# README → “PLANTS VS ZOMBIES”

## INTRODUCTION

This documentation is about a project inspired by “Plants vs Zombies” games that I finished this semester using OpenGL in Microsoft Visual Studio for the front-end part and Object-Oriented Programming (OOP) principles in C++ for the back-end part. This game presents a blend of strategy and action, being based on the interaction with the player, who must defend his grid (home) from the enemies.

## OPENGL

OpenGL is the industry's most widely used, supported and best documented 2D/3D graphics API making it inexpensive & easy to obtain information on implementing OpenGL in hardware and software. There are numerous books, tutorials, online coding examples, coding seminars, and classes that document the API, Extensions, Utility Libraries, and Platform Specific Implementations. This helped me to implement the interface of my game, given the fact that it was the first time working with OpenGL.

## OOP

The foundation of my game's code lies in the principles of Object-Oriented Programming (OOP). C++ makes easy the implementation of OOP concepts such as encapsulation, inheritance, and polymorphism, allowing to create a structured and modular design. By organizing my code around classes and objects, I enhance code readability, promote reusability, and simplify the overall development and maintenance process.

## VISUAL STUDIO

Visual Studio provides an intuitive environment and advanced debugging tools making it the ideal IDE that support the needs of C++ developers. It also works out great with the framework for rendering 2D/3D graphics provided by OpenGL.

## GAME LOGIC

The game logic is based on the interaction with the player. The “plants” in my game are diamonds and the “zombies” are hexagons. The player is: collecting stars that appear at random intervals on the screen by pressing on the button mouse, placing diamonds in the grid by means of drag & drop only if he collected enough stars for that diamond, deleting diamonds from the grid when they are no longer necessary.

In this game, you navigate three rows with three columns each. Hexagon enemies appear from the right, moving left at random intervals. If a hexagon reaches the end of a row (the red rectangular), you lose a life. Lose three lives, and the game ends.

To combat the hexagons, the player strategically places a diamond in one of the three cells in each row (drag & drop). The diamond launches projectiles that move to the right along the row in which the diamond is placed. When a hexagon is hit by three projectiles, it disappears, temporarily saving the situation.

However, the player faces two challenges. They cannot place as many diamonds as they wish, as each diamond costs a certain number of stars. Stars appear at random intervals on the screen, and the player must collect them. Additionally, each diamond can only harm a specific type of hexagon based on its color.

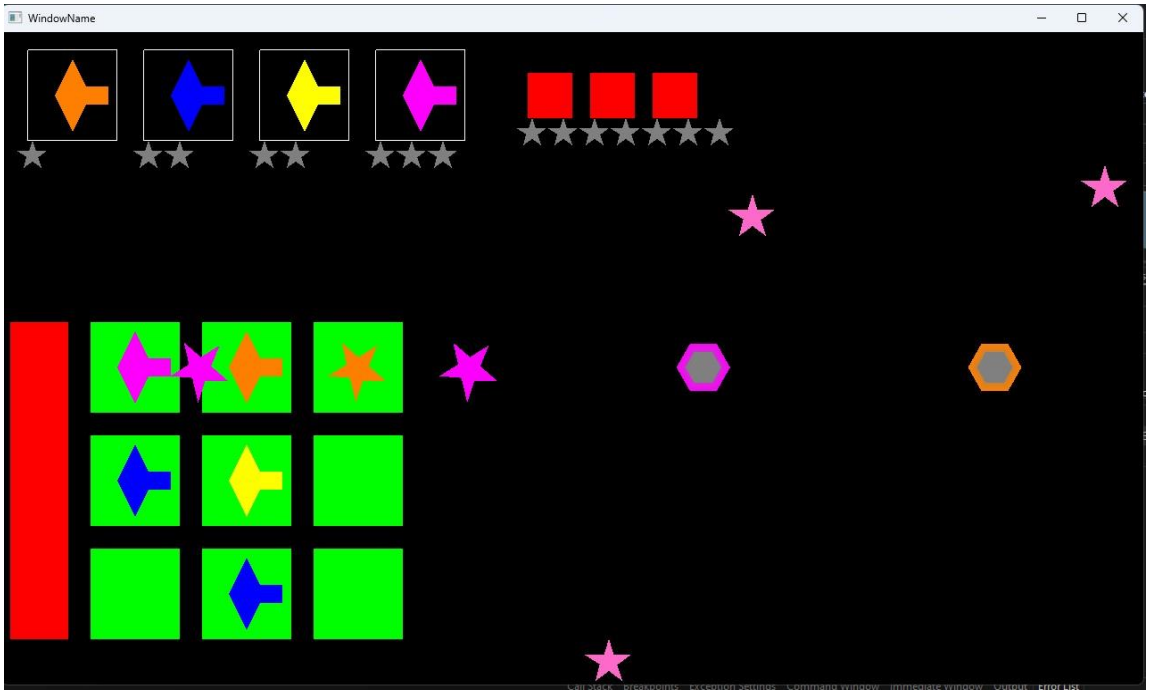


Fig. 1

The collected stars (click right on the pink stars on the screen) appear under the lives left (red squares). When a diamond is dragged and dropped by the player, a certain number of stars (the exact number as under the diamond) is deleted from the total number of stars and this can be seen under the red squares.

## Front-end/Back-end

Each element on the graphic interface (Fig. 1) is created with OpenGL using Mesh class and the struct VertexFormat. Then each element (star, rectangle, diamond, etc.) is added to the Mesh List so it can appear on the screen.

Then the static elements are rendered on the screen, such as the diamonds in the upper left corner, or the red rectangle in the bottom left corner, or the green grid with rows and columns. The rest of the elements, that need animations for the game to work properly, are transformed into objects using the classes I have implemented in C++ and are stored in vectors to be easier to manipulate and then are also rendered on the screen one by one, depending on the animations each of them will do.

The animations, such as dragging & dropping the diamonds, or the translation of the hexagons to the left, the rotation of the projectiles (stars that are thrown by the diamonds in the grid) are made using operations of rotation, translation and scaling.

The functions I have implemented for player interaction are using the mouse coordinates each moment of time and verifying which part of the mouse is pressed to do the command in accordance. (e.g. when a star that appears randomly on the screen is clicked with the right part of the mouse, it is put in the vector with stars and used to buy diamonds by the player.

In summary, OpenGL is used to visually render the game, and OOP is applied to structure the game into manageable objects. Initialization sets up the initial game state, rendering continuously draws the objects onto the screen, animations bring them to life, interactions respond to user inputs, and randomization adds variety. This combination creates an interactive and visually appealing game experience.