Nasza pierwsza gra - kurs programowania AmigaOS i C - część

Asman

W dziewiątym, ostatnim odcinku cyklu, zajmiemy się edytorem, który to umożliwi nam tworzenie plansz. Dodamy także największy możliwy wynik, czyli tabelę najlepszych wyników.

Pierwsza kwestia nasuwająca się na myśl w sprawie edytora: czy ma być osobnym programem, czy też powinniśmy go wbudować w grę. My skupimy się na tym drugim podejściu. Zatem dodamy odpowiedni wpis w menu tak, aby użytkownik widział, że jest dostępny edytor i po kliknięciu nastąpi przejście do edycji etapów. Tutaj powinno się wybrać jedną z możliwych kostek - a jest ich kilka - i za pomocą myszki nanieść w odpowiednie miejsce. Oczywiście musi być możliwy w jakiś sposób wybór planszy i, najważniejsza opcja, zapisanie tak powstałej planszy gry. Wygląda na niedużo, lecz niestety czeka nas sporo pracy, bo to co napisałem jest bardzo ogólne i dosyć mgliste. Warto na przykład zadać sobie pytanie gdzie owe kostki, które ma wybierać użytkownik, będą umieszczone. Na wszystkie te pytania mam nadzieję uda mi się odpowiedzieć.

Jeżeli chodzi o kostki, to umieścimy je w osobnym okienku, które będziemy musieli otworzyć, obsłużyć sygnały pochodzące od niego i zamknąć, gdy już skończymy edycję i zechcemy przejść do ekranu tytułowego. Wyobraźmy sobie, że już jest zrobione okno edytora, teraz chcemy zająć się obsługą sygnałów. Jak pamiętamy to w boxo.c jest główny rdzeń gry zawarty w funkcji loop i tam też umieścimy kod obsługi, kładąc go przed wywołaniem Wait.

```
if (g_bEditMode) {    sigSet |= g_nEditorWinSignal; }
```

Jak widzimy, nie ma tu nic magicznego. Nawiasem mówiąc, zmienną g_bEditMode umieszczamy w module boxo, dodajemy za pomocą operatora alternatywy bitowej numer sygnału do wcześniejszych. Jasne jest, że dodajemy go tylko, gdy warunek jest spełniony, co będzie równoważne z otwartym już oknem edytora tak, aby istniał sygnał dla tego okna. Nie możemy bezpośrednio dodać tego numeru, tak jak poprzednie, gdyż okno edytora może jeszcze wtedy nie istnieć i, jak dobrze pamiętam podczas testowania, to gra zawiśnie. Sama obsługa sygnałów pochodzących od edytora wygląda podobnie jak w przypadku dema gry.

```
if (g_bEditMode) {    if (signals & g_nEditorWinSignal)    {      g_bEditMode = signalsEditorWindow();    } }
```

Dodam tylko, że gdy signalsEditorWindow zwróci FALSE, to nastąpi zamknięcie edytora. Zanim przejdziemy do modułu editor, opiszę zmiany, jakie dotknęły moduł tile, gdzie trzymane są procedury rysujące po oknie gry kostki, napisy i znaki. Aby możliwe było rysowanie po innym oknie, możemy każdą funkcję zaopatrzyć w dodatkowy parametr struct RastPort* pRPort. Wystarczy wtedy zmienić wszystkie aktualne wywołania tych procedur. Istnieje też inny sposób - trochę bardziej skomplikowany. Zamiast jednej funkcji PrintChar, zróbmy dwie - jedna dla gry (PrintGameChar) a druga dla edytora (PrintEditChar). Do tego będziemy potrzebować dwóch wskaźników na strukturę RastPort (jedna dla gry, a druga dla edytora). Niewątpliwym minusem, poza powtórzeniem funkcji, będzie konieczność ustawiania tych wskaźników i ten ciężar udźwigną moduły window i editor. Pozostaje jeszcze dodanie wpisu "editor" do menu w title, który uruchomi edytor, za pomocą znanego nam wskaźnika na funkcję.

W końcu dotarliśmy do sedna, czyli do modułu editor. Tutaj, podobnie jak w poprzednich odcinkach, dodajemy go w typowy sposób - dwa nowe pliki i aktualizacja makefile, nie zapominając go umieścić w initTable zaraz po inicjacji konfiguracji. Kod inicjujący i kończący są niewielkie. Pierwszy z nich nic nie robi poza zwróceniem RT_OK, a drugi ustawia tryb edycji na FALSE i jeśli to możliwe, zamyka okno samego edytora. Przypatrzmy się bliżej funkcji uruchamiającej edytor (to właśnie wskaźnik na nią jest umieszczony w title).

```
void StartEditor(void) {    g_bEditMode = TRUE;    m_pEditWin = (struct Window*)OpenWindowTagList(NULL,
m_editWinTags);    if (NULL == m_pEditWin)    {      g_pFnc = &Title;      KillEditor();      return ;    }
```

Asman


```
g_nEditorWinSignal = 1L UserPort->mp_SigBit;    m_pRpEdit = m_pEditWin->RPort;    g_pEditRPort = m_pRpEdit;
showLevelToEdit();    printEditor();     m_nTile = TILE_EMPTY;    m_bTileChanged = FALSE;    m_nSaveCnt = 0;
g_pFnc = &editorLoop; }
```

Na początku ustawiamy tryb edycji na TRUE, dzięki temu w pętli głównej w boxo będziemy mogli odbierać i analizować sygnał dla edytora. Dalej następuje otwarcie okna, gdzie umieścimy kostki z gry i krótką informację z pomocą, mówiącą między innymi, jakich klawiszy można używać. W przypadku niepowodzenia przy otwieraniu okna, powracamy do obrazka tytułowego. Następnie ustawiamy numer sygnału potrzebny przy komunikacji z funkcją loop, a wskaźnik na strukturę RastPort używaną w oknie edytora zapamiętujemy dla funkcji rysujących po nim. Metoda showLevelToEdit pokazuje planszę do edycji w oknie gry, bo tam będziemy ją zmieniać a printEditor ma za zadanie narysować wszystkie rzeczy związane z oknem edycji, czyli po lewej stronie kostki a po prawej pomoc. Zmienna m_nTile przechowuje aktualny numer kostki będący w edycji i na starcie będzie to pusty klocek. Zadaniem m_bTileChanged, będzie poinformowanie funkcji rysującej kostkę, że użytkownik zmienił ją na inną. Ostatnia zmienna m_nSaveCnt będzie używana podczas zapisywania etapów. Na końcu znanym nam sposobem, przechodzimy do pętli głównej edytora. Z opisu funkcji StartEditor powoli rysuje się obraz, jak edytor mniej więcej będzie działać. Odrobinę go jednak rozszerzę. Jak wiemy będą dwa okna: pierwsze to okno gry, na którym będziemy widzieć planszę i za pomocą myszki będziemy nanosić kostki, tworząc tym samym nowy poziom. Dodatkowo, aby ułatwić życie twórcy, pod myszką będziemy widzieć wybrany klocek, a klikając lewym przyciskiem myszy będziemy stawiać kostkę, a prawym ją usuwać (czyli stawiać kostkę pustą). Drugie okno to okno edytora. Tu będziemy mieć możliwość wybrania kostki i zobaczenia pomocy. Sam wybór kostki będzie możliwy na dwa sposoby: za pomocą klawiatury, używając klawiszy od 1 do 5 oraz za pomocą myszki, klikając zwyczajowo na obrazek przedstawiający kostkę. Zapis poziomu będzie możliwy tylko za pomocą myszki. Aby użytkownik był pewny, że zapis się powiódł, ikonka przedstawiająca dyskietkę, na chwilę zmieni się i pojawi się na niej napis "ok". W przeciwnym razie będzie to napis "bad". Przejdźmy do głównej pętli edytora.

```
static void editorLoop(void) {    checkExitFromEditor();    checkMouseInEditor();    checkKeysInEditor();
 checkMouseInGame();    changeEditTile();    animateEditorArrow(); }
```

Nie będę podawał kodu dla wywoływanych funkcji, bo zajęło by to zbyt dużo miejsca, przedstawię za to dwie najciekawsze procedury: checkKeysInEditor i changeEditTile. Ujmując rzecz z grubsza to w tej funkcji zachodzą trzy rzeczy. Pierwsza i najważniejsza to procedury z przedrostkiem check – obsługa interfejsu użytkownika, czyli przetwarzanie zdarzeń pochodzących od zarówno od klawiatury, jak i myszki. Mamy tu trzy funkcje do obsługi edytora i jedną obsługującą grę, bo przecież tam klikamy myszką stawiając odpowiednią kostkę. Spójrzmy na checkKeysInEditor.

```
static void checkKeysInEditor(void) {    struct KeyEdit    {      int key;      int pos;      int tile;    };     struct KeyEdit
keyTable[] =     {      {KEY_1, EDIT_EMPTY_POS_Y, 0},      {KEY_2, EDIT_WALL_POS_Y, 1},      {KEY_3,
EDIT_ROCK_POS_Y, 2},      {KEY_4, EDIT_SHIP_POS_Y, 3},      {KEY_5, EDIT_EXIT_POS_Y, 4},    };     if (g_bLeft)
   {      g_bLeft = FALSE;       if (g_nLvlNumber > 0)      {         g_nLvlNumber--;         showLevelToEdit();      }    }    else if
(g_bRight)    {      g_bRight = FALSE;       if (g_nLvlNumber
```

W tej funkcji sprawdzamy klawisze i podejmujemy stosowną akcję, czyli dla klawiszy od 1-5 będzie to zmiana kostki, a w przypadku kursorów, zmiana poziomu. Aby ułatwić sobie zadanie i uczynić kod odrobinę bardziej przejrzystym, zamieniłem ciągi instrukcji if else na pętle. Do tego celu potrzebowałem struktury składającej się z kodu klawisza, pozycji pionowej, potrzebnej, aby pokazać użytkownikowi, co aktualnie zostało wybrane i na ostatnim miejscu umieściłem numer kostki. W ferworze walki są to magiczne liczby, które powinno się zamienić na stałe z pliku tile.h, co pozostawiam czytelnikowi. Mając strukturę, tworzymy tablicę, aby w pętli się po niej przesuwać, co też jest robione pod koniec funkcji. W klasyczny sposób, czyli za pomocą "ifów", sprawdzamy klawisze odpowiadające za kursory (lewy i

Asman

prawy) i przy udziale warunku sprawdzającego czy czasem nie wyszliśmy poza ustaloną liczbę plansz, zmieniamy poziomy.

Najbardziej tajemnicza funkcja wywoływana w editorLoop to changeEditTile. Odpowiada ona za poprawne wyświetlenie kostki na ekranie gry, przy czym dopóki użytkownik nie naciśnie przycisku myszki, to poziom nie ulega zmianie. Poziom gry składa się z kostek 16x16 i w takiej siatce stawiamy też kostki. Zatem jeśli pobieramy pozycję myszki, to musimy ją odpowiednio "przyciąć", aby funkcja stawiająca kostkę, wpasowała ją dokładnie tam, gdzie należy. Dlatego wykonujemy małą sztuczkę związaną z iloczynem bitowym - zamiast dzielenia całkowitego przez 16, mnożymy przez 16. Ten wybieg działa niestety tylko dla potęg dwójki, ale tak się szczęśliwie składa, że 16 nią jest. Czy to nie przypadek? Jasne, że nie. Wybierając szerokość i wysokość kostek powinniśmy kierować się potęgami dwójki. Wtedy możemy wykorzystać różne ciekawe ich własności (na przykład 2^n + 2^n = 2^(n+1), czy też przesunięcia bitowe w lewo czy w prawo zamiast dzieleń i mnożeń). To taka dygresja. Wracając jednak do funkcji changeEditTile. Po pobraniu pozycji, musimy sprawdzić czy mieścimy się w wymiarach poziomu. Jeśli nie, to wychodzimy z procedury. Ostatni warunek sprawdzający wykonujemy tylko wtedy, gdy nowa pozycja zmieniła się o przynajmniej jedną kostkę od starej pozycji chyba, że klocek został zmieniony przez użytkownika, to wtedy musimy go podmienić, tak by widział, że nastąpiła zmiana. Po spełnieniu warunku przywracamy kostkę ze starej pozycji i stawiamy wybraną przez użytkownika na nowej pozycji. Do pełni sukcesu pozostała do opisania funkcja animateEditorArrow, która, jak łatwo się domyślić, animuje strzałkę w oknie edytora, informując, co aktualnie mamy wybrane. Oczywiście to nie wszystkie funkcje w editor, ale pozostałe są na tyle proste, że po tylu lekcjach czytelnik będzie w stanie je zrozumieć samodzielnie.

```
static void changeEditTile(void) {    const WORD nPosX = GetGameMouseX() & ~0x0f;     if (nPosX
((g_nLvlTHeight-1)*g_nTileWidth))    {       return;    }     BOOL bChanged = nPosX != m_nOldPosX || nPosY !=
m_nOldPosY || m_bTileChanged;     if (bChanged)    {       m_bTileChanged = FALSE;       UBYTE* pLvl = g_pAllLevels +
g_nLvlNumber * (g_nLvlTWidth*g_nLvlTHeight);       UBYTE* p = pLvl + m_nOldPosX / 16 + (m_nOldPosY / 16) *
g_nLvlTWidth;       PutGameTile(m_nOldPosX, m_nOldPosY, *p);       m_nOldPosX = nPosX;       m_nOldPosY = nPosY;
     PutGameTile(m_nOldPosX, m_nOldPosY, m_nTile);    } }
```

Nadszedł czas na dodanie tabeli najlepszych wyników. Oczywiście, aby go zapamiętać, zapiszemy go w postaci pliku o znajomo brzmiącej nazwie highscore. Sprawa z pozoru błaha, ale i tu czyha na nas parę niebezpieczeństw. Przede wszystkim brak wyżej wymienionego pliku przez przypadkowe czy to celowe skasowanie, niewłaściwa zawartość bądź długość pliku, uniemożliwiając tym samym poprawne wyświetlenie i walidację. Zaczynając od samego początku - najlepszy wynik musimy w jakiś sposób przechowywać - nazwijmy go g_nHighscore. Oczywistym jest, że musi być on w stanie przechowywać także maksymalną ilość punktów, którą można zdobyć w grze, a co za tym idzie obydwie zmienne zarówno g_nScore, jak i g_nHighscore muszą być tego samego typu. Najlepszym rozwiązaniem będzie umieszczenie najlepszego wyniku w osobnym module. Zwyczajowo dodajemy dwa nowe pliki highscore.c i highscore.h do naszego projektu, aktualizujemy zasady w makefile i dodajemy odpowiednią linię w initTable w pliku boxo.c tuż za inicjacją bibliotek. Sam moduł będzie bardzo skromny, bo będzie składał się tylko z dwóch funkcji: InitHighscore, KillHighscore. Tam też umieścimy zmienną g_nHighscore. Oto i ciało modułu highscore:

```
#include "highscore.h" #include "fileIO.h"  int g_nHiscore; static char* name = "highscore";
/*==========================================================================*/ int
InitHighscore(void) {    g_nHiscore = 0;    ReadFile(name, (UBYTE*)&g_nHiscore, sizeof(int));    return RT_OK; }
/*--------------------------------------------------------------------------*/ void KillHighscore(void) {    WriteFile(name,
(UBYTE*)&g_nHiscore, sizeof(int)); } /*--------------------------------------------------------------------------*/
```

Wydawać by się mogło, że w InitHighscore zerowanie zmiennej nie jest potrzebne, bo przecież zaraz później ReadFile wczyta do niej wartość z pliku. Nic bardziej mylnego. Wystarczy sobie zdać sprawę, że funkcja czytająca może nie

Asman

zadziałać, gdyż plik nie istnieje, dlatego właśnie ustawiamy na zero zmienną g_nHighscore. Konstrukcja wskaźnik na adres zmiennej umożliwi procedurze ReadFile wczytanie wartości bezpośrednio w g_nHighscore. A trzeci parametr w wywołaniu funkcji oblicza rozmiar zmiennej. KillHighscore ma tylko jedno zadanie - zapisać najlepszy wynik do pliku i to czynimy wykorzystując do tego celu WriteFile.

Oprócz modułu highscore trzeba zadbać jeszcze o prawidłowe działanie najlepszego wyniku w naszej grze. Pierwszym miejscem, gdzie powinna być logika z tym związana, to miejsce, gdzie nowa gra się rozpoczyna, czyli u nas w funkcji NewGame. Zaraz za inicjalizacja początkowej ilości punktów dodajemy prosty warunek sprawdzający czy punkty nie są większe niż aktualny najlepszy wynik i jeśli tak jest, to wtedy highscore jest takie samo jak punkty. Drugim i ostatnim miejscem, gdzie powinno się sprawdzić punkty i najlepszy wynik to miejsce, kiedy gracz doszedł do wyjścia i uzyskał dodatkowe punkty. W tym przypadku postępujemy w taki sam sposób jak poprzednio. Na zakończenie warto wspomnieć, że wyświetlanie highscore znajduje się w bottomPanel w funkcji UpdateHiscoreOnPanel.

Myślę, że po dziewięciu odcinkach temat pierwszej gry w języku C został opisany, gdybym o czymś zapomniał a jest jeszcze warte napisania, w ramach tego tematu, to proszę o informacja na forum PPA. Dziękuje i zachęcam do zadawania pytań oraz eksperymentowania z kodem.

Artykuł oryginalnie pojawił się w czternastym numerze Polskiego Pisma Amigowego.