Juego Final

Juego en Qt informática II

FRANKLIN ANDRES ANACONA

Departamento de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia Medellín

Motivación.

- Teniendo las habilidades adquiridas a lo largo del curso mediante la programación en C++ y la implementación del paradigma de programación orientada a objetos en el entorno de desarrollo Qt Creator.
- Experimentar la capacidad y el control para crear objetos reales y simular sistemas físicos al crear un videojuego de manera independiente.
- Utilizar las destrezas adquiridas durante el curso para abordar proyectos futuros que impulsen el crecimiento personal y profesional de los involucrados en el desarrollo del videojuego.
- Diseñar y desarrollar un juego atractivo que destaque por su estética y jugabilidad

Objetivos.

- Diseñar un videojuego con un enfoque original, que resalte por su creatividad.
- Aplicar los conocimientos adquiridos en el curso de Informática II durante el semestre en el desarrollo del videojuego.
- Implementar sistemas físicos en el juego.
- Entregar un videojuego con jugabilidad sólida, un diseño visual atractivo y la introducción de elementos originales en el desarrollo del videojuego.

- Manejar el desarrollo del juego en repositorio.

Concepto y descripción del juego.

Nombre del juego: **Gallos Bros.**

Concepto:

Gallos Bros es un juego sencillo que combina elementos de Mario Bros con las emocionantes peleas de gallos. Es un juego de supervivencia y combate en cual debes sobrevivir en un escenario similar al de Mario Bros donde debes evitar ataques de pelotas de caen como pelotas de goma y proyectiles con movimiento rectilíneo uniforme . Superado el escenario anterior cambiaras a otra escena donde deberás derrotar a un gallo enemigo similar a tu personaje este combate será similar a las peleas de gallos.

Descripción:

La jugabilidad del juego será en una vista en 2D la travesía será de pasar por dos eventos para ganar. El juego empezara escogiendo tu gallo. cada raza tendrá ciertas características apariencia, salto, ataque, vida y velocidad.

En el **primer escenario** será un estilo de vista y jugabilidad de Mario Bros en el cual el gallo podrá moverse izquierda a derecha y saltar. El gallo estará en el lado izquierdo de la pantalla y avanzara eliminando ciertos enemigos que le aparecerán. La manera de eliminar a los enemigos será disparando proyectiles.

también del lado derecho de la pantalla saldrá proyectiles que el gallo debe esquivar para proteger su vida y de la esquina superior derecha caerán bolas la cuales caerán con rebote elástico (Similar a cuando se deja caer una pelota de goma) las cuales el gallo deberá esquivar. El juego pasara al siguiente escenario cuando resista por 2 minutos este primer escenario sin ser afectado toda su vida. Ver figura1

Figura 1

segundo escenario ya superado el primer escenario se mostrará el segundo escenario en el cual el juego se basará en las peleas de gallos. Como en las peleas de gallos habrá una escena similar a una arena de pelea la vista también será 2D. ahora tu gallo puede golpear cuando está a cierta distancia además podrá disparar, pero con la condición de que debe recoger proyectiles que cada cierto tiempo estarán disponibles en la arena. en la arena tendremos que derrotar a otro gallo el cual puede golpear cuando está a cierta distancia además este puede lanzar fuego el cual lo dispara de manera directa (como una bala) y disparo con movimiento parabólico. Ver figura2

El segundo gallo tendrá una cierta vida la cual cuando más daño le hagas aumentará su velocidad y el daño de sus ataques. La prueba se supera cuando acabes con toda la vida del gallo enemigo.

Figura 2

Modelamiento y Definición De 0bjetos.

Modelamiento

El juego empezará escogiendo tu gallo aquí se utilizará un menú. Seguido crear un mapa el cual estará constituido por rectángulos que harán la simulación de paredes y obstáculos.

Ya en el juego le debemos dar ciertas características al personaje velocidad, ataque, vida y su posición inicial que empezara en la parte inferior izquierda de la pantalla. Con características asignadas al personaje le deberos dar diferentes capacidades accionarias en el juego las cuales serán moverse hacia izquierda y derecha, realizar ataques con proyectiles y saltar que esta regido por la gravedad.

Para los ataques generados en el juego hacia el jugador se utilizarán modelos físicos para el lanzamiento de proyectiles con tiro parabólico y el lanzamiento de bolas que caerán con rebote elástico.

Listado de objetos a utilizar:

- Clase gallo: esta será para jugador principal como para el Bot mecanizado del gallo enemigo de la escena 2
- Clase pared: esta servirá en la escena 1 y 2 para delimitar nuestra escena además de servir para la jugabilidad en el enfrentamiento (cubrirnos de ataques o recoger proyectiles que estarán en parte alta de la pantalla)
- Clase bala: esta se usará para generar los disparos en las escenas 1 y 2 para el lanzamiento de proyectiles tanto para el uso de jugador principal como para gallo enemigo y en la primera escena se autogenerarán estos disparos para intentar alcanzar al jugador.
- Clase bola De Goma: para las bolas que caen con rebote elástico en la escena 1 se usara esta clase
- Clase fuego: esta clase será para el uso del Bot de disparos parabólico la cual tendrán una vista de fuego

- Clase iniciar Sección: registrar usuarios en documento

Clase Gallo

Este tendrá ciertos atributos que serán:

- **Velocidad:** su velocidad de movimiento en la pantalla
- **Salto:** su capacidad de alcance al realizar un salto
- **Vida:** su cantidad de resistencia a ataques
- **Posx**: Posición en x
- **Posy:** Posición en y
- **Tamaño:** será su longitud de ancho y alto al manejar su imagen en un rectángulo
- **Tipo de raza:** este será un entero el cual indica que tipo de raza para el jugador con la cual tendrá ciertos atributos y apariencia

Para los métodos necesitamos:

- moverJugador: la cual permitirá moverse en direcciones arriba, abajo, izquierda, derecha y además realizar saltos
- disparar: realizar disparas tanto para el jugador principal como para el enemigo
- incinerar: este solo será del uso del enemigo final la cual lo que hace es lanzar fuego con un movimiento parabólico
- gallo(): constructor
- ~gallo(): destructor
- Setpox: dar posición en x
- Setposy: dar posición en y
- Getposx: obtener posición en x
- Getposy: obtener posición en y
- Funciones graficas de Qt

Cuando se usa el constructor también las funciones graficas para darle apariencia distinta a el personaje para el cual cada apariencia tendrá distintos atributos. Las apariencias o razas disponibles serian:

- **GUACHARAKO**

- VELOCIDAD:9
- VIDA :10
- ATAQUE:6
- SALTO: 6

- **COLORADO**

- VELOCIDAD:10
- VIDA: 7
- ATAQUE:8
- SALTO: 6

- **FERRERO ROJO**

- VELOCIDAD: 8
- VIDA:8
- ATAQUE:8
- SALTO: 7

Clase Pared

Este tendrá ciertos atributos que serán:

- **Posx**: Posición en x
- **Posy:** Posición en y
- **Tamaño:** será su longitud de ancho y alto al manejar su imagen en un rectángulo

Para los métodos necesitamos:

- Setpox: dar posición en x
- Setposy: dar posición en y
- Getposx: obtener posición en x
- Getposy: obtener posición en y
- Funciones grafica de Qt

Los métodos relacionados con las posiciones de (x) y de (y) los usaremos para genera cuadros aleatorios en el mapa y para detectar colisiones con ellos.

Clase Bala

Este tendrá ciertos atributos que serán:

- **Velocidad:** su velocidad de movimiento en la pantalla aumentara con el avance del juego
- **Posx**: Posición en x
- **Posy:** Posición en y
- **Tamaño:** será su longitud de ancho y alto al manejar su imagen en un rectángulo
- **Daño:** esta cantidad aumentara a medida que avance el juego

Para los métodos necesitamos:

- disparar: esta función es para los disparos que genera el programa
- posición: donde se graficará esto para simular los movimientos
- Setpox: dar posición en x
- Setposy: dar posición en y
- Getposx: obtener posición en x
- Getposy: obtener posición en y
- Mover: función para dar dirección en la que se va a mover la bala
- Funciones graficas de Qt

Clase Fuego

Este tendrá ciertos atributos que serán se tendrá en cuenta el movimiento parabólico:

- **Velocidad Inicial:** su velocidad de movimiento en la pantalla
- **Angulo:** Angulo de lanzamiento
- **Velocidad en x:** velocidad respecto a eje x
- **Velocidad en y:** velocidad respecto a eje y

- **Posx inicial**: Posición en x ira cambiando con el movimiento
- **Posy inicial:** Posición en y ira cambiando con el movimiento
- **Tamaño:** será su longitud de ancho y alto al manejar su imagen en un rectángulo también se dará de distintos tamaños para que el campo de daño sea más grande

Para los métodos necesitamos:

- disparar: realizar disparos hacia el jugador desde el Bot
- calcular velocidad: calcular velocidad con formula
- calcular posición: posición en la que se estará
- actualizar velocidad: nueva velocidad para simular el movimiento parabólico
- comprobar distancia: esta función comprueba la distancia ideal para realizar el tiro hacia el jugador cuando sea correcta entonces se realiza el disparo.
- Setpox: dar posición en x
- Setposy: dar posición en y
- Getposx: obtener posición en x
- Getposy: obtener posición en y
- Funciones graficas de Qt

Clase Bola de Goma

para esta se tendrá en cuenta el **rebote elástico:**
para el rebote elástico necesitamos ciertos atributos:

- **Velocidad en x:** su velocidad de movimiento en la pantalla respecto a eje x
- **Velocidad en y:** su velocidad de movimiento en la pantalla respecto a eje y
- **Aceleración en x:** variación de velocidad sobre t
- **Aceleración en y:** variación de velocidad sobre t
- **Posx:** Posición en x ira cambiando con el movimiento
- **Posy:** Posición en y ira cambiando con el movimiento
- **Tamaño:** será su longitud de ancho y alto al manejar su imagen en un rectángulo también se dará de distintos tamaños para que el campo de daño sea más grande

Los métodos serian:

- Mover: para dar movimiento a la bola desde la esquina superior derecha
- Setpox: dar posición en x
- Setposy: dar posición en y
- Getposx: obtener posición en x
- Getposy: obtener posición en y
- SetVelocidadx: para cambio de velocidad en rebote elástico
- SetVelocidady: para cambio de velocidad en rebote elástico
- GetVelocidadx: obtener velocidad
- GetVelocidady: obtener velocidad
- SetAceleracionx: cambiar aceleración para el rebote elástico
- SetAceleraciony: cambiar aceleración para el rebote elástico
- GetAceleracionx: obtener aceleración para hacer el cambio con fórmula
- Funciones graficas de Qt

Clase Inicio De Sección y Registro

SignIn //Iniciar sección.
SignUn //Nueva cuenta.
LoadGame //Nuevo juego
DeleteGame //Borrar progreso del juego
Users()//administrar información de los usuarios

Depuraciones:

2.1. CLASE GALLO:

```
Class: GALLO public QGraphicsItem
{
  //Atributos
  int posx, posy, tamaño,velocidad; //
int salto;
int raza;
int vida;


public:
  GALLO();


  GALLO(int x , int y , int t);
  QRectF boundingRect() const;//LIMITES DEL RECTANGULO QUE DELIMITA TU OBJETO
  //Paint como se mostrará pintado
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
        QWidget *widget= nullptr);

  void Mover();
  void saltar();


  int getPosx() const;
  void setPosx(int newPosx);
  int getPosy() const;
  void setPosy(int newPosy);
  int getVelocidad() const;
  void setVelocidad(int newVelocidad);
```

```
};
```

2.2. CLASE PARED

```
class pared :  public QGraphicsItem
{
   int posx,posy,tamaño;
public:
   pared();
   pared(int x, int y , int tamañox, int tamañoy);
   QRectF boundingRect() const;//LIMITES DEL RECTANGULO QUE DELIMITA TU OBJETO
   //paint como se mostrara pintado
   void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
           QWidget *widget= nullptr);

   int getPosx() const;
   void setPosx(int newPosx);
   int getPosy() const;
   void setPosy(int newPosy);
};
```

2.3. CLASE BALA:

```
class BALA: public QGraphicsItem
{

private:
   int x;  int velocidad ;
   int y; int daño;
   int tamaño;

public:
   BALA(int rad);
   ~BALA();

   int getX() const;
   void setX(int newX);
   int getY() const;
   void setY(int newY);

   //Definir posicion graficar
   void posicion();
```

```
    void posicion(int newX, int newY);

    void disparar(int);

    QRectF boundingRect() const;
    void paint(QPainter *painter,
            const QStyleOptionGraphicsItem *option, QWidget *widget);

};
```

2.4. BOLA DE GOMA:

```
class BOLA_DE_GOMA
{
public:
    BOLA_DE_GOMA(float px_=0, float py_=0, float vx_=0, float vy_=0, float rad_=10);

    float getPx() const;
    void setPx(float value);

    float getPy() const;
    void setPy(float value);

    float getVx() const;
    void setVx(float value);

    float getVy() const;
    void setVy(float value);

    float getAx() const;
    void setAx(float value);

    float getAy() const;
    void setAy(float value);

    float getRad() const;

    void mover(float dt);

private:
```

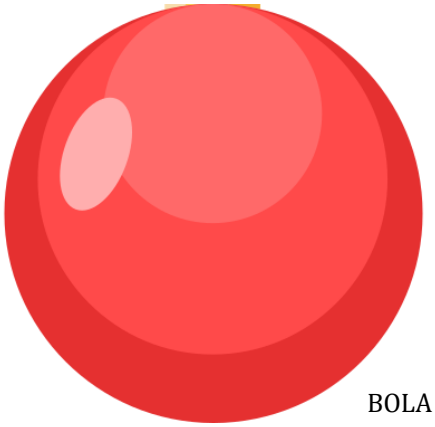
```
  float px;
  float py;
  const float rad;
  float vx;
  float vy;
  float ax;
  float ay;
};
```

2.5. Clase Fuego

```
Class Fuego
{
        // Atributos
        double posx,posy,angulo,velx,vely;
        int daño;

public:

  QRectF boundingRect() const;
  void paint(QPainter *painter,
       const QStyleOptionGraphicsItem *option, QWidget *widget);

fuego();
fuego(double x , double y , double vel, double angulo);
calcularVelocidad();
calcularPosicion();
actualizarVelocidad();
shot();
getposx();
getposy();
setposx();
setposy();

}
```

3. Anexos Imágenes Para Utilizar

BALA



BOLA DE GOMA