Descripción de Clases – Descubrimiento de América

Para el desarrollo de una implementación en un lenguaje orientado a objetos es necesario establecer clases con las capas de abstracción necesarias, que facilitarán en gran medida la realización del juego. Dadas las justificaciones pertinentes se procede a mostrar las clases que se percibieron como necesarias para desarrollar el juego.

- **Reglas de juego:** Esta clase va a jugar el papel de centinela en cuanto a los eventos del juego, va a ser la encargada de permitir el movimiento del barco, a su vez revisará colisiones, en general, será la encargada de disparar todos los sucesos relevantes del juego (menús, procesar los archivos de guardado, entre otros).

- **Físicas:** Esta clase se encargará de manejar las interacciones físicas entre los componentes del juego, dentro de ella estarán los tres componentes físicos requeridos en el juego.

- **Barco:** Como se puede ver en esta clase se encargará de representar el barco, allí se podrán establecer variables relacionadas con las mejoras del barco, a su vez como las animaciones (hereda de Físicas).

- **Animaciones:** Será la clase encargada de manejar todos los gifs de los componentes gráficos del juego, sea el barco o los obstáculos de la escena.

- **Obstáculos:** Corresponde a la clase que representa los obstáculos que le impedirán al barco seguir con su camino, o bien las monedas que aumentarán su dinero (hereda de Físicas).

- **Partida:** Gestiona la partida guardada, se encarga de abrir, cerrar, leer y escribir archivos, a su vez emite señales a las reglas de juego para mostrar los menús.

- **Ventanas:** Gestiona la aparición y desaparición de ventanas, contiene todos los widgets, botones y labels que aparecerán en las escenas.

Reglas de juego – Atributos y Métodos

Atributos

VECTORES.

QVector<QLabel *> labels: Contiene todos los labels editables.

QVector<QLabel *> image_labels: Contiene todos los labels de imágenes.

QVector<QPushButton *> shop_buttons: Tiene los botones del menu de compra.

QVector<QGraphicsView *> interfaces: Contiene los graphics view de cada widget.

QVector<QGraphicsScene *> scenes: Contiene las escenas donde se dispondrán los componentes gráficos.

PARTIDA GUARDADA.

partida *saves: Instancia de partida, que servirá para gestionar la partida guardada.

//TIMERS DE JUEGO.

QTimer *game_timer: Temporizador que cuenta los segundos y activa un slot cuando llega su timeout.

QTimer *change_scene_timer: Temporizador que permite cambiar de escena en un intervalo de tiempo dado.

QTimer *dispatch_obstacles_timer: Temporizador que significará el intervalo en el que los enemigos van a aparecer.

QTimer *wave_timer: Temporizador para saber cuándo se debe emitir la señal de oleaje.

SHORTS CON ESTADOS DEL JUEGO.

unsigned short current_scene: Indica el indice de la escena actual en el vector de escenas.

unsigned short in_scene_obstacles: Indica los obstaculos en escena.

unsigned short next_scene: Indica la siguiente escena que se mostrará.

unsigned short current_stage: Indica la stage (escena de gameplay) actual.

unsigned short game_time_counter: Contiene los segundos restantes hasta la finalización de la stage.

PRECIOS DE LA TIENDA.

unsigned int shop_prices[3];

bool twister_death: Booleano de control que representa si el barco chocó con un tornado o no.

QMediaPlayer *reproductor: MediaPlayer que permitirá reproducir la música de ambiente del juego.

QAudioOutput *output: Dispositivo de audio de salida, ligado al reproductor.

barco *ship: Instancia del barco que controla el jugador.

VECTOR DE OBSTACULOS, TODOS ALOJADOS EN EL HEAP.

QVector<obstaculo*> moving_obstacles: Obstáculos en movimiento.

QVector<obstaculo*> removed_obstacles: Obstáculos removidos.

QVector<obstaculo*> shop_obstacles: Barcos de muestra que se encuentran en la tienda.

VECTOR DE QSTRING.

QVector<QString> stage_messages: Mensaje que aparecerá en cada pantalla de carga.

Métodos

reglas_juego(QVector<QGraphicsView *> &graphics, QVector<QPushButton *> &shop_buttons, partida *partida): Constructor de la clase.

~reglas_juego(): Destructor de la clase.

void setup(): Crea las escenas, las asocia a un GraphicsView y verifica la existencia de una partida guardada.

void key_pressed(int key): Gestiona el evento keyPressed, verificando si se presionó una tecla asociada al juego, y realización operaciones en base a eso.

CONEXIONES

void initial_conections(): Conexiones que se establecen al crear la instancia.

void stage_connections(): Conexiones que se establecen al configurar la stage.

void obstacle_connections(obstaculo *obstacle): Conexiones para todo obstáculo.

LIBERACION

void dispose_obstacles(): Se libera la memoria de todos los obstáculos.

void dispose_removed_obstacles(): Se libera la memoria de los obstáculos removidos solamente.

void dispose_shop_obstacles(): Se libera la memoria de los barcos de muestra en la tienda del juego.

void dispose_scenes(): Se libera la memoria de las escenas creadas.

void dispose_image_labels(): Se libera la memoria del vector de labels de imágenes.

SETUP

void setup_stage(): Configuración inicial para cada stage.

void setup_shop(): Inicializa los obstáculos de la tienda, se edita el texto de los labels de la escena de menú de compra.

EVENTOS DEL JUEGO

bool is_colliding(QGraphicsProxyWidget *widget): Revisa si el widget dado colisiona con el barco.

void gameover(): Dispara las acciones que se hacen para el gameover.

void win(): Dispara las acciones que se hacen para la victoria.

CAMBIOS DE ESCENA, CARGA DE MENUS, ETCETERA.

void switch_scenes(): Oculta la escena actual y muestra la siguiente.

void main_menu(): Configura el menú principal y lo muestra.

void main_menu_load(): Configura el menú principal con la opción de cargar partida y lo muestra.

void show_middle_message(QString text): Muestra la pantalla de carga y un mensaje dado.

void handle_menu_compra(): Abre o cierra el menu de compra, y actualiza la información que hay en el mismo.

void update_shop(unsigned short blocked_buttons): Realiza las operaciones necesarias para actualizar el menu de compra.

void handle_end_stage(): Gestiona la finalización de una stage, realizando las operaciones necesarias para mostrar la siguiente, o bien mostrar la escena de victoria.

void stop_game(): Detiene las variables del juego, con el fin de que no se disparen eventos.

void loadMenu(bool dato): Slot que activa la funcion de menu principal o menu de carga, dependiendo del booleano dado.

void try_move(QPoint future_pos, QGraphicsProxyWidget *widget, bool crash_happening): Slot que administra los movimientos del juego, disparando eventos de choques, entre otros.

void dispatch_obstacles(): Slot que administra la generación de enemigos en el stage.

void wave_event(): Slot que dispara una señal para comenzar el movimiento armónico.

void outside_removal(obstaculo *obs): Slot que remueve a un obstáculo de la escena.

void change_scene(): Slot que administra los cambios de escena, y ejecuta las operaciones para iniciar una stage nueva.

void timer_changed(): Slot asociado al timer ‘game_timer’, disminuye el Contador de segundos y actualiza el label que muestra los mismos. También ejecuta operaciones cuando el contador llegue a 0.

void handle_mcu_finish(): Slot que active el gameover si es que el movimiento circular termina.

void manage_shop_buttons(): Slot ideado para administrar el clickeo de un boton del menu de compra, permitiendo subir de nivel, emitir sonidos, entre otros.

void start_game(): Inicia el juego despues de que el usuario haya pasado por el menu principal o el menu de carga.

DISPARADOR DE FISICAS / SEÑALES QUE DISPARAN EVENTOS FÍSICOS EN EL JUEGO.

void crash(QGraphicsProxyWidget *widget);

void shm();

void mcu();

void crash_ship(float speed);

void change_speed(short direction);

CAMBIOS EN LA ESCENA.

void hide_screen(unsigned short screen): Oculta un widget dado.

void show_screen(unsigned short screen): Muestra un widget dado.

void hide_button(unsigned short button): Oculta un botón dado.

void shoot_label_change(unsigned short label_index, QString new_text, bool is_aligned): Señal que dispara un evento que cambiar el texto de un label dado, y decidir si el texto está alineado o no.

void shoot_button_change(unsigned short button_index): Señal que dispara un evento para cambiar la visibilidad de un botón dado.

Físicas – Atributos y Métodos

Atributos

const float SPEED_MIN: Velocidad mínima.

const float DEFAULT_SPEED: Velocidad por defecto.

unsigned short max_speed: Velocidad máxima.

MAS

float frequence: Frecuencia del movimiento armónico simple.

double phase: Fase inicial del movimiento armónico simple.

float amplitude: Amplitud del movimiento armónico simple.

double initial_time: Tiempo inicial del movimiento armónico simple (tiempo en el que alcanza la posicion inicial).

double initial_x: Posicion inicial donde empieza el movimiento armónico simple.

MCU

float center_x: Centro del movimiento circular en x.

float center_y: Centro del movimiento circular en y.

float angle: Angulo actual.

float radius: Radio del movimiento circular.

float mcu_speed: Velocidad angular inicial.

MRU / TRABAJO

float pos_x: Posicion x actual.

float pos_y: Posicion y actual.

float mass: Masa

float speed: Velocidad

float ship_force: Fuerza ejercida al barco.

bool crash_happening: Booleano que verifica si está sucediendo un choque actualmente.

bool shm_happening: Booleano que verifica si hay movimiento armónico simple actualmente.

unsigned short crash_counter: Contador de choque.

unsigned short shm_counter: Contador de movimiento armónico simple.

unsigned short mcu_counter: Contador de movimiento circular.

Métodos

fisicas(int pos_x, int pos_y, float mass, float ship_force, unsigned short SPEED_MAX): Constructor de la clase físicas.

float mru(short direction): Devuelve la distancia recorrida en la dirección dada.

float trabajo(short direction): Devuelve la velocidad que tendrá el barco después de que se le ejerza una fuerza dada, en la dirección indicada.

float shm_x(unsigned int actual_time): Calcula la posición del cuerpo en un movimiento armónico simple.

QPoint mcu(): Calcula las coordenadas X y Y del objeto en un movimiento circular.

SETTERS Y GETTERS.

float getSpeed() const;

void setSpeed(float newSpeed);

float getPos_x() const;

void setPos_x(float newPos_x);

float getPos_y() const;

void setPos_y(float newPos_y);

float getMass() const;

void setMass(float newMass);

float getShip_force() const;

void setShip_force(float newShip_force);

unsigned short getMax_speed() const;

void calculate_initial_time(): Calcula el tiempo en el que el objeto estaría en una posición inicial en un MAS, tomando como x = 0 el pixel X = 350.

Barco – Atributos y Métodos

Atributos

int actual_animation: Animación actual.

unsigned int money: Dinero.

unsigned short level: Nivel actual.

unsigned short hp: Puntos de vida actuales.

animations *ship_animations: Animaciones del barco.

QTimer *crash_timer: Temporizador que permite realizar operaciones durante un choque.

QTimer *shm_timer: Temporizador que permite realizar operaciones durante el movimiento armónico simple.

QTimer *mcu_timer: Temporizador que permite realizar operaciones durante el movimiento circular.

Métodos

barco(unsigned short level, unsigned short hp, int pos_x, int pos_y): Constructor de la clase barco.

~barco(): Destructor de la clase barco.

void move(int key): Metodo que administra el movimiento del barco según la tecla ingresada.

void level_up(unsigned short level): Realiza las operaciones necesarias para mejorar el barco, y subirlo de nivel.

void stop_movement(): Detiene cualquier tipo de movimiento que tenga el barco (que no sea el movimiento con las teclas).

void crash_timeout(): Slot que realiza operaciones durante un choque.

void shm_timeout(): Slot que realiza operaciones durante el movimiento armónico simple.

void mcu_timeout(): Slot que realiza operaciones durante el movimiento circular.

void ask_move(QPoint future_pos, QGraphicsProxyWidget *widget, bool crash_happening): Señal que se emite para verificar si el barco puede moverse a una posicion dada.

void mcu_finished(): Señal que es emitida para indicar que el movimiento circular terminó, recibida por un slot de reglas de juego.

LOS SIGUIENTES METODOS PERMITEN INICIAR ACCIONES DE MOVIMIENTO DEL BARCO.

void start_crash(float speed)

void start_shm(

void start_mcu()

SETTERS Y GETTERS

unsigned short getMoney() const

void addMoney(unsigned short newMoney)

void setMoney(unsigned short money)

unsigned short getLevel() const

void setLevel(unsigned short newLevel)

unsigned short getHp() const

void setHp(unsigned short newHp)

Animaciones – Atributos y Métodos

Atributos

QString file_prefix: Prefijo del archivo que contiene la animación.

QLabel *main_label: Label que será añadido a la escena.

QVector<QMovie *> movies: Vector que contiene las aniamciones.

unsigned short animations_number: Numero de animaciones.

unsigned short max_pixels: Tamaño que tendrá el label.

Métodos

animations(QString file_prefix, int number_animations, unsigned short max_pixels): Constructor de clase, permite tener varias animaciones.

animations(QString route, unsigned short max_pixels): Constructor de clase para una sola animación.

~animations(): Destructor de clase.

void change_animations(QString file_prefix, int number_animations): Borra las animaciones actuales y las cambia por unas nuevas.

void set_animation(unsigned short animation_number): Le establece una animacion dada al label principal.

void initialize_movies(): Configura las animaciones en el vector de animaciones, y establece una de ellas en el label.

void setup_label(unsigned short max_pixels): Configura el label en cuanto tamaño y estilo se refiere.

void dispose_movies(): Libera memoria de las animaciones (QMovie).

QLabel *getMain_label(): Devuelve el label principal, que contiene una QMovie.

Obstaculos – Atributos y Métodos

Atributos

VARIABLES DE ESTADO

bool is_dangerous: Dice si el obstaculo es peligroso o no.

bool is_twister: Dice si es un tornado o no.

unsigned short issued_damage: Indica el daño que el obstáculo le puede hacer al barco.

animations *obstacle_animations: Animaciones del obstáculo.

TIMERS FISICOS

QTimer *movement_timer: Temporizador del movimiento usual.

QTimer *crash_timer: Temporizador para el movimiento durante el choque.

AUDIO

QString audio_route: La ruta donde se encuentra el audio que se reproducirá.

QMediaPlayer *reproductor: Reproductor de medios.

QAudioOutput *output: Dispositivo de salida de medios.

Métodos

void play_own_music(): Reproduce el audio que indica el atributo ‘audio_route’.

void start_movement(): Inicia el movimiento usual del obstáculo.

void stop_movement(): Detiene todo tipo de movimiento, sea de choque o usual.

void determine_audio_route(unsigned short obs_number): Establece la ruta de audio para el obstáculo actual.

SLOTS FISICOS

void handle_timeout(): Slot que permite realizar operaciones relacionadas con el movimiento usual del obstáculo.

void crash_timeout(): Slot que permite realizar operaciones relacionadas con el choque del obstáculo.

void start_crash(QGraphicsProxyWidget *widget): Inicia el choque.

void change_speed(short direction): Cambia la velocidad del movimiento del obstáculo.

void surpassed_limit(obstaculo *obs): Señal emitida cuando el obstáculo sobrepasa el limite de la escena.

void crash_management();

void ask_move(QPoint future_pos, QGraphicsProxyWidget *widget, bool crash_happening): Señal que se emite para verificar si el obstaculo puede moverse a una posicion dada.

GETTERS Y SETTERS.

animations *getObstacle_animations()

bool getIs_dangerous() const

void setIs_dangerous(bool newIs_dangerous)

bool getIs_out_scene() const

void setIs_out_scene(bool newIs_out_scene)

bool getIs_twister() const

void setIs_twister(bool newIs_twister)

unsigned short getIssued_damage() const

void setIssued_damage(unsigned short newIssued_damage)

Partida – Atributos y Métodos

Atributos

QString nombreArchivo: Nombre del archivo que contiene la partida guardada.

QFile archivo: Archivo donde está la partida guardada.

bool error: Booleano que indica si hubo un error al cargar partida o no.

bool wants_load: Indica si el usuario quiere cargar o no partida.

QVector<int> data: Datos extraidos del archivo de partida guardada.

Métodos

void save_load():Carga la partida guardada, o en su defecto carga datos por defecto al vector de datos.

void save_data(): Guarda en el archivo los datos que están contenidos en el vector de datos.

void update_data(unsigned short level, unsigned int coins, unsigned short stage, unsigned short hp, unsigned short time): Actualiza el vector de datos.

void exists_file(): Emite una señal diciendo si existe el archivo de partida guardada o no.

bool check_integrity(QString line, unsigned short parameter): Verifica la integridad de las lineas del archivo, decidiendo si cumplen los parámetros establecidos o no.

void load_defaults(): Carga al vector de datos los parámetros por defecto.

void open_file(): Abre el archivo en modo lectura/escritura.

void error_occurred(const QString &errorMessage): Emite una señal que indica que hubo un error en el proceso de carga de datos, esta señal esta conectada con un slot en ventana, que muestra un QMessageBox.

LOS METODOS SIGUIENTES SIRVEN PARA CAMBIAR LA VARIABLE DE ESTADO 'wants_load' a true o false respectivamente.

void allow_file_load();

void forbid_file_load();

SETTERS Y GETTERS.

bool getError() const

void setError(bool newError)

QVector<int> getData() const

void setData(const QVector<int> &newData)

Partida – Atributos y Métodos

Atributos

INSTANCIAS EN PILA UI (INFORMACION DE TODOS LOS UI).

Ui::Menu1 menu_nuevo

Ui::Menu2 menu_cargar

Ui::Stage stages

Ui::MenuCompra menu_compra

Ui::MiddleMessage middle_message

WIDGETS EN HEAP.

QWidget *menu_nuevo_widget

QWidget *menu_cargar_widget

QWidget *middle_message_widget

QWidget *stages_widget

QWidget *menu_compra_widget

VECTORES DERIVADOS DE LOS WIDGET, SE BORRAN CUANDO BORRO EL WIDGET.

QVector<QGraphicsView *> forms (Contiene los qgraphicsview asociados a cada widget)

QVector<QLabel *> labels (contiene todos los labels editables).

QVector<QPushButton *> shop_buttons (Contiene todos los botones de la tienda)

QVector<QPushButton *> hideable_buttons (Contiene un solo boton, el de salir de la escena de gameover y victoria.

QVector<QWidget *> widgets (Los widgets asociados a cada UI).

reglas_juego *game: Instancia de las reglas de juego.

QFont Font: Fuente que se usará en absolutamente todos los label.

partida *partidas: Instancia de partida, que permite administrar la partida guardada.

Métodos

ventana(QWidget *parent = nullptr): Constructor de clase.

~ventana(): Destructor de clase.

void configure_graphics(QGraphicsView *graph): Establece las configuraciones iniciales para un graphics view dado.

void keyPressEvent(QKeyEvent *event) override: Metodo que administra los eventos de teclas presionadas, remitiendo cualquier evento a las reglas de juego.

void setup_game_rules(): Se configuran todos los aspectos necesarios para realizar una instancia de reglas de juego (se forma el vector de graphicsView y se crea una instancia de partida).

void setup_font(): Configura la fuente para que este lista para usar en cualquier momento.

void conexiones(): Realiza las conexiones necesarias con respecto a ventanas.

void setup_buttons(): Se configuran los botones para que sean invisibles.

void dispose_widgets(): Se eliminan todos los widgets, por lo tanto todo lo que sea hijo de un widget dado, por ejemplo botones, labels, etcétera.

void close_window(): Slot que cierra la ventana cuando reciba la señal

void change_label_text(unsigned short label_index, QString text, bool is_aligned): Slot que cambia el texto de un label dado cuando recibe la señal.

void enable_button_visibility(unsigned short button): Vuelve un botón dado visible.

void hide_widget(unsigned short number): Oculta un widget dado.

void show_widget(unsigned short number): Muestra un widget dado.