Nasza pierwsza gra - kurs programowania AmigaOS i C - częś

Asman

(c) Polski Portal Amigowy (www.ppa.pl)

W ósmej odsłonie tworzenia gry skupimy się na następujących rzeczach: ulepszymy menu w obrazku tytułowym, które zaczęliśmy w poprzednim odcinku, dodamy możliwość konfigurowania klawiszy poprzez menu Options oraz uatrakcyjnimy rozgrywkę umieszczając licznik punktów w dolnym panelu.

Aby zająć się porządnie konfiguracją klawiszy, musimy zrobić odpowiednie menu z możliwością edycji pól tak, aby były widoczne zmiany dokonane przez gracza. W siódmej części w title.c w bardzo prosty sposób napisaliśmy takie przykładowe menu. Niestety nie jest ono wystarczająco elastyczne dla naszych nowych potrzeb, więc niewiele myśląc zmienimy je. Początkującym twórcom takie zmiany mogą się kojarzyć z niewłaściwą drogą, bo przecież wystarczy tylko nieznacznie rozbudować to, co zaczęliśmy robić, dodając brakujące elementy, a to, co już jest zostawić, skoro zostało to już przetestowane. Ja uważam inaczej z dwóch powodów. Po pierwsze, ulepszanie przez rozbudowanie przy złym podejściu może skutkować zawiłym kodem, a po drugie warto programować na różne sposoby (a nuż wymyśli się coś zupełnie nowego). W każdym razie tworzymy nowe menu. Standardowo dodajemy nowy moduł menu do naszego projektu. Myślę, że nie sprawi to problemu czytelnikowi, bo temat ten przewijał się w paru odcinkach. Najważniejszą rzeczą w menu będzie odpowiednia struktura, która ułatwi nam życie zarówno przy wyświetlaniu, jak i w poruszaniu się po menu. Będzie to struktura odpowiedzialna za jedną linię menu.

```
typedef struct {    int posX;    int posY;    char* name;    int color; } MenuEntry;
```

Widzimy, że jest to bardzo intuicyjne podejście. Mamy tu pozycję poziomą, pionową, kolor i ciąg znaków do wyświetlenia. Umieszczenie koloru na końcu struktury ma znaczenie praktyczne – służy zmniejszeniu ilości błędów podczas programowania oraz nie musimy się zastanawiać czy najpierw jest kolor a potem pozycje x i y, czy też najpierw są pozycje a później kolor. Mając takie MenuEntry możemy teraz sobie stworzyć przykładowe menu, robiąc odpowiednią tablicę, którą umieściłem w title.c.

```
static MenuEntry m_menuTitle[] = {    {MENU_POS_X, MENU_START_POS_Y, "start", COLORTXT_LIGHT},
 {MENU_POS_X, MENU_OPTIONS_POS_Y, "options", COLORTXT_NORMAL},    {MENU_POS_X,
MENU_EDITOR_POS_Y, "editor", COLORTXT_NORMAL},    {MENU_POS_X, MENU_EXIT_POS_Y, "exit",
COLORTXT_NORMAL}, };
```

Nietrudno się domyślić – MENU_POS_X i MENU_START_POS_Y to przykładowe pozycje zdefiniowane za pomocą dyrektywy #define. Słowo komentarza należy się kolorom. Aby menu na starcie wyglądało jak należy, pierwsza jego linia powinna mieć inny kolor niż pozostałe, aby było wiadomo, że zaczynamy od tego miejsca. Zastanówmy się teraz, w jaki sposób będziemy realizować przemieszczanie się po menu. Oczywiście będziemy się poruszać za pomocą joysticka lub klawiatury. Oprócz tego nasza gra nabrałaby kolejnych rumieńców, gdyby można było użyć do tego celu również myszki. Tym zajmiemy się chwilę później. Jak wiemy, jeden kolor mamy inny w tablicy m_menuTitle i jeśli będziemy odpowiednio nim manipulować, umieszczając kolor COLORTXT_LIGHT w różnych wierszach, to rozwiążemy problem poruszania się po menu. Co więcej – nasze menu zawiera też informacje o ograniczeniach, czyli kiedy gracz już nie może iść do góry czy też do dołu (mówią o tym pozycje pionowe w pierwszej i ostatniej linii w tablicy). Załóżmy, że wystąpił ruch w naszym menu – wtedy musimy na samym początku zamienić kolor ówczesnego podświetlenia na normalny kolor, czyli innymi słowy musimy zgasić podświetloną wcześniej pozycję w menu. Potem trzeba tylko podświetlić napis w nowej pozycji i gotowe. Oto funkcja realizująca to zadanie.

```
void MenuUpdate(MenuEntry* menuTable, int amount, int nPosY) {    int i = 0;    while (i
```

Samo wyświetlenie całego menu jest banalne i nie będę go przedstawiał, gdyż sprowadza się to do jednej pętli, w której wyświetlamy napis o znanej pozycji i kolorze. Ponieważ wcześniejsza wersja funkcji odpowiedzialnej za pokazanie

Asman

(c) Polski Portal Amigowy (www.ppa.pl)

tekstu na ekranie, znajdująca się w module tile, nie miała możliwości zmiany koloru, to ona także została napisana od nowa. Aby ułatwić sobie pracę z tekstem, dodałem też dwie dodatkowe wersje funkcji pokazujące tekst w normalnym i podświetlonym kolorze. Pojawiła się też nowa funkcja – BackgroundToBlack, której zadaniem jest zrobienie czarnego tła w całym oknie. Oprócz wyświetlenia menu będziemy potrzebować też funkcję inicjującą menu, która przechodząc w pętli wszystkie pozycje menu ustawi podświetlenie na pierwszej.

Jak zaznaczyłem już wcześniej, dodamy obsługę myszki w naszym menu. Zaczniemy najpierw od umieszczenia niezbędnych rzeczy w module window. Tam bowiem będą zbierane eventy o przyciskach urządzenia i tam też dołączymy funkcje zwracające pozycje poziomą i pionową myszki. W każdym razie powiększymy ilość tagów okna, dorzucając WA_RMBTrap, który blokuje dostęp do menu (mam tu na myśli główny pasek na samej górze w Workbenchu) dla prawego przycisku naszego elektronicznego stworzenia, a także dopisujemy IDCMP_MOUSEBUTTONS w tagu WA_IDCMP, umożliwiając systemowi generowanie zdarzeń o przyciskach. Oczywiście musimy umieścić odpowiedni blok kodu, który przeanalizuje zdarzenie IDCMP_MOUSEBUTTONS, a sam kod polega na sprawdzeniu odpowiednich stałych w msg_code, takich jak SELECTUP – dla wciśniętego lewego przycisku myszki, SELECTDOWN – dla puszczonego. Podobnie będzie dla prawego przycisku – tu mamy MENUUP i MENUDOWN. W window.c dokładamy funkcje GetMouseY i GetMouseX zwracające pozycję poziomą i pionową myszki, które pobieramy odpowiednio z pól GZZMouseX i GZZMouseY ze struktury Window. Tak uzbrojeni możemy przystąpić do właściwej procedury zawierającej obsługę menu wraz z pozycją y myszy. A zatem na samym początku pobieramy wspomnianą pozycję pionową. Aby dobrać się do linii w menu, musimy, tak na chłopski rozum, pozycję podzielić przez wysokość fonta, wziąć część całkowitą i pomnożyć przez wysokość fonta. Dzięki temu otrzymamy jedną pionową pozycję w obrębie danej linii menu. Ja zastosowałem tutaj sztuczkę, aby uniknąć dzieleń i mnożeń. Bo jak pamiętamy – w naszej grze wysokość fontów wynosi 8 pikseli. Te 8 pikseli w zmiennej posY zajmuje 3 pierwsze bity (licząc oczywiście od lewej). A więc, aby osiągnąć jedną pozycję w linii, wystarczy te 3 bity skasować w zmiennej. I dokładnie to robimy wykorzystując do tego celu negację i koniunkcję (and). Wprowadzamy też zmienną przetrzymującą starą wartość pozycji m_nOldPosY, aby było możliwe usunięcie starego podświetlenia menu oraz ze względów wydajnościowych (przy małych ruchach myszki nie ma potrzeby aktualizacji podświetlania). Jeśli faktycznie użytkownik wykonał ruch w kierunku zmiany linii w menu, to wówczas pozostaje nam sprawdzić nasze menu i pozycję pionową, aby wiedzieć którą linię musimy zaktualizować.

```
static WORD m_nOldPosY = 0;  void MenuMousePosY(MenuEntry* menuTable, int amount) {    const WORD posY =
GetMouseY() & ~0x07;     if (m_nOldPosY != posY)    {       int i;       m_nOldPosY = posY;        for (i = 0; i
```

W powyższym bloku kodu pojawiła się tajemnicza funkcja ArrowUpdate, która zajmuje się aktualizacją strzałki widocznej z lewej strony. Cały wcześniejszy kod dotyczący obsługi strzałki został przeniesiony z modułu title do arrow i właśnie nim zajmiemy się teraz. Najważniejszą funkcją jest ArrowInit, której zadaniem jest inicjalizacja niezbędnych rzeczy do poprawnego funkcjonowania strzałki. Są to pozycja pozioma i pionowa a także granice - dolna i górna. Oprócz tego zerujemy licznik opóźniający animację i ustawiamy początkowy znak z fontów, który przedstawia graficzny symbol strzałki. Pozostałe funkcje to: ArrowAnimate, czyli serce naszego modułu, służące do animacji strzałki, ArrowDown i ArrowUp przemieszczają znak do dołu bądź do góry o jedną linię, która ma stałą wielkość 8 pikseli. Za pomocą funkcji ArrowGetPosY możemy z innego modułu dowiedzieć się o aktualnej pozycji pionowej, a ArrowUpdateY aktualizuję pozycję y.

W module options znajdziemy menu odpowiedzialne za opcje w naszej grze. W poprzednim odcinku dodaliśmy konfigurację dla klawiszy i kolejnym bardzo naturalnym krokiem jest właśnie możliwość ich zmiany. Modyfikacja klawiszy w dowolny sposób musi pociągać za sobą odzwierciedlenie w postaci napisu, jaki to klawisz użytkownik wybrał, a zatem potrzebujemy tablicy zawierającej wszystkie możliwe klawisze. Przedstawię kawałek takiej tablicy, bo zwyczajnie szkoda miejsca na takie tabelki.

Asman

(c) Polski Portal Amigowy (www.ppa.pl)

Na marginesie warto wiedzieć, jaki kod klawisza przychodzi w zdarzeniu IDCMP_RAWKEY. Jest on w msg_code. Siedem młodszych bitów mówi o klawiszu, dla przykładu ESC ma kod 0x45, a ostatni 7 bit (licząc od zera) mówi o tym czy klawisz został naciśnięty. Jeśli gracz wciśnie i puści ESC, to zostaną wygenerowane dwa zdarzenia IDCMP_RAWKEY i msg_code będzie zawierał 0xC5 (0x45+0x80), a później będzie to 0x45.

```
const char *lv_all_keys_tab[] = {   "`",          /*;00 */   "1",          /*;01 */   "2",          /*;02 */   "3",          /*;03 */   "4",
/*;04 */
```

Mamy tu tablicę zawierającą wskaźniki do napisów i poruszając się po indeksach tejże tablicy mamy napis odpowiadający klawiszowi. Jeśli gracz wciśnie na klawiaturze jedynkę, to odpowiada to pierwszemu (liczymy od zera w tym przypadku) elementowi naszej tablicy. Ułożenie napisów w tej tablicy nie jest przypadkowe – jest ono zgodne z kodami otrzymywanymi za pomocą zdarzenia IDCMP_RAWKEY. Swoją drogą, zbieraliśmy tylko niektóre kody klawiszy w window.c – pora to zmienić. Najpierw definiujemy w module input tablicę przechowującą wszystkie klawisze – będzie ich aż 128. Oczywiście klawiszy na klawiaturze jest mniej, ale na 7 bitach tyle można pomieścić wartości.

```
UBYTE nKeyValue = 1; if (msg_code & 0x80) {     nKeyValue = 0; } g_bKeys[msg_code & 0x7f] = nKeyValue;
```

Powyższy kod wstawia do tablicy jedynkę o indeksie równym kodowi klawisza – o ile gracz nacisnął klawisz. Jeśli ten sam klawisz został puszczony, to wówczas w to miejsce powędruje zero. W input.c doszły dwie nowe funkcje wspierające klawiaturę: ClearKeysTable czyści tablicę przechowującą kody klawiszy a GetFirstPressedKey zwraca pierwszy klawisz, jaki został wciśnięty, a jeśli zostało wciśniętych kilka klawiszy, to zostanie zwrócona wartość odpowiadająca najniższemu kodowi klawisza. Czyli jeśli gracz wcisnął jednocześnie klawisze 3 i ESC, to po pierwsze przyjdą dwa zdarzenia IDCMP_RAWKEY i dla każdego inny indeks tablicy klawiszy zostanie ustawiony na jeden, a GetFirstPressedKey zwróci kod odpowiadający klawiszowi 3, czyli w tym przypadku właśnie 3. A co jeśli żaden klawisz nie został wciśnięty? Co w takim przypadku funkcja zwróci? Jasne jest, że nie może zwrócić poprawnej wartości z zakresu 0-127. Oznaczałoby to, że jednak klawisz został wciśnięty – przyjąłem, że będzie to 255. Mając odpowiednie funkcje w module input, możemy zabrać się za napisanie pętli głównej w menu options. Najważniejsze w tej pętli to oczekiwanie na przycisk czy to joysticka, czy to myszki oraz czekanie na ruch dół/góra. Jedyne co jest zawsze robione w ciele funkcji, to animacja strzałki. Szybko omówmy przypadki łatwiejsze, które to w zasadzie aktualizują menu zgodnie z tym, co gracz zrobił. Jeśli skierował joystick do góry, to strzałka wędruje do góry i aktualizujemy podświetlanie menu. Bardzo podobnie to wygląda dla wychylenia w dół kontrolera. Słowo wyjaśnienia dlaczego korzystamy z konstrukcji if else zamiast używać tylko if. Naiwnie rozumując, ruch myszki i ruch joystickiem są niezależne od siebie i nie powinny generować problemu. Z początku też tak myślałem, ale szybko się okazało, że strzałka może być w innym miejscu niż podświetlenie menu i wygląda to co najmniej nieciekawie. Przejdźmy do przypadku, gdy użytkownik wcisnął przycisk klawiatury. Na początku odtwarzany jest dźwięk. Następnie pobieramy pozycję pionową strzałki, sprawdzamy której linii menu to dotyczy i jeśli udało się ją odnaleźć, to wyświetlamy komunikat proszący o wciśnięcie klawisza. Potem następuje przejście do procedury czekającej na puszczenie klawisza. Odbywa się to w typowy dla nas sposób – poprzez ustawienie wskaźnika na funkcję. Sama procedura czekająca, wykorzystuje do tego celu niepoprawną wartość 255, wykonując się w kółko aż żaden klawisz nie będzie wciśnięty. Po spełnieniu tego warunku czeka na wciśnięcie klawisza, pokazuje na ekranie ten klawisz w postaci napisu i powraca do pętli głównej w menu options.

```
static void optionsLoop(void) {    if (g_bFire || g_bMouseLeft)    {       PlaySfx(SND_CLUNK);       g_bFire = FALSE;
    g_bMouseLeft = FALSE;       int nPosY = ArrowGetPosY();       int i;        for (i = 0; i
```

Asman

(c) Polski Portal Amigowy (www.ppa.pl)

A na deser punktacja. Umieścimy ją na dole, a więc tym samym musimy powiększyć trochę okno zmieniając jedną zmienną w window.c. Poprzednia wersja gry wykorzystywała całą przestrzeń okna i nie było miejsca na punkty. Oprócz punktów na dole umieścimy też ilość żyć. Przeważnie w grach startuje się z wynikiem 0, a grając zdobywa się punkty. W naszym przypadku to standardowe podejście się nie sprawdzi, bo nie do końca wiadomo za co przydzielać punkty. Przecież nie można dostać punktów za to, że statek uderzył w kamień. punktów za to że statek uderzył w kamień. Dlatego w naszej grze zrobimy na odwrót, gracz dostanie na starcie każdej planszy pewną ilość punktów a grając, punkty będą ubywać w zależności w jaki obiekt nasz statek uderzy. Wtedy to lepsi gracze, którzy to odnajdą krótszą drogę będą mieli więcej punktów. Logika punktacji powinna się znaleźć w module game i tam też umieszczamy zmienną g_nScore, odpowiedzialną za to. Do zwiększania i zmniejszania punktów posłużą dwie funkcje: decreaseScore i increaseScore.

```
static void decreaseScore(int amount) {    g_nScore -= amount;    if (g_nScore  9999)    {      g_nScore = 9999;    } }
```

W obu przypadkach sprawdzamy czy licznik punktów nie osiągnął ograniczenia. Ma to znaczenie przy wyświetlaniu punktów – będziemy wiedzieli ile minimalnie i maksymalnie miejsca zajmie punktacja. W tym przypadku będzie to najmniej jeden znak a najwięcej cztery. Skoro mówimy o pokazywaniu punktów, to w module bottomPanel umieściłem funkcję PrintBottomPanel, która między innymi rysuje wynik uzyskany przez gracza.

W dzisiejszym odcinku to tyle. Zachęcam jak zwykle do eksperymentowania z kodem i do zadawania pytań na forum PPA.

Artykuł oryginalnie pojawił się w trzynastym numerze Polskiego Pisma Amigowego.

www.ppa.pl