42: Una guida completa a So Long

`So_long` è il primo gioco che costruirai nei tuoi 42 cursus. È un progetto divertente, in quanto ti consente di utilizzare le tue risorse grafiche e animazioni. Ti aiuterà anche in modo significativo a superare `cub3d` quando arrivi lì, poiché viene utilizzato anche l'MLX.

La parte principale del tuo programma comunicherà con MLX e X11, e il resto è un po' di analisi e gestione degli errori. Faremo un'immersione profonda nella MiniLibX e poi passeremo a vedere come possiamo integrarla all'interno del nostro progetto. Scaviamo!

Cos'è l'MLX?

MiniLibX, o MLX, è un framework creato da Olivier Crouzet su X11, un sistema di finestre sviluppato nel 1984! L'MLX è un'API C adatta ai principianti per interagire con il sistema X11 sottostante. Diamo un'occhiata ad alcune delle funzioni che potresti utilizzare.

- `mlx_init`: Inizializza la libreria MLX. Deve essere chiamato prima di utilizzare qualsiasi altra funzione.
- `mlx_new_window`: crea una nuova istanza di finestra.
- `mlx_hook`: registra gli eventi.
- `mlx_loop`: passa sopra il puntatore MLX, attivando ciascun hook in ordine di registrazione.
- `mlx_xpm_file_to_image`: converte un file XPM in un puntatore immagine MLX.
- `mlx_put_image_to_window`: Mette la tua immagine sullo schermo alle coordinate date.
- `mlx_destroy_image`: Libera l'immagine.
- `mlx_destroy_window`: Libera l'istanza della finestra.
- `mlx_destroy_display`: Libera MLX.

Esamineremo ogni funzione in modo più dettagliato in seguito, ma se desideri maggiori informazioni su queste funzioni, ti consiglio di visitare 42Docs, poiché hanno svolto un ottimo lavoro documentando l'MLX (collegato in risorse aggiuntive).

Installazione dell'MLX

Collegare l'MLX al tuo codice è piuttosto complicato, quindi cercherò di spiegarlo sia con una macchina macOS che con Linux (funziona anche sui computer della scuola). Questi esempi presuppongono che l'MLX sia installato nel progetto in una cartella denominata`mlx`. Se non hai installato MLX, puoi installarlo eseguendo questo comando:

```
git clone https://github.com/42Paris/minilibx-linux.git mlx
```

Questo scaricherà l'MLX in una cartella chiamata`mlx`, che ti aiuterà a seguire questa parte successiva. Se vuoi facilitare il processo di installazione/compilazione, puoi usare 42 CLI, che dispone di ottimi strumenti per l'MLX.

Mac OS

X11 dipende da`xquartz`, di cui avrai bisogno installare da Homebrew affinché venga compilato correttamente.

Sarà necessario includere le seguenti intestazioni nella compilazione dei file oggetto:

```
# Contiene i file di intestazione X11 e MLX
INCLUDE = -I/opt/X11/include -Imlx

.c.o:
	$(CC) $(FLAG) -c -o $@ $<$(INCLUDE)
```

E il seguente frammento viene utilizzato per collegare le librerie e i framework richiesti:

```
# Collega X11 e MLX e usa OpenGL e AppKit
MLX_FLAGS = -Lmlx -lmlx -L/usr/X11/lib -lXext -lX11 -framework OpenGL -framework
AppKit

$(NAME):$(OBJS)
	$(CC) $(FLAG) -O$(NAME) $(OBJS) $(MLX_FLAGS)
```

Linux

X11 e MLX dipendono da diversi pacchetti, che possono essere tutti installati con il seguente comando:

```
sudo apt-get install gcc Fare xorg libxext-dev libbsd-dev
```

Questo installerà tutti i pacchetti richiesti (alcuni dei quali potresti già avere).

Quindi dovrai includere le intestazioni richieste nei tuoi file oggetto, che assomigliano a questo:

```
# Contiene i file di intestazione X11 e MLX
INCLUDE = -I/usr/include -Imlx

.c.o:
	$(CC) $(FLAG) -c -o $@ $<$(INCLUDE)
```

E quindi collegare le librerie richieste:

```
# Collega X11 e MLX
MLX_FLAGS = -Lmlx -lmlx -L/usr/lib/X11 -lXext -lX11
```

```
$(NAME):$(OBJS)
	$(CC) $(FLAG) -O$(NAME) $(OBJS) $(MLX_FLAGS)
```

Compilazione universale

Se sviluppi su entrambe le piattaforme, ti suggerisco di far funzionare il tuo Makefile universalmente tra di loro. Questo può essere ottenuto controllando `uname`. Ecco uno snippet del mio Makefile:

```
# [...]

ifeq ($(shell uname), Linux)
	INCLUDES = -I/usr/include -Imlx
else
	INCLUDES = -I/opt/X11/include -Imlx
endif

MLX_DIR = ./mlx
MLX_LIB = $(MLX_DIR)/libmlx_$(UNAME).a
ifeq ($(shell uname), Linux)
	MLX_FLAGS = -Lmlx -lmlx -L/usr/lib/X11 -lXext -lX11
else
	MLX_FLAGS = -Lmlx -lmlx -L/usr/X11/lib -lXext -lX11 -framework OpenGL
-framework AppKit
endif

# [...]

all: $(MLX_LIB) $(NAME)

.c.o:
	$(CC) $(CFLAGS) -c -o $@ $< $(INCLUDES)

$(NAME): $(OBJS)
	$(CC) $(CFLAGS) -o $(NAME) $(OBJS) $(MLX_FLAGS)

$(MLX_LIB):
	@make -C $(MLX_DIR)

# [...]
```

Nota: collega sempre le tue librerie esterne DOPO i tuoi file oggetto! Non farlo può portare a un `unidentified reference`.

Alluvione che riempie la mappa

Ora che il tuo Makefile è completo, puoi passare all'analisi. La parte principale riguarda questa regola: "Devi controllare se c'è un percorso valido nella mappa".

Questo significa due cose:

1. Il giocatore deve accedere all'uscita.
2. Il giocatore, prima di uscire dalla mappa, è in grado di raccogliere tutti i collezionabili.

Questo può essere fatto in una singola funzione ricorsiva, che segue uno schema simile al seguente pseudo-codice:

```
if (all_collectables_collected && exit_count == 1)
	return map_valid;
if (on_wall)
	return map_invalid;
if (on_collectable)
	collectables++;
if (on_exit)
	exits++;
replace_current_position_with_wall;
if (one_of_the_four_adjacent_directions_is_possible)
	return map_valid;
return map_invalid;
```

Sviluppo con MLX

Consiglio vivamente di utilizzare il più possibile la memoria allocata nello stack, poiché renderà la tua vita molto più semplice quando dovrai uscire dal programma.

Voglio anche condividere la struttura dei dati che ho utilizzato in questo incarico, poiché credo che aiuterà molti di voi a non cadere nelle stesse trappole che ho fatto io.

```
typedef struct s_data
{
	void		*mlx_ptr; // MLX pointer
	void		*win_ptr; // MLX window pointer
	void		*textures[5]; // MLX image pointers (on the stack)
	t_map		*map; // Map pointer (contains map details - preferably kept on the stack)
}	t_data;
```

Ora con quello fuori mano, scaviamo!

Inizializzazione

Per iniziare a lavorare con MLX, dobbiamo prima inizializzare la libreria MLX. Sotto il cofano, questo crea una nuova struttura che contiene tutti i dati richiesti per il corretto funzionamento dell'MLX.

```
#include "mlx/mlx.h"
#include <stdlib.h>

int main(void)
{
	void *mlx_ptr;

	mlx_ptr = mlx_init();
	if (!mlx_ptr)
		return (1);
	free(mlx_ptr);
	return (0);
}
```

In questo esempio, noterai che abbiamo impostato il `mlx_ptr` al risultato del `mlx_init` funzione. Potrebbe tornare `NULL` se si è verificato un problema con X11, che dovrai gestire.

Creare la tua prima finestra

Dopo aver istanziato la libreria MLX, sarai in grado di creare una finestra. Useremo il `mlx_new_window` funzione, che ha il seguente prototipo:

```
void *mlx_new_window(void *mlx_ptr, int size_x, int size_y, char *title);
```

Noterai che possiamo fornire le dimensioni, che ti torneranno utili quando caricherai le tue trame personalizzate. Per ora, però, forniamo solo alcune variabili statiche:

```
#include "mlx/mlx.h"
#include <stdlib.h>

int main(void)
{
	void *mlx_ptr;
	void *win_ptr;

	mlx_ptr = mlx_init();
	if (!mlx_ptr)
		return (1);
	win_ptr = mlx_new_window(mlx_ptr, 600, 400, "hi :)");
	if (!win_ptr)
		return (free(mlx_ptr), 1);
	mlx_destroy_window(mlx_ptr, win_ptr);
	mlx_destroy_display(mlx_ptr);
	free(mlx_ptr);
```

```
	return (0);
}
```

Questo dovrebbe aprire una finestra e quindi chiuderla immediatamente, dato che subito dopo chiameremo i metodi destroy. Per prevenire questo comportamento e dire all'MLX di aspettare, dobbiamo conoscere gli hook e gli eventi!

Ascolto di eventi con hook

Gli hook sono una parte essenziale dell'MLX, in quanto consentono di ascoltare i cambiamenti e gli eventi. Ecco un paio di hook principali che utilizzerai `so_long`:

- `mlx_loop`: registra qualsiasi hook e ascolto definiti in precedenza. Ciò impedisce anche il comportamento predefinito di distruggere la finestra al momento della sua creazione.
- `mlx_hook`: consente di ascoltare eventi X11 nativi, come movimenti del mouse, pressioni di tasti, interazione con la finestra e altro... (elenco completo qui)

Con questi due hook, sarai in grado di ascoltare l'input dell'utente e modificare i dati di conseguenza.

Un'implementazione di base di questo potrebbe essere simile a questa:

```
#include "mlx/mlx.h"
#include <stdio.h>
#include <stdlib.h>
#include <X11/X.h>
#include <X11/keysym.h>

typedef struct s_data
{
	void *mlx_ptr;
	void *win_ptr;
} t_data;

int on_destroy(t_data *data)
{
	mlx_destroy_window(data->mlx_ptr, data->win_ptr);
	mlx_destroy_display(data->mlx_ptr);
	free(data->mlx_ptr);
	exit(0);
	return (0);
}

int on_keypress(int keysym, t_data *data)
{
	(void)data;
	printf("Pressed key: %d\\n", keysym);
```

```
        return (0);
}

int main(void)
{
        t_data data;

        data.mlx_ptr = mlx_init();
        if (!data.mlx_ptr)
                return (1);
        data.win_ptr = mlx_new_window(data.mlx_ptr, 600, 400, "hi :)");
        if (!data.win_ptr)
                return (free(data.mlx_ptr), 1);

        // Register key release hook
        mlx_hook(data.win_ptr, KeyRelease, KeyReleaseMask, &on_keypress, &data);

        // Register destroy hook
        mlx_hook(data.win_ptr, DestroyNotify, StructureNotifyMask, &on_destroy, &data);

        // Loop over the MLX pointer
        mlx_loop(data.mlx_ptr);
        return (0);
}
```

Dopo aver istanziato MLX e la finestra, registriamo due hook:

- Un gancio di rilascio chiave, che chiama il `on_keypress` funzione ogni volta che l'utente rilascia un tasto. Questo include chiavi normali come `A`, `a` e `1`, ma anche alcune chiavi di sistema, come `ESC`, `ACCEDERE`, e `DEL`.
- Un distrutto notifica kook, che chiama `on_destroy` dopo che l'utente ha chiuso la finestra.

Noterai anche che l'ultimo argomento di ogni hook è l'argomento che vogliamo fornire al nostro gestore di hook. In questo caso, abbiamo passato un riferimento a `dati`, che contiene informazioni sui nostri puntatori MLX, ma puoi passare tutto ciò che desideri.

> Il `KeyRelease` e `DestroyNotify` le costanti sono eventi X11 nativi e possono essere importate tramite le intestazioni X11.

Caricamento di trame

Prima di caricare le tue trame, potresti notare che non ne hai. Ecco un modo rapido per ottenere alcune risorse grafiche di base e prepararle per MLX:

1. Devi prima andare a prurito.ioe scegli alcune risorse per il tuo gioco (puoi anche crearne di tue in Photoshop, assicurati solo di esportarle come PNG).
2. Esporta ogni texture in un'immagine PNG.

3. Vai a Convertio, che prenderà le tue immagini e le trasformerà in un formato di file XPM.

Con queste immagini XPM appena create, ti consiglio di definire sia la dimensione del riquadro che i percorsi per ciascuna delle tue risorse in un file di intestazione. Ciò porterà a un codice più leggibile e il tuo futuro autovalutatore ti ringrazierà per questo.

Il caricamento delle texture nell'MLX è un processo piuttosto semplice. Userai `mlx_xpm_file_to_image` per caricare i tuoi file XPM dal tuo `/assets` cartella. Ha il seguente prototipo:

```
void *mlx_xpm_file_to_image(void *mlx_ptr, char *filename, int *width, int *height);
```

Ti consigliamo di memorizzare il puntatore dell'immagine restituito da questa funzione, poiché in seguito ne avrai bisogno quando dirai all'MLX cosa visualizzare sullo schermo.

Rendering alla tua finestra

Finalmente! Ora puoi iniziare a eseguire il rendering di trame e*Vedere* qualcosa sullo schermo. Per questo, c'è la funzione MLX `mlx_put_image_to_window`, che, come suggerisce il nome, mette un'immagine sulla finestra. Ecco il suo prototipo:

```
int mlx_put_image_to_window(void *mlx_ptr, void *win_ptr, void *img_ptr, int x, int y);
```

L'argomento del puntatore dell'immagine è il `void *` tornato da `mlx_xpm_file_to_image`, e `x` & `y` corrispondono alla posizione dello schermo dal pixel in alto a sinistra della texture.

Movimento di manipolazione

Quando un giocatore si muove, ci sono due cose che devi controllare:

1. La prossima mossa è valida?
2. La mossa successiva provoca un evento speciale (vincere la partita, raccogliere un collezionabile o, nel caso di un bonus, colpire un nemico)?

Dopo aver delineato questi due casi, è necessario apportare modifiche per gestirli separatamente. Inoltre, è necessario assicurarsi che l'utente finale veda le modifiche in tempo reale. Ciò significa che una volta che l'utente preme un tasto, si aspetta*qualcosa*, che si tratti di un movimento grafico come il movimento del giocatore o di un cambiamento radicale, come l'uscita dal programma dopo aver vinto o morto.

Informazioni sul rendering delle trame

Dovrai assicurarti di rendere correttamente le tue risorse. Dovrai prendere in considerazione la stratificazione (cioè il mio giocatore dovrebbe essere dietro o davanti al mio pavimento?) e il rendering condizionale, come non rendere l'uscita se il giocatore non ha tutti i collezionabili. Puoi anche aggirare alcuni di questi problemi personalizzando le tue risorse per includere lo sfondo, che semplificherà il tuo codice.

Errori comuni

- Rendering di tutte le trame in ogni fotogramma. Questo consuma memoria e dopo un (molto lungo) tempo causerà il crash del programma.
- Non utilizzare le costanti per aumentare la leggibilità/manutenibilità. Sebbene non sia un bug, non è una buona abitudine indicizzare esplicitamente in un array con un numero fisso, utilizzare una stringa semplice come percorso dell'asset o ripetere un valore arbitrario come dimensione del riquadro.
- Casi limite per il controllo della mappa. Pensa ad autorizzazioni non valide, una mappa con troppe poche colonne/righe o persino una riga vuota al centro della mappa.
- Andare all'uscita senza tutti i collezionabili non dovrebbe terminare il gioco.