Podstawy programowania w LabView (5)

Pierwsza aplikacja

Do tej pory poznaliśmy środowisko programistyczne, podstawowe elementy języka programowania graficznego, obsługę plików tekstowych oraz zasady tworzenia wykresów. W tej części przygotowując nieskomplikowaną grę – Bitwa Morska – nauczymy się pracy z menadżerem projektu, przygotowywania podprogramów, dodawania opisów funkcji, konfiguracji okien naszej aplikacji i generowania plików exe.

Wszystkie zrobione do tej pory przykłady mieściły się na jednym diagramie. Oczywiście w rzeczywistości aplikacje są znacznie bardziej rozbudowane. Umieszczenie całego diagramu w jednym oknie może stać się niemożliwe, a już na pewno program taki byłby bardzo trudny do analizy i wyszukania ewentualnych błędów. Dlatego aplikacje pisane w LabView mają strukturę hierarchiczną. Każda może zostać podzielona na podprogramy (subVi) mogące zawierać kolejne podprogramy i tak aż do funkcji najniższego poziomu. Strukturę hierarchiczną programu i powiązania pomiędzy podprogramami można zobaczyć wybierając z menu górnego *View* → *VI Hierarchy*. Aby program mógł być umieszczony na diagramie jako *subVi* powinien mieć zdefiniowane zaciski służące do wymiany danych z nadrzędnym *VI* oraz ikonę symbolizującą go na diagramie.

Przygotowując dużą aplikację należy dobrze przemyśleć i podzielić zadania na małe, łatwe do zrealizowania fragmenty. Zapanowanie nad całą aplikacją ułatwia trzymanie się kilku zasad:

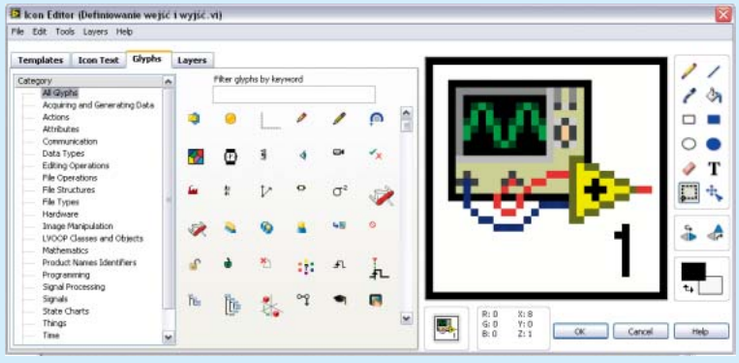
- Każdy program należy podzielić na fragmenty, których kod będzie w całości widoczny na ekranie monitora. Znacznie ułatwia to jego analizę i wyszukanie błędów.
- Każdy program należy dokładnie przetestować i dodać do niego krótki opis, może on być widoczny w okienku pomocy kontekstowej.
- Fragmenty kodu realizujące wspólne zadania najlepiej gromadzić w bibliotece tak aby przeniesienie jednego pliku zapewniało przeniesieni wszystkich wewnętrznych i powiązanych ze sobą funkcji.
- Zalecam korzystanie z Menadżera Projektów. Znacznie ułatwia to zapanowanie nad dużą liczbą plików. Dzięki niemu nie musimy za każdym razem szukać na dysku potrzebnej funkcji. Wystarczy przeciągnąć ją z okna projektu na diagram.
- Proponuje włączenie okna *Context Help*. Wyświetla ono krótki opis funkcji i przekazywanych parametrów.
- Diagram powinien być uporządkowany tak aby połączenia nie zachodziły na siebie, były możliwie krótkie i wchodziły do ikony na wysokości zacisku do którego są podłączone.
- Z lewej strony diagramu umieszczajmy kontrolki a z prawej wskaźniki, przepływ informacji zawsze od lewej do prawej.
- Nazwy podprogramów powinny sugerować realizowaną funkcję.
- W nawiasach należy umieścić wartości domyślne, jeśli wejście może zostać niepodłączone.

Definiowanie wejść i wyjść

Każdy podprogram – aby mógł być wykorzystany jako funkcja w nadrzędnym Vi – musi mieć zdefiniowane zaciski wejściowe i wyjściowe. Zaciski przekazują dane bezpośrednio do kontrolek lub wskaźników znajdujących się na panelu czołowym. Wykonanie takich połączeń jest możliwe tylko w oknie panelu czołowego. W prawym górnym rogu panelu czołowego na **rysunku 46** widzimy dwa małe okienka zewnętrzne przedstawiające wygląd ikony funkcji na diagramie, natomiast wewnętrzne rozkład i ilość wyprowadzeń. Liczba zacisków zależy od liczby parametrów funkcji. Zwykle z lewej strony łączymy parametry wejściowe, czyli kontrolki na naszym panelu czołowym, a z prawej strony parametry wyjściowe czyli wskaźniki.

Rysunek 46. Definiowanie zacisków subVi

Odpowiedni rozkład zacisków można przygotować poprzez kliknięcie prawym kla-



Rysunek 47. Edytor ikon

wiszem myszki w okienku terminali i wybór z menu lokalnego opcji dodaj terminal *(Add Terminal)* lub usuń terminal *(Remove Terminal)*. Można również skorzystać z zdefiniowanych wzorców wybierając opcję *Patterns*. Połączenie zacisku ze zmienną wykonuje się przez kliknięcie na wybranym zacisku, gdy kursor przybierze postać szpulki, a następnie na elemencie znajdującym się na panelu czołowym, z którym chcemy połączyć dany terminal. Poprawne połączenie sygnalizowane jest zmianą koloru terminala z czarnego na zgodny z typem przekazywanej zmiennej. W starszych wersjach *LabView* należało w okienku ikony wybrać *Show Connector*, aby zobaczyć okienko Terminali.

Edycja ikon

Wygląd ikony powinien sygnalizować funkcję realizowaną przez subVi. Środowisko oferuje nam edytor pozwalający w szybki i prosty sposób przygotować odpowiednią ikonę. Edytor oprócz prostego rysowania udostępnia gotowe symbole. **Rysunek 47** przedstawia okno edytora ikon z widocznymi po lewej stronie symbolami. Możemy je otworzyć klikając na okienku ikony w prawym górnym rogu i wybierając z menu lokalnego *Edit Icon*.

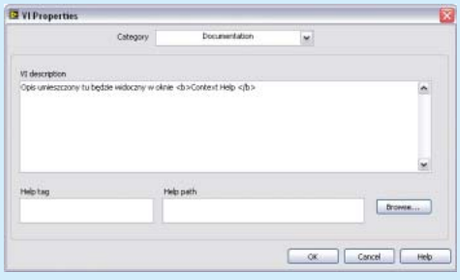
Opisy i komentarze

Jak wspominałem wcześniej każdy nasz program powinien być opisany. Proponuję dodawać komentarze wewnątrz diagramu lub panelu czołowego. Ułatwia to jego analizę i dokonanie modyfikacji w przypadku konieczności dostosowania go do innych aplikacji. Komentarze jest korzystnie dodawać za pomocą etykiety dostępnej w palecie *Decorations* → *Label*. Jest to tekst, który nie podlega kompilacji.

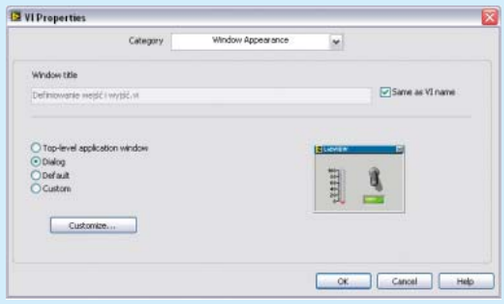
Korzystając z możliwości dodawania opisów w właściwościach vi, należy umieścić tam krótki opis realizowanej funkcji i zacisków wejściowych. Będzie on widoczny w okienku pomocy kontekstowej (*Context Help*) pokazanym na **rysunku 48**, gdy najedziemy myszką na ikonę funkcji w oknie diagramu lub projektu.

W tym celu klikamy w okienku ikony i wybieramy *VI Properties*. Przechodzimy do kategorii *Documentation* w polu *VI description* wpisujemy opis. Korzystając z przycisku *Browse…* możemy wskazać plik pomocy, najczęściej typu html, ale akceptowane są też inne formaty: hlp, chm, htm. Otwarcie tego pliku jest możliwe po kliknięciu w oknie pomocy kontekstowej w link *Detailed help*. **Rysunek 49** przedstawia okno właściwości Vi, w którym dodajemy opis, wyświetlany później w okienku pomocy kontekstowej.

Rysunek 48. Okno pomocy kontekstowej



Rysunek 49. Okno właściwości Vi kategoria Documentation



Rysunek 50. Okno właściwości Vi kategoria Windows Appearance

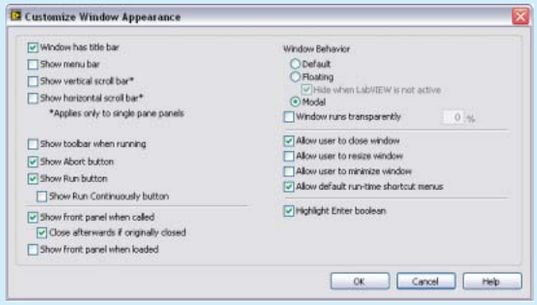
Konfiguracja okna uruchomionej aplikacji

Wygląd ona programu po uruchomieniu aplikacji można dopasować do naszych potrzeb. Dostęp do ustawień mamy w oknie *VI Properties* w kategorii *Windows Appearance* (**rysunek 50**). *Windows title* określa nazwy aplikacji widocznej w pasku tytułu. Domyślnie jest to nazwa VI, ale można wpisać tam własną. Poniżej mamy cztery opcje do wyboru:

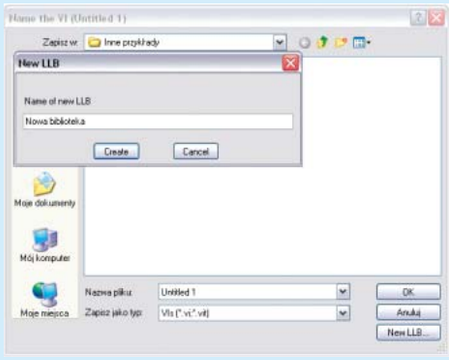
- *Top-level application window* – wygląd okna jest typowy dla programu głównego widoczny jest pasek tytułu z przyciskami do minimalizacji maksymalizacji i zamknięci okna, oraz górne menu aplikacji.
- *Dialog* – typowy wygląd dla okien wyświetlających komunikaty widoczny jest pasek tytułu.
- *Default* – widoczne są prawie wszystkie elementy paski i przyciski.
- *Custom* – niestandardowe ustawienia dostępne pod przyciskiem *Customize…*

Rysunek 51 przedstawia okno właściwości *Customize Windows Appearance*. Udostępnia ono wszystkie opcje konfiguracji okna aplikacji. Ustawiamy w nim, które z elementów mają być widoczne oraz jak zachowuje się okno względem innych okien:

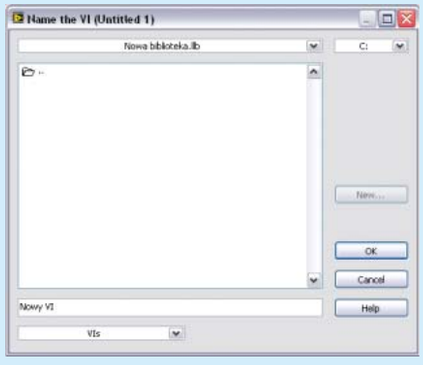
- *Window has title bar* – jest widoczny pasek tytułu.
- *Show menu bar* **– widoczne jest menu aplikacji.**
- *Show vertical scroll bar* **– widoczny jest pionowy pasek przewijania okna.**
- *Show horizontal scroll bar* **– widoczny jest poziomy pasek przewijania okna.**
- *Show toolbar when running* **–** gdy program jest uruchomiony widoczny jest pasek narzędziowy.
- *Show Abort button* **– wyświetla przycisk Abort.**
- *Show Run button* **– wyświetla przycisk Run.**
- *Show Run Continuously button* **– wyświetla przycisk Run Continuously.**



Rysunek 51. Okno Customize Windows Appearance.

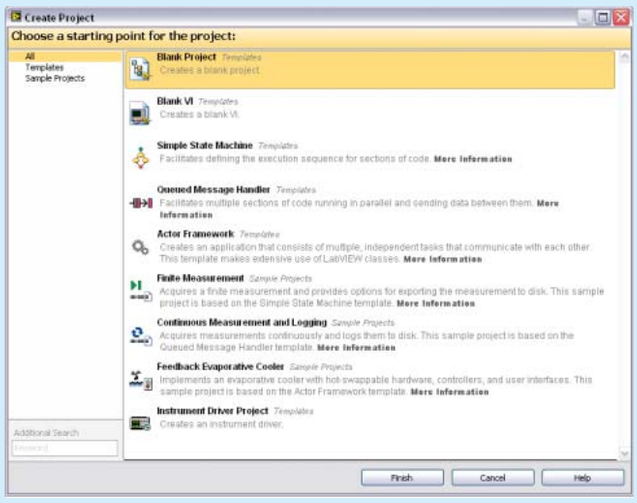


Rysunek 52. Okno dialogowe podczas tworzenia nowej biblioteki



Rysunek 53. Okno biblioteki

- *Show front panel when called* **– pokazuje panel po uruchomieniu programu.**
- *Close afterwards if originally closed* **– zamyka panel po zatrzymaniu programu.**
- *Show front panel when loaded* **– pokazuje panel po załadowaniu programu.**
- *Window Behavior* **– określa zachowanie się okna aplikacji względem innych okien.**
- *Allow user to close window* **– pozwala na zamknięcie okna przyciskiem na pasku tytułu.**
- *Allow user to resize window* **– pozwala na zmianę rozmiaru okna przyciskiem na pasku tytułu.**
- *Allow user to minimize window* **– pozwala na minimalizację okna przyciskiem na pasku tytułu.**
- *Allow default run-time shortcut menus* **– pozwala korzystać z menu lokalnego.**



Rysunek 54. Okno kreowania nowego projektu

Grupowanie przyrządów wirtualnych w biblioteki

Korzystnie jest zgromadzić wszystkie przyrządy wirtualne realizujące wspólne funkcje w bibliotece. Dzięki temu przenosząc plik biblioteki dysponujemy wszystkimi powiązanymi ze sobą funkcjami. Aby to zrobić należy wybrać z menu *File* → *Save* w okienku dialogowym (**rysunek 52**) wybieramy *New LLB* podajemy nazwę biblioteki i wybieramy *Create*. Otworzy się okno biblioteki (**rysunek 53**), w którym podajemy nazwę instrumentu wirtualnego i wybieramy *OK*. Od tej pory zapisując nowe programy do naszej biblioteki wskazujemy ją jak katalog na dysku.

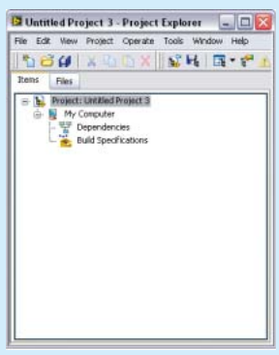
Menadżer Projektu

Pracując z dużymi projektami korzystnie jest pracować z Menadżerem Projektu. Ułatwia on zapanowanie nad dużą ilością plików i generowanie plików wynikowych. Okno, w którym możemy wykreować projekt otwiera się po wybraniu z menu *File* → *Create Project…* wyświetlone zostanie okno jak na **rysunku 54**. W tym miejscu wspomnę, że oprócz utworzenia nowego projektu czy pliku vi można skorzystać z kilku przykładowych projektów. Po wybraniu jednego z nich zostanie utworzony nowy projekt, zawierający kilka charakterystycznych dla siebie plików np. może być to podstawowa obsługa interfejsu użytkownika zawierająca pętle główną, strukturę zdarzeń i przycisk kończący działanie aplikacji, pozostaje dodanie własnych funkcji. Szczegółowy opis projektu, który zostanie utworzony możemy zobaczyć wybierając *More Information* dla interesującego nas projektu. Aby utworzyć nowy pusty projekt wybieramy *Blank Project* a następnie *Finish*. Zostanie wyświetlone okno projektu jak na **rysunku 55**. Możemy tutaj dodawać nowe pliki i biblioteki, dołączać istniejące, oraz wygenerować plik instalacyjny czy uruchamiany.

Przykładowa aplikacja

Aby przećwiczyć wszystkie etapy tworzenia programu, przygotujmy przykładową aplikację. Proponuję wykonanie prostej gry Bitwa Morska. Rozgrywka będzie wykonywać się między graczem a komputerem. Postaram się użyć tylko tych elementów programowania, które zostały opisane w powyższym kursie. Żeby program nie był zbyt skomplikowany przyjąłem kilka założeń, które pozwoliły znacznie go uprościć:

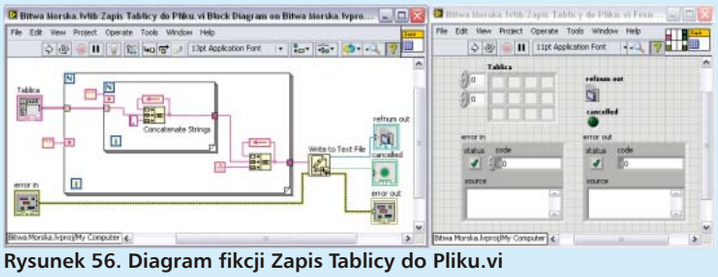
- Gra odbywa się na polu o rozmiarach 10×10, będzie to tablica dwuwymiarowa z elementami typu *String*.
- Okręt jest oznaczmy literą „o”.
- Kilka literek „o” umieszczonych obok siebie mogą symbolizować duży okręt, ale nie są one z sobą grupowane.
- Puste pole jest oznaczone spacją.
- Niecelny strzał znakiem „–”.
- Celny strzał znakiem „x”.
- Na każdej planszy znajduje się 20 znaków symbolizujących okręt.
- Gra kończy się po dwudziestu celnych strzałach w okręty przeciwnika.



Rysunek 55 Okno projektu

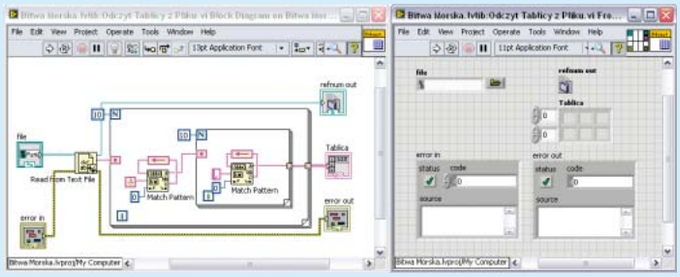
- Na ekranie widoczne są dwie plansze, własna z wszystkimi okrętami i komputera, na której wyświetlone są tylko pola, w które oddano strzał. Pozostałe są zamaskowane.

Grę podzieliłem na dwa programy pierwszy jest główną aplikacją umożliwiającą rozgrywkę. Natomiast drugi pozwala na ustawienie okrętów na planszy, wczytanie planszy komputera, oraz przygotowanie i zapis do pliku planszy komputera.

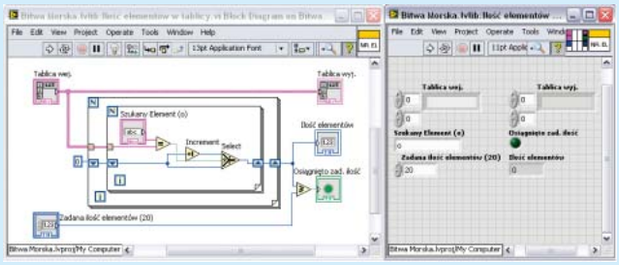


Rysunek 56. Diagram fikcji Zapis Tablicy do Pliku.vi

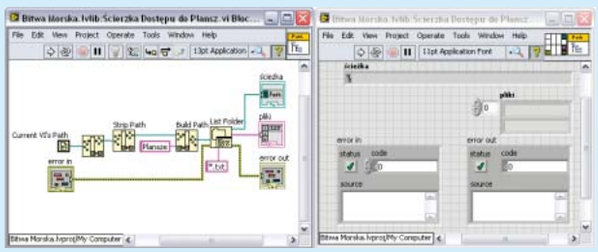
Przygotowanie programu rozpoczynamy od utworzenia nowego projektu i zapisania go na dysku. Klikając w oknie projektu na ikonie *My Computer* z menu lokalnego wybieramy *New* → *Library*. Zostanie utworzona ikona symbolizująca bibliotekę. Wybierając z menu lokalnego *Save* → *Save* należy nadać jej nazwę np. *Bitwa Morska*. Powstała w ten sposób biblioteka w rzeczywistości nie przechowuje w sobie plików, ale grupuje pliki, które mogą być rozrzucone po dysku. Klikając na utworzonej bibliotece z menu lokalnego wybieramy *New* → *VI* zostanie otwarty nowy plik. Należy wybrać *File* → *Save*, a w wyświetlonym oknie dialogowym proponuję wybrać *New LLB…* i wpisać nazwę naszej biblioteki *Bitwa Morska*, następnie wybrać *Create*. Na dysku zostanie zapisana biblioteka, jednocześnie wyświetli się okno, w którym należy podać nazwę naszego pliku np. *Bitwa Morska TopApp*. Będzie to panel główny naszej gry. Teraz klikając w menadżerze projektu na symbolu biblioteki z menu lokalnego należy wybrać New > VI, zostanie otwarty nowy plik który proponuję zapisać w naszej bibliotece pod nazwą *Nowa Gra*. Będzie to program umożliwiający przygotowanie nowej gry. Na razie zostawmy te programy i przygotujmy podprogramy pomocnicze (subVi), wykorzystamy je później do tworzenia aplikacji. Jako pierwszy przygotujemy program zapisujący do pliku tekstowego tablicę z pozycją statków. Poszczególne elementy wiersza będą oddzielone od siebie przecinkami, natomiast wiersze znakiem nowej linii. Na **rysunku 56** przedstawiono diagram takiej funkcji.



Rysunek 57. Diagram funkcji Odczyt Tablicy z Pliku.vi



Rysunek 58. Diagram funkcji Ilość elementów w tablicy.vi



Rysunek 59. Diagram funkcji Ścieżka Dostępu do Plansz.vi

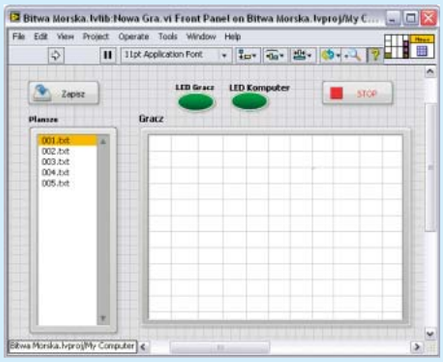
Zapis do pliku realizowany jest funkcją *Write to Text File* bez przekazania ścieżki dostępu. Dlatego po jej wywołaniu otworzy się okno dialogowe, w którym należy wpisać nazwę pliku wraz z *rozszerzenie .txt*. Należy też zdefiniować zaciski funkcji, poprzez kliknięcie w okienku terminali na wybranym polu a następnie na elemencie panelu czołowego, który chcemy z nim połączyć. Proponuję w właściwościach *VI* w kategorii *Documentation* dodać krótki opis funkcji i zacisków.

Przygotujmy teraz funkcje odczytującą wcześniej zapisane dane, z pliku tekstowego i konwertującą je do tablicy tak, aby po jej wywołaniu otrzymać tablicę dwuwymiarową. Rysunek 57 przedstawia odpowiedni diagram.

Przygotujmy teraz funkcję liczącą ile określonych elementów znajduje się w tablicy. Przyda się ona do określenia ilości okrętów jak również celnych strzałów. Jej diagram został przedstawiony na **rysunku 58**. Funkcja jest prosta dlatego nie będę opisywał jej szczegółowo. Ale zwróćmy uwagę na dwie kontrolki. Szukany Element (o) i Zadana ilość elementów (20). W nawiasach zaznaczyłem wartości domyśle. W przypadku nie podłączenia tych zacisków do żadnej zmiennej pobrane zostaną właśnie te wartości. Aby to było możliwe należy wykonać dwie czynności:

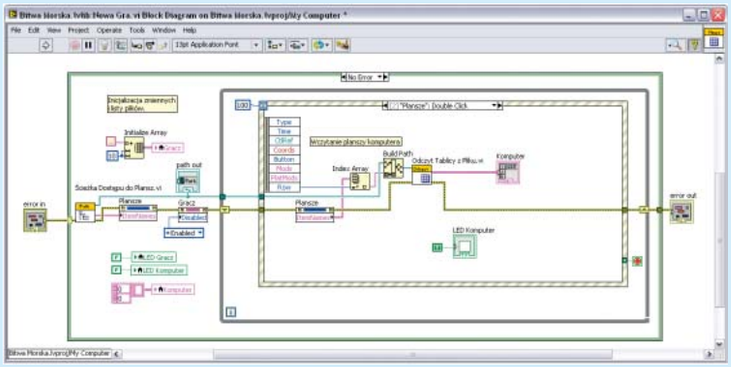
- wpisać wartości do kontrolek, następnie kliknąć na kontrolce *(w panelu czołowym)* lub ikonie *(na diagramie)* i z menu lokalnego wybrać *Data Operations* → *Make Current Value Default*. W ten sposób wartość wpisana aktualnie do kontrolki staje się domyślną.
- w okienku terminali kliknąć na zacisku związanym z daną kontrolką i z menu lokalnego wybrać *This Connection Is* → *Optional*. Dzięki temu kompilator pozwoli zostawić wejście niepodłączone, a do funkcji zostaną przekazane wartości domyślne.

Kolejna funkcja zwraca ścieżkę dostępu do katalogu zawierającego zapisane plansze komputera oraz ich listę. Założyłem, że plansze będą przechowywane w katalogu Plansze znajdującym się w tym samym katalogu co nasz program. **Rysunek 59** przedstawia diagram funkcji. Podobny przykład był robiony w poprzedniej części kursu

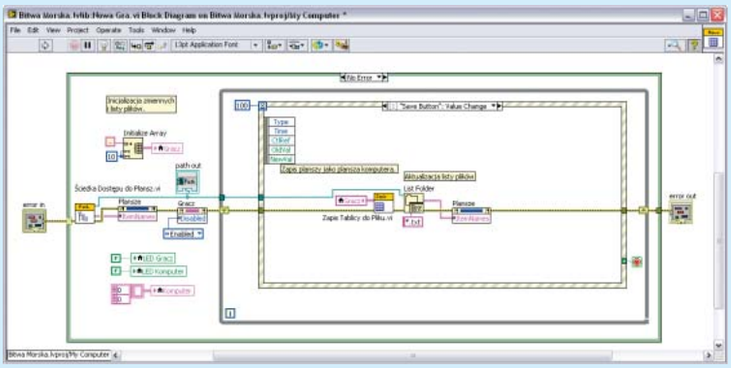


Rysunek 60. Panel czołowy funkcji Nowa Gra.vi

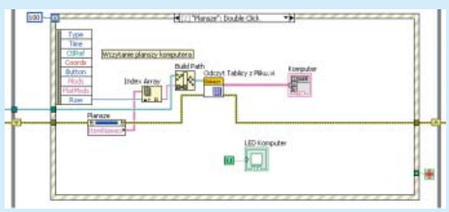
dlatego wspomnę tutaj tylko, że do określenia katalogu należy dwukrotnie użyć funkcji *Strip Path*. Ponieważ plik zapisany jest w bibliotece, więc musimy z ścieżki dostępu „odciąć" nazwę pliku i biblioteki. Korzystając z *List Folder* otrzymujemy listę plików tekstowych znajdujących się w katalogu.



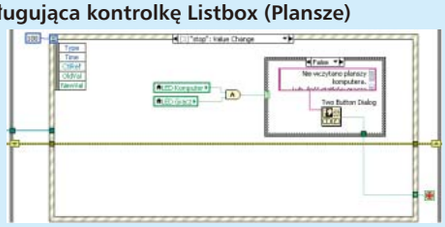
Rysunek 61. Diagram funkcji Nowa Gra.vi



Rysunek 62. Zawartość struktury Event obsługująca przycisk Zapisz



Rysunek 63. Zawartość struktury zdarzeń obsługująca kontrolkę Listbox (Plansze)



Rysunek 64. Zawartość struktury zdarzeń wykonywana po naciśnięciu przycisku STOP

Powróćmy do wcześniej zapisanego pliku *Nowa Gra*. Mamy już wszystko, aby przygotować tę część. Można zacząć od przygotowania panelu czołowego, **rysunek 60** przedstawia przykładowy panel czołowy. Ja korzystałem z elementów zawartych w palecie *Silver*. Potrzebne będą z palety *Silver* ➔ *Boolean* dwie diody i dwa przyciski.

Jedną diodę nazwałem *LED Komputer*. Zapis do niej wartości *TRUE* sygnalizuje wczytanie planszy komputera. Drugą *LED Gracz*. Wartość *TRUE* oznacza, że na planszy znajduje się 20 elementów symbolizujących okręt. Z palety *Silver* ➔ *List, Table & Tree kontrolka Listbox* i wskaźnik *Table*.

Kontrolka *Listbox (Plansze)* wyświetla listę plików tekstowych znajdujących się w katalogu plansze. Wskaźnik *Table(Gracz)* służy jako plansza do rozmieszczenia okrętów. Kliknięcie w wybraną komórkę wpisuje do odpowiedniego elementu tablicy znak „o". Na panelu czołowym znajduje się jeszcze jedna tablica, do której wczytywane są plansze komputera. Nie jest ona widoczna, ponieważ została umieszczona poza widzianym obszarem okna, aby gracz nie mógł zobaczyć okrętów komputera. Musi natomiast być podłączona do zacisku wyjściowego, aby przekazać dane funkcji nadrzędnej.

Gdy panel przedni jest gotowy należy przejść do diagramu. **Rysunek 61** przedstawia diagram z widocznym oknem struktury zdarzeń odpowiedzialnym za wpisywanie symboli okrętów do tablicy. Z lewej strony następuje inicjalizacja wszystkich zmiennych i wczytanie listy plików znajdujących się w katalogu Plansze. Korzystając z *Property Node (ItemNames)* zapisujemy do kontrolki *Listbox (Plansze)* listę plików. Aby to zrobić należy kliknąć na ikonie kontrolki i z menu lokalnego wybrać *Create* ➔ *Property Node* ➔ *ItemNames*. Po wstawieniu na diagram trzeba jeszcze zmienić terminal, aby możliwe było zapisanie do niego wartości. Robimy to wybierając z menu lokalnego *Change To Write*. Analogicznie wstawiamy *Property Node* dla *Table (Gracz)* wybierając z listy odpowiednie właściwości. Widoczna na rysunku ramka struktury zdarzeń wykona się po kliknięciu wewnątrz tabeli Gracz. Na początku korzystając z odpowiedniego *Property Node (EditPos)* odczytujemy, w której komórce nastąpiło kliknięcie, następnie w tabeli *Gracz* zamieniamy element na odpowiedniej pozycji wpisując znak „o" i sprawdzamy ile takich znaków znajduje się w tabeli. Jeśli jest ich dwadzieścia zawarta w strukturze *Case* funkcja *One Button Dialog* wyświetli odpowiedni komunikat, a *Property Node (Disabled)* ustawi tabelę jako nieaktywną.

Rysunek 62 przedstawia zawartość ramki zdarzeń wykonywanej po naciśnięciu na przycisk *Zapisz*. Wywołana jest tutaj wcześnie przygotowana funkcja zapisująca tabelę do pliku tekstowego, oraz uaktualniona zostaje lista plików w kontrolce *Plansze*.

Na **rysunku 63** jest przedstawiony diagram struktury zdarzeń wykonywany po podwójnym kliknięciu w kontrolkę *Listbox (Plansze)*. Z zacisku *Row* na lewej krawędzi struktury pobieramy numer wiersza, w którym nastąpiło

kliknięcie. Property *Node (ItemNames)* dostarcza tablicy zawierającej listę plików, *Index Array* zwraca nazwę pliku, która jest dopisywana do ścieżki dostępu. Wcześniej przygotowana funkcja *Odczyt Tablicy* z *Pliku.vi* odczytuje plik i przekazuje jego zawartość do niewidocznego na panelu czołowym wskaźnika *Komputer*.

Rysunek 64 przedstawia okno struktury zdarzeń, które wykona się po naciśnięciu na przycisk *Stop*. Korzystając z zmiennych lokalnych sprawdzamy stan diod LED sygnalizujących wczytanie plansz, gdy wartość którejkolwiek z nich jest równa *False* jest wyświetlany odpowiedni komunikat. W przeciwnym razie program się zamyka.

Proszę zwrócić uwagę, że na diagramie znajdują się dwa elementy *error in* i *error out*. Nie są one konieczne, ale chciałem pokazać w jaki sposób zrobić obsługę błędów. Jeśli we wcześniejszych procedurach powstał błąd, przekazana o tym informacja za pomocą kontrolki *error in* pozwoli na skierowanie programu do ramki *error* struktury *case*, gdzie można przygotować obsługę błędu. Gdy nie ma błędów to wykonuje się zawartość ramki *No Error*. W naszym przypadku strukturą tą został objęty cały program.

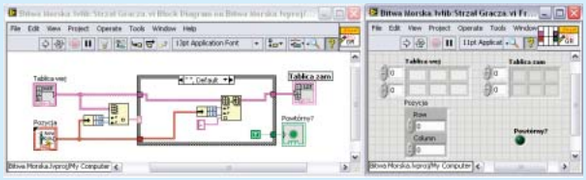
Program ten będzie wywoływany z widocznym panelem czołowym. Po naciśnięciu przycisku *Stop* panel się zamknie. Aby to było możliwe należy kliknąć prawym klawiszem myszki w okienku ikony wybrać *Vi Properties* i w kategorii *Window Appearance* ustawić *Dialog*. Przed przygotowaniem ostatniego elementu gry zrobimy jeszcze trzy pomocnicze funkcje.

Pierwsza to *Strzał Gracza.vi*, jej diagram przedstawia **rysunek 65**. Funkcja ma za zadanie wpisanie do tablicy odpowiedniego symbolu, gdy na wskazanej pozycji znajduje się znak spacji wpisuje znak „–”, gdy znajduje się znak okrętu „o” wpisuje „x” oznaczający trafienie. Dodatkowo sprawdza czy wskazana komórka nie zawiera znaku „–” lub „x” oznaczających, że oddano w nią już strzał. Jeśli tak to sygnalizuje to wartością FALSE w wskaźniku *Powtórny?*. Przyda się to przy następnej funkcji.

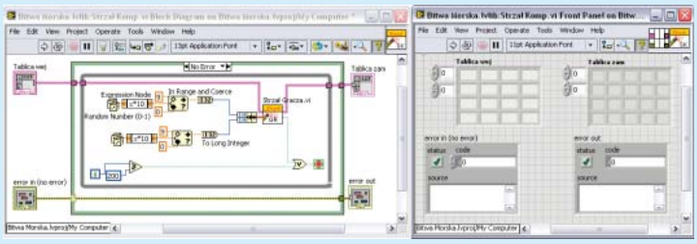
Druga to *Strzał Komp.vi* jej diagram przedstawia **rysunek 66** funkcja ma za zadanie oddanie strzału do planszy gracza. Aby nie komplikować diagramu, zastosowałem proste podejście. Korzystając z *Random Number* losuję dwie przypadkowe liczby z przedziału od 0 do 1. Następnie zostają one pomnożone przez 10. W ten sposób otrzymuję liczby większe od 1. Przy pomocy *In Range and Coerce* ograniczam ich zakres do przedziału od 0 do 9 następnie konwertuję je do liczb całkowitych. W ten sposób otrzymuje losowe współrzędne strzału. Korzystając z funkcji *Strzał Gracza.vi* zapisuję odpowiedni symbol do tablicy. Zwraca ona informację czy na danej pozycji był oddany strzał, jeśli tak, to ponawiam losowanie. Pętla *while* może tutaj wykonać się maksymalnie 200 razy, co oznacza do 200 ponownych prób. Proponuję napisanie bardziej inteligentnego algorytmu.

Trzecią i ostatnią już funkcją, którą musimy przygotować jest *Maskowanie Statków.vi*. Jej zadaniem jest ukrycie pozycji okrętów na planszy komputera. Diagram przedstawia **rysunek 67**. Funkcja przeszukuje tablicę wejściową i w miejsce znaku okrętu „o” wpisuje spację.

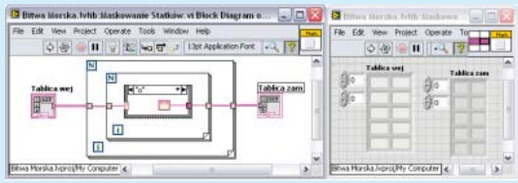
Oczywiście wszystkie wcześniej przygotowane funkcje, mające być użyte jako subvi, muszą mieć zdefiniowane zaciski wejściowe i wyjściowe. Pozwoli im to pobierać i przekazywać dane do nadrzędnego vi z którego są



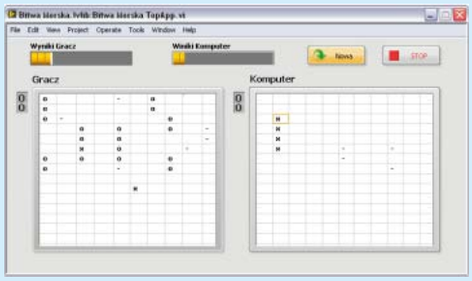
Rysunek 65. Diagram funkcji Strzał Gracza.vi.



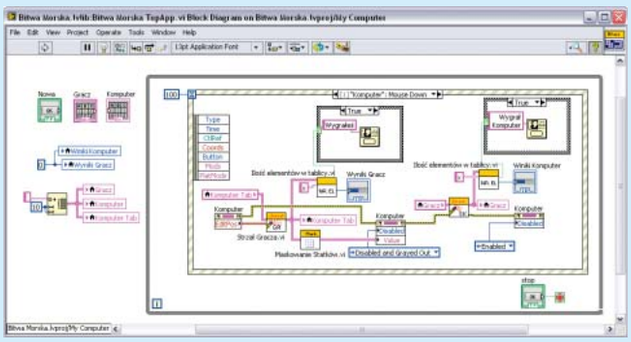
Rysunek 66. Diagram funkcji Strzał Komp



Rysunek 67. Diagram funkcji Maskowanie Statków.vi



Rysunek 68. Przykładowy panel główny

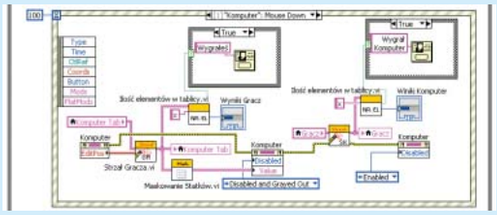


Rysunek 69. Diagram gry z strukturą zdarzeń wykonywaną po naciśnięciu przycisku Nowa

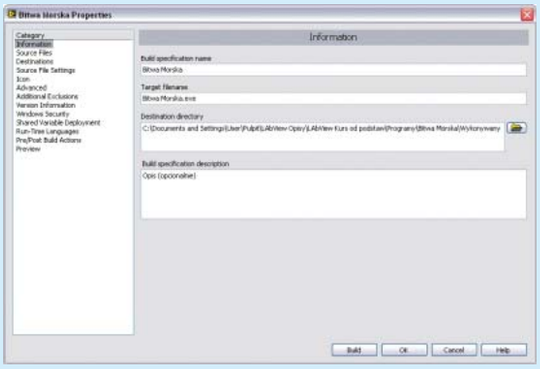
wywoływane. Możemy z tego zrezygnować tylko w nadrzędnej aplikacji która nie będzie wywoływana przez inny nadrzędny program.

Pozostało jeszcze przygotowanie panelu głównego. **Rysunek 68** przedstawia przykładowy wygląd panelu głównego. Oprócz widocznych elementów znajduje się tutaj jeszcze jedna tablica o nazwie *Komputer Tab* przechowująca pozycję statków komputera. Można ją wstawić poza widocznym obszarem okna lub ustawić jako ukrytą klikając na ikonie w oknie diagramu i z menu lokalnego wybierając *Hide Indicator*.

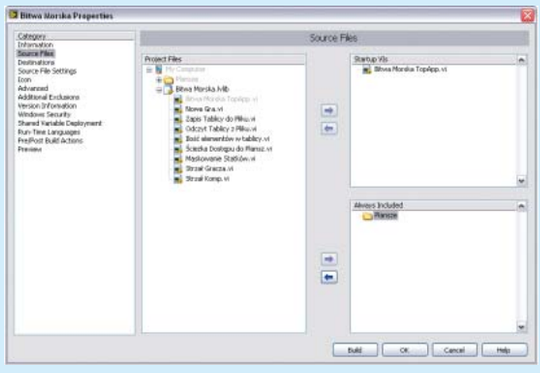
Diagram naszej gry przedstawia **rysunek 69**. Z lewej strony przed pętlą for następuje inicjalizacja wszyst-



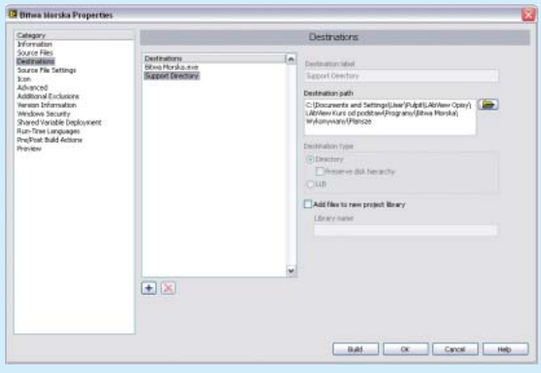
Rysunek 70. Ramka struktury zdarzeń wykonywana po kliknięciu w planszy Komputer



Rysunek 71. Okno kategorii Information



Rysunek 72. Okno kategorii Source Files



Rysunek 73. Okno kategorii Destinations

kich zmiennych. Wewnątrz pętli znajduje się struktura zdarzeń na rysunku widać ramkę, w której następuje wywołanie wcześniej przygotowanego programu *Nowa Gra.vi*. Po jego zamknięciu zwracane wartości są zapisywane do odpowiednich kontrolek. Dodatkowo, czyszczona jest plansza komputera, plansza z okrętami komputera zapisywana jest do ukrytej zmiennej *Komputer Tab*. Natomiast z okrętami gracza, poprzez zmienną lokalną *Gracz*, do planszy gracza.

Na **rysunku 70** przedstawiony został fragment diagramu znajdującego się w strukturze zdarzeń i wykonywanego po kliknięciu w planszę *Komputer*. Na początku, poprzez *Property Node (EditPos)* pobierany jest numer komórki, w który nastąpiło kliknięcie. Z ukrytej kontrolki *Komputer Tab* poprzez zmienną lokalną jest pobierana tablica zawierająca pozycję okrętów komputera, funkcja *Strzał Gracza.vi* wpisuje na wskazanej pozycji odpowiedni znak zależny od celności strzału. Zmodyfikowana tablica jest zapisywana ponownie do zmiennej. Jednocześnie ta sama tablica po zamaskowaniu pozycji okrętów wyświetlana jest w tablicy *Komputer* na panelu czołowym. Wykonuje się to poprzez *Property Node (Disabled) (Value)*, w tym samym miejscu tablica ustawiana jest jako nieaktywna.

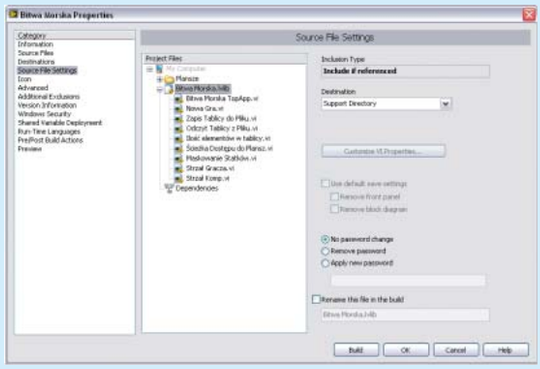
Funkcja *Ilość elementów w tablicy.vi* sprawdza ile oddano celnych strzałów i wyświetla wynik gracza. W przypadku, gdy celnych strzałów jest 20, funkcja *One Button Dialog* umieszczona w strukturze *Case* wyświetla komunikat „Wygrałeś". Następnie jest wywoływana funkcja *Strzał Komp.vi*, do której przez zmienną lokalną przekazana jest zawartość tablicy gracza. Wynik zapisany zostaje również do tej tablicy i wyświetlony na panelu czołowym. Podobnie jak poprzednio sprawdzana jest ilość celnych strzałów.

Program jest już gotowy, ale należy skonfigurować wygląd okna aplikacji i sposób uruchamiania. W tym celu należy otworzyć okno *Vi Properties* w kategorii *Window Appearance* ustawić *Top-level application window*. Ukryje to zbędne ikony paska narzędziowego. Proponuję jeszcze przejść do kategorii *Execution* i zaznaczyć *Run when opened*. Wówczas po otwarciu okna nasz program natychmiast się uruchomi. Tak przygotowany program będzie można uruchomić tylko dysponując LabView. Jeśli chcemy przenieść go na inny komputer musimy przygotować plik wykonywalny (*.exe).

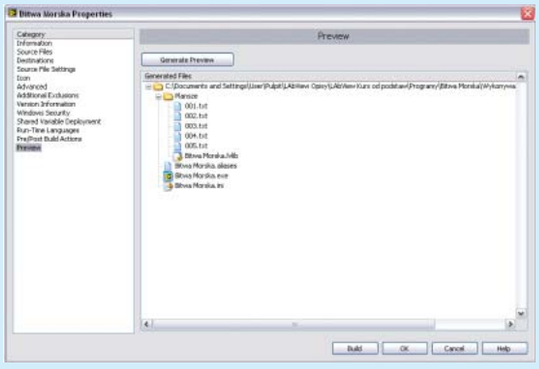
Generowanie plików exe

Programy przygotowane w LabView nie są całkowicie samodzielnymi programami. Do ich uruchomienia konieczne jest zainstalowanie środowiska uruchomieniowego LabView Run-Time w odpowiedniej wersji, zgodnej z LabView, w którym przygotowana została aplikacja. Środowisko to instaluje się zwykle wraz z LabView. Jeśli program ma być uruchomiony na innym komputerze możliwe jest pobranie go z strony producenta. Po jego zainstalowaniu wystarczy skopiować przygotowaną aplikację i uruchomić ją jak każdy program pod systemem Windows. Oczywiście LabView umożliwia przygotowanie pliku instalacyjnego zawierającego wszystkie niezbędne elementy do zainstalowania i uruchomienia aplikacji na dowolnym komputerze, ale standardowo dostępne jest to tylko w najwyższych wersjach LabView Professional i Developer Suite.

Na przykładzie naszej gry pokażę tylko niezbędne czynności pozwalające prawidłowo wygenerować plik exe. Rozpoczynamy od kliknięcia w oknie projektu na ikonie *Build Specifications* i wybraniu z menu lokalnego *New* → *Application (exe)*. Otworzy się okno konfiguracyjne jak na **rysunku 71**. W polu *Build specification name* wpisujemy nazwę konfiguracji widoczną później w menadżerze projektu np. *Bitwa Morska*. W polu *Target Filename* wpisujemy nazwę wygenerowanego pliku exe np. *Bitwa Morska.exe*. W polu *Destination directory* określamy katalog, w którym zostaną zapisane utworzone pliki. W polu *Build specification description* możemy dodać krótki opis danej konfiguracji. W kategorii *Source Files* (**rysunek 72**) określamy, który z plików jest główną aplikacją. Widzi-



Rysunek 74. Okno kategorii Source File Settings



Rysunek 75. Okno kategorii Preview

my to w oknie *Startup Vis*, będzie on otwierany jako nasz program. W polu *Always Included* umieszczamy pliki, które powinny być dołączone do programu. W naszym wypadku jest to folder zawierający zapisane ustawienia statków komputera. Musi być on wcześniej dodany do projektu. Pozostałe pliki będące podprogramami w naszej aplikacji zostaną dodane automatycznie.

W kategorii *Destinations* (**rysunek 73**) należy tak jak na rysunku w polu *Destination path* ostatni element ścieżki zmienić z *data* na *Plansze*. Jest to konieczne ze względu na sposób wyznaczania ścieżki dostępu do katalogu *Plansze*. W kategorii *Source File Settings* (**rysunek 74**) w polu *Destination* musimy wybrać opcję *Support Directory*. Jest to konieczne, aby zachować konfigurację plików pozwalającą prawidłowo wyznaczyć ścieżkę do katalogu *Plansze*. Dla tej konfiguracji wszystkie *subvi* zostaną zgromadzone w oddzielnej bibliotece. Możliwe jest również zgromadzenie wszystkich funkcji w jednym pliku exe.

Teraz można przejść do kategorii *Preview* (**rysunek 75**) i klikając w przycisk *Generate Preview* zobaczyć, jakie pliki zostaną wygenerowane i jaka będzie ich organizacja. Po wciśnięciu *Build* nasz program zostanie wygenerowany.

Podsumowanie

Mam nadzieją, że ułatwiłem naukę programistom początkującym w tym środowisku. Starałem zwrócić uwagę na szczegóły oczywiste dla doświadczonych programistów, a początkującym mogące sprawiać problemy. Możliwości środowiska są ogromne i pozostało wiele zagadnień, które warto poruszyć. Zachęcam do analizy przykładów dostarczonych z oprogramowaniem. Są dobrze opisanie i nieskomplikowane.

Wiesław Szaj
wszaj@prz.edu.pl