hummingbird\_directionX, hummingbird\_directionY).**
* **Result:**

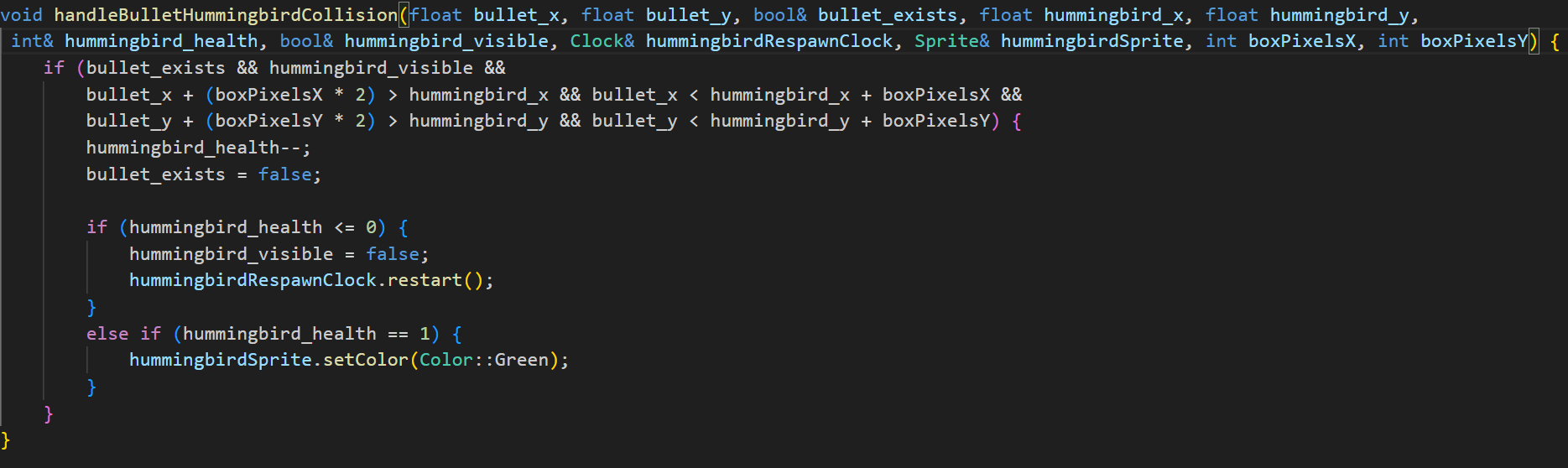
**The hummingbird mechanics in this code are designed to make the hummingbird move in a combination of deterministic (towards the closest honeycomb) and stochastic (random movement) ways. Here's the sequence of behaviors:**

1. **Movement towards the closest honeycomb: The hummingbird always tries to move towards the closest honeycomb if it's not paused. It calculates the direction vector and moves in that direction.**
2. **Pausing: The hummingbird pauses at random intervals (with a 1% chance per movement cycle) for 1 second, adding unpredictability.**
3. **Random movement: If no honeycomb is found or if the hummingbird is not actively searching for one, it will move randomly in one of 9 directions.**
4. **Eating honeycomb: When the hummingbird's bounding box overlaps with a honeycomb, the honeycomb is marked as eaten (deactivated), removing it from the game.**
5. **The directions possible for movement of humming bird is:**
6. ****

**Health conscious humming bird:**

**When bullet hits the humming bird its health decreases and turns green on 2nd bullet shot indicating its low health and the next shot will be its last and it will be dead so sad:(**

**But theres a twist it will respawn after an interval healthy and fit :)**

****

### **+File Handling for High Scores**

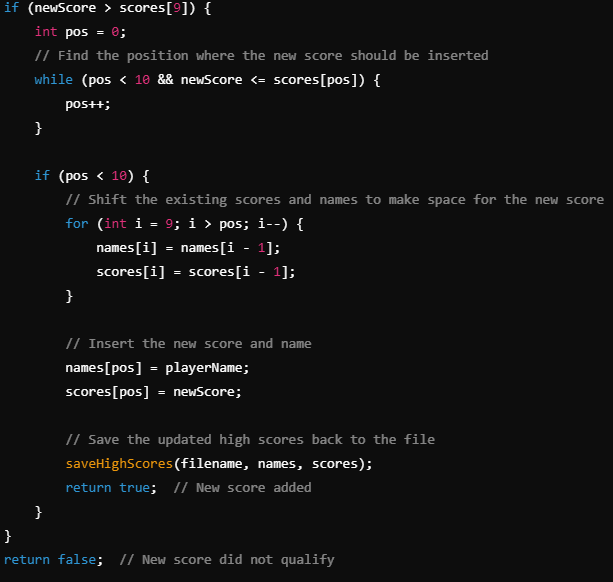
In this game, high scores are stored and retrieved from a text file (highscores.txt). The code uses **file input/output** (I/O) operations,and string library , to load and save the top 10 scores and player names. This is handled through functions like loadHighScores, saveHighScores, and addNewScore.

Following is the highlighting feature of highscore tracking system:

**Purpose**: This function checks if the player's score qualifies for the top 10 high scores and inserts it in the correct position.

**Flow**:

1. It first loads the existing high scores using the loadHighScores function.
2. It checks if the new score is higher than the lowest score in the top 10 (scores[9]).
3. If the new score qualifies:
   * The function finds the correct position to insert the new score by iterating through the existing scores.
   * It shifts the lower scores down to make space for the new score.
   * It inserts the new name and score at the correct position.
4. After updating the scores, it calls saveHighScores to save the new high scores back to the file.
5. The function returns true if the score was added, and false if it was not.





A screenshot of a game

Description automatically generated

A screenshot of a game

Description automatically generated

A screenshot of a video game

Description automatically generated

A screenshot of a video game

Description automatically generated

A screenshot of a game

Description automatically generated