Informe de Video Juego

Theory B

video juego en lenguaje C++

John Edisson Chamorro Coral

Daniela Alvarez Bernal

Despartamento de Ingeniería Electrónica y
Telecomunicaciones
Universidad de Antioquia
Medellín
Febrero 21 de 2022

Índice

1. Descripción del Video Juego

El video juego consiste en un personaje (Mr Traveller) el cual va atravesando una serie de obstaculos hasta llegar a la meta. La razon por la que el Mr Traveller esta en constante huida es debido a que el tiempo se le va a acabando y el, para poder sobrevivir, deberá llegar a la meta en donde se garantiza que el tiempo no lo va a afectar y podra continuar con vida. En el transcurso del juego, mientras pasa los obstaculos en los mapas planteados, el podra encontrar puntos de recarga de tiempo (checkpoints) para que pueda llegar a la meta sin inconvenientes o contratiempos. Como otra opcion de ayuda estará un objeto resortado el cual le brindara mayor aceleración en el salto a Mr Traveller
El personaje tendra movilidad absoluta, es decir podra moverse para cualquiera de los cuatro puntos cardinales, ademas de poder realizar saltos con un efecto parabolico. En cuanto la manera de defenderse de los enemigos u obstaculos, este contará con bombas las cuales que efecutarán un movimiento parabolico y que al momento de su explosion eliminaran dichos obstaculos.
Finalmente el mapa donde se desemvuelve el personaje, contara con una serie de obstaculos que deberan ser atravesados por medio de saltos, o explosiones, o evitando su contacto.
Los enemigos tendran movimiento y repetitivo, y se ubicaran en tierra o aire dentro del mapa. Otro obstaculo será una bola de fuego la cual estará atada a una cadena en un punto estatico y rotara con movimiento circular uniforme buscando impactar al Mr Traveller.

2. Clases

Como se describió anterior mente, el juego constara de una serie de personajes y objetos los cuales deberan ser modelados en C++ para su posterior interacción e integración dentro del el desarrollo final

2.1. ¿Cómo se modelará el personaje principal?

El personaje pricipal o Mr Traveller tendrá un movimiento rectilineo en cualquiera de los cuatro puntos cardinales, además de poder realizar saltos parabolicos. Para ello se necesitará estos atributos:

- float posicionX : Se encargará de definir la posición en x dentro del escenario.
- float posicionY: Se encargará de definir la posición en y dentro del escenario.
- float velocidadX: se encargará de definir la velocidad en x a la que se mueve el personaje.
- float velocidadY: se encargará de definir la velocidad en y a la que se mueve el personaje.

- float aceleracionX: se encargará de definir la aceleración en x a la que se mueve el personaje.
- float aceleracionY: se encargará de definir la aceleración en y a la que se mueve el personaje cuando realiza un salto.
- float velocidadPaso: Se encargará de definir la velocidad de paso con la que le personaje se desplazará caminando hacia la izquierda o derecha del escenario.
- float dt: Esta será la variable que permitirá realizar una actualizacion de las posiciones, velocidades y aceleraciones cuando el personaje realice un salto.
- float ANG:Será una costante que define el angulo inicial del movimiento parabolico dentro del salto.
- float MASA: Definirá la masa del personaje para tener un correcto modelamiento del mismo con las escuaciones a emplear.
- float G:Definirá la constante de gravedad para tener un correcto modelamiento del mismo con las escuaciones a emplear.
- int ancho: Es el ancho que ocupara el personaje dentro de la escena.
- int alto: Es el alto que ocupara el personaje dentro de la escena.
- int vidas: Establece la cantidad de vidas que el personaje tendrá a lo largo de la partida.
- int puntaje: Se almacenará el puntaje que vaya adquiriendo el jugador a medida que avanza en la partida.
- Qpixmap *pixmap : Se encargará definir un objeto de tipo pixmap para poder insertar una imagen.
- QTimer *timer: para establecer la velocidad con la que itera entre las imagenes.

Ahora bien, para poder ir cambiando los valores que corresponden a cada accion del jugador y poder graficarlo con sprints, el personaje contará con los siguientes metodos:

- constructor(parametros): Sobrecarga de constructor para definir valores iniciales en los atributos del personaje.
- void setPx(): Se encargará de establecer la posicion del jugador, en el eje x, cuando este se esté desplazando.
- void setPy(): Se encargará de establecer la posicion del jugador, en el eje y, cuando este se esté desplazando.

- float getPx(): Para obtener la posición actual en el eje x, del personaje.
- float getPY():Para obtener la posición actual en el eje y, del personaje.
- int getPuntaje():Para obtener el puntaje que en el momento lleve el personaje.
- void setPuntaje():Para establecer el puntaje al personaje.
- int getVidas():Para obtener las vidas que en el momento lleve el personaje.
- void moveRight(): Para actualizar el movimiento hacia la derecha del personaje.
- void moveLeft():Para actualizar el movimiento hacia la izquierda del personaje.
- void moveUp(): Para actualizar el movimiento hacia arriba del personaje.
- void moveDown():Para actualizar el movimiento hacia abajo del personaje.
- void actualizarSalto(float dt): Para ir actualizando las posiciones del personaje cuando este realice un salto.
- actualizarImagen(): Se encargará de generar sprite al jugador.
- QRectF boundingRect() const: El cual definirá los limites del jugador para su representacion dentro del escenario.
- void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget): Metodo que me permitirá graficar al jugador.

2.2. ¿Cómo se modelará el enemigo volador?

Está clase estará distribuida a lo largo de todo el mapa y tendrá un movimiento periódico de derecha a izquierda en un espacio limitado. Si el personaje principal colisiona con este objeto perderá una vida. Su punto debil será su pefil superior ya que por ahi el jugador principal podra aplastarlo y asi acabaría con la vida de este personaje; para logra lo anterior se definen los siguientes atributos:

- float posicionX : Se encargará de definir la posición en x dentro del escenario.
- float posicionY: Se encargará de definir la posición en y dentro del escenario.
- float velocidadPaso: Se encargará de definir la velocidad de paso con la que le personaje se desplazará hacia la izquierda o derecha del escenario.
- int ancho: Es el ancho que ocupara el personaje dentro de la escena.

- int alto: Es el alto que ocupara el personaje dentro de la escena.
- Qpixmap *pixmap : Se encargará de definir un objeto de tipo pixmap para poder insertar una imagen;
- QTimer *timer: para establecer la velocidad con la que itera entre las imagenes.

Ahora bien, para poder ir cambiando los valores que corresponden al movimiento y visualizacion del objeto, se definiran los siguientes metodos:

- constructor(parametros): Sobrecarga de constructor para definir valores iniciales en los atributos del objeto.
- void setPx(): Se encargará de establecer la posicion del objeto, en el eje x, cuando este se esté desplazando.
- void setPy(): Se encargará de establecer la posicion del jugador, en el eje y, cuando este se esté desplazando.
- float getPx(): Para obtener la posición actual en el eje x, del objeto.
- float getPY():Para obtener la posición actual en el eje y, del objeto.
- void moveRight(): Para actualizar el movimiento hacia la derecha del objeto.
- void moveLeft():Para actualizar el movimiento hacia la izquierda del objeto.
- actualizarImagen(): Se encargará de generar sprite al objeto.
- QRectF boundingRect() const: El cual definirá los limites del objeto para su representacion dentro del escenario.
- void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget): Metodo que me permitirá graficar el objeto.

2.3. ¿Cómo se modelará el enemigo terrestre?

El enemigo terrestre tendrá un movimiento rectilineo uniforme unicamente hacia la izquierda o derecha del escenario, y estará con movimiento independiente en la busqueda del personaje principal con el fin de colisionarlo y quitarle la vida. Su punto debil será su pefil superior ya que por ahi el jugador principal podra aplastarlo y asi acabaría con la vida de este personaje; Para lograr lo anterior se le define estos atributos:

- float posicionX : Se encargará de definir la posición en x dentro del escenario.
- float posicionY: Se encargará de definir la posición en y dentro del escenario.

- float velocidadPaso: Se encargará de definir la velocidad de paso con la que le personaje se desplazará hacia la izquierda o derecha del escenario.
- int ancho: Es el ancho que ocupara el personaje dentro de la escena.
- int alto: Es el alto que ocupara el personaje dentro de la escena.
- Qpixmap *pixmap : Se encargará de definir un objeto de tipo pixmap para poder insertar una imagen;
- QTimer *timer: para establecer la velocidad con la que itera entre las imagenes.

Ahora bien, para poder ir cambiando los valores que corresponden al movimiento y visualizacion del personaje, se definiran los siguientes metodos:

- constructor(parametros): Sobrecarga de constructor para definir valores iniciales en los atributos del personaje.
- void setPx(): Se encargará de establecer la posicion del jugador, en el eje x, cuando este se esté desplazando.
- void setPy(): Se encargará de establecer la posicion del jugador, en el eje y, cuando este se esté desplazando.
- float getPx(): Para obtener la posición actual en el eje x, del personaje.
- float getPY():Para obtener la posición actual en el eje y, del personaje.
- void moveRight(): Para actualizar el movimiento hacia la derecha del personaje.
- void moveLeft():Para actualizar el movimiento hacia la izquierda del personaje.
- actualizarImagen(): Se encargará de generar sprite al jugador.
- QRectF boundingRect() const: El cual definirá los limites del jugador para su representacion dentro del escenario.
- void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget): Metodo que me permitirá graficar al jugador.

2.4. ¿Cómo se modelará la bomba?

La bomba será un objeto que utilizará el personaje principal para defenderse de los enemigos y superar obstáculos. Ejecutará un movimiento parabolico y al momento de su explosion eliminará dichos obstáculos; Para esto se utilizarán los siguientes atributos:

- float posicionX : Se encargará de definir la posición en x dentro del escenario.

- float posicionY: Se encargará de definir la posición en y dentro del escenario.
- float velocidadX: se encargará de definir la velocidad en x a la que se mueve el objeto.
- float velocidadY: se encargará de definir la velocidad en y a la que se mueve el objeto.
- float aceleracionX: se encargará de definir la aceleración en x a la que se mueve el objeto.
- float aceleracionY: se encargará de definir la aceleración en y a la que se mueve el objeto.
- float dt: Esta será la variable que permitirá realizar una actualizacion de las posiciones, velocidades y aceleraciones cuando el objeto esté en movimiento.
- float ANG:Será una variable que define el angulo del movimiento parabólico cuando se lanza la bomba .
- float MASA: Definirá la masa del objeto para tener un correcto modelamiento del mismo con las escuaciones a emplear.
- float G:Definirá la constante de gravedad para tener un correcto modelamiento del mismo con las escuaciones a emplear.
- int ancho: Es el ancho que ocupara el objeto dentro de la escena.
- int alto: Es el alto que ocupara el objeto dentro de la escena.

Ahora bien, para poder ir cambiando los valores y poder graficar el objeto, este contará con los siguientes metodos:

- constructor(parametros): Sobrecarga de constructor para definir valores iniciales en los atributos del objeto.
- void setPx(): Se encargará de establecer la posicion del objeto, en el eje x, cuando este se esté desplazando.
- void setPy(): Se encargará de establecer la posicion del objeto, en el eje y, cuando este se esté desplazando.
- float getPx(): Para obtener la posición actual en el eje x, del objeto.
- float getPY():Para obtener la posición actual en el eje y, del objeto.
- QRectF boundingRect() const: El cual definirá los limites del objeto para su representacion dentro del escenario.
- void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget): Metodo que me permitirá graficar el objeto.

- void actualizardatos(): Para actualizar la posición y la velocidad del objeto cuando este es lanzado.

2.5. ¿Cómo se modelará el fuego?

El fuego propio de la explosion que tendrá la bomba despues de ser lanzada, no tendrá un movimiento como tal, este será un objeto estatico con una forma especial la cual abaracara un area con el fin de colisionar con objetos dentro del mapa; Para lograr lo anterior se le define estos atributos:

- float posicionX : Se encargará de definir la posición en x dentro del escenario.
- float posicionY: Se encargará de definir la posición en y dentro del escenario.
- int ancho: Es el ancho que ocupara el personaje dentro de la escena.
- int alto: Es el alto que ocupara el personaje dentro de la escena.
- Qpixmap *pixmap : Se encargará definir un objeto de tipo pixmap para poder insertar una imagen;
- QImage *imagen: Se encargará definir un objeto tipo imagen para poder representarla en el escenario;

Ahora bien, para poder ir cambiando los valores que corresponden a la visualizacion del fuego, se definiran los siguientes metodos:

- constructor(parametros): Sobrecarga de constructor para definir valores iniciales en los atributos del personaje.
- QList¡fuego*¿explosion(int posx, int posy): Una lista donde se almacenaran los objetos de tipo fuego, con diferente posicion para poder definir el area de afectacion de dicha explosion.
- QRectF boundingRect() const: El cual definirá los limites del jugador para su representacion dentro del escenario.
- void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget): Metodo que me permitirá graficar al jugador.

2.6. ¿Cómo se modelará el resorte?

El personaje principal podrá utilizar este objeto para tener una mayor aceleracion en el salto y asi poder superar algunos obstáculos; Para lograr lo anterior se le define estos atributos:

- double posx:Se encargará de definir la posición en x dentro del escenario.
- double posY: Se encargará de definir la posición en y dentro del escenario.

- int ancho: Es el ancho que ocupara el resorte dentro de la escena.
- int alto: Es el alto que ocupara el resorte dentro de la escena.
- double Masa: Es la masa del objeto que está saltando sobre el resorte.
- double alpha: Es el Coeficiente de amortiguamiento.
- bouble expo: Es los valores que va teniendo alpha al pasar el tiempo.
- double angu: Es los valores de la frecuencia a la que se mueve el resorte al pasar el tiempo.
- double K: constante de elasticidad del resorte.
- double W: Es la frecuencia a la que se mueve el resorte.
- Qpixmap *pixmap : Se encargará definir un objeto de tipo pixmap para poder insertar una imagen;
- QTimer *timer: para establecer la velocidad con la que cambia su posición.

Ahora bien, para poder ir cambiando los valores que corresponden al movimiento y visualizacion del objeto, se definiran los siguientes metodos:

- double getPx(): Para obtener la posición actual en el eje x, del objeto.
- double getPY():Para obtener la posición actual en el eje y, del objeto.
- void setMasa(float newMasa): Para obtener la masa del objeto que está saltando sobre el resorte.
- void activarMovimiento(): Para conectar el Qtimer con el slot "calcularPosicion" y llamar este slot cada cierto tiempo.
- void actualizarValores(): Para calcular el valor que va teniendo el coeficiente de amortiguamiento y los valores de la frecuencia a la que se mueve el resorte al pasar el tiempo.
- void calcularPosicion(): Es un slot que se encarga de calcular la posición en Y del resorte dentro del escenario.
- QRectF boundingRect() const: Definirá los limites del resorte para su representacion dentro del escenario.
- void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget): Metodo que me permitirá graficar el resorte.
- resorte(double posx, double posy,double k): Sobrecarga de constructor para definir valores iniciales en los atributos del resorte.

2.7. ¿Cómo se modelará las bases de datos?

Esta clase se utilizará para guardar la partida, el puntaje del jugador y dato en general del juego, además de definir objetos dentro del mapa para darles esa creacion inicial; Esta clase no tendrá atributos.
Para realizar todo esto se empleará los siguientes metodos:

- void getStaticObjects(string fileName,QList¡T*¿object): Para leer y guardar en una lista la posición de todos los objetos estaticos del mapa.
- getDinamicObjects(string fileName,QList¡T*¿object): Para leer y guardar en una lista la posición inicial de todos los objetos dinamicos del mapa.
- void getPartida(): Para obtener la partida guardada.
- void setPartida(): Para establecer una nueva partida.

2.8. ¿Cómo se modelará utilidades?

Este objeto se utilizará para ahorar codigo con algunos procedimientos muy utilizados en el manejo de archivos como convertir de string a int, convertir de int a string etc, por lo tanto los objetos que manejen archivos heredarán de esta clase.
Esta clase no tendrá atributos.
Para realizar todo esto se empleará los siguientes metodos:

- static int contadorDigitos(string numero): Para contar los dígitos de un número.
- static int conversionStr2Int(string numero):Para convertir de string a entero.
- static QString conversionInt2Str(int numero):Para convertir de entero a string.

2.9. ¿Cómo se modelará los mapas?

Esta clase va a graficar lo que se haya leído en la base de datos.

2.10. ¿Cómo se modelará la bola de fuego con movimiento circular?

La bola de fuego será un obstaculo que el jugador tendrá que superar evitando el contacto con este, esto lo puede hacer saltando sobre la bola, si el juagador colisiona con la bola perderá una vida. Este objeto se moverá con un movimiento rectilíneo uniforme; Para lograr lo anterir se definen los siguientes atributos:

- float posicionX : Se encargará de definir la posición en x dentro del escenario.
- float posicionY: Se encargará de definir la posición en y dentro del escenario.
- int radioMovimiento: Es el radio de la circunferencia que describe el movimiento.
- float posicionAng: Se encargará de definir la posición angular de la bola de fuego.
- float rapAngular: Se encargará de definir la rapidez angular con la que gira el objeto.
- int ancho: Es el ancho que ocupara la bola de fuego dentro de la escena.
- int alto: Es el alto que ocupara la bola de fuego dentro de la escena.
- float dt: Esta será la variable que permitirá realizar una actualizacion de las posiciones y velocidades del objeto en movimiento circular .
- Qpixmap *pixmap : Se encargará definir un objeto de tipo pixmap para poder insertar una imagen.
- QTimer *timer: para establecer la velocidad con la que itera entre las imagenes.

Ahora bien, para poder ir cambiando los valores y poder graficar el objeto, este contará con los siguientes metodos:

- constructor(parametros): Sobrecarga de constructor para definir valores iniciales en los atributos del objeto.
- void setPx(): Se encargará de establecer la posicion del objeto, en el eje x, cuando este se esté desplazando.
- void setPy(): Se encargará de establecer la posicion del objeto, en el eje y, cuando este se esté desplazando.
- float getPx(): Para obtener la posición actual en el eje x, del objeto.
- float getPY():Para obtener la posición actual en el eje y, del objeto.
- QRectF boundingRect() const: El cual definirá los limites del objeto para su representacion dentro del escenario.
- void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget): Metodo que me permitirá graficar el objeto.
- void actualizardatos(): Para actualizar la posición y la velocidad del objeto.

2.11. ¿Cómo se modelará la ventana de inicio ?

3. Herencias

3.1. Clase enemigos

Los objetos .enemigo voladorz .enemigo terrestre"heredarán los atributos y métodos de esta clase.
Sus atributos serán los atributos que tienen en común estas dos clases.

- float posicionX : Se encargará de definir la posición en x dentro del escenario.
- float posicionY: Se encargará de definir la posición en y dentro del escenario.
- float velocidadPaso: Se encargará de definir la velocidad de paso con la que le personaje se desplazará hacia la izquierda o derecha del escenario.
- int ancho: Es el ancho que ocupara el personaje dentro de la escena.
- int alto: Es el alto que ocupara el ersonaje dentro de la escena.
- Qpixmap *pixmap : Se encargará de definir un objeto de tipo pixmap para poder insertar una imagen;
- QTimer *timer: para establecer la velocidad con la que itera entre las imagenes.

Sus métodos serán los métodos que tienen en común estas dos clases.

- void setPx(): Se encargará de establecer la posicion del objeto, en el eje x, cuando este se esté desplazando.
- void setPy(): Se encargará de establecer la posicion del jugador, en el eje y, cuando este se esté desplazando.
- float getPx(): Para obtener la posición actual en el eje x, del objeto.
- float getPY():Para obtener la posición actual en el eje y, del objeto.
- void moveRight(): Para actualizar el movimiento hacia la derecha del objeto.
- void moveLeft():Para actualizar el movimiento hacia la izquierda del objeto.
- actualizarImagen(): Se encargará de generar sprite al objeto.
- QRectF boundingRect() const: El cual definirá los limites del objeto para su representacion dentro del escenario.

3.2. clase utilidades

Los objetos que manejen archivos heredarán de esta clase.
Esta clase no tendrá atributos.
Para realizar todo esto se empleará los siguientes metodos:

- static int contadorDigitos(string numero): Para contar los dígitos de un número.
- static int conversionStr2Int(string numero):Para convertir de string a entero.
- static QString conversionInt2Str(int numero):Para convertir de entero a string.

3.3. clase movimiento parabólico

Los objetos que se muevan en un movimiento parabólico heredarán de esta clase. utilizará los siguientes atributos:

- float posicionX : Se encargará de definir la posición en x dentro del escenario.
- float posicionY: Se encargará de definir la posición en y dentro del escenario.
- float velocidadX: se encargará de definir la velocidad en x a la que se mueve el objeto.
- float velocidadY: se encargará de definir la velocidad en y a la que se mueve el objeto.
- float aceleracionX: se encargará de definir la aceleración en x a la que se mueve el objeto.
- float aceleracionY: se encargará de definir la aceleración en y a la que se mueve el objeto.
- float dt: Esta será la variable que permitirá realizar una actualizacion de las posiciones, velocidades y aceleraciones cuando el objeto esté en movimiento.
- float ANG:Será una variable que define el angulo del movimiento parabólico cuando se lanza la bomba .
- float G:Definirá la constante de gravedad para tener un correcto modelamiento del mismo con las escuaciones a emplear.

Se utilizará los siguientes metodos:

- void setPx(): Se encargará de establecer la posicion del jugador, en el eje x, cuando este se esté desplazando.

- void setPy(): Se encargará de establecer la posicion del jugador, en el eje y, cuando este se esté desplazando.
- float getPx(): Para obtener la posición actual en el eje x, del personaje.
- float getPY():Para obtener la posición actual en el eje y, del personaje.
- void actualizarPos(float dt): Para ir actualizando las posiciones del objeto cuando este realice un movimiento parabólico.