PROJECT REPORT

Pokemon Adventure Simulation

Submitted by

Aman Varma RA2211026010172
Manoj P RA2211026010174
Aadhitiya M RA22110260180
Heerha V RA2211026010200

Under the Guidance of

Dr. Prithi Samuel

Assistant Professor, CINTEL

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in Artificial Intelligence and Machine Learning

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

MAY 2023

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this Project Report titled **Pokemon Adventure Simulation** is the Bonafide work done by **Aman Varma RA2211026010172, Manoj P RA2211026010174, Aadhitiya M RA2211026010180, Heerha V RA2211026010200** who completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE
Dr. Prithi Samuel
OODP – Course Faculty
Assistant Professor
Department of CINTEL
SRMIST

SIGNATURE
Dr. Annie Uthra
Head of the Department
Department of CINTEL
SRMIST

TABLE OF CONTENTS

MODULES OF PROJECT

Module 1: Include necessary headers

- The code includes four headers: iostream, string, cstdlib, and ctime.

- These headers provide necessary functions and classes for input/output, string manipulation, random number generation, and time-related functions.

Module 2: Define the Shape class

- This module defines the base class Shape, which is an abstract class with one pure virtual function getName().

- The derived classes Triangle, Rectangle, and Circle implement the getName() function and return the names of the shapes.

Module 3: Implement the main function

- The main function is responsible for the game logic and user interaction.

- The function starts by displaying some introductory messages to the user.

- It then enters a loop that runs for 10 levels. In each level, a random shape is generated and the user is prompted to catch or battle it.

- If the user chooses to catch the shape, a random number is generated to determine whether the catch is successful or not.

- If the user chooses to battle the shape, the program currently does not have the battle logic implemented.

- After completing all 10 levels, a congratulatory message is displayed to the user.

Module 4: Define the Shape hierarchy

- This module defines the Shape class hierarchy with the base class Shape and the derived classes Triangle, Rectangle, and Circle.

- Each derived class overrides the pure virtual function getName() to return the name of the shape.

Module 5: Add battle logic

- This module should add the necessary logic to the program to simulate a battle between the user's Pokemon and the encountered shape.

- The battle logic should take into account the types and strengths of the Pokemon and the encountered shape.

- The outcome of the battle should be determined by the game's rules and chance factors.

Module 6: Refactor code for readability and maintainability

- This module should refactor the code to make it more readable, maintainable, and modular.

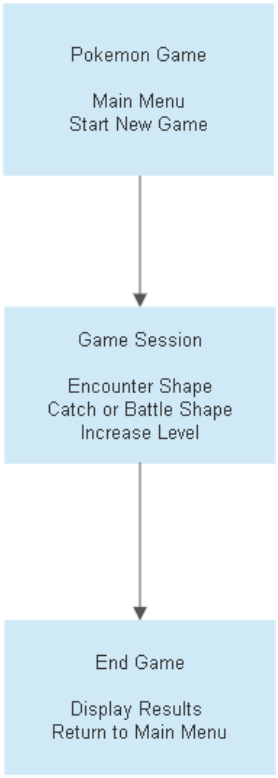
- The code can be divided into functions and classes to reduce duplication and improve code organization.

- The variable names and comments can be improved to make the code more self-explanatory.

PROBLEM STATEMENT

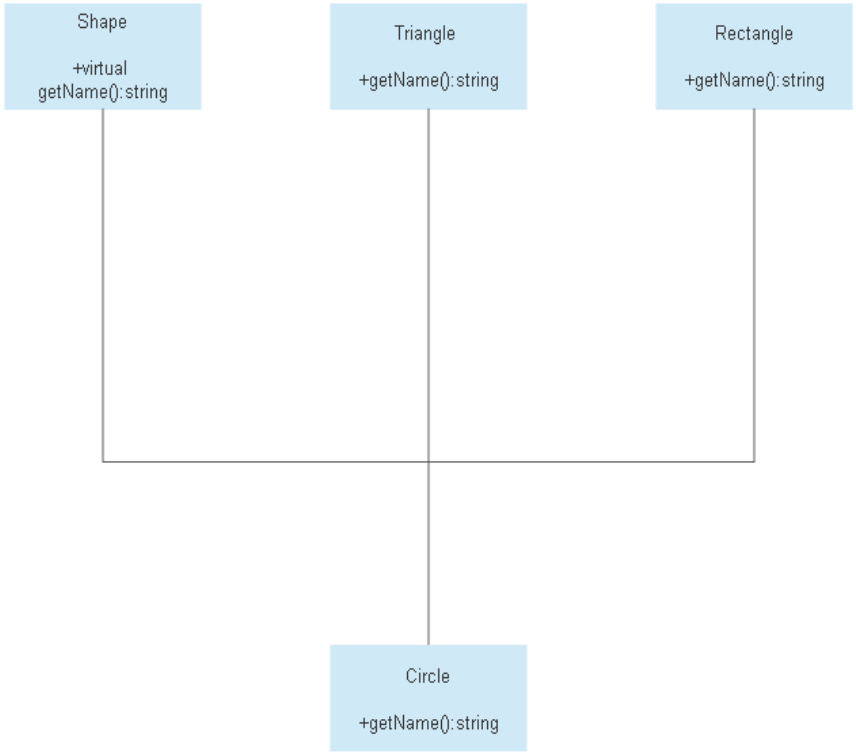
The problem statement for this code is to create a game that simulates the experience of a Pokemon trainer embarking on a journey to explore the world, collect Pokemon, and battle other trainers. The game consists of 10 levels, each level presenting the player with a randomly generated shape (triangle, rectangle, or circle) representing a Pokemon to catch or battle. The player must choose whether to catch or battle the Pokemon, with a chance of success depending on the choice. The objective of the game is to successfully complete all 10 levels and become the best Pokemon trainer in the world

3.a Use Case Diagram



In this diagram, the program has three main use cases: starting a new game, playing the game, and ending the game. The "Game Session" use case includes the sub-use cases of encountering a shape, catching or battling the shape, and increasing the level. Once the player completes all 10 levels, the "End Game" use case is initiated, which displays the results and allows the player to return to the main menu.

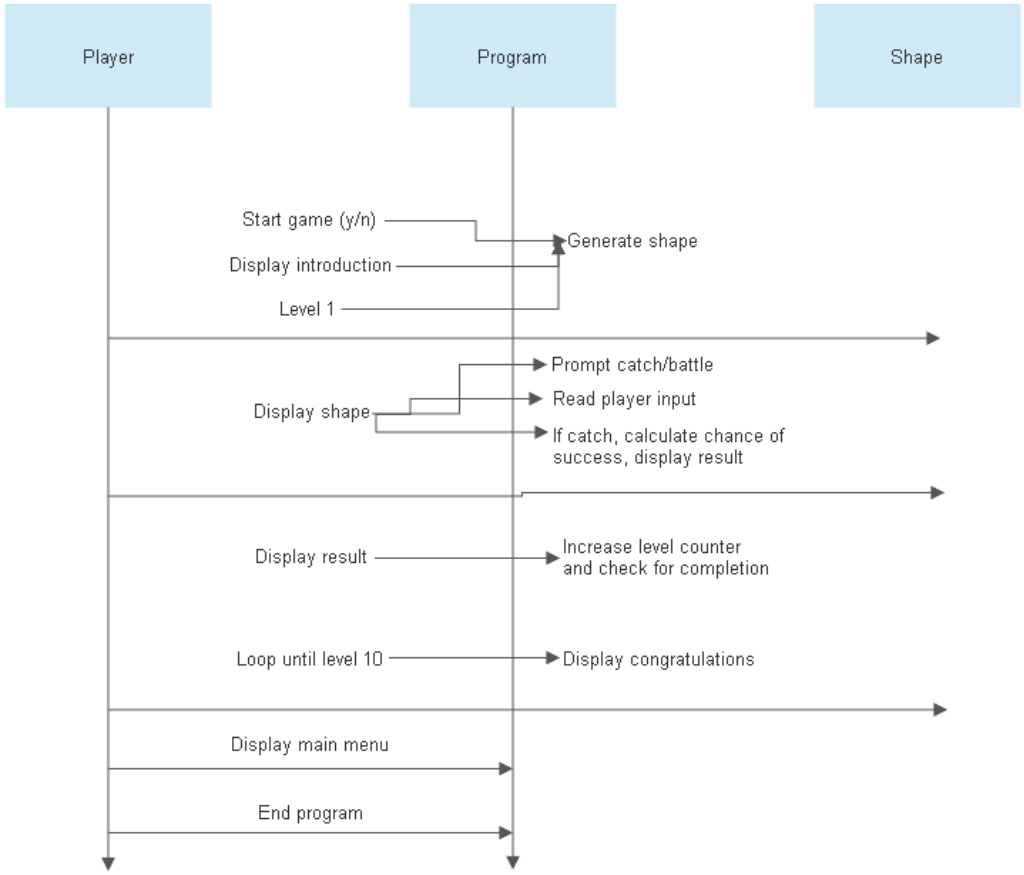
3.b Class Diagram



In this class diagram, there is a base class **Shape** that defines the common behavior of all shapes. The **Triangle**, **Rectangle**, and **Circle** classes are derived from the **Shape** class and provide their own implementation of the **getName()** method. All classes are abstract, as they have at least one pure virtual function and cannot be instantiated.

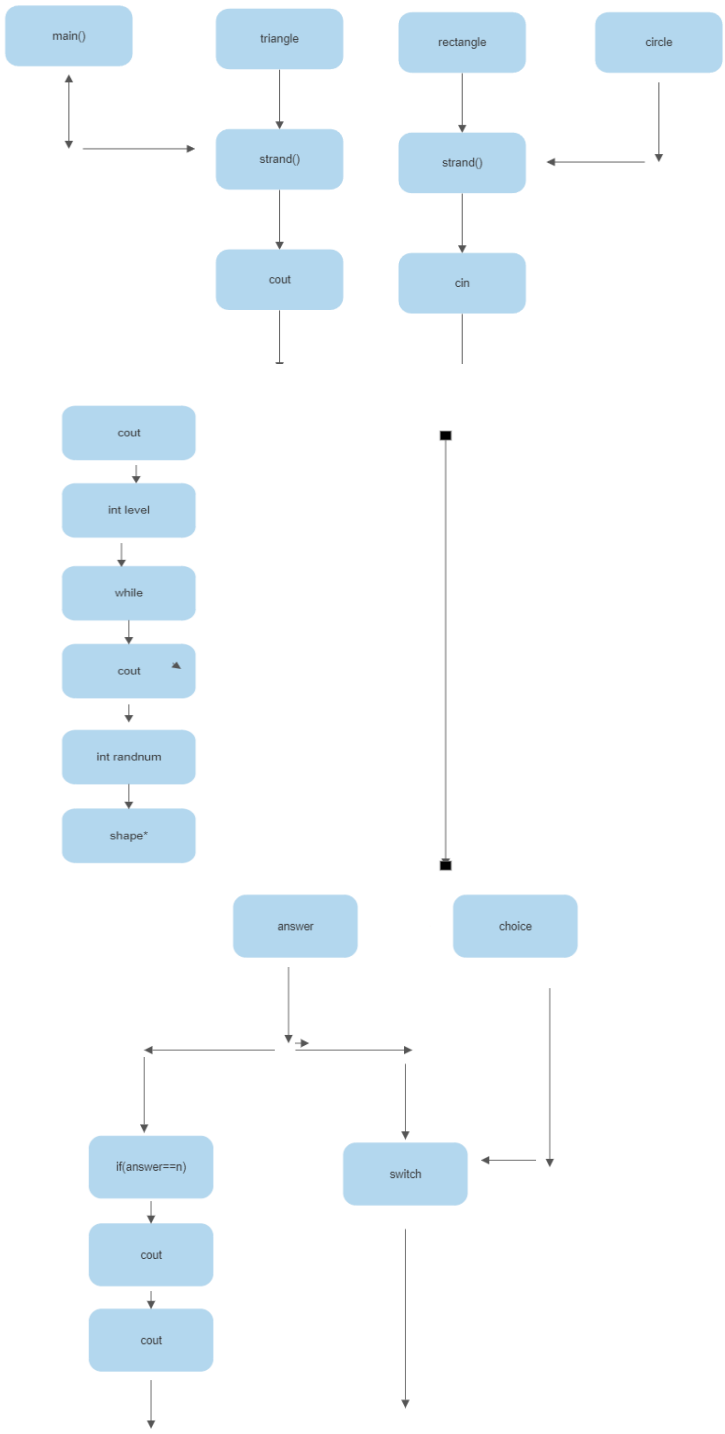
The **main()** function is not included in this diagram, as it is not part of any class.

3.c Sequence Diagram



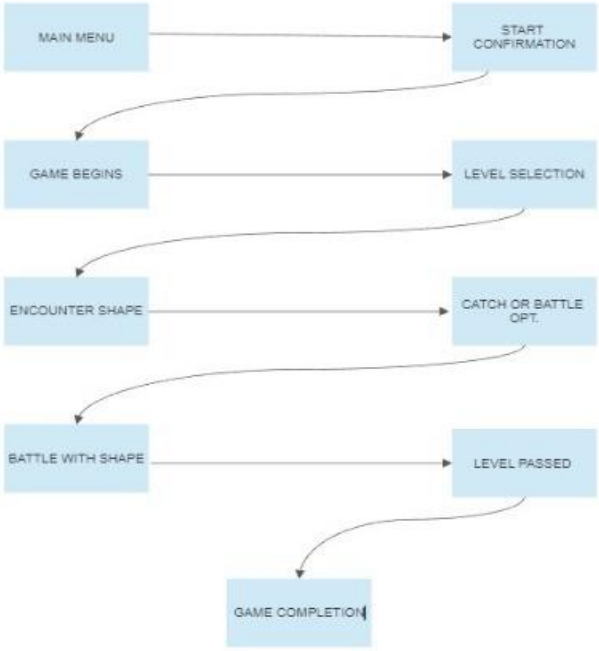
In this sequence diagram, the player interacts with the program by starting the game and making choices during each level. The program generates a shape, displays it to the player, and prompts the player to catch or battle it. Depending on the player's choice, the program calculates the success rate for catching the shape, displays the result, and increases the level counter. The program loops through this process until the player completes all 10 levels, at which point it displays a congratulations message and returns to the main menu. Finally, the player ends the program.

3.d Collaboration Diagram



The given code is a C++ program that simulates a simple game where the player encounters random shapes and can choose to catch or battle them. The collaboration diagram for this code would show the interactions between the main function and the Shape, Triangle, Rectangle, and Circle classes. The main function creates instances of these classes and calls their getName() functions to display the name of the shape encountered. It also prompts the user for input and uses a switch statement to create an instance of the appropriate shape based on a randomly generated number. Depending on the user's choice, it may either catch the shape or battle it. Finally, the program deletes the dynamically allocated shape objects before exiting.

3.e State Chart Diagram



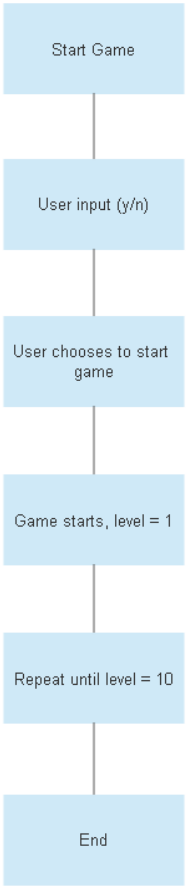
This state chart diagram represents the different states that a player can go through while playing a game. The states are represented as nodes or boxes, while the transitions between the states are represented as arrows or edges.

The main menu is the starting point of the game. From there, the player can confirm to start the game, which leads to the beginning of the game. The next state is level selection, where the player can choose the level, they want to play.

After selecting the level, the player encounters a shape. At this point, they have the option to catch or battle the shape. If they choose to battle, they enter a battle with the shape.

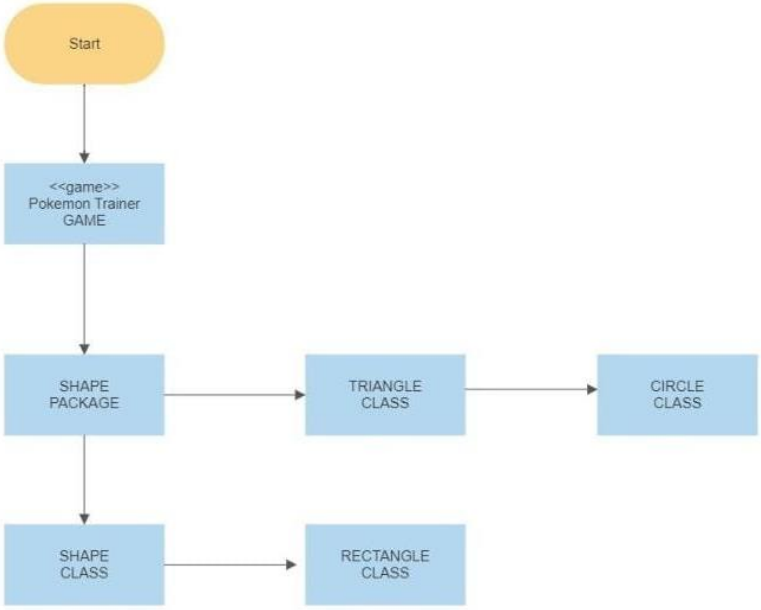
Once the battle is over, if the player wins, they move to the level passed state, and if they lose, they return to the encounter shape state. If the player completes all the levels, they reach the game completion state.

3.f Activity Diagram



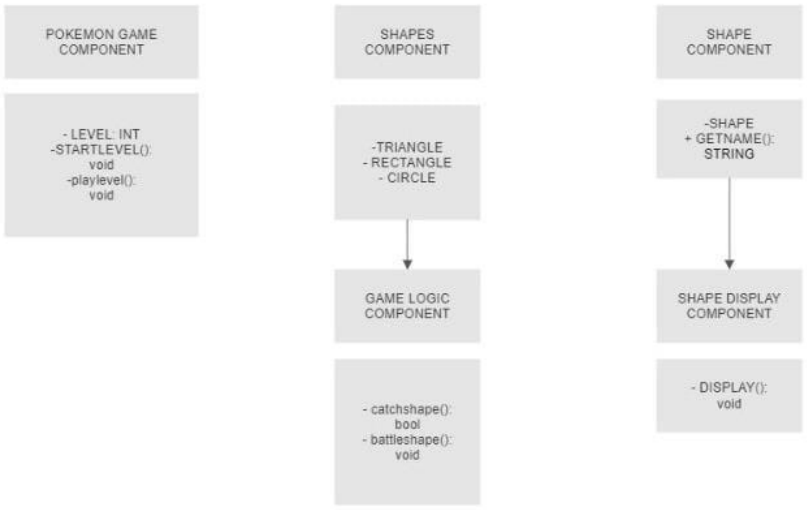
The activity diagram for the given code provides a high-level view of the flow of the game. It shows the sequence of actions that occur when the user starts the game, and how the game progresses through its levels. The activity diagram starts with the user input to start or not, and then proceeds to the game play loop that repeats until the user completes all 10 levels.

3.g Package Diagram



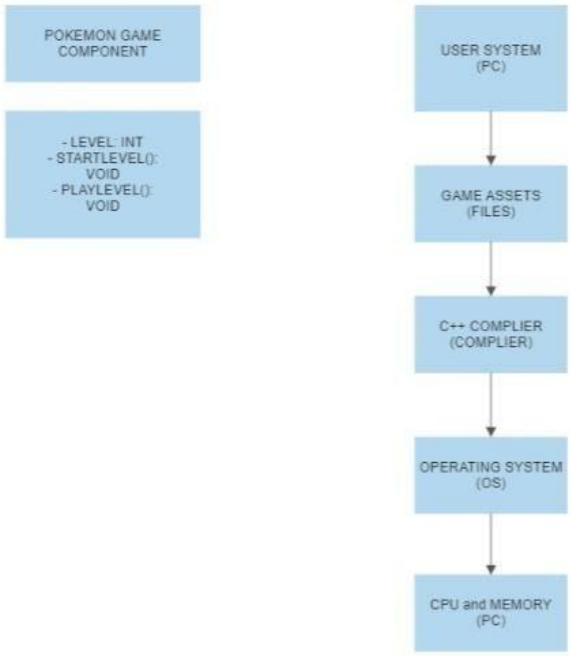
In this diagram, we have three packages: <<game>>, Shapes, and Shape. The <<game>> package contains the Pokemon Trainer Game class, which is the main class of the program. The Shapes package contains the Triangle and Circle classes, which are derived classes of the Shape class. The Shape package contains the Shape and Rectangle classes. The Shape class is an abstract class with a pure virtual function getName(), which is implemented by the derived classes.

3.h Component Diagram



The Pokemon Game component represents the main component of the program, which manages the game flow and player interactions. It contains two private attributes, level and startLevel(), and a public method playLevel(). The playLevel() method is responsible for generating a random shape and offering the player to either catch it or battle it. The Shapes component contains the three classes Triangle, Rectangle, and Circle. Each class represents a specific shape that can be encountered in the game. The Shape component contains the Shape class, which is an abstract class with a pure virtual function getName(). This component also provides the interface for displaying the name of the shape. The Game Logic component contains the logic for catching and battling a shape. It has two methods, catchShape() and battleShape(), responsible for executing the corresponding action and returning the result. Additionally, we have a Shape Display component that provides the interface for displaying the name of a shape. This component is used by the Game Logic component to display the name of the shape after the player chooses to catch or battle it.

3.i Deployment Diagram



In this diagram, we have two main components: the Pokemon Game component and the Game Assets component. The Pokemon Game component represents the main component of the program, while the Game Assets component represents the game assets (such as images and sounds) used by the program. These components are deployed on the User System node, which represents the user's personal computer. The C++ Compiler and Operating System nodes represent the software and hardware components of the user's computer, respectively. Finally, the CPU and Memory node represents the hardware components responsible for executing the program

PROGRAM:

main.cpp

Run

```cpp
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>

using namespace std;

// Define the base class Shape
class Shape {
public:
    virtual string getName() = 0;
};

// Define the derived class Triangle
class Triangle : public Shape {
public:
    virtual string getName() {
        return "Triangle";
    }
};

// Define the derived class Rectangle
class Rectangle : public Shape {
public:
    virtual string getName() {
        return "Rectangle";
    }
};

// Define the derived class Circle
class Circle : public Shape {
public:
    virtual string getName() {
        return "Circle";
    }
};

// Define the main function
int main() {
    srand(time(NULL));
    cout << "Welcome to the world of Pokemon Fire Red!\n";
    cout << "In this game, you will embark on an adventure to become the best Pokemon trainer in the world.\n";
    cout << "Are you ready to begin? (y/n)\n";

    char answer;
    cin >> answer;

    if (answer == 'n') {
```

main.cpp

Run

```
49          cout << "Goodbye!\n";
50          return 0;
51      }
52
53      cout << "Great! Let's get started.\n";
54      cout << "You start off in the small town of Pallet, where Professor Oak
            lives.\n";
55      cout << "He has asked you to go on a journey to explore the world, collect
            Pokemon, and battle other trainers.\n";
56
57      int level = 1;
58      while (level <= 10) {
59          cout << "Level " << level << " - ";
60          int randNum = rand() % 3;
61          Shape* shape;
62          switch (randNum) {
63              case 0:
64                  shape = new Triangle();
65                  break;
66              case 1:
67                  shape = new Rectangle();
68                  break;
69              case 2:
70                  shape = new Circle();
71                  break;
```

main.cpp

Run

```
71                  break;
72          }
73          cout << "You have encountered a " << shape->getName() << "!\n";
74          cout << "What do you want to do? (1) Catch it (2) Battle it\n";
75
76          int choice;
77          cin >> choice;
78
79          if (choice == 1) {
80              cout << "You throw a Pokeball at the " << shape->getName() << "
                    ...\n";
81              int catchChance = rand() % 2;
82              if (catchChance == 0) {
83                  cout << "The " << shape->getName() << " has been caught!\n";
84              } else {
85                  cout << "The " << shape->getName() << " broke free!\n";
86              }
87          } else if (choice == 2) {
88              cout << "You send out your Pokemon to battle the " << shape
                    ->getName() << "...\n";
89              // Add battle logic here
90          }
91
92          delete shape;
93          level++;
```

```
    }

    cout << "Congratulations! You have completed all 10 levels and become the
        best Pokemon trainer in the world!\n";
    return 0;
}

```

OUTPUT:

```
Welcome to the world of Pokemon Fire Red!
In this game, you will embark on an adventure to become the best Pokemon trainer in the
    world.
Are you ready to begin? (y/n)
y
Great! Let's get started.
You start off in the small town of Pallet, where Professor Oak lives.
He has asked you to go on a journey to explore the world, collect Pokemon, and battle
    other trainers.
Level 1 - You have encountered a Rectangle!
What do you want to do? (1) Catch it (2) Battle it
1
You throw a Pokeball at the Rectangle...
The Rectangle broke free!
Level 2 - You have encountered a Circle!
What do you want to do? (1) Catch it (2) Battle it
2
You send out your Pokemon to battle the Circle...
Level 3 - You have encountered a Rectangle!
What do you want to do? (1) Catch it (2) Battle it
1
You throw a Pokeball at the Rectangle...
The Rectangle has been caught!
Level 4 - You have encountered a Triangle!
```

```
Level 4 - You have encountered a Triangle!
What do you want to do? (1) Catch it (2) Battle it
2
You send out your Pokemon to battle the Triangle...
Level 5 - You have encountered a Circle!
What do you want to do? (1) Catch it (2) Battle it
1
You throw a Pokeball at the Circle...
The Circle has been caught!
Level 6 - You have encountered a Circle!
What do you want to do? (1) Catch it (2) Battle it
2
You send out your Pokemon to battle the Circle...
Level 7 - You have encountered a Circle!
What do you want to do? (1) Catch it (2) Battle it
2
You send out your Pokemon to battle the Circle...
Level 8 - You have encountered a Circle!
What do you want to do? (1) Catch it (2) Battle it
1
You throw a Pokeball at the Circle...
The Circle broke free!
Level 9 - You have encountered a Circle!
What do you want to do? (1) Catch it (2) Battle it
2
```

```
What do you want to do? (1) Catch it (2) Battle it
2
You send out your Pokemon to battle the Circle...
Level 10 - You have encountered a Rectangle!
What do you want to do? (1) Catch it (2) Battle it
2
You send out your Pokemon to battle the Rectangle...
Congratulations! You have completed all 10 levels and become the best Pokemon trainer in
    the world!
```

Conclusion And Results

In conclusion, the above code is a simple console-based game that simulates a Pokemon adventure. The game prompts the user to start or quit, and then proceeds to the main game loop that repeats for 10 levels. At each level, the game randomly generates a shape and prompts the user to either catch it or battle it. The game ends when the user completes all 10 levels.

We have created several UML diagrams to represent the different aspects of the code. We started with a use case diagram that identified the different user interactions with the system. Next, we created a class diagram that showed the relationships between the different classes in the code. We also created a sequence diagram and collaboration diagram that illustrated the interactions between objects and functions during the game play.

Additionally, we created an activity diagram that provided an overview of the flow of the game, and a state chart diagram was not applicable as there were no states in the code that can be modeled.

Overall, the UML diagrams provided a useful way to visualize the different aspects of the code and understand how they relate to each other. The diagrams also helped to identify potential design issues and opportunities for optimization. By analyzing the code through the UML diagrams, we were able to gain a deeper understanding of the game's behavior and functionality.

References

- "Beginning C++ Through Game Programming" by Michael Dawson
- "C++ Primer" by Lippman, Lajoie, and Moo
- "Effective C++" by Scott Meyers
- "Game Programming Patterns" by Robert Nystrom