Dear ImGui: pragmatyczne podejście do programowania interfejsów użytkownika

Kiedyś ktoś nieco przekornie stwierdził, że lenistwo jest motorem wszelkiego postępu. Oczywiście jest w tym stwierdzeniu sporo przesady, ale też można w nim znaleźć ziarno prawdy. Otóż istnieje pewna kategoria lenistwa, którą ja nazywam „lenistwem pragmatycznym". Podstawową zasadą pragmatycznego lenia jest słynna reguła DRY (skrót od angielskich słów „don't repeat yourself", czyli: „nie powtarzaj się"). I tutaj należy uczciwie przyznać: ten rodzaj lenistwa często prowadzi do zwiększenia efektywności pracy i do powstawania ciekawych rozwiązań. Dziś chciałbym zaprezentować czytelnikowi Dear ImGui: bibliotekę służącą do tworzenia interfejsów użytkownika, która powstała w duchu tak właśnie postrzeganego lenistwa. Jeśli chcesz się przekonać, co z tego wynikło, to zapraszam do lektury!

DAWNO, DAWNO TEMU...

Mniej więcej w latach 2008–2013 miałem okazję pracować nad szeregiem projektów związanych z tworzeniem interfejsów użytkownika dla gier oraz dla narzędzi wspomagających ich tworzenie. I pomimo tego, że z czasem miałem coraz więcej doświadczenia związanego z realizacją tego typu projektów, to ciągle byłem niezadowolony z efektów mojej pracy. Przy każdym kolejnym przedsięwzięciu, które tworzyłem, część związana z interfejsem użytkownika – nawet pomimo tego, że moja wiedza i umiejętności z czasem rosły – koniec końców z pięknie uporządkowanej, czystej grządki zamieniała się w pole pełne chwastów. Byłem coraz bardziej sfrustrowany – praca w takim środowisku nie sprawiała mi przyjemności. Tak się złożyło, że w okolicach 2013 roku zmieniłem pracę i skupiłem się na innych wyzwaniach. Pozostał jednak pewien niedosyt. Olśnienie przyszło kilka lat później, kiedy dość przypadkowo natrafiłem na niepozorną bibliotekę: Dear ImGui. Nie jest ona bynajmniej panaceum na wszystkie bolączki związane z budowaniem GUI, jednakże leżąca u jej podstaw koncepcja programowania interfejsów użytkownika w trybie natychmiastowym (ang. *immediate mode GUI*) jest genialna w swojej prostocie, a jednocześnie rozwiązuje szereg istotnych problemów w tej dziedzinie. Dla mnie zrozumienie tej koncepcji stanowiło moment pewnego myślowego przełomu, olśnienia. Zrozumiałem wtedy, że powodem opisanego wyżej stanu rzeczy nie był brak wiedzy czy staranności. Problem tkwił w tym, że koncepcja leżąca u podstaw architektury bibliotek służących do tworzenia graficznych interfejsów użytkownika, z których korzystałem, generowała zbyt duży poziom złożoności. W efekcie stosowanie rozwiązań opartych na tej koncepcji kończyło się powstawaniem rozwiązań nazbyt skomplikowanych, pełnych nadmiarowości oraz redundancji. Zacznijmy więc od początku i omówmy najpierw pokrótce...

...KLASYCZNE PODEJŚCIE DO TWORZENIA INTERFEJSÓW UŻYTKOWNIKA

Praca z typową biblioteką służącą do budowania interfejsów użytkownika składa się z trzech kroków:

1. Budowanie drzewa kontrolek (komponentów interfejsu użytkownika: przyciski, pola edycyjne, pola wyboru itd.); proces tworzenia takiego drzewa może przebiegać na różne sposoby: czasami programista buduje je ręcznie, innym razem tworzone jest ono na podstawie jakiejś zewnętrznej definicji – często wygenerowanej przez zewnętrzne narzędzie służące do projektowania interfejsu.
2. Aktualizacja drzewa kontrolek: krok ten wykonywany jest w głównej pętli programu i z punktu widzenia programisty aplikacji jest on zazwyczaj kompletnie nieprzezroczysty – w tle dzieje się jakaś „magia", dzięki której kontrolki odpowiednio reagują na zdarzenia (np. naciśnięcia klawisza lub kliknięcia myszką na ekranie).
3. Obsługa funkcji zwrotnych: funkcje te wywoływane są w odpowiedzi na pojawiające się zdarzenia. Na przykład kliknięcie myszką na fragmencie okna, gdzie znajduje się przycisk, powoduje wywołanie funkcji zwrotnej pokroju `OnButtonClicked()` z odpowiednimi parametrami (np. ID kontrolki), dzięki którym programista może zorientować się, który przycisk został aktywowany, i podjąć odpowiednią akcję. Gdy mamy do czynienia z dynamicznym interfejsami, to wewnątrz funkcji zwrotnych często umieszcza się kod modyfikujący drzewo kontrolek (tworzenie/usuwanie/aktywowanie/dezaktywowanie kontrolek itp.).

Już czytając powyższy opis, trudno nie oprzeć się wrażeniu, że tak skonstruowany system może być trudny i mało intuicyjny w obsłudze ze względu na decentralizację logiki obsługującej drzewo kontrolek. Śledzenie i debugowanie tej logiki (szczególnie w świetle braku transparencji na poziomie aktualizacji drzewa kontrolek, co w praktyce oznacza, że jesteśmy w stanie przechwytywać tylko wyrwane z kontekstu funkcje zwrotne) bywa koszmarem (kto tego sam nie doświadczył, ten nie zrozumie...). Dramat zaczyna się w momencie, gdy projekt rozwijany w ten sposób przekroczy pewną masę krytyczną. Wtedy poziomu złożoności nie da się już okiełznać (warto w tym przypadku pamiętać, że w większych projektach kod taki jest zazwyczaj rozwijany przez więcej niż jedną osobę). Kończy się to zazwyczaj katastrofą, wyrywaniem włosów z głowy oraz powtarzaniem pytania: