Universidad del Valle de Guatemala — Jorge Daniel Barrientos Pineda 18714
Electrónica Digital 2 — Emilio Eduardo Velasquez Sanchez 18352

Proyecto 3

Explicación Código

sounds.h

`play_note`:
permite reproducir una sola nota, dada su frecuencia, su duración y la velocidad a la que se reproducirá.

`play_sound`:
es en esencia play_note en un loop para reproducir todo el array de notas que se le pase.

hal.h

Esta es la “hardware astraction layer” se encarga de todo el input y lo expone con un sencillo api.

Define cada tipo de input en un enum para facilitar su uso y evitar errores.

En un estructura guarda el estado de cada input en la forma de un booleano, así se puede estructurar el programa como si fueran “eventos” y usar condiciones sencillas de la forma “if (UP)”

`input_init`:
Se encarga de inicializar los pines necesarios como input.

`read_inputs`:
Lee todos los inputs y los organiza en la estructura de eventos.
Para los inputs analógicos de los joysticks, utiliza los macros `DEAD_ZONE` y `MID_POINT` para determinar a que evento corresponden.

`get_event`:
Devuelve el valor del evento que se le pase, su propósito es evitar una asignación accidental al evento en caso de una condición mal escrita.

`hal_debug`:
Envía por serial los valores de todos los eventos, para debugeo. Vuelve lento el programa en general.

map.h

Contiene las estructuras y funciones para crear los mapas del juego.

Node:
Estructura de nodo, contiene información de posición, distancia al nido de fantasmas, si posee pastilla, tipo de pastilla y enlaces a nodos vecinos. Base de la navegación de jugadores y NPCs.

node_new:
Crea un nuevo nodo por medio de malloc, inicializa sus valores y devuelve el puntero.

node_link:
Enlaza dos nodos bilateralmente. A partir de los valores de posición de los nodos determina la dirección que se encuentran una del otro y asigna sus punteros al enlace correspondiente.
Opera sobre el listado de nodos del mapa y el indice de los nodos en dicho listado.

map1_init:
Crea, inicializa y enlaza todos los nodos necesarios para crear el mapa.
Se llama map1 como “future proof” para facilitar la adición de mas mapas al juego.

map_debug_draw:
Para propósitos de debugeo. Dibuja todos los nodos del mapa y los enlaces que cada uno tiene con otros. Bastante lento.

game_object.h

game_object:
Pseudo clase de los jugadores, npcs y otros elementos del juego. Contiene informacion de posicion, tamaño del sprite, rangos de movimiento, puntaje, vidas, direction, control de sprites, lisatado de sprites, nodo actual, contadores para animaciones y powerups, etc.

game_object_init:
Inicializa los valores del objeto.

game_object_add_x y game_object_add_y:
Modifican la posicion del objeto manteniendolo dentro de los limites establecidos por los argumentos max y min.

game_object_set_pos:
Setea directamenta la posicion del objeto.

game_object_set_sprite:
Setea el indice del sprite a utilizar, la direccion en que debe realizarse espejado y actualiza el ancho y alto del objeto con los del sprite.

game_object_direction:
Setea la direccion del objeto.

`game_object_is_on_node`:
Revisa si el objeto se encuentra sobre el nodo activo.

`game_object_set_node`:
Setea el nodo activo delobjeto y actualiza los rangos de movimiento según la existencia y posicion de nodos vecinos.

`game_object_move`:
Principal control de movimiento. Mueve el objeto 1 pixel a la vez. Evaluá si puede moverse en la dirección seleccionada, configura el nodo activo si llega al siguiente nodo. Actualiza el contado de pasos que controla la animación de caminar.

`game_object_update_index`:
Control de animacion. Configura el sprite a mostrar según dirección, powerup, si esta muerto y control de la animación de caminar.

`game_object_random_move`:
Genera movimiento a azar. Si se encuentran en un nodo selecciona una dirección al azar excepto la dirección por la que vino de lo contrario sigue moviéndose en linea recta. También evita que entren al nido de fantasmas cuando no deben.

`game_object_ghost_move`:
Movimiento de los fantasmas, si no están muertos moverse al azar, si se encuentran entre nodos, seguir moviéndose en linea recta, si están muertos buscar la ruta al nido, si estan en el nido moverse de lado a lado hasta revivir, también actualiza el contador de pasos muertos que controla cuando reviven.

`game_object_pacman_move`:
Movimiento del jugador. Si esta entre nodos sigue en linea recta. Controla el devorado de pastillas, activa/descativa el powerup y evita que entre al nido de fantasmas.

`game_object_collision`:
Revisa si el jugador y los fantasmas estan suficientemente cerca para ser contacto. Dependiendo de si el powerup esta activo o no, le quita vida al jugador o mata al fantasma y aumenta el puntaje del jugador.

`main.cpp`

`setup`:
Se prepara el serial, modulo spi, la tarjeta SD, los pines de input/output, la pantalla, los objetos de juego (jugadores, fantasmas y display de vidas).

`loop`:
El loop es en efecto un FSM, según el valor de estado, ejecuta `game_menu`, `game_play`, `game_over`. Siempre ejecuta `read_input` para mantener actualizados los eventos.

`game_menu`:
: dibuja la pantalla de inicio con `draw_sd_img` para cada pieza de la imagen. Se queda a la espera de que se seleccione el modo de juego mientras reproduce la musica de inicio. Una vez se seleccoina modo de juego, limpia la pantalla llama `start_conditions` y cambia el estado a `game_play.`

`game_play`:
: Dibuja el mapa al iniciar, actualiza la dirección de los jugadores en base al input, mueve los jugadores, setea los fantasmas en modo vulnerable si alguno de los jugadores tiene el powerup, evaluá si hubieron colisiones, si alguno de los jugadores pierde una vida ejecuta la animación de muerte del jugador, actualiza los sprites de todos, reproduce el efecto de sonido del jugador y si ambos jugadores pierden todas sus vidas o ya no hay pastillas en el mapa cambia el estado a `game_over` y renderiza todos los objetos.

`game_over`:
: Muestra un mensaje de victoria o derrota y los puntajes según el modo de juego y la cantidad de vidas de los jugadores.

`render_game_object`:
: Dibuja en la pantalla el sprite del objeto o en caso de estar oculto dibuja un rectángulo negro.

`render_pellet`:
: Dibuja las pastillas en el mapa según la información del nodo. Si no hay pastilla dibuja un rectángulo negro.

`render_lives`:
: Dibuja en la pantalla el contador de vidas. Por eficiencia solo se ejecuta cuando hubo cambio en las vidas (`old_lives` no es igual a `lives`). Según la cantidad de vidas oculta los elementos correspondientes.

`render_score`:
: Dibuja en la pnatalla el puntaje de cada jugador solo si hay cambios (`old_score` no es igual a `score`).

`render_game`:
: Dibuja todos los elementos del juego. Itera sobre las listas de elementos y llama las funciones de renderizado adecuadas.

`draw_sd_img`:
: Dibuja en la pantalla una imagen guardada en la SD. Lee todo el contenido del archivo a ram, parsea los primero dos elementos del archivo (ancho y alto) y con ese asigna con `malloc` espacio para el bitmap. Luego parsea el resto del archivo asignando los valores en la memoria de bitmap, renderiza el bitmap y libera con `free` la memoria de bitmap. El parseo se realiza con las funciones de la librería estandar `strtoul` (string to unsigned long) y `strtok` (string token, para dividir en substrings según un delimitador).

`start_conditions`:

Setea el nodo y posición inicial de cada objeto.

`play_death_anim`:

Reproduce los sprites de la animación de muerte del jugador y las notas del efecto de sonido de muerte.

`draw_background`:

Dibuja el mapa desde las sd, utilizando `draw_sd_img`.