6.8. Statyczne elementy GUI

Ten podrozdział to prawdopodobnie najbardziej fascynujący etap naszej podróży przez wciąż obce ziemie wxWidgets. Choć są w nich jeszcze zakątki, o których nie śnią jeszcze nawet nasi najwytrawniejsi kartografowie, odsłaniamy te obszary biblioteki, które sprawią, że w coraz mniejszym stopniu będzie ona jawić się jako *terra incognita*. Wkraczamy w niezwykle bogaty świat kontrolek graficznego interfejsu użytkownika.

Kontrolki są istotą interakcji użytkownika z programem komputerowym. To one pośredniczą w nieustannej wymianie danych, niezależnie od tego, czy chodzi o ich prezentację, pobranie od użytkownika, czy też ze wskazanych przez niego źródeł zewnętrznych.

Podział kontrolek GUI, jaki chcę Ci zaproponować, nie jest innowacyjny i jest oparty na naturze samych kontrolek oraz doświadczeniu zdobytym przez wiele lat przez społeczność wxWidgets. Ponadto trzeba mieć na uwadze, że kolejne tematy pojawiają się na stronach książki w miarę potrzeby ich wprowadzania, dzięki czemu możemy uniknąć zbytniego informacyjnego przesilania. Nie jest powiedziane zatem, że temat, który nie znalazł dla siebie miejsce tutaj, nie pojawi się dalej. Pracując nad swoistą taksonomią elementów graficznego interfejsu użytkownika, jakie goszczą w ogromnych pokładach biblioteki, trudno również uniknąć skojarzeń z rozwiązaniami w tym zakresie, jakie zostały zaproponowane przez innych autorów – zwłaszcza tych, spod których ręki wyszło monumentalne dzieło *Cross-Platform GUI Programming with wxWidgets*[7]. Jeżeli potrzebujesz usystematyzowanego i analitycznego opracowania na temat wxWidgets, trudno jest polecić coś bardziej właściwego i trafnego.

Na podstawie przytoczonego dzieła i dzięki wieloletniej praktyce w pracy z wxWidgets wśród bogactwa klas wxWidgets związanych z programowaniem graficznego interfejsu użytkownika na własne potrzeby pojmowania świata wxWidgets wyodrębniłem kilka grup elementów GUI. Pierwszą z nich stanowią **podstawowe kontrolki GUI**, wśród których swoje miejsce znalazły **kontrolki statyczne**. Składają się na nie kontrolki służące jedynie do prezentowania jakichś informacji, z którymi użytkownik nie może wejść w interakcję (statyczne napisy, obrazki, paski postępu, inne graficzne elementy interfejsu). Oprócz nich wachlarz elementarnych kontrolek GUI tworzą **podstawowe kontrolki dynamiczne** służące do obsługi czynności, jakie użytkownik może podjąć w programie podczas standardowej pracy. Znajdują się wśród nich wszelkie przyciski oraz kontrolki, których celem jest pobranie od użytkownika jakichś prostych i zwykle pojedynczych wartości. Drugą grupę tworzą **zaawansowane kontrolki GUI**, których zadaniem jest oparta na zaawansowanych mechanizmach prezentacja danych połączona zwykle z możliwością pełnej interakcji użytkownika z programem. Znalazły się w niej wszelkie listy, drzewa, arkusze. Trzecią grupę, choć nazywanie ich kontrolkami może być sporne, tworzą różnego rodzaju kontenery, których zadaniem jest organizacja kontrolek w obrębie okna programu w różnych przeznaczonych do tego celu strukturach.

W kolejnych punktach zapoznam Cię z wybranymi i niezbędnymi do kontynuacji pracy nad naszą grą kontrolkami statycznymi, które stanowią fundament tworzonych samodzielnie graficznych interfejsów użytkownika.

6.8.1. Klasa wxStaticText

W zakresie tworzenia graficznych interfejsów użytkownika żadna klasa w praktyce nie występuje tak licznie jak klasa ***wxStaticText***, której obiekty reprezentują wszelkiego rodzaju statyczne teksty wyświetlane w obszarze klienta, w tym przeróżne etykiety i opisy. Powszechność klasy *wxStaticText* idzie nieodłącznie w parze z niebywałą łatwością jej stosowania. A oto, w jaki sposób można z niej korzystać:

```
wxStaticText *st = new wxStaticText(panel, wxID_ANY, wxT("Co ma wisieć, nie utonie."),
   wxPoint(15, 15), wxSize(200, 25), wxALIGN_CENTER_HORIZONTAL);
```

[7] Mowa oczywiście o wspomnianym wcześniej i wydanym w 2006 r. przez Pearson Education, Inc. niemal encyklopedycznym opracowaniu biblioteki wxWidgets, Panów Juliana Smarta, Kevina Hocka i Stefana Csomora.

Spróbuj dodać ten kod do naszej gry (tuż po tym, jak dodajesz do niego panel *wxPanel*) i skompiluj program. Po jego uruchomieniu powinieneś zobaczyć okno z pięknie wypisaną sentencją. Zwróć uwagę na to, w jaki sposób zachowuje się nasz nowy napis w oknie programu, gdy próbujesz je skalować, a także, w jaki sposób są interpretowane współrzędne, w których został umieszczony.

Jak widzisz, kolejne argumenty konstruktora *wxStaticText* nie odbiegają znacznie od ogólnego schematu. Wierzę, że nie stanowią już dla Ciebie żadnej tajemnicy.

W związku z tym, że rodzicem naszego tekstu jest główny panel okna, współrzędne są liczone od lewego górnego rogu panelu, a sam tekst jest na „sztywno" do nich uwiązany. Zwróć też uwagę na to, że rozmiar kontrolki przewyższa faktyczny rozmiar wyświetlanego tekstu w pikselach, dlatego został on wyśrodkowany w jej obrębie za pomocą stylu ***wxALIGN_CENTER_HORIZONTAL***. Oczywiście możesz to zmienić, korzystając z pozostałych możliwości, czyli ***wxALIGN_LEFT***, ***wxALIGN_RIGHT*** czy wielu innych, o których możesz poczytać w dokumentacji klasy.

Wypróbuj zmianę współrzędnych, zmianę stylów tekstu, a także sprawdź, co się stanie, gdy wpiszesz tekst dłuższy niż pojemnościowe możliwości kontrolki. Uzyskane w ten sposób doświadczenie będzie nieocenione, a wyciągnięte wnioski wyznaczą dalszą drogę na wiele lat pracy z wxWidgets.

Mimo statycznej natury kontrolki możesz zmieniać wyświetlany w niej tekst w trakcie działania programu. Metodą, która realizuje to zadanie, jest ***SetLabel()*** przyjmująca nowy ciąg znaków jako argument. Jej przykładowe użycie może wyglądać tak:

```
st->SetLabel(wxT("Nowy tekst..."));
```

Podobnie w każdej chwili możesz pobrać tekst wyświetlany w kontrolce. W tym celu użyj bezargumentowej metody ***GetLabel()***:

```
wxString str = st->GetLabel();
```

Poznanie natury i zachowania obiektów klasy *wxStaticText* jest niezbędne do efektywnego tworzenia programów wieloplatformowych. Jest to niezwykle istotne zwłaszcza w kontekście różnic, jakimi charakteryzują się wykorzystane w różnych systemach operacyjnych środowiska graficzne. Różnice w renderowaniu poszczególnych czcionek czy interpretacji niektórych danych związanych z umiejscowieniem i rozmiarem kontrolki mogą być źródłem nieoczekiwanych i irytujących problemów. Jednym z rozwiązań takiej sytuacji jest stosowanie relacyjnego projektowania interfejsów użytkownika przy użyciu różnych klas *sizerów*. Bardziej szczegółowe informacje na ten temat znajdziesz w dalszej części książki.

6.8.2. Klasa wxStaticBitmap

Klasa ***wxStaticBitmap*** służy do wyświetlania statycznych obrazów w obszarze klienta, przy czym obowiązują tu wszelkie zasady wyświetlania grafiki, o których mówiłem w jednym z poprzednich podrozdziałów. W zależności od zastosowanego formatu graficznego pliku źródłowego grafika może zostać wyświetlona z pełną obsługą przezroczystości lub nie.

Użycie klasy *wxStaticBitmap*, podobnie jak omówionej wcześniej klasy *wxStaticText*, jest banalne, a jej konstruktor, nieodbiegający znacznie od ogólnego schematu, nie pozostawia żadnych wątpliwości i nie skrywa żadnych tajemnic, z którymi moglibyśmy sobie nie poradzić.

Aby się o tym przekonać, posłużymy się *Wisielcem* i do jego obszaru klienta wstawimy ikonę programu w pełnej okazałości. Poniższy kod powinien załatwić sprawę w całości.

```
wxStaticBitmap *sbmp = new wxStaticBitmap(panel, wxID_ANY,
   wxBitmap(wxGetAppFile("gfx/icon.ico"), wxBITMAP_TYPE_ICO),
   wxPoint(250, 15), wxSize(64, 64));
```

Jeśli chcesz, przygotuj teraz inny obrazek testowy, jednak podczas jego wczytywania pamiętaj o zastosowaniu odpowiedniego makra inicjującego procedury obsługi dla Twojego formatu. Sprawdź też, co się stanie, gdy określony przez Ciebie rozmiar kontrolki będzie inny niż rozmiar obrazka źródłowego.