

Universidad Autónoma de San Luis Potosí

Facultad de Ingeniería

Ciencias de la Computación

Ingeniería en Computación

Documentación Proyecto Final

Profesor: Ing. Rodríguez González Omar

Alumno(s): Rodriguez Zavala José María Sebastian

Materia: Graficación por Computadora

Semestre: 2021-2022/II

02/06/2021

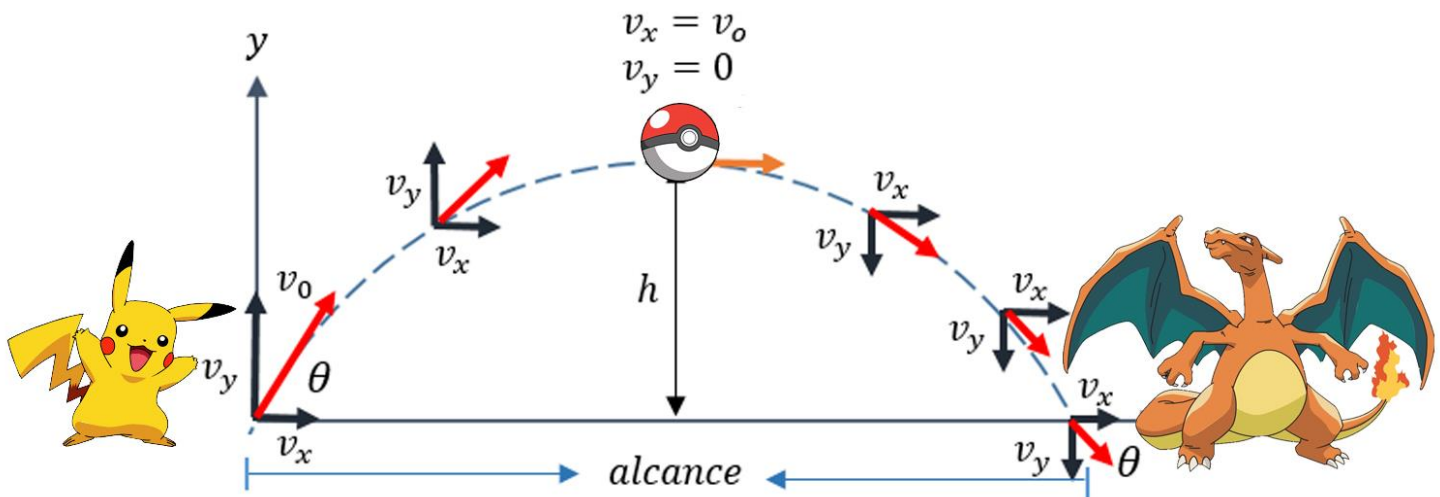
Introducción

Este proyecto esta conformado por diferentes programas, primero se vio el lector de archivos .obj, después se vio la parte de la graficación en OpenGL y al final la parte de mover la cámara con *eye* y *camera*.

El proyecto hace uso de diferentes herramientas tales como Armadillo y GLFW, las cuales son utilizadas para el calculo de normales, uso de vectores, dibujo del objeto y movimiento de la cámara.

El proyecto consiste en una simulación del famoso juego Pokémon GO, donde se simula el lanzamiento de una Pokeball hacía un Pokémon, esto se logra utilizando cálculo de las variables de tiro parabólico, lo cual busca exponer como se muestra un proyectil en el aire al ser lanzado.

Este proyecto está pensado para ser software gratuito principalmente para escuelas mostrando la utilidad y representación del tiro parabólico.



Desarrollo

Empezamos a modelar el proyecto con base al lector de archivos .obj, donde se calculaban las caras, vertices y aristas de un objeto. Posteriormente hacíamos experimentos en clase sobre que tanto se podía hacer y que no referente a la lectura de objetos. Utilizamos la herramienta de *Blender* para poder modelar diferentes objetos y poder reducir sus caras en caso de que sea un objeto pesado.

Una vez terminamos de leer y almacenar un objeto, utilizamos la librería de Armadillo, con la cual pudimos hacer cálculos y ver más a fondo las funciones, tales como lectura de columnas, de renglones, matrices, etc.

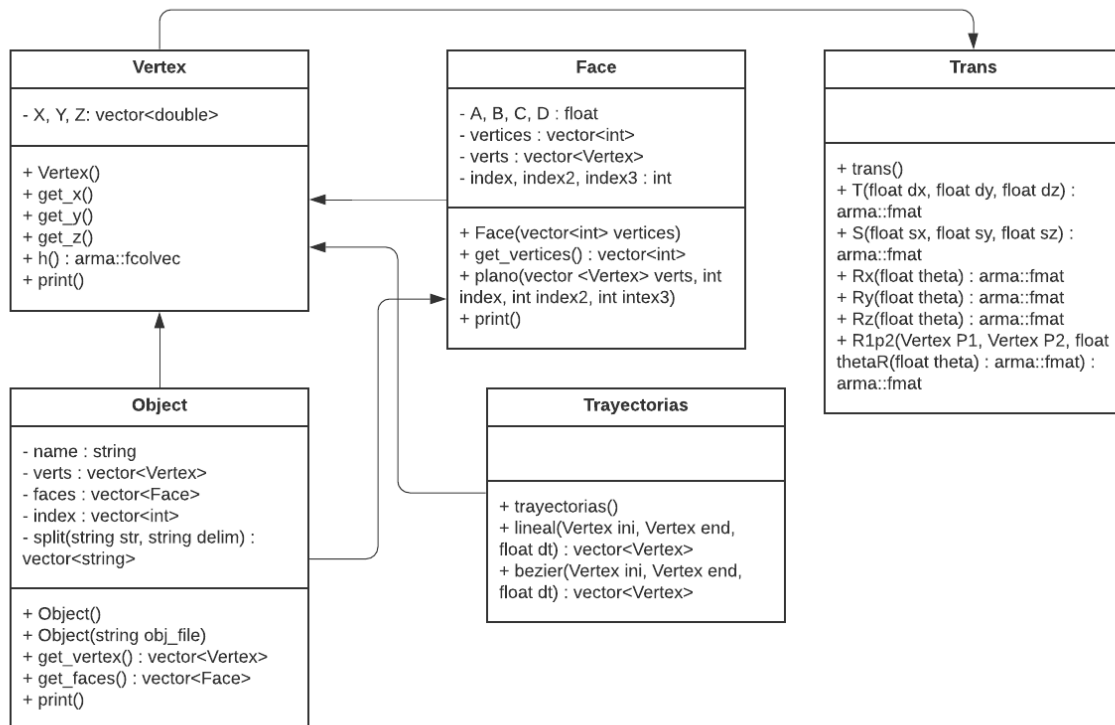
Posteriormente vino la parte gráfica, utilizamos OpenGL y lo primero que vimos fue graficación con base a vectores, después vimos lectura de archivos .obj (que se complementa con el programa inicial) y pudimos graficarlos.

Finalmente, dichos objetos pudimos darle movimiento, o bueno “simular” movimiento calculando la posición de la cámara y del ojo del a misma, pudiendo cambiar la perspectiva de la vista en pantalla.

Diagrama UML de Lector de Objetos

Empezamos el proyecto programando un lector de objetos tipo .obj, este es un formato utilizado para objetos tridimensionales que contienen coordenadas en X, Y, Z.

Los archivos obj pueden ser exportados en *Blender* y en diferentes herramientas de modelado 3D. Esto nos sirve y nos complementa con la librería de Armadillo pudiendo calcular sus vertices, aristas y caras.



El diagrama UML busca representar lo visto e implementado en el lector de archivos obj.

Cómo se puede observar, tenemos las clases principales como lo son **Object**, que este sirve para guardar el objeto a leer, pudiendo almacenar objetos tridimensionales.

Tenemos la clase **Vertex** que es el pilar de este proyecto puesto a que como todos los objetos tienen coordenadas X, Y, Z, lo que hace Vertex es guardar y almacenar dichas coordenadas para su uso.

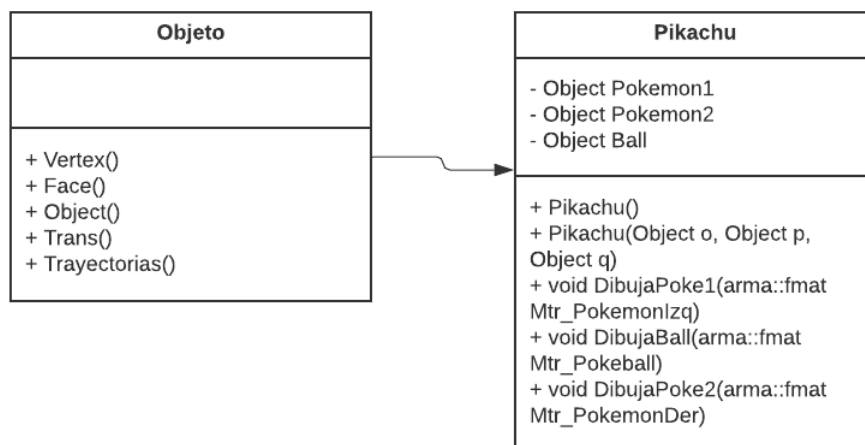
La clase **Face** se complementa a la clase Vertex, ya que esta clase su principal función es juntar los vertices del objeto y guardarlos en un arreglo de vertices, por

eso en cuanto a variables tenemos tres tipos de índice, cada uno una coordenada y el dicho arreglo de vértices

La clase **Trans** de transformación es la clase principal a la hora de dibujar en pantalla los modelos obj, ya que utilizamos principalmente el método **S** que se refiere a *escalado*, ya que para poder tener los objetos con el tamaño deseado podemos escalarlos. Después tenemos el método **T** que nos sirve para *trasladar* el objeto, este método nos sirve para el lanzamiento de la *Pokeball* y el movimiento del primer *Pokémon*.

Finalmente tenemos la clase **Trayectorias**, que nos sirve para calcular trayectorias lineales y curvas de Bézier.

Diagrama de clases del proyecto



El diagrama UML del proyecto busca explicar de manera visual la relación entre clases.

Empezamos por la clase **Objeto** en la cual por comodidad y solución de errores decidí juntar todas las clases anteriormente explicadas del lector de archivos obj en una clase *maestra* para tener fácil acceso durante el código.

Después tenemos la clase **Pikachu** la cual lleva el nombre del *Pokémon* principal de la saga y esta puede guardar tres objetos diferentes, el primer *Pokémon* a escoger, la *Pokeball* y el segundo *Pokémon* el cual se le lanzará la *Pokeball*.

Explicación OpenGL

Existen muchos tipos de fabricantes de tarjetas de video (tales como Intel, Nvidia, AMD, etc.).

Cada compañía diseña su hardware de la manera que ellos piensan que es la correcta. Pero, a veces eso significa que cuando tú estas utilizando dicha tarjeta gráfica las implementaciones son diferentes. Los comandos pueden ser diferentes o la manera de acceder a ellos pueden cambiar. En lo (malos) viejos tiempos un programador tenia que escribir diferentes versiones de su código en el ámbito gráfico para cada tipo de tarjeta de video que lo pudiera correr.

DirectX y OpenGL ofrecieron una solución para eso. Ellos presentaron un estándar para que los programadores lo utilicen. Si usted quiere dibujar un cubo y cubrirlo con una textura, entonces usted solo tiene que escribir el código una vez.

Los *frameworks gráficos* de OpenGL traducirán su código a cualquier tipo de hardware que quiera referirse.

Ellos también tienen la ventaja de crear una condición de que si una parte del hardware no soporta cierta característica entonces este será ejecutado en software (no en hardware), aunque, será más lento, pero funcionará.

Dicho esto, ¿qué función tiene OpenGL? Muy sencillo, tomemos como ejemplo que yo quiero hacer un cubo 3d. ¿Tengo que leerme toda la documentación de todas las tarjetas gráficas que existen? No, para eso existe OpenGL. Hacemos el cubo 3d y se traduce, puesto a que la mayor parte de las tarjetas gráficas que existen en el mercado entienden lo que es OpenGL.

Podemos decir que OpenGL consiste en una serie de librerías y rutinas de clases por lo que, OpenGL es una API de bajo nivel que proporciona una interfaz de hardware de gráficos.

Explicación del proyecto

El proyecto tomó como referencia la popular franquicia **Pokémon**, específicamente la saga **Let's Go**.

Ya que en dicho juego me llamó la atención el lanzamiento de *Pokeball* que cuenta con un tiro parabólico. Este tiro consiste en el movimiento de un objeto donde su movimiento no se basa solamente en el eje X o en el eje Y, sino que lo hace con ambos ejes para simular un movimiento. Para calcular el tiempo lo que se hizo fue despejar la fórmula:

$$t = \frac{2 * v * \text{sen}(\theta)}{g}$$

Para calcular la distancia máxima se despejó la fórmula:

$$X_{max} = \frac{Vo^2 * \text{sen}(\theta)}{g}$$

Lo que hace el proyecto es simular el lanzamiento de una *Pokeball* al capturar un *Pokémon*, lo que se hace es dar un ángulo dado de 45° con una velocidad inicial de 40 m/s.

Una vez entendemos esto, necesitamos los puntos de la distancia máxima en X, la distancia máxima en Y y el tiempo que se tardó en ser disparado el proyectil. Para obtener estos puntos se utilizó la fórmula en X:

$$x = ((Vel * \cos(\theta)) * t)$$

Y para obtener los puntos en Y se utilizó la fórmula:

$$(Vel * \text{sen}(\theta) * t) - \left(\frac{t^2 * g}{2} \right)$$

Recordemos que tenemos que escalar los resultados para el dibujado de las trayectorias.

Problemas/Experimentos

Problemas:

Existieron muchos problemas a la hora de realizar este proyecto.

Principalmente, trabajé con Windows en un 80% del semestre, cabe destacar que cursar esta materia en Windows es difícil, porque se tiene que hacer muchos pasos y nada te asegura que compile tu programa.

Mi principal problema fue que a la hora de instalar Armadillo no podía lograr compilar el programa, reinstalé muchas veces Armadillo y no pude hacerlo funcionar.

En el lector de archivos obj tuve como problema al momento de guardar los vertices y separarlos, dependiendo de por que letra empieza, si g, f, o, etc.

Pude solucionar estos problemas al cambiarme a Linux en una máquina virtual, instalar Armadillo y OpenGL fue sencillo y en cuestión de menos de 7 pasos para cada uno.

En cuanto a problemas dentro del entorno Linux fueron que no podía compilar directamente con Visual Studio Code, me dejaba compilar y correr el programa de *Hello World!*. Pero a la hora de compilar el lector de archivos obj o el test compilador de armadillo no me daba permiso, buscando en foros de internet (como *StackOverflow*), me recomendaron utilizar la instrucción de ***chmod 777*** y si podía compilar, solamente que no podía correr el programa porque me daba errores dentro del archivo ejecutable.

Pude solucionar este problema compilando con el comando ***make*** y pude ejecutar el archivo normalmente.

Tuve otro problema a la hora de usar las librerías creadas y los archivos .hpp. Estando dentro de la carpeta *include* no podía compilar y me daba errores. Realmente no se por que si tenia todo bien, solamente pude solucionar este problema creando un archivo .hpp maestro y un archivo .cpp maestro y dejándolo en la carpeta principal.

En cuanto al código del proyecto, el primer problema que tuve fue a la hora de calcular los puntos, ya que intenté aplicar la fórmula, pero Visual Code me daba errores, pude solucionarlo creando una función para la fórmula.

El segundo problema fue cuando al utilizar las fórmulas para calcular el seno y el coseno, éstas eran calculadas en radianes. Se pudo corregir convirtiendo el ángulo multiplicándolo por Pi y dividirlo entre 180.

El tercer problema fue a la hora de leer los modelos, ya que directamente en el código escribía el modelo a cargar, pero a la hora de leer la cadena de los *Pokémon*

tuve problemas para concatenar **obj/** y **.obj** ya que estos me daban basura. Probé con limpiar el buffer de lectura y aumentar el tamaño de las cadenas.

El cuarto problema fue cuando se guardan los colores para dibujar a los *Pokémon* dependiendo de que modelo se elige. Tenía la función dentro de un **do while** y la condición era que mientras que la lectura sea diferente de *null* y sea igual al nombre de un *Pokémon*, pero me seguía arrojando basura. Solucioné el problema mandando por referencia variables que se usarán para pintar.

Experimentos:

Por la parte de experimentos, jugué mucho con los ángulos, las velocidades y los modelos.

En *Blender* tuve reducir las caras de las figuras y escalarlas de manera que a la hora de leer el archivo no tarde mucho.

Prueba 1:

Velocidad inicial: 0 m/s

Ángulo: 30



Como podemos ver, al no tener velocidad inicial, la *Pokeball* no se mueve ni genera trayectoria.

Prueba 2:

Velocidad inicial: 30 m/s

Ángulo: 60

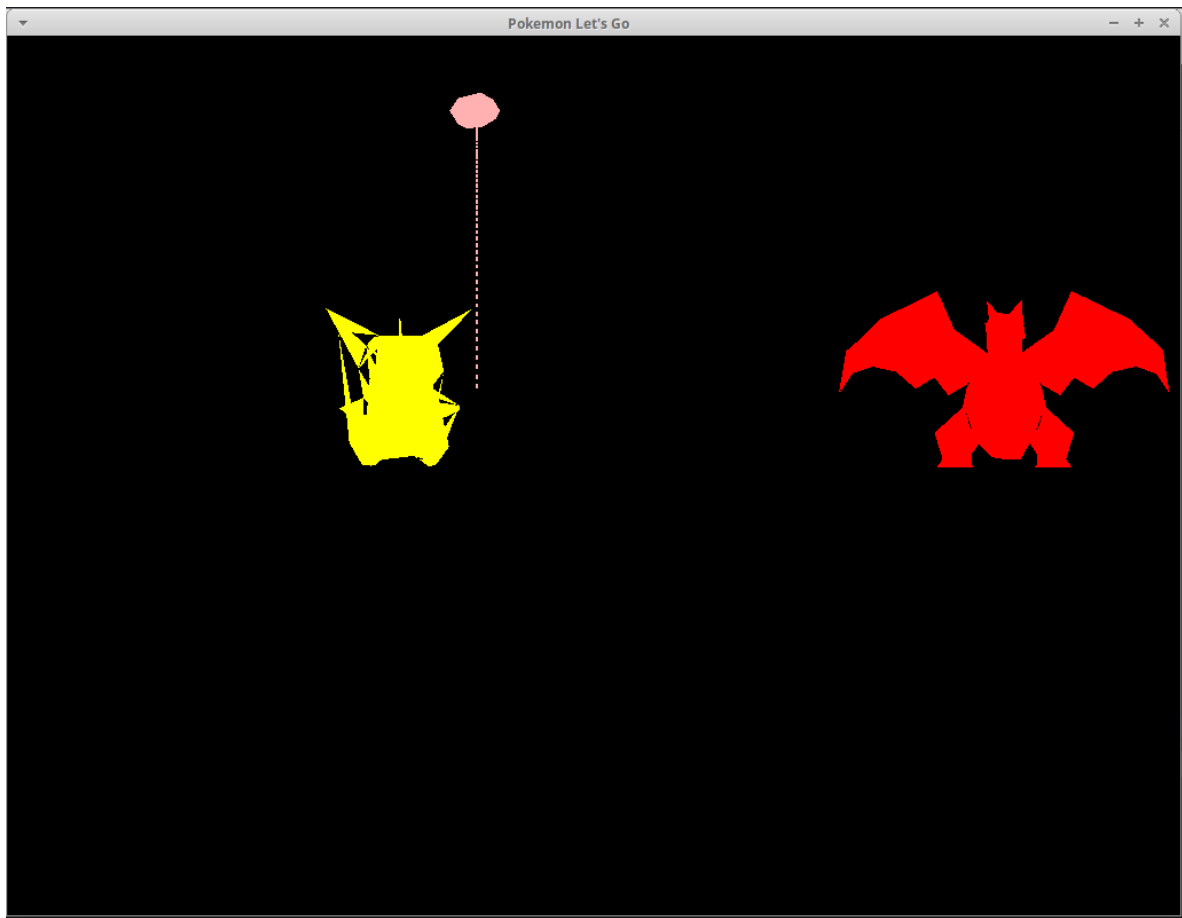


Podemos observar que teniendo un ángulo mayor pero una velocidad menor no alcanza a capturar a **Charizard**

Prueba 3:

Velocidad inicial: 50 m/s

Ángulo: 90

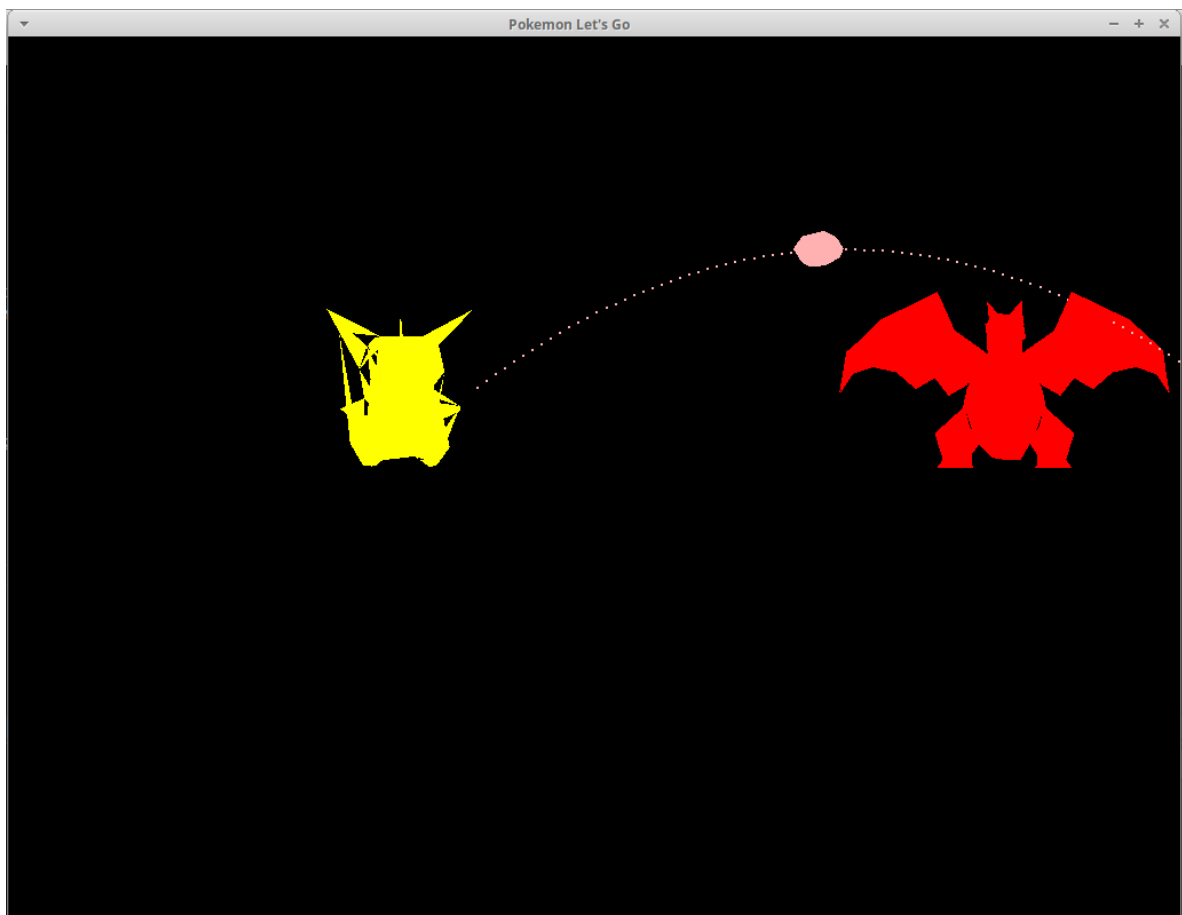


Se puede observar que, a pesar de la velocidad inicial, aunque sea el ángulo de 90 grados, siempre va a ir en línea recta sin movimiento en el eje X.

Prueba 4:

Velocidad inicial: 50 m/s

Ángulo: 45



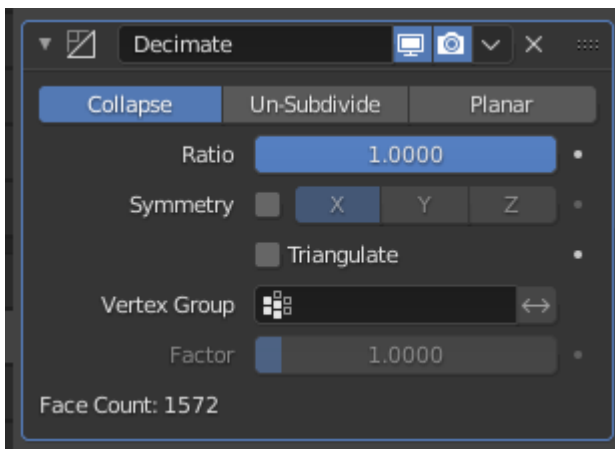
Se puede observar que se acerca mucho al ángulo y la velocidad inicial correcta.

Blender:

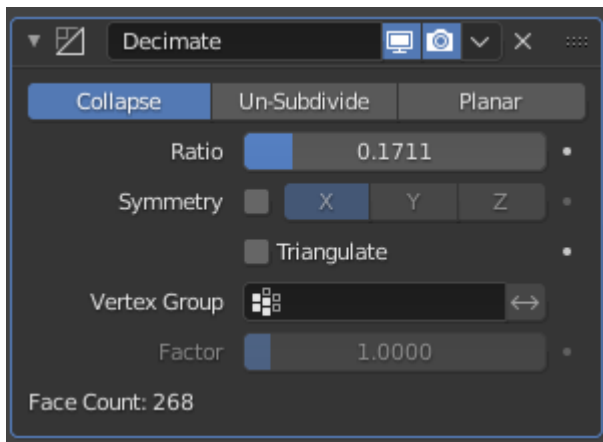
En el modelo de Charizard podemos ver el .obj original



Pero en propiedades del objeto se puede ver la cantidad de caras que tiene que en su totalidad son 1572, cosa que será muy pesado para leer en el lector de archivos obj.



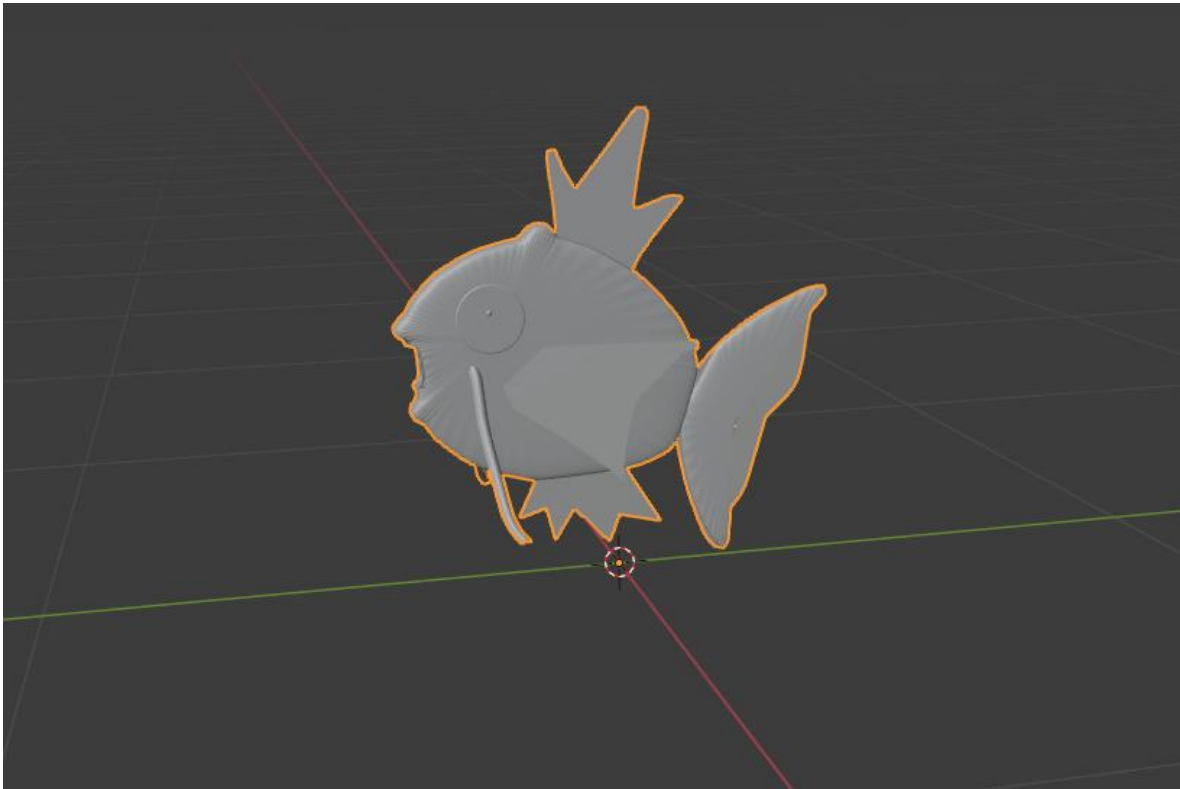
Lo que hice fue reducir el **Ratio** para disminuir la cantidad de caras de la figura.



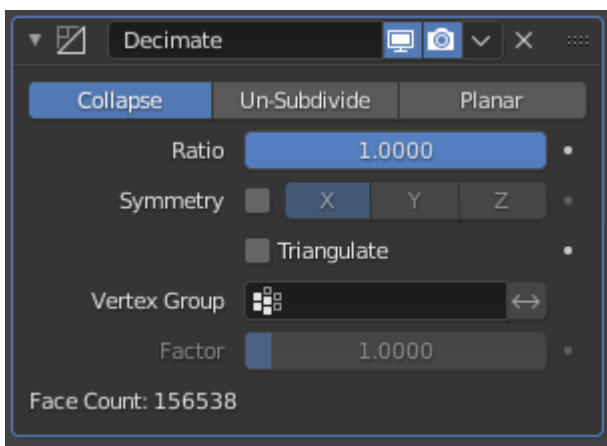
268 caras es algo que el lector de archivos puede leer con facilidad. Pero el modelo de Charizard perdió a su vez caras, aunque sigue siendo visible y entendible de que *Pokémon* se trata.



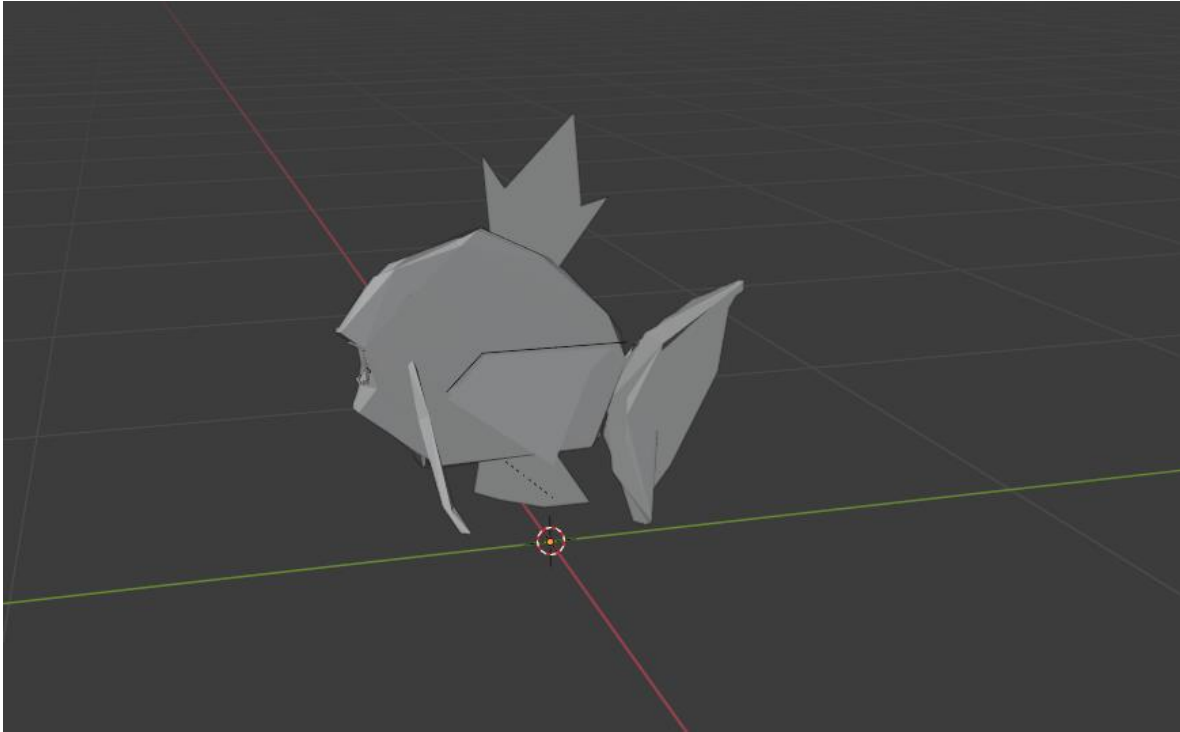
Lo mismo realicé con todos los modelos que utilicé. Algunos archivos tenían muchas caras, un ejemplo es Magikarp.



Que tiene una cantidad de caras de 156,538.



Pude reducir las caras dando un **Ratio** de 0.001, quedando como una figura visible y entendible.



Conclusión

Como conclusión podemos observar que con las librerías de Armadillo y GLFW podemos entender mejor como se trabaja y se entiende dichas librerías complementadas al proyecto.

Las clases del proyecto fueron programas y creadas para uso específico del desarrollo de este proyecto, se utilizaron todas las funciones creadas.

El proyecto se realizó y se concluyó de manera adecuada al temario. Se aplicaron los conocimientos adquiridos a lo largo del semestre, junto con las fórmulas de tiro parabólico.

Podemos ver este proyecto como un punto de partida a la graficación por computadora, el uso de librerías existentes como OpenGL y no quebrarnos la cabeza intentando crear algo que podemos utilizar de una herramienta como GLFW.

La materia de Graficación por Computadora tiene muchas matemáticas y temas que se deben de entender antes de cursar la materia por lo que se recomienda llevar la materia de Calculo B, Calculo A y Algebra B para entender a lo que se refiere por el uso de matrices y multiplicación de matrices.

Comentarios Finales

Finalmente, como comentario final, estuve muy preocupado porque al inicio de semestre no sabia como llevar la materia, pero con lo explicado por el profesor y lo que se vio en el temario me llamó mucho la atención, me sorprendió y me interesé por los temas vistos.

Me terminó gustando más el entorno de Linux porque fue sencillo la implementación de Armadillo y OpenGL.

Tuve muchos problemas con la materia, pero consultándolo con personas que me pudieron ayudar a entender temas se pudo solucionar.

Referencias

Dave Shreiner, Graham Sellers, John Kessenich, Bill Licea-Kane, "The Khronos OpenGL ARB Working Group", 2013.

James D.,. Foley, Andries van Dam, Steven K. Feiner and John F. Hughes, ADDISON.WESLEY PUBLISHING COMPANY,1990

José L. Fernández, "Movimiento Parabólico", <https://www.fisicalab.com/apartado/movimiento-parabolico#contenidos>, 2017

[Cluster_1]. (2013, Febrero 06). ELI5: DirectX, OpenGL, etc. [Online forum post]. Retrieved from https://www.reddit.com/r/explainlikeimfive/comments/180vv4/eli5_directx_opengl_e tc/

Manual de usuario

Introducción

El simulador de *Pokémon Let's GO* es programa que consiste en calcular tiros parabólicos con base a un ángulo y velocidad dada. También simula el lanzamiento en tiro parabólico de una *pokeball*.

Requerimientos

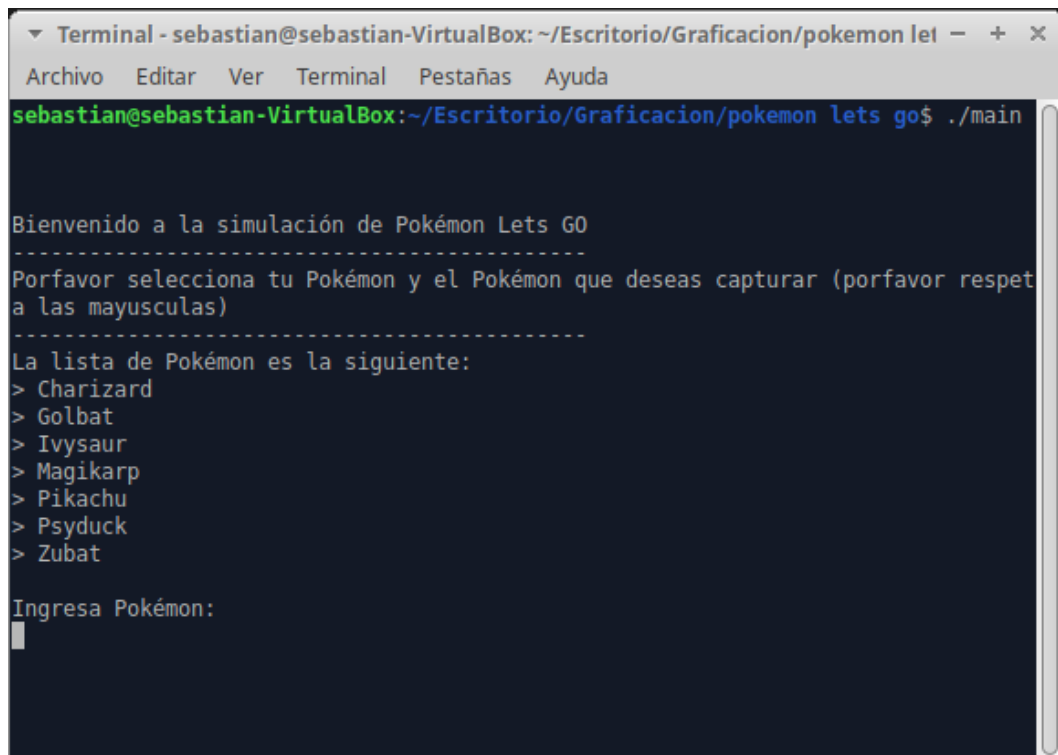
El programa necesita dos librerías importantes.

- Armadillo
- GLFW

Para poder instalar estas librerías es necesario consultarlo en su manual correspondiente para Windows, Linux o MacOS.

Ejecución

El programa se corre con el comando ***./main*** y a continuación te pide ingresar tu *Pokémon* y el *Pokémon* deseado a capturar.

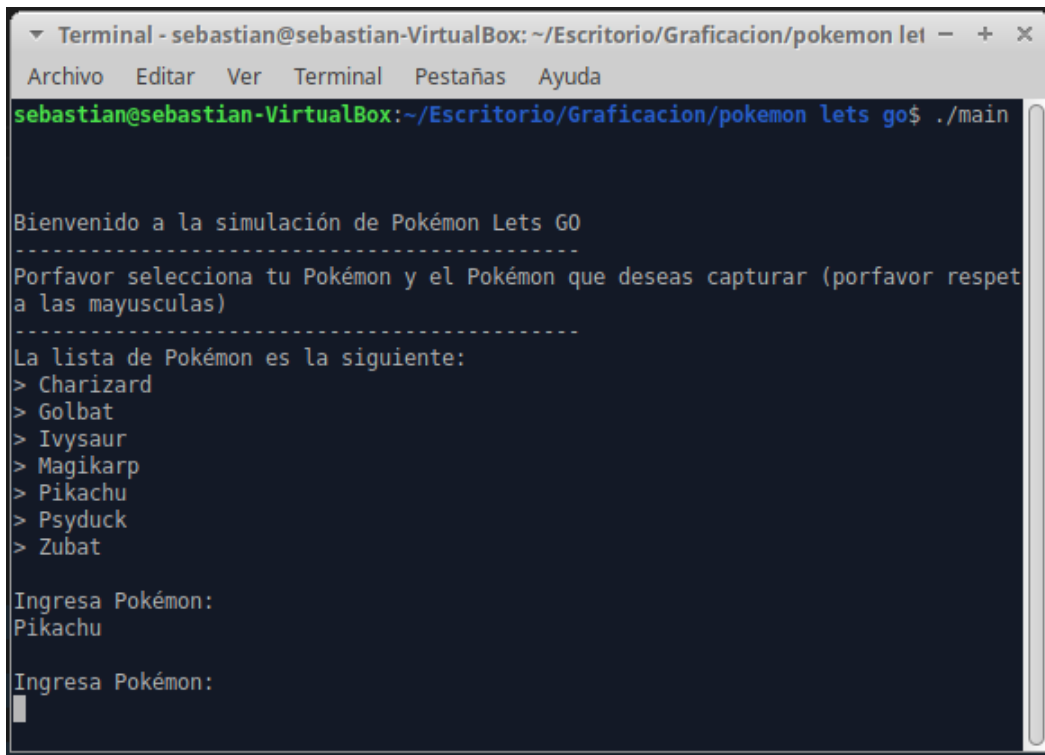


```
▼ Terminal - sebastian@sebastian-VirtualBox: ~/Escritorio/Graficacion/pokemon let - + x
Archivo  Editar  Ver    Terminal  Pestañas  Ayuda
sebastian@sebastian-VirtualBox:~/Escritorio/Graficacion/pokemon lets go$ ./main

Bienvenido a la simulación de Pokémon Lets GO
-----
Porfavor selecciona tu Pokémon y el Pokémon que deseas capturar (porfavor respet
a las mayusculas)
-----
La lista de Pokémon es la siguiente:
> Charizard
> Golbat
> Ivysaur
> Magikarp
> Pikachu
> Psyduck
> Zubat

Ingresa Pokémon:
█
```

Al ingresar el primer *Pokémon* te pide el segundo.



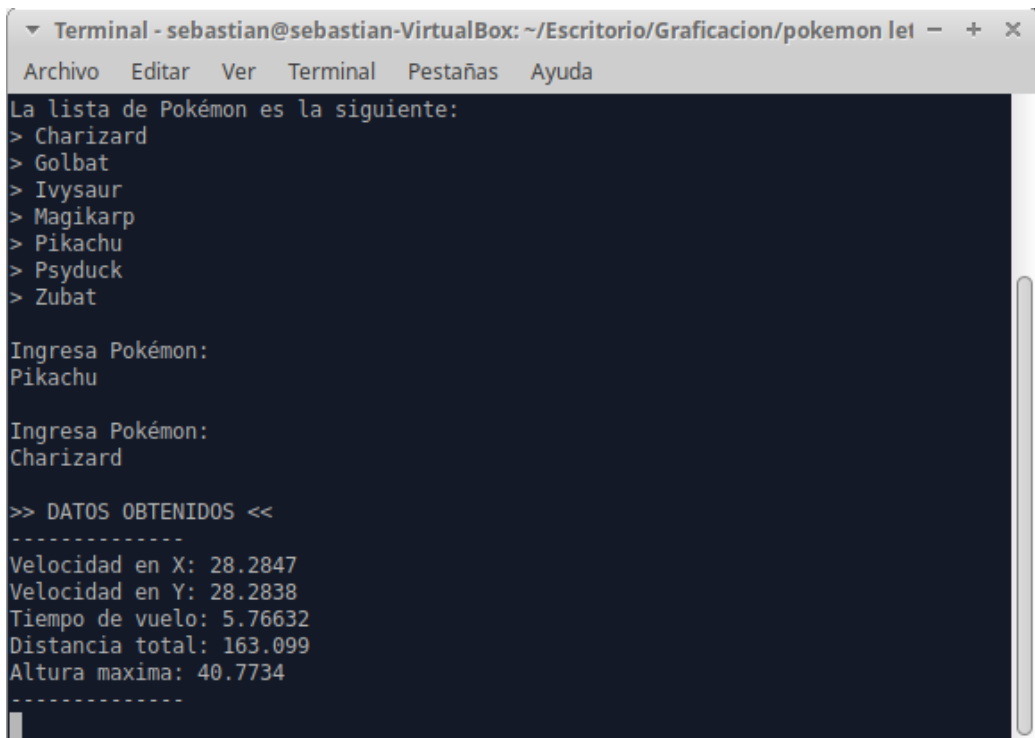
```
Terminal - sebastian@sebastian-VirtualBox: ~/Escritorio/Graficacion/pokemon lets go - + x
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
sebastian@sebastian-VirtualBox:~/Escritorio/Graficacion/pokemon lets go$ ./main

Bienvenido a la simulación de Pokémon Lets GO
-----
Porfavor selecciona tu Pokémon y el Pokémon que deseas capturar (porfavor respet
a las mayusculas)
-----
La lista de Pokémon es la siguiente:
> Charizard
> Golbat
> Ivysaur
> Magikarp
> Pikachu
> Psyduck
> Zubat

Ingresa Pokémon:
Pikachu

Ingresa Pokémon:
█
```

Una vez ingresado el segundo, en consola te muestra los datos obtenidos como lo son la velocidad en X, la velocidad en Y, el tiempo de vuelo, la distancia total y la altura máxima.



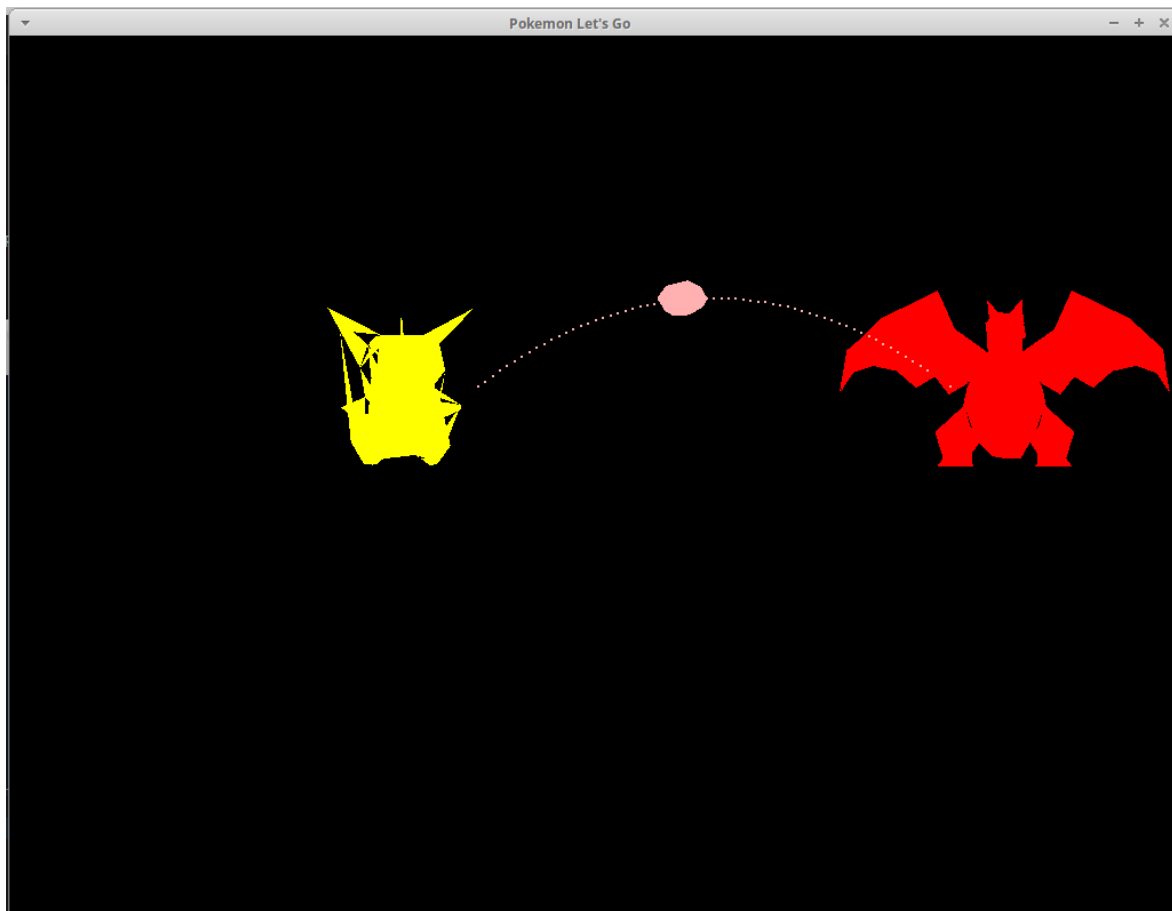
```
Terminal - sebastian@sebastian-VirtualBox: ~/Escritorio/Graficacion/pokemon lets go - + x
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
La lista de Pokémon es la siguiente:
> Charizard
> Golbat
> Ivysaur
> Magikarp
> Pikachu
> Psyduck
> Zubat

Ingresa Pokémon:
Pikachu

Ingresa Pokémon:
Charizard

>> DATOS OBTENIDOS <<
-----
Velocidad en X: 28.2847
Velocidad en Y: 28.2838
Tiempo de vuelo: 5.76632
Distancia total: 163.099
Altura maxima: 40.7734
-----
█
```

Una vez mostrado los datos obtenidos, se genera la ventana de GLFW donde se cargan los archivos obj ingresados.



Se busca que el simulador sea de ayuda para escuelas mostrando el funcionamiento del tiro parabólico.