

# AI Sokoban

nosty, Last 2026

Cześć! Chcę rozpocząć projekt nowego programu na Atari 800XL (procesor 6502). Będziesz moim głównym programistą. Proszę, abyś stosował się ściśle do poniższych zasad technicznych i wniosków wyciągniętych z poprzednich projektów. Używamy asemblera XASM.

#### ZASADY SKŁADNI I FORMATOWANIA (XASM):

1. Definicje: Używaj `equ` dla stałych i rejestrów, `dta` dla danych, `org` dla adresu startowego.
2. Formatowanie: Etykiety i deklaracje `equ` zawsze od początku linii (bez wcięć), instrukcje wcięte (tab/spacje).
3. Dane (WAŻNE): W instrukcji `dta` NIE stawiaj spacji po przecinkach.  
Poprawnie: `dta 1,2,3,4`  
Źle: `dta 1, 2, 3, 4` (powoduje błąd kompilatora).
4. Adresowanie: W instrukcjach typu `lda label,x` również nie stawiaj spacji po przecinku.
5. Przesunięcia bitowe: Pamiętaj o jawnym wskazywaniu akumulatora, np. `lsl @` zamiast `lsl`.
6. Komentarze w kodzie pisz nie używając polskich znaków.

#### OGRANICZENIA SPRZĘTOWE I ARCHITEKTURA:

1. Skoki warunkowe: Pamiętaj o limicie zasięgu (-128/+127). Przy długim kodzie, jeśli skok jest daleki, stosuj "mosty":  
Zamiast: `bne DalekaEtykieta` (błąd)  
Stosuj: `beq Omin\_Jmp`, `jmp DalekaEtykieta`, `Omin\_Jmp`.
2. Klawiatura: Rejestr `CH` (\$02FC) przechowuje wewnętrzne kody Atari (Internal Keycodes), a nie ASCII! (np. 'N'=\$23, 'G'=\$3D, Spacja=\$21). Po odczytaniu klawisza należy zresetować rejestr wpisując do niego #\$FF.
3. DLI (Display List Interrupts): Procedury DLI muszą być ekstremalnie krótkie i szybkie. Używaj ich tylko do zmiany kolorów rejestrów sprzętowych (`COLPF0`-`COLPF3`). Dłuższa logika w DLI zepsuje wyświetlanie. Pamiętaj o `pha`/`pla` i kończeniu `rti`.
4. VBI/Pętla Główna: Ciężkie operacje (jak scrollowanie pamięci ekranu) wykonuj na samym końcu pętli głównej, PO narysowaniu duszków (PMG), aby uniknąć efektu mrugania ("Race the Beam").
5. Tryby Graficzne:
  - Pamiętaj o specyfice trybów ANTIC (np. w trybie 15 jeden bajt to 4 piksele, 2 bity na piksel).
  - Zawsze sprawdzaj, który rejestr koloru (`COLOR0`-`COLOR3`) odpowiada za daną parę bitów (00, 01, 10, 11).
6. Pamięć:
  - Kod programu: zazwyczaj od \$2000.
  - Display List i bufor ekranu: Umieszczaj bezpiecznie powyżej kodu (np. od \$6000), by kod ich nie nadpisał.
  - PMG: Wymaga wyrównania do granicy 2KB (\$4000, \$4800 itd.). Pamiętaj o zerowaniu pamięci PMG przed użyciem.

Na razie nie twórz żadnego kodu zanim nie wyjaśnię ci więcej.

Gotowy?

Chcę byś stworzył w assemblerze kompletną grę na Atari typu "Sokoban", czyli grę logiczną z widokiem z góry, gdzie bohater (którym sterujemy) musi umieścić skrzynki na właściwych miejscach pchając je przed sobą.

Zasady gry:

1. Pole gry podzielone jest na kwadraty.
2. Bohater (B.) porusza się skokowo między sąsiednimi polami (kwadratami) góra/dół/prawa/lewa. Ruchy po przekątnej nie są dozwolone.
3. B. może pchać przed sobą jedną skrzynkę. Nie może ciągnąć skrzynki. Nie można pchać dwóch lub więcej skrzynek.
4. Na polu gdy oznaczamy miejsca (sloty) gdzie mają znaleźć się skrzynki by wygrać dany poziom. Nie ma znaczenia, która skrzynka znajdzie się na którym słocie.
5. Skrzynek jest tyle samo co slotów. Warunek wygranej: wszystkie sloty są zajęte przez skrzynki.

Wykonanie, szczegóły techniczne:

1. Stwórz trzy ekrany i zaprojektuj dla nich odpowiednie Display Listy:

1.1 Ekran Startowy. 24 linie w trybie tekstowym hi-res. Na górze wysrodkowany tytuł gry (wymyśl go). Niżej opisane w skrócie co jest zadaniem gracza (może być po angielsku). Na dole napis "Press Fire to Start". Po naciśnięciu "fire" w joysticku przechodzimy do Ekranu Gry.

1.2 Ekran Gry. 24 linie w trybie tekstowym \$05 antic (4-kolorowym). W tym trybie kolor koduje kombinacja 2 bitów, (00, 01, 10, 11), a piksele są kwadratowe.

1.2.1 Ponieważ chcemy by elementy gry były kwadratami, to każdy z nich będą tworzyły 2 znaki ustawione poziomo obok siebie.

1.2.2 Maksymalna wielkość mapy danego poziomu to 20x12 (gdzie każdy element tworzą dwa znaki obok siebie).

1.2.3 Mapy poszczególnych poziomów będą miały różny rozmiar.

1.2.4 Na mapie poziomu wykorzystamy następujące elementy:

- ; 0 - podłoga (po której bohater może się poruszać)
- ; 1 - pusty slot na skrzynkę
- ; 2 - skrzynka w słocie
- ; 3 - skrzynka (poza slotem) na polu gry
- ; 4 - ściana (bohater nie może przez nią przechodzić)
- ; 5 - początkowa pozycja gracza na pustym słocie
- ; 6 - początkowa pozycja gracza na podłodze
- ; 9 - tło (spacje)

1.2.5 Przepisz definicję znaków z systemu operacyjnego w nowe miejsce w pamięci i zmodyfikuj, tworząc graficzną reprezentację dla każdego elementu z punktu 1.2.4. Pamiętaj, że każdy z tych elementów będzie złożony z 2 znaków. Zaprojektuj prostą, czytelną grafikę, tak by elementy kluczowe ściany, podłoga, skrzynki, bohater wyraźnie się różniły. Pamiętaj jednak, że każdy z elementów ma mieć kształt kwadratu. Wykorzystaj dobrze kolory (3 + kolor tła).

1.2.6 Stwórz procedurę przepisującą mapę do obszaru roboczego, który będzie służył Ci do obsługi całej logiki.

1.2.7 Stwórz procedurę rysującą znakowe pole gry na podstawie mapy (każde pole to 2 znaki). Rysować zaczynasz od lewego górnego rogu.

1.2.8 Pamiętaj aktualną pozycję bohatera. Napisz obsługę ruchu bohatera po mapie w obszarze roboczym. Zasady:

1.2.8.1 Jeśli obok B. w kierunku wychylenia joysticka znajduje się podłoga, to bohater zmienia pozycję.

1.2.8.2 Jeśli obok B. w kierunku wychylenia joysticka znajduje się skrzynka, a za nią podłoga, to bohater zmienia pozycję i skrzynka zmienia pozycję (bohater pcha skrzynkę).

1.2.8.3 Dotychczasową pozycję bohatera wypełniamy odpowiednim elementem (podłoga-0 lub pusty slot-1) na podstawie oryginalnej mapy poziomu.

1.2.8.4 Jeśli pchamy skrzynkę, to na dotychczasowej pozycji skrzynki będzie stał bohater.

1.2.8.5 Jeśli w wyniku pchania skrzynka znajdzie się na ślocie, to zmieniamy ją na mapie z 3 (skrzynka poza slotem), na 2 (skrzynka w ślocie). Analogicznie, jeśli zepchniemy skrzynkę ze slotu, to zmieniamy element na mapie roboczej z 2 na 3.

1.2.9 Po ruchu wykonujemy sprawdzenie, czy poziom został ukończony: na mapie roboczej ma nie być pozycji 1 (pusty slot). Wtedy ładujemy kolejny poziom.

1.2.10. Naciśnięcie klawisza ESC powoduje reset poziomu (np. gdy gracz się zablokuje): ponowne przepisanie mapy do obszaru roboczego.

1.2.11 Naciśnięcie klawisza SPACJA powoduje przeskoczenie na kolejny poziom, bez konieczności ukończenia obecnego.

1.2.12 Wprowadź zabezpieczenie, by bohater nie przemieszczał się błyskawicznie o jedno pole co ramkę. Jeśli gracz będzie trzymał wychylony joystick to między kolejnymi ruchami musi minąć co najmniej 16 ramek.

1.3. Po ukończeniu ostatniego poziomu ładujemy trzeci ekran tekstowy hi-res z napisem "CONGRATULATION! YOU WON!" na środku ekranu. Po naciśnięciu fire przechodzimy do ekranu startowego 1.1

UWAGA. Moje zalecenia i procedury, które opisałem to zapewne nie wszystkie konieczne rzeczy do wykoania. Zanim zaczniesz pracę przemyśl wszystko dokładnie i uzupełnij moje pomysły. Nie spiesz się. Przemyśl wszystko dwa razy. Pamiętaj, że twoim zadaniem jest stworzyć działającą poprawnie grę.

Podaj definicje 4 poziomów. Stwórz na ich postawie linie dla

232323 0 0 0 2323232323  
2323 0 0 ! 0 23 0 0 0 0  
23 0 0 ! ! 0 0 0 ! 0  
0 0 ! 2 ! ! ! 0  
0 ! ! ! 2 ! 0 ! 0  
0 0 0 ! 2 0 0 0 ! 0  
2323 0 ! 0 0 0 0 0  
23 0 0 ! 0 0 0 0 0  
23 0 ! ! ! ! ! 0 0 23  
23 0 ! ! ! ! ! 0 2323  
23 0 0 0 0 0 0 232323

SOKOBAN

PUSH BOXES

PRESS FIRE

