Emulator Chip-8

Michał Góraj 171937

Wstęp

Emulator to program symulujący działanie określonej platformy sprzętowej przez inny, niekompatybilny system, na którym jest uruchamiany. Wykonuje on kod przeznaczony do wykonania na danym sprzęcie, na przykład konsoli. Dzięki zastosowaniu emulatora możliwe jest uruchomienie dowolnego programu przeznaczonego na daną platformę.

Dzięki emulatorom możemy ocalić dobra kultury (głównie gry, ale nie tylko) które zostały stworzone do użycia na przestarzałym dziś sprzęcie, który nie jest już produkowany. Za pomocą emulatorów możemy np. „młócić w platformówki z PlayStation 1 na telefonie”. Emulatory to zwykle bardzo złożone programy, które muszą odtworzyć co do najmniejszych szczegółów strukturę emulowanego sprzętu na urządzeniu o zupełnie innej architekturze, pozbawionej wielu mechanizmów dostępnych oryginalnej maszynie. Producenci sprzętu również używają czasem takich rozwiązań sprzedając „unowocześnione” wersje starego, popularnego niegdyś sprzętu – wspomniane PlayStation 1 doczekało się po 23 latach od wydania zminiaturyzowanej wersji PlayStation Classic, a NES po 33 wersji NES Classic Mini – tego typu produkty używają zawierają nowsze komponenty, a gry na nich są emulowane programowo. Kolejne konsole firmy Sony do PS3 zawierały emulatory poprzednich wersji konsol stacjonarnych; Nintendo realizuje podobną funkcjonalność poprzez usługę Virtual Console – również realizowaną poprzez emulację starszych systemów.

Chip-8 to nigdy nie był „prawdziwy” system, ale interpretowany język programowania stworzony w latach 70. przez Josepha Weisbeckera. Został pomyślany tak, by był bardzo nieskomplikowany, dzięki czemu jego emulacja jest niezwykle prosta. Jednocześnie stworzony program może stanowić świetną bazę do emulacji bardziej złożonych systemów, podobnych w ogólnym zarysie. W Chip-8 urządzenia zewnętrzne są mapowane do 4KB pamięci, istnieje tylko 35 instrukcji procesora, nie występuje mechanizm przerwań, a rozdzielczość czarno-białego wyświetlacza to 64x32 (zaledwie 2048 pikseli).

Wytworzenie dowolnego emulatora wymaga dokładnego studiowania dokumentacji urządzenia – niestety bardzo często jest ona dostępna publicznie (popularne systemy mają opisy dostępne w internecie, uzyskane za pomocą inżynierii wstecznej). W przypadku Chip-8 najlepsze źródła to artykuł na **anglojęzycznej Wikipedii** i dokumentacja **Cowgod's Chip-8 Technical Reference v1.0**.

Opis kodu projektu

Projekt został wykonany w języku C++ za pomocą Microsoft Visual Studio 2017. Proste instrukcje graficzne i odbiór wejścia użytkownika został wykonany przy użyciu biblioteki SDL 2.0.10. Kod źródłowy składa się z trzech plików: main.cpp, chip8.h i chip8.cpp. Program stanowi interpreter poleceń Chip-8.

Obiekt klasy Chip8 jest wirtualną reprezentacją emulowanego urządzenia. W pliku nagłówkowym chip8.h mieści się deklaracja tej klasy, której pola reprezentują poszczególne elementy sprzętu. I tak na przykład cała pamięć urządzenia jest reprezentowana jako tablica bajtów (char, zmienna znakowa w C++ ma rozmiar jednego bajtu):

```
unsigned char memory[MEMORY_SIZE];
```

Chip-8 ma 4KB pamięci, więc rozmiar tablicy jest zdefiniowany stałą:

```
#define MEMORY_SIZE 4096
```

Analogicznie jako tablicę zmiennych znakowych zdefiniowane jest 16 rejestrów ogólnego przeznaczenia. Jako że słowo w Chip-8 ma długość 2 bajtów, stos, służący wyłącznie do zapisu adresów powrotu z instrukcji skoku reprezentujemy jako tablicę zmiennych dwubajtowych unsigned short. Jako zmienne zapiszemy również licznik programu (pc), licznik

stosu (sp), rejestr indeksu (I), tablicę stanów przycisków, mapę bitową ekranu czy stan liczników zegara.

Znajdują się tu również deklaracje funkcji realizujących podstawowe działanie urządzenia, których definicje znajdują się w chip8.cpp:

- initialize() ustawiająca prawidłowe wartości początkowe zmiennych klasy Chip8 (wypełnianie rejestrów zerami, ustawienie licznika programu na stałe miejsce początku wykonywania instrukcji z pamięci 200H, wpisanie zestawu czcionek do pamięci)

- loadGame(const char* filename) ładująca dane z pliku o podanej ścieżce do pamięci Chip8 (na odpowiednie bajty)

- emulateCycle() realizująca pojedynczy cykl procesora(wczytanie instrukcji do wykonania z pamięci, zdekodowanie i wykonanie jej, aktualizacja licznika programu)

- timersTick() emulująca pojedyncze cyknięcie zegara (aktualizacja wartości dwóch liczników – opóźnień i dźwięku)

W pliku main.cpp znajduje się funkcja main, gdzie znajduje się wejście do programu i zawarta jest ogólna logika działania emulatora. Zadeklarowano tu również strukturę Display, która zawiera wszystkie zmienne potrzebne do zrealizowania prawidłowego wyświetlania ekranu Chip-8 za pomocą SDL i zdefiniowano funkcje obsługujące wyświetlacz i przekazujące dane o wciśniętych przez użytkownika klawiszach do pamięci urządzenia w odpowiedniej dla nich formie.

Program wynikowy chip8.exe przyjmuje jako argumenty: ścieżkę dostępu do pliku z odpalaną grą i mnożnik rozdzielczości – np. podanie liczby 10 sprawi, że każdy „piksel” Chip-8 będzie wyświetlany jako kwadrat 10x10 pikseli o tym samym kolorze.

Funkcja main zajmuje się: inicjalizacją pól Chip8, załadowaniem gry z pliku do pamięci emulowanego urządzenia, stworzeniem wyświetlacza o odpowiedniej rozdzielczości i realizacją pętli emulacji wraz z możliwością zamknięcia programu odpowiednim klawiszem (Esc).

Pętla emulacji

Dopóki użytkownik nie zdecydował się zakończyć działania programu, wykonujemy partiami po 9 instrukcji-cyklów procesora, przekazujemy obiektowi klasy Chip8 dane o wciśniętych przez użytkownika klawiszach, w razie potrzeby aktualizujemy stan wyświetlacza i usypiamy program na pozostały czas jednej klatki wyświetlania. Chip8 nie ma wyspecyfikowanej częstotliwości taktowania zegara, ale większość programów napisanych dlań zakładało taką rzędu kilkuset Hz – w implementacji założono taktowanie zegara procesora Chip-8 jako 540 MHz. Częstotliwość taktowania zegara obecnych komputerów i ogólnie szybkość działania procesorów jest znacznie, znacznie większa i różni się między procesorami. Dlatego by osiągnąć zadowalającą prędkość i stabilne 60 klatek na sekundę dokonano powyższych zabiegów (9 * 60 = 540). Warto zauważyć, że taktowanie zegara procesora i taktowanie zegara opóźnień oraz dźwięku to dwie różne rzeczy – to drugie jest zapisane w dokumentacji, częstotliwość wynosi 60Hz.

Dekodowanie instrukcji

Lwią część kodu zajmuje dekodowanie instrukcji procesora i ich emulowanie. Dekodowanie realizowane jest przez dużą instrukcję switch z kolejnymi zagnieżdżonymi takimi instrukcjami, gdzie kolejne bity kodu instrukcji są przyrównywane do odpowiednich wartości. Po rozpoznaniu wykonany jest kod emulujący wykonanie danej instrukcji.

Przykład:

```
case 0x8000:
    switch (opcode & 0x000F)
    {
      •••
      case 0x0004:                                  // 8XY4 : Adds VY to VX. VF is set to 1 when there's a carry,
                                                    // and to 0 when there isn't.
          if (VY > (0xFF - VX))                     // V[Y] > 0xFF - V[X] <=>  V[Y] + V[X] > 0xFF
          {                                         // (but there won't be an overflow on 1-byte long variables)
              V[0xF] = 1;                           // Carry Flag = 1, there's a carry
          }
          else
          {
              V[0xF] = 0;                           // Carry Flag = 0
          }
          VX += VY;
          pc += 2;
      break;
      •••
    }
```

Kod instrukcji 8XY4 oznacza, że wartość pierwszych 4 bitów to (szesnastkowo) 8, ostatnich 4 to 4 – w ten sposób instrukcja jest rozpoznawana. Środkowe bity wyznaczają numery rejestrów, gdzie znajdują się odpowiednie dane. Instrukcja 8XY4 oznacza dodanie wartości z rejestru ogólnego przeznaczenia o numerze Y, czyli w naszym programie V[Y] – element z tablicy V o indeksie Y. Analiza zbioru instrukcji dla procesora może doprowadzić do zaobserwowania zależności, według której indeksy określane jako X Y są zapisane zawsze na tych samych bitach instrukcji, w kodzie emulatora zdefiniowano stałe, upraszczające zapis:

```
#define VX V[(opcode & 0x0F00) >> 8]
#define VY V[(opcode & 0x00F0) >> 4]
```

By emulować działanie tej instrukcji nie wystarczy jednak tylko dodać do wartości zapisanej w tablicy V pod indeksem X wartość zapisaną w tablicy V pod indeksem Y. Należy również ustawić flagę przepełnienia, reprezentowaną jako ostatni element tablicy rejestrów V[15] (V[0xF]).

Zrzut ekranu włączonego programu z grą Space Invaders: