HOOK HIT

El juego consta de un personaje el cual debe derribar ciertos objetivos que se encuentran realizando diferentes movimientos los cuales en su trayectoria pueden llegar a causar daño al personaje principal, sin embargo, algunos de estos objetivos pueden beneficiar su partida o incluso llegar a perjudicarla; El personaje a su vez deberá derribar los objetivos con un gancho que se extiende a lo largo del eje y de la pantalla. Además, a lo largo de los niveles el personaje se va enfrentando diferentes adversidades que ponen a prueba sus reflejos a modo de mejorar en cada partida jugada.

MOTIVACIÓN

La motivación para este videojuego es poner a prueba todo lo aprendido en clase y laboratorio para desarrollar un videojuego completo y sin errores, esto de cierto modo también nos reta a la investigación y correcta documentación para con el desarrollo de cualquier actividad. Desde que iniciamos a programar, soñamos en tener los conocimientos necesarios para crear diversos programas útiles para la sociedad. El constante conocimiento adquirido cuando programamos hace que cada código sea bueno para practicar y aprender nuevas cosas ya que si tenemos dudas en algo podemos dirigirnos a la documentación.

DEFINICIÓN DE OBJETOS

Personaje: este tiene la capacidad de moverse hacia los lados y además puede saltar. A modo de defensa de los diferentes objetivos, este puede disparar un proyectil en dirección perpendicular al piso hacia arriba, este cuenta con un contador de vidas que varía en función del daño recibido o de las ventajas adquiridas.

class **personaje**

{

private:

    float posX;

    float posY;

    float masa;

```
        float VX;

        float VY;

        float angulo;

        float AX;

        float AY;

        float G;

        float K;//resistencia del aire

        float e;//coeficiente de restitución

        float V;//vector de velocidad

        float dt;//variación del tiempo

public:

        personaje(float posX_,float posY_,float velX_,float velY_,float mass,float radio,float
k_,float e_);

        void actualizar();

        float getPX() const;

        float getPY() const;

        float getMass() const;

        float getR() const;

        float getVX() const;

        float getVY() const;

        float getE() const;

        void set_vel(float vx,float vy, float px,float py);

}
```

Proyectil: estos son generados por el Personaje y su función principal es acabar con los diferentes objetivos y una vez ha logrado su cometido esta desaparece. Estos proyectiles son ilimitados, sin embargo, estos solo pueden ser generados una vez que se ha alcanzado el techo.

class **pelota**

{

private:

float posX;

float posY;

float Velocidad;

public slots:

void movimiento();

public:

proyectil(float posx_,float posy_,float masa_,float velocidad);

float **getposX**() const;

float **getposY**() const;

float **setVelocidad**(float vel);

}

Pelota: estas son un tipo de objetivo que pueden ser destruidas por el usuario con un disparo del personaje principal, su característica principal es que rebotan de forma parabólica y en caso de existir una colisión con el personaje principal este pierde una vida.

class **pelota**

{

private:

float posX;

float posY;

float masa;

float radio;

float VX;

float VY;

float angulo;

float AX;

```
        float AY;

        float G;

        float K;//resistencia del aire

        float e;//coeficiente de restitución

        float V;//vector de velocidad

        float dt;//variación del tiempo

public:

        pelota(float posX_,float posY_,float velX_,float velY_,float mass,float radio,float
k_,float e_);

        void actualizar();

        float getPX() const;

        float getPY() const;

        float getMass() const;

        float getR() const;

        float getVX() const;

        float getVY() const;

        float getE() const;

        void set_vel(float vx,float vy, float px,float py);

}
```

Globo: este es un tipo de objetivo que pueden ser destruido por el usuario al general un proyectil, su movimiento es rectilíneo sobre el eje x y su función principal es que al ser destruido este puede soltar una ventaja para el jugador o una desventaja

```
class globo

{

private:

        float posX;

        float posY;

    float Velocidad;
```

public slots:

    void movimiento();

public:

  globo(float posx_,float posy_, float velocidad);

    float **getposX**() const;

    float **getposY**() const;

}

Plataforma: es el elemento sobre el cual el personaje puede desplazarse, los elementos como la pelota y el globo pueden chocar con esta que genera una colisión elástica

class **plataforma**

{

private:

    float posX;

    float posY;

    float ancho;

  float alto;

public:

  plataforma(float posx_,float posy_, float ancho_, float alto_);

}

Pared: restringe el paso del personaje y objetivos momentáneamente hasta que el personaje haya terminado con una misión en específico

class **pared**

{

private:

    float posX;

    float posY;

    float ancho;

```
    float alto;
public:
    pared(float posx_,float posy_, float ancho_, float alto_);
        }
```

vida: varía su estado en función de daño recibido o por ventajas que recoge el personaje

```
class vida: public QGraphicsTextItem
{
private:
        int vida;
public:
        Health(QGraphicsItem * parent = 0);
        void decrease();
        int getHealth();
};
```

Puntaje: se encarga de contabilizar cada disparo exitoso por el personaje a sus objetivos

Potenciador: este puede mejorar aspectos del jugador en función de facilitar sus funciones

```
class puntaje: public QGraphicsTextItem
{
private:
        int Puntaje;
public:
        puntaje(QGraphicsItem * parent = 0);
        void increase();
        int getPuntaje();
};
```

Ventaja: esta se encarga de mejorar ciertos atributos del personaje y su disparo como lo puede ser, la velocidad del personaje, la velocidad del disparo, el radio del disparo etc.

class **ventaja**

{

private:

float posX;

float posY;

float velocidad;

int numero_ventaja;

public slots:

void **movimiento**();

public:

ventaja(float posx_,float posy_, float velocidad, int numero);

float **getposX**() const;

float **getposY**() const;

}

Desventaja: esta se encarga de complicar ciertos atributos del personaje, como la velocidad del personaje.

class **desventaja:**

{

private:

float posX;

float posY;

float velocidad;

int numero_desventaja;

public slots:

void **movimiento**();

public:

desventaja(float posx_,float posy_, float velocidad, int numero);

float **getposX**() const;

float **getposY**() const;

}

Archivo: se encarga de guardar datos de partidas jugadas como puntajes con sus respectivos usuarios y niveles jugados.

class: **archivo**

{

private:

string nombre;

int puntaje;

int nivel;

public:

archivo(string nombre_,int puntaje_,int nivel_);

string **getNombre()** const**;**

METODOLOGÍA

Idea general del videojuego: Plantear una idea de las funcionalidades de cada objeto en su punto más básico, con el fin de general un sistema de mecánicas que puedan ser aprovechadas a largo plazo.

Creación de repositorio: Crear un sistema de guardado para el continuo desarrollo del videojuego, para que de ese modo se logre recuperar copias previas en caso de afectar negativamente el videojuego.

Objetos y escenas del juego: Crear los objetos que se utilizarán a lo largo del videojuego con su respectiva funcionalidad básica.

Idea de las funciones de cada objeto: Desarrollar las ideas sobre el movimiento o físicas que lleva cada objeto y cómo contribuye en la dinámica del videojuego.

Funciones básicas del juego: Implementar las mecánicas y sistemas dentro del videojuego con una idea más clara del comportamiento de todo el sistema.

Funciones de carga y guardado de archivo: Implementar la carga y descarga de archivos para guardar el proceso de un usuario en el juego.

Funciones avanzadas del juego: Plantear funciones alternas de algunos objetos o niveles adicionales en el videojuego para contribuir con la noción de dificultad.

Multijugador: Implementar las funciones de otro personaje.

Prueba del programa: A lo largo de la construcción del código se harán pruebas o se ejecutará el videojuego para observar y analizar el avance del mismo.

Entrega del programa: Una vez se considere al videojuego con un correcto funcionamiento, se procede a dar por concluido el desarrollo del videojuego y como paso siguiente se realiza la respectiva entrega del proyecto.

CRONOGRAMA

Diciembre 21-27	-Modelación del movimiento del personaje principal con sus respectivas funciones y acciones tales como disparar y saltar. -Creación del movimiento de las pelotas y a su vez el daño que causan estas al jugador.
Diciembre 28 Enero 3	-Creación de paredes, plataformas y la interacción del personaje y las pelotas con estos. -Creación de las mecánicas del proyectil. -Interacción entre los disparos del personaje y las pelotas.
Enero 4-10	-Creación completa de las mecánicas de las pelotas con sus respectivas interacciones con los demás objetos. -Creación de un HUD (Heads-Up Display (Presentación de Información)), que permita visualizar las vidas, puntuación y ventajas que adquiere el personaje principal
Enero 11-17	-Modelamiento de los globos como su movimiento y sus respectivas interacciones con respecto a los disparos del personaje principal.

Enero 18-24	-Creación del sistema de guardar partidas, cargar partidas y crear partidas.
Enero 25 - 31	-Creación del modo multijugador. -Mejoramiento de aspectos estéticos del videojuego, tanto en visuales como auditivos.

Integrantes:

Jorge Sebastián Arroyo Estrada, grupo 7

Javier José León Rodriguez, grupo 6