

Olimpiada de Inovare și Creație Digitală

UI_engine

Documentația proiectului

1. Introducere

UI_engine a fost creat cu scopul de a realiza executabile care să aibă o interfață grafică realizată total de mine, are suficiente funcționalități pentru a crea o aplicație cu o interfață 2D, care să asculte acțiunile utilizatorului și să ofere un răspuns vizual cât mai rapid.

Ce este UI_engine mai exact?

UI_engine este o librărie statică care poate fi folosită împreună cu codul vostru C++ pentru a adăuga la aplicația voastră o interfață grafică rapidă și ușor de folosit cu doar câteva linii de cod. De asemenea, datorită faptului că este construită deasupra librăriei SDL2, aplicația voastră poate fi compatibilă cu multe platforme, fără să facă codul vostru mai complex sau mai greu de înțeles.

De ce avem nevoie pentru a crea un proiect cu UI_engine?

Tehnologiile necesare unui proiect ce folosește UI_engine sunt: *Visual Studio* și versiunile construite ale librăriilor *SDL2*, *SDL2_ttf*, *SDL2_image*. Pentru a putea rula executabila generată de Visual Studio este nevoie de fișierele *.dll găsite în folderele librăriilor menționate anterior sub *lib/\$\{arhitectura țintă\}*, plasate lângă executabila.

Tehnologiile folosite în acest proiect:

Visual Studio Community 2022 [<https://visualstudio.microsoft.com/vs/community>]

Împreună cu Desktop Development for C++ [în Visual Studio Installer]

C++ 20 [<https://en.cppreference.com/w/cpp/20>]

Și Standard Template Library (STL) [https://en.wikipedia.org/wiki/Standard_Template_Library]

Simple DirectMedia Layer 2 (SDL2) [<https://www.libsdl.org/>]

Alături de SDL2_ttf [https://wiki.libsdl.org/SDL2_ttf/FrontPage]

Și SDL2_image [https://wiki.libsdl.org/SDL2_image/FrontPage]

2. Cum funcționează UI_engine?

[a] UI_engine folosește pentru fiecare element o clasă care moștenește funcțiile și variabilele clasei UI: Datorită acestei clase putem avea elemente diferite, menținând o structură omogenă, similară între toate elementele. Acest aspect ne ajută pentru a crea un sistem de rendering stabil și ușor.



```
1 class UI {
2 public:
3
4     UI() {};
5     ~UI() {};
6     virtual float width() {};
7     virtual void width(float w) {};
8     virtual float height() {};
9     virtual void height(float h) {};
10    virtual float transparency() {};
11    virtual void transparency(float t) {};
12    virtual float x() {};
13    virtual void x(float x) {};
14    virtual float y() {};
15    virtual void y(float y) {};
16    virtual void show() {};
17    virtual void hide() {};
18    virtual void render() {};
19    virtual void destroy() {};
20    float getParentPosX() {};
21    float getParentPosY() {};
22    void toggleEvents() {}
23
24 protected:
25
26     friend class UI_window;
27     friend class UI_collection;
28     virtual void execute_event(UI_window* _window, const SDL_Event& _event) {};
29     void init() {}
30     void finit() {}
31     bool used;
32     bool isVisible;
33     bool getsEvents;
34     SDL_FRect* interface;
35     float transparencyVal;
36     UI* parent;
37
38 public:
39     struct UI_eventData {};
40     typedef std::function<void(UI_eventData)> UI_event;
41     UI_event onMousePress;
42     UI_event onMouseRelease;
43     UI_event onMouseMove;
44     UI_event onKeyPress;
45     UI_event onKeyRelease;
46     UI_event onScroll;
47 };
48
```

Imagini din structura clasei UI

După cum vedeți, cuvântul *virtual* este folosit foarte des în interiorul clasei UI, motivul fiind utilitatea acestui operator în UI_engine. Clasele ce moștenesc UI, au aceleași elemente de bază dar în cazul unei imagini, funcția *render()* va trebui să transfere sau să copieze fiecare pixel pe

ecran pe când dacă am avea un simplu pătrat render() va trebui doar sa seteze un numar de pixeli dintr-o regiune cu o singură culoare. Așadar avem multe funcții care pot fi modificate de utilizator pentru a crea elemente total unice și după nevoie. Pe lângă aceste funcții avem și variabile de tip UI_event care de fapt pot stoca funcții date de utilizator pentru a răspunde în diferite moduri la acțiunile utilizatorului.

Datorită clasei UI avem clasele următoare:

```
1 class UI_frame {};  
2 class UI_image {};  
3 class UI_text {};  
4 class UI_collection {};
```

[b] A doua parte importantă este sistemul ce orchestrează elementele UI și derivatele lor. Prima parte a acestui sistem are grijă la afișarea elementelor pe ecran. Toate elementele ce trebuie să fie afișate sunt stocate în *UI_vectorRenderList* itemOrder* respectiv *UI_mapRenderList* items* (variabile globale accesate de clasa *UI_window* prezentată în subpunctul **[c]**), funcția principală *processEvents(UI_window* uiWindow)* (din *UI_engine.h* și *UI_engine.cpp*) transmite toate evenimentele primite iar după efectuează afișarea elementelor din elementul *UI_window* transmis ca argument acestei funcții.

```
1 namespace UI_engine {  
2  
3     void processEvents(UI_window* uiWindow) {  
4         SDL_Event _event;  
5         while (uiWindow->status() != stopped) {  
6             uiWindow->update_framerate();  
7             if (uiWindow->framerate_limit())  
8             {  
9                 uiWindow->finish_framerate();  
10                while (SDL_PollEvent(&_event)) {  
11                    uiWindow->execute_sdlEvents(_event);  
12                }  
13                uiWindow->render();  
14            }  
15        }  
16    }  
17 };  
18  
19 }
```

[c] Și nu în ultimul rând, elementul *UI_window* este ultima parte importantă din *UI_engine*, acest element ne ajută la crearea și gestionarea unei ferestre capabilă de a proiecta interfața

tuturor elementelor noastre. Pe partea de Application Program Interface, dezvoltatorul poate stoca, obtine și manipula elementele UI create de el într-un mod cât mai ușor.

Creații și exemple cu UI_engine:



```
1 #include "SDL.h"
2 #include "SDL_Image.h"
3 #include "SDL_ttf.h"
4 #include "UI_engine.h"
5
6 using namespace UI_engine;
7
8 //UI_engine Hello World!
9
10 int main(int argc, char* argv[]) {
11     UI_window window("exampleWindow",
12         SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
13         400, 400, SDL_WINDOW_SHOWN, SDL_RENDERER_ACCELERATED,
14         144.0);
15
16     UI_frame frame_example(0.0f, 0.0f, 35.0f, 35.0f, 1.0f, { 0,0,0,255 });
17     frame_example.filled();
18
19     window.add(&frame_example);
20     processEvents(&window);
21     return 0;
22 }
```

UI_engine Hello World! Codul pentru a crea un pătrat negru pe ecran



```
1 #include "SDL.h"
2 #include "SDL_Image.h"
3 #include "SDL_ttf.h"
4 #include "UI_engine.h"
5
6 #include<thread>
7 #include<cstdlib>
8 #include<vector>
9
10 using namespace UI_engine;
11
12 //UI_engine Cube Rain
13
14 bool END = false;
15
16 void move_cube(std::vector<UI_frame> v) {
17     while (END == false) {
18         for(auto& cube: v){
19             cube.x(rand() % 1000);
20             cube.y(rand() % 1000);
21         }
22     }
23 }
24
25 int main(int argc, char* argv[]) {
26
27     UI_window window("exampleWindow",
28         SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
29         400, 400, SDL_WINDOW_SHOWN, SDL_RENDERER_ACCELERATED,
30         144.0);
31
32     std::vector<UI_frame> cubes;
33     for (int i = 0; i < 100; ++i) {
34         UI_frame* frame = new UI_frame(0.0f, 0.0f, 35.0f, 35.0f, 1.0f, {
35             0,0,0,255 });
36         frame->filled();
37         cubes.push_back(*frame);
38         window.add(frame);
39     }
40
41     std::thread random(move_cube, cubes);
42     processEvents(&window);
43     END = true, random.join();
44     return 0;
45 }
```

UI_engine Cube Rain. Codul pentru avea 100 de pătrate ce se mișcă pe tot ecranul folosind un thread.



```
1 #include "SDL.h"
2 #include "SDL_Image.h"
3 #include "SDL_ttf.h"
4 #include "UI_engine.h"
5
6 using namespace UI_engine;
7
8 //UI_engine Click Counter
9
10 int main(int argc, char* argv[]) {
11
12     UI_window window("exampleWindow",
13         SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
14         400, 400, SDL_WINDOW_SHOWN, SDL_RENDERER_ACCELERATED,
15         144.0);
16
17     UI_text click_counter(window.width()/2.0f, window.height()/2.0f, 1.0f, "0");
18     click_counter.x(click_counter.x() - (click_counter.width() / 2.0f));
19     click_counter.y(click_counter.y() - (click_counter.height() / 2.0f));
20
21     click_counter.onMouseMove = [&](UI::UI_eventData d) {
22         UI_text* ui = (UI_text*)d.w->object("click_counter");
23         ui->x(d.e.motion.x);
24         ui->y(d.e.motion.y);
25         ui = NULL;
26     };
27
28     click_counter.onMouseRelease = [&](UI::UI_eventData d) {
29         UI_text* ui = (UI_text*)d.w->object("click_counter");
30         ui->text(std::to_string(stoi(ui->text()) % 100 + 1));
31         ui = NULL;
32     };
33
34     click_counter.toggleEvents();
35
36     window.add(&click_counter);
37
38     processEvents(&window);
39     return 0;
40 }
```

UI_engine Click Counter. Exemplul pentru evenete și UI_window.object()

3. Cum putem adăuga UI_engine la proiectul nostru sau cum îl putem modifica?

[a] Pentru a adăuga UI_engine la orice sursă sau proiect de al nostru în Visual Studio este nevoie de pachetul binary *RELEASE* al UI_engine-ului găsit la următorul link:

[RELEASE] [https://github.com/boroicamarius/UI_engine/releases/tag/v1.0.0]

Instrucțiunile pentru a include librăria pot fi găsite în același link sau reproduceți următorii pași.

PREREQUISITE: librăriile SDL2, SDL2_ttf, SDL2_image descărcate deja în calculator

1. Creați un nou proiect gol în Visual Studio
2. Adăugați un fișier *.cpp cu numele pe care îl doriți
3. Descărcați versiunea RELEASE de pe github prin linkul de mai sus
4. În **Project Properties > C/C++ > Additional Include Directories** adăugați:
 - `${path_to_UI_engine}\include,`
 - `${path_to_SDL2}\include,`
 - `${path_to_SDL2_ttf}\include,`
 - `${path_to_SDL2_image}\include`

Unde `${path_to_xx}` este locația librăriei în calculator.

5. În **Project Properties > VC++ Directories > Library Directories** adăugați:
 - `${path_to_UI_engine}\lib\${target_architecture},`
 - `${path_to_SDL2}\lib\${target_architecture},`
 - `${path_to_SDL2_ttf}\lib\${target_architecture},`
 - `${path_to_SDL2_image}\lib\${target_architecture}`

Unde `${target_architecture}` este arhitectura în care vreți
construi aplicația.

6. Adăugați în **Project Properties > Linker > Additional Dependencies:**
 - `UI_engine.lib,`
 - `SDL2.lib,`
 - `SDL2main.lib,`
 - `SDL2_image.lib,`
 - `SDL2_ttf.lib`

Cum putem modifica UI_engine?

Trebuie reproduceți aceași pași din subpunctul [a] din această categorie, însă trebuie șterse `${path_to_UI_engine}\include` și `UI_engine.lib`, iar proiectul trebuie să fie de tipul *.lib

Iar asta e UI_engine.

Mulțumiri celor ce au creat următoarele softuri:

Imaginile au fost generate folosind *carbon* [<https://carbon.now.sh/>]

Github stochează tot proiectu repo [https://github.com/boroicamarius/UI_engine]

Tot proiectul a fost scris în **Visual Studio** și **C++** referințe mai sus.

De asemenea, mulțumesc de la cei ce m-am inspirat sau am învățat să realizez acest proiect:

The Chernob - [Making and Working with Libraries in C++ \(Multiple Projects in Visual Studio\)](#)



Lazyfoo.net - <https://lazyfoo.net/tutorials/SDL/>

StackOverflow – diferite întrebări pentru a face proiectul meu să funcționeze