Politechnika Wrocławska

Wydział Informatyki i Zarządzania
kierunek studiów: Informatyka
specjalność: -

Praca dyplomowa – inżynierska

Emulacja przenośnej konsoli gier video z użyciem języka C# na platformie Windows

Bartłomiej Szcześniak

słowa kluczowe:
emulacja
C#
Windows

krótkie streszczenie:
Celem pracy jest zaprojektowanie i implementacja emulatora konsoli Nintendo Game Boy. Efektem jest powstały emulator, zdolny do uruchomienia oprogramowania stworzonego dla Game Boy-a.

opiekun pracy dyplomowej	dr inż. Marek Kopel	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	*ocena*	*podpis*

Ostateczna ocena za pracę dyplomową

Przewodniczący Komisji egzaminu dyplomowego	.....	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	*ocena*	*podpis*

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**
a) kategorii A (akta wieczyste)
b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)
** niepotrzebne skreślić*

pieczątka wydziałowa

Wrocław

2019

Streszczenie

Celem pracy jest zaprojektowanie i implementacja emulatora konsoli Nintendo Game Boy. Na dokument ten składają się opisy kolejnych etapów tworzenia emulatora. Game Boy został tutaj opisany z funkcjonalnego punktu widzenia. Na podstawie znalezionych opisów, analiz i amatorskich dokumentacji, został przygotowany projekt emulatora. Projekt ten został zaimplementowany, zweryfikowany i częściowo poprawiony, co zostało opisane w kolejnych rozdziałach tej pracy. Efektem jest powstały emulator, zdolny do uruchomienia oprogramowania stworzonego dla Game Boy-a. W dokumencie tym zaproponowano również możliwe przyszłe ścieżki rozwoju, które umożliwią przyspieszenie i poprawę pracy programu, jak i rozszerzenie jego funkcjonalności.

Abstract

Goal of this thesis is design and implementation of Nintendo Game Boy video game console. This document covers description of phases leading to creation of working emulator. Game Boy has been described from functional point of view. Based on found descriptions, analysis and amateur documentation, design of the emulator was prepared. This design has been implemented, verified and partially corrected, what has been described in following chapters of this document. As result of this thesis, emulator was created, able to run software created for Game Boy. This document contains proposal of future development paths, which could make this emulator faster and expand its functionality.

Spis treści

Streszczenie 3
Abstract 3
Spis treści 4
1. Wprowadzenie 6
1.1. Cel pracy 6
1.1. Zakres pracy 6
2. Emulator 7
2.1. Emulacja, a symulacja 7
2.2. Rodzaje emulacji 7
2.2.1. Interpretacja 7
2.2.2. JIT 7
2.2.3. Kompilacja przed wykonaniem 7
3. Nintendo Game Boy 8
3.1. Opis sprzętowy 8
3.1.1. Procesor 8
3.1.2. Wyświetlacz 8
3.1.3. Podsystem graficzny 8
3.1.4. Podsystem dźwięku 8
3.1.5. Mapa pamięci 9
3.1.6. DMA 9
3.1.7. Wejście/Wyjście 9
3.2. Opis nośników pamięci 10
3.2.1. Nagłówek kartridża 10
3.2.2. Typy kontrolerów pamięci 11
4. Opis Projektu 12
4.1. Aplikacja GbEmuApp 12
4.2. Biblioteka GbCore 13
4.2.1. GbSystem 13
4.2.2. Catridge 13
4.2.3. Input 14
4.2.4. MemProxy 14
4.2.5. MBCMemAccess 15
4.2.6. CpuCore 15
4.2.7. GpuCore 17
5. Opis Implementacji 20

5.1. Wybrane technologie implementacji 20
5.1.1. C# 20
5.1.2. WPF 20
5.2. Napotkane problemy 20
5.2.1. Błędy dokumentacji 20
5.2.2. Skomplikowany podsystem graficzny 20
5.2.3. Emulacja z dokładnością do cyklu procesora 21
5.3. Zastosowane ulepszenia 21
5.3.1. Debugger 21
5.3.2. Mechanizm logowania 22
5.3.3. Ulepszona reprezentacja przestrzeni adresowej 22
5.3.4. Trzymanie przetworzonych kafelków w pamięci podręcznej 22
5.3.5. Przeniesienie emulacji podsystemu graficznego do osobnego wątku. 23
5.3.6. Ograniczenie wpływu części służącej logowaniu na działanie wersji produkcyjnej 23
6. Weryfikacja działania 24
6.1. ROM-y testowe 24
6.2. Programy homebrew 25
6.2.1. Yvar's GB snake 25
6.2.2. Evoland 26
7. Możliwości rozwoju 27
7.1. Zbadanie i poprawna implementacja emulacji z dokładnością do cyklu procesora 27
7.2. Implementacja dynamicznej rekompilacji 27
7.3. Poprawa wyglądu i funkcjonalności GbEmuApp 27
7.4. Zastosowanie filtrów graficznych w celu poprawy jakości generowanego obrazu 27
7.5. Implementacja obsługi dźwięku 28
7.6. Rozszerzenie i reorganizacja funkcjonalności debuggera 28
7.7. Dodanie wsparcie dla Super Game Boy-a (SGB) i Game Boy-a Color (GBC). 28
7.8. Zapisywanie stanu 28
8. Podsumowanie 29
9. Bibliografia 30
10. Spis rysunków 31

1. Wprowadzenie

1.1. Cel pracy

Celem pracy jest napisanie aplikacji pozwalającej uruchamiać ROM-y[1] przeznaczone dla mobilnej konsoli gier komputerowych - Nintendo Game Boy.

1.1. Zakres pracy

W zakresie implementacji projektu mieści się interfejs użytkownika i właściwy emulator, działający na poziomie wystarczającym w celu uruchomienia ogólnodostępnych programów weryfikacyjnych. Z zakresu tego z góry zostaje wyłączona emulacja dźwięku z powodu ograniczeń czasowych.

[1] ROM - zrzut pamięci kartridża, najczęściej zawierający kod binarny programu.

2. Emulator

Jak podaje [1], “emulatorem nazywamy oprogramowanie, które sprawia, że system komputerowy (system gospodarza), zachowuje się jak inny system komputerowy (system gościa). Emulator zwykle pozwala uruchomić oprogramowanie lub skorzystać z urządzeń przeznaczonych dla systemu gościa, na systemie gospodarza.”

2.1. Emulacja, a symulacja

Emulacja jest procesem mającym na celu zastąpienie emulowanego urządzenia. Emulator nie musi dokładnie odzwierciedlać wewnętrznych procesów zachodzących w urządzeniu emulowanym. Symulator natomiast nie musi i zwykle nie jest w stanie zastąpić urządzenia, procesu które symuluje. Jego najważniejszym zadaniem jest jak najdokładniejsze zamodelowanie jakiegoś urządzenia lub procesu, a nie jego zastąpienie.

2.2. Rodzaje emulacji

2.2.1. Interpretacja

Interpretacja polega na interpretowaniu każdego kolejnego rozkazu architektury emulowanej i wykonania w postaci kodu architektury docelowej. Jest to najwolniejsza metoda emulacji, pozwala jednak na stworzenie kodu łatwego do przeniesienia pomiędzy różnymi architekturami docelowymi.

2.2.2. JIT

JIT (Just In Time compilation - kompilacja w czasie wykonania) [2] polega tłumaczeniu kodu architektury emulowanej do architektury docelowej w momencie, kiedy jest on wykonywany. Kod przetłumaczony już do architektury docelowej jest zapisywany do pamięci podręcznej w blokach ograniczonych rozgałęzieniami ścieżki wykonania. W momencie ponownego natrafienia przez emulator na instrukcję, której adres bloku znajduje się w pamięci podręcznej, wykonuje się dany blok, zamiast interpretować na nowo kolejne rozkazy. Metoda jest ta ma o wiele mniejszy narzut niż interpretacja, jednak w swojej najprostszej wersji jest wrażliwa na architektury w których dozwolona jest automodyfikacja kodu wykonywalnego. W takim wypadku wymaga mechanizmu, który będzie oznaczać bloki kodu które wymagają reinterpretacji z uwagi na ich dezaktualizacje.

2.2.3. Kompilacja przed wykonaniem

Kompilacja przed wykonaniem (inna nazwa: AOT - Ahead Of Time compilation) [3] polega na przetłumaczeniu kodu architektury emulowanej do kodu architektury docelowej przed jego wykonaniem. Rezultat takiej operacji można później zapisać w postaci pliku wykonywalnego zawierającego już tylko kod architektury docelowej. Metoda ta ma najmniejszy narzut wydajnościowy spośród wymienionych, gdyż po jednokrotnym uruchomieniu pozwala na wielokrotne korzystanie z rezultatów, jednak nie jest często stosowana z powodu ograniczenia: kod wykonywalny nie może być modyfikowany w trakcie działania programu.

3. Nintendo Game Boy

Nintendo Game Boy [4] jest 8-bitową przenośną konsolą gier video stworzoną przez firmę Nintendo. Został wprowadzony do sprzedaży w 1989 roku w Japonii i rok później pozostałych regionach świata. Konsola odniosła światowy sukces, sprzedając się do momentu premiery swojego następcy, w liczbie ponad 64 mln sztuk.[2]

3.1. Opis sprzętowy

Jak podaje [5], funkcjonalnie Game Boy-a można podzielić na następujące podzespoły.

3.1.1. Procesor

Procesor Game Boy-a bazowany jest na chipie Zilog80, który z kolei wywodzi się z Intel-a 8080. W porównaniu do układu Ziloga dodano kilka instrukcji, kilka innych usunięto, a dla kilku zmieniono kody binarne. Układ użyty w konsoli działa z częstotliwością 1.048576MHz, natomiast zegar, na którym działa cały system pracuje z prędkością 4 razy większą (4.194304MHz).

3.1.2. Wyświetlacz

Game Boy jest wyposażony w monochromatyczny wyświetlacz o rozdzielczości 160x144 pikseli. Jest on w stanie wyświetlić 4 odcienie szarości. Oryginalnie wyświetlacz oferował żółte podświetlenie w późniejszych wersjach urządzenia jednak, zrezygnowano z niego dla tradycyjnego białego.

3.1.3. Podsystem graficzny

Podsystem graficzny konsoli działa równolegle do procesora. Pozwala na renderowanie wyświetlanych treści na 3 dostępne sposoby:

a) Tło - Bufor tła zawiera 256x256 pikseli reprezentowanych przez 32x32 kafelków o wymiarach 8x8 punktów. Część bufora wyświetlaną na ekranie kontrolują 2 rejestry specjalne określające współrzędne ekranu na buforze - SCREENX i SCREENY. Bufor ten wspiera zawijanie, czyli w przypadku, kiedy współrzędne ekranu będą ustawione tak aby ekran wykraczał poza przestrzeń bufora, brakujące miejsce zostanie uzupełnione za pomocą odpowiedniej grafiki z jego drugiego końca.
a) Okno - Bufor okna, różni się od bufora tła, tylko sposobem wyliczania pozycji na ekranie. Służą do tego 2 rejestry - WNDPOSX i WNDPOSY, które określają pozycję okna w stosunku do aktualnej pozycji ekranu wyświetlacza konsoli.
b) Sprite-y - Game Boy pozwala na wyświetlenie na ekranie do 40 dodatkowych sprite-ów, z ograniczeniem maksymalnie 10-ciu w jednej linii. Każdy z nich może mieć włączoną przezroczystość, a także inny sposób mapowania palety szarości.

3.1.4. Podsystem dźwięku

Konsola pozwala na generowanie dźwięku w 4 możliwych trybach:

a) Fala kwadratowa z funkcją obwiedni
b) Fala kwadratowa z funkcją sweep i obwiedni
c) Fala zapisana w pamięci RAM
d) Generator szumu

---

[2] Po premierze następcy - Game Boy-a Color, Nintendo we wszystkich podsumowaniach sprzedaży publikowało tylko sumę sprzedaży oryginalnego Game Boy-a i Game Boy-a Color. W momencie wycofania ze sprzedaży suma ta wyniosła ponad 118 mln [14]

Każdy z tych trybów można skierować do jednego albo obu dostępnych kanałów, w których są miksowane.

3.1.5. Mapa pamięci

Game Boy korzysta z 16-bitowej przestrzeni adresowej. Pozwala ona na zmapowanie dostępu do większości funkcji konsoli. Z powodu zapotrzebowania na większą ilość pamięci powstały specjalne kontrolery wbudowane w kartridże, pozwalające zmieniać banki pamięci zmapowane w części obszarów obsługiwanych przez kontrolery. Przestrzeń adresowa dzieli się na następujące bloki [tab. 3.1]:

Tabela 3.1Mapa przestrzeni adresowej Game Boy-a (źródło: [5])

Adres początkowy	Adres końcowy	Opis bloku
0x0000	0x3FFF	ROM Bank 0
0x4000	0x7FFF	Wymienialny ROM Bank
0x8000	0x9FFF	Video RAM
0xA000	0xBFFF	Wymienialny RAM Bank
0xC000	0xDFFF	Wewnętrzny RAM Bank
0xE000	0xFDFF	Echo wewnętrznego Banku RAM
0xFE00	0xFE9F	OAM RAM
0xFEA0	0xFEFF	Nieużywana przestrzeń
0xFF00	0xFF4B	Pamięć rejestrów specjalnych
0xFF4C	0xFF7F	Nieużywana przestrzeń
0xFF80	0xFFFE	Wewnętrzny RAM Bank
0xFFFF	0xFFFF	SRejestr kontroli przerwań

3.1.6. DMA

W konsoli zaimplementowane prosty mechanizm DMA (ang. Direct Memory Access) pozwalający na szybkie kopiowanie z 160 bajtów danych z pamięci RAM do pamięci OAM.

3.1.7. Wejście/Wyjście

Urządzeniami wejścia/wyjścia konsoli są:

- Wyświetlacz, opisany powyżej
- 8-przyciskowa klawiatura, złożona z 4 przycisków kierunkowych, przycisków A, B oraz Start, Select
- Port szeregowy, służący do połączenia do 4 konsol ze sobą
- Port na kartridże
- Syntezator dźwięku, opisany powyżej

3.2. Opis nośników pamięci

Nośnikami programów dla Game Boy-a są kartridże. Są to małe układy elektroniczne, wyposażone w pamięć ROM i RAM. Ze względu na ograniczenia przestrzeni adresowej Game Boy-a często wyposażano je w dodatkowe układy zwane kontrolerami banków pamięci (MBC - Memory Bank Controller) służącymi do przełączania pomiędzy bankami pamięci mapowanymi na tą samą przestrzeń adresową. Kontrolery te sterowane są zapisami do pamięci ROM, które ze względu na specyfikę tego rodzaju pamięci zwykle zostałyby zignorowane. Kartridże były dodatkowo często wyposażane w małe baterie, które służyły podtrzymaniu zasilania pamięci RAM po odłączeniu od konsoli. Pozwalało to na implementacje zapisu danych użytkownika w programach.

3.2.1. Nagłówek kartridża

Każdy ROM Game Boy-a posiada nagłówek, dostępny pod adresem 0x100, składający się z następujących elementów [tab. 3.2]:

Tabela 3.2 Definicja nagłówka ROM-u (źródło: [5])

Adres początkowy	Adres końcowy	Opis
0x100	0x103	Punkt wejściowy programu. Game Boy zaczyna wykonywanie od instrukcji zawartej pod adresem 0x100
0x104	0x133	Bitmapa zawierająca logo Nintendo. Jeżeli nie jest ona taka sama jak mapa zapisana wewnątrz konsoli, procesor zatrzyma działanie, zawieszając działanie konsoli
0x134	0x143	Tytuł programu, zapisany wielkimi literami w ASCII
0x144	0x145	Kod Licencji (nowy) wskazujący na wydawcę programu
0x147	0x147	Kod wskazujący na typ kontrolera banku pamięci zastosowanego w kartridżu
0x148	0x148	Kod wskazujący na ilość pamięci ROM w kartridżu
0x149	0x149	Kod wskazujący na ilość pamięci RAM w kartridżu
0x14A	0x14A	Kod regionu dystrybucji
0x14B	0x14B	Kod licencji (stary)
0x14C	0x14C	Numer wersji gry
0x14D	0x14D	Suma kontrolna nagłówka ROM-u
0x14E	0x14F	Suma kontrolna całego ROM-u

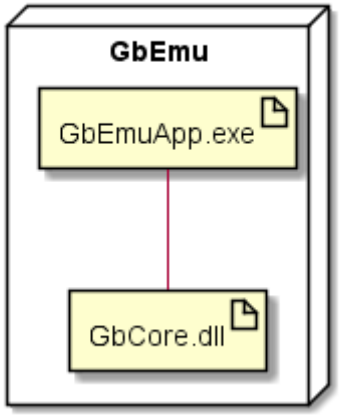
3.2.2. Typy kontrolerów pamięci

Kartridże z oprogramowaniem wydanym na Game Boy-a wykorzystują następujące rodzaje kontrolerów:

- MBC0 - Kontroler, który nie posiada jakiejkolwiek funkcjonalności związanej z przełączaniem banków pamięci. Pozwala na zmapowanie do 32KB pamięci ROM
- MBC1 - Pierwszy kontroler, który wspiera przełączanie się między bankami pamięci. Obsługiwał do 2 MB pamięci ROM.
- MBC3 - Kontroler, który zawierał zintegrowany zegar czasu rzeczywistego. Zegar ten działał (dzięki dodatkowej bateryjce) nawet po odłączeniu od konsoli, dzięki czemu programy mogły go wykorzystywać w różnym celu (np. Uzależnieniu rozgrywki w grze od aktualnego dnia tygodnia)
- Inne - Powstało jeszcze wiele innych kontrolerów, jak MBC2, MBC5, HuC1 (z portem podczerwieni), czy MMM01, jednak nie były one szeroko wykorzystywane, jak i również dokumentacja do nich nie jest tak szeroko dostępna. Warto wspomnieć również o istnieniu kontrolerów dedykowanych konkretnym alternatywnym zastosowaniom, jak Game Boy Pocket Camera, czy Sonar.

4. Opis Projektu

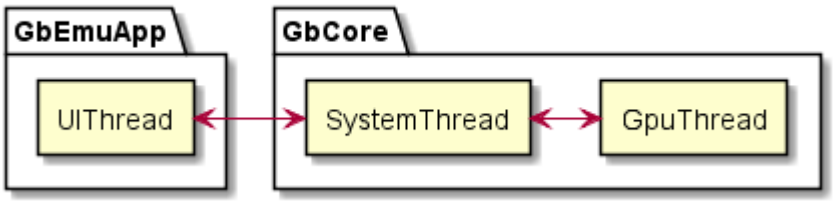
Projekt zakłada podział emulatora na dwie części (rys. 4.1): aplikacje (interfejs użytkownika) o nazwie "GbEmuApp.exe" i bibliotekę zawierającą funkcjonalność właściwą emulatora "GbCore.dll", implementującą API pozwalające na wykorzystanie jej przez aplikacje. Taki podział projektu pozwala na łatwe przeniesienie go na inne platformy, dzięki oddzieleniu części kodu dedykowanej dla danej platformy (tutaj: interfejs użytkownika), od kodu który może być wykorzystany na każdej innej bez żadnych zmian.



Rysunek 4.1 Podział emulatora na artefakty

W celu zapewnienia szybkiego działania i responsywności, emulator będzie działać w 3 wątkach(rys. 4.2):

a) UIThread - odpowiedzialny za responsywność interfejsu użytkownika
b) SystemThread - odpowiedzialny za emulacje procesora i koordynację emulacji
c) GpuThread - odpowiedzialny za emulacje działania układu graficznego konsoli



Rysunek 4.2 Podział emulatora na wątki z uwagi na ich artefakt pochodzenia

4.1. Aplikacja GbEmuApp

Aplikacja jest częścią emulatora zapewniającą interfejs użytkownika.
Do jej zadań należą:

- Wyświetlanie informacji o załadowanym ROM-ie, odczytanych z jego nagłówka przy pomocy biblioteki GbCore
- Pozwolenie użytkownikowi na wybranie ROM-u do załadowania
- Uruchomienie emulacji z wykorzystaniem biblioteki GbCore
- Wyświetlanie w czasie rzeczywistym grafiki wyrenderowanej przez bibliotekę GbCore użytkownikowi
- Przechwytywanie naciśnięć klawiszy i przekazywanie ich do biblioteki GbCore.

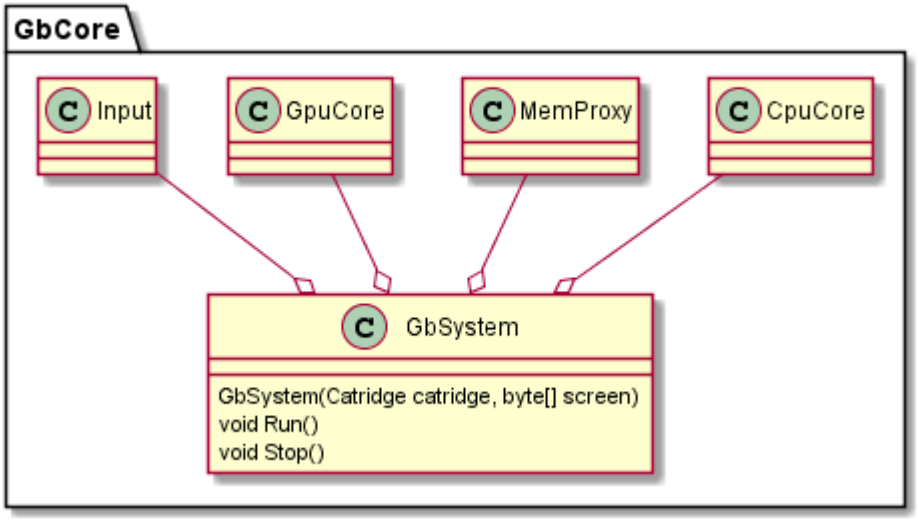
4.2. Biblioteka GbCore

Biblioteka GbCore to element będący właściwą częścią emulatora. Jej celami jest emulacja konsoli Nintendo Game Boy, w stopniu umożliwiającym uruchomienie programów stworzonych dla tego systemu, jak i zapewnienie interfejsu pozwalającego na wykorzystanie biblioteki Aplikacje GbEmuApp.

4.2.1. GbSystem

Główną klasą biblioteki jest GbSystem, która reprezentuje działanie całego systemu. Jest odpowiedzialna za uruchomienie, koordynację i wymianę informacji pomiędzy podległymi jej klasami funkcjonalnymi(rys. 4.3):

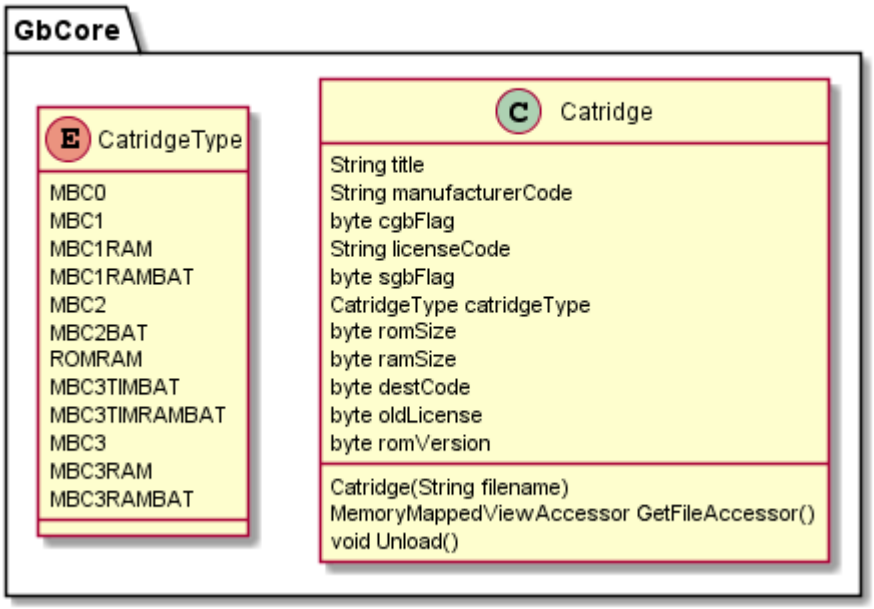
a) Input - Odpowiedzialna za obsługę emulacji przycisków konsoli
b) GpuCore - odpowiedzialna za emulacje podsystemu graficznego konsoli
c) MemProxy - odpowiedzialna za mapę pamięci urządzenia jak i również emulacje funkcjonalności nie pokrytych przez pozostałe urządzenia
d) CpuCore - odpowiedzialna za emulacje procesora, interpretacje i wykonywanie jego instrukcji



Rysunek 4.3 Model klasy dla klasy GbSystem

4.2.2. Catridge

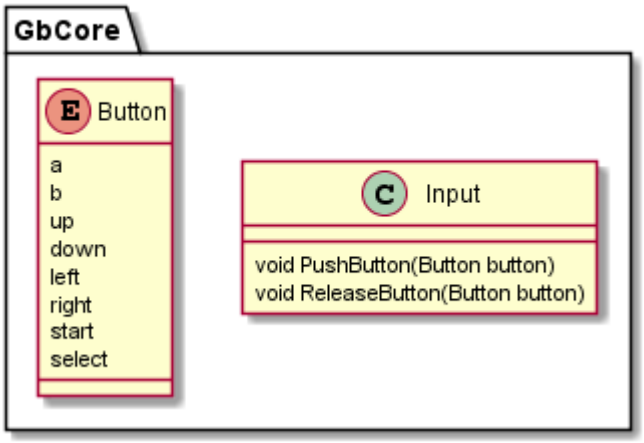
Klasa Catridge (rys. 4.4) reprezentuje wybrany przez użytkownika ROM uruchomiony w emulatorze. W zakresie jej funkcjonalności znajduje się wczytywanie pliku, interpretacja nagłówka ROM-u, jak i zwalnianie zasobów.



Rysunek 4.4 Model klasy Catridge

4.2.3. Input

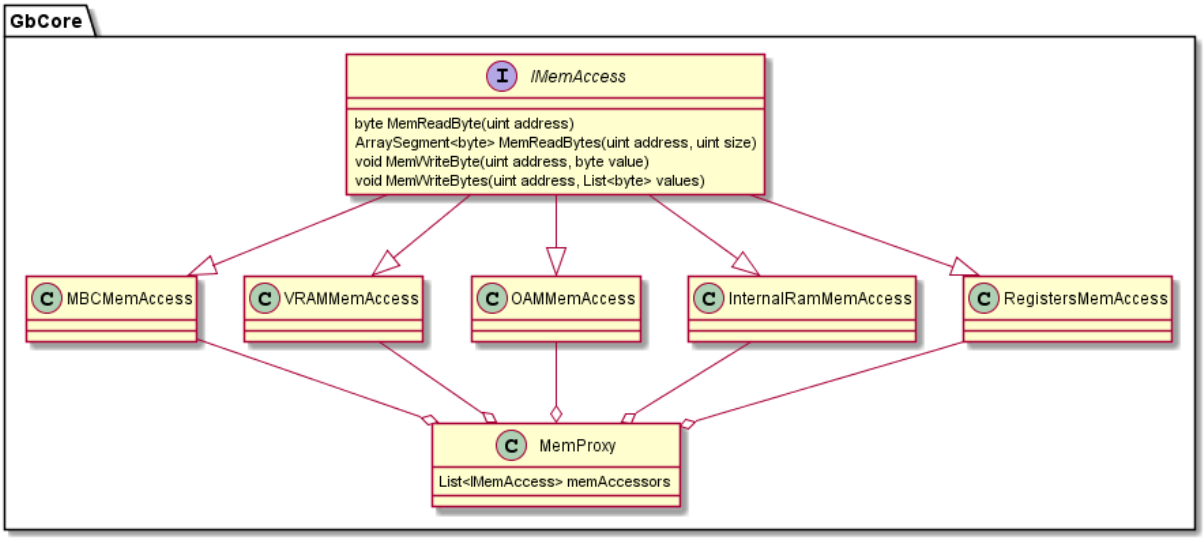
Input (rys. 4.5) to najprostszy z podsystemów, którego celem jest emulacja przycisków konsoli. Jest to spełnione poprzez interfejs udostępniony aplikacji pozwalający wcisnąć lub odpuścić wybrany przycisk.



Rysunek 4.5 Model klasy Input

4.2.4. MemProxy

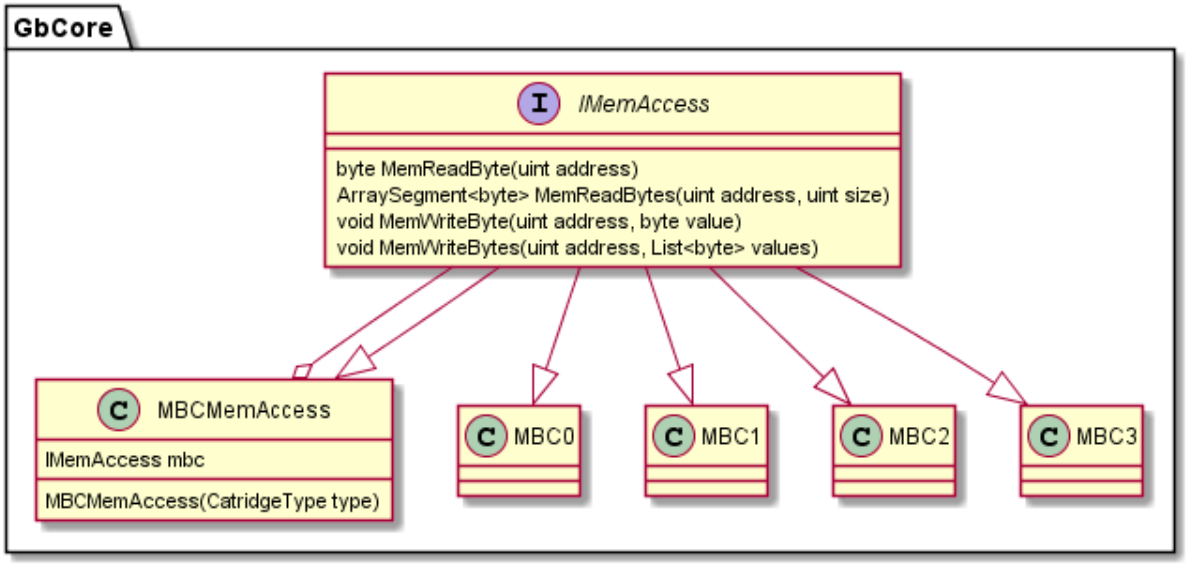
Mem proxy (rys. 4.6) to podsystem odpowiedzialny za emulacje przestrzeni adresowej systemu jakim jest Game Boy. Poprzez niego realizowane są wszystkie odczyty i zapisy, z i do pamięci. Składa się na niego kilka różnych klas implementujących wspólny interfejs, zwanych dalej akcesorami. Akcesory te emulują różne urządzenia reprezentowane w przestrzeni adresowej Game Boy-a, m.in. pamięć wideo, kartridż, pamięć RAM. Z uwagi na prostotę działania, dokładniej zostanie opisany tylko MBCMemAccess



Rysunek 4.6 Model klasy MemProxy

4.2.5. MBCMemAccess

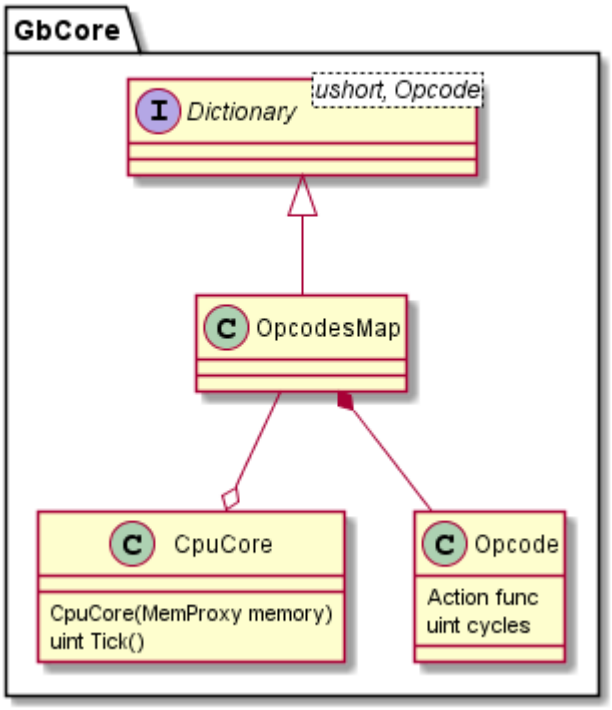
MBCMemAccess (rys. 4.7) jest to klasa - akcesor - emulująca kartridż z programem w przestrzeni adresowej konsoli. Implementuje ona wzorzec projektowy proxy [6] - na podstawie typu kartridża tworzy akcesor właściwy dla danego typu kontrolera pamięci, a następnie deleguje do niego wszystkie odniesienia.



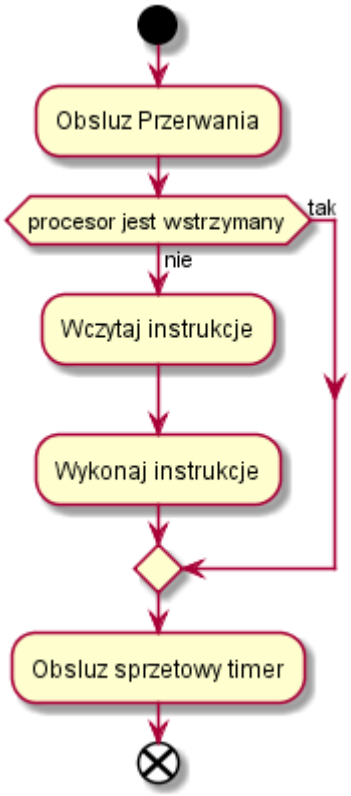
Rysunek 4.7 Model klasy MBCMemAccess

4.2.6. CpuCore

Klasa CpuCore (rys. 4.8) odpowiedzialna jest za emulacje cyklu (“tick”) działania procesora (rys. 4.9). Zawiera ona mapę rozkazów procesora, która powiązuje je z akcją, którą należy wykonać oraz ilością cykli potrzebną na wykonanie danego rozkazu.



Rysunek 4.8 Model klasy CpuCore



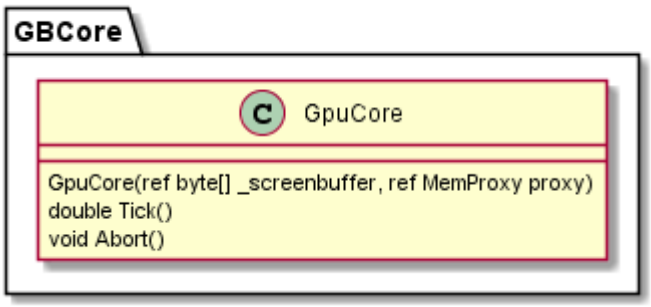
Rysunek 4.9 Sekwencja przetwarzania instrukcji przez procesor

4.2.7. GpuCore

Klasa (rys. 4.10) ta emuluje podsystem grafiki. Została ona zaprojektowana przy użyciu wzorca projektowego “Thread Pool Pattern” [7], z założeniem istnienia tylko jednego wątku roboczego. Funkcjonalnie podzielona jest na dwie części:

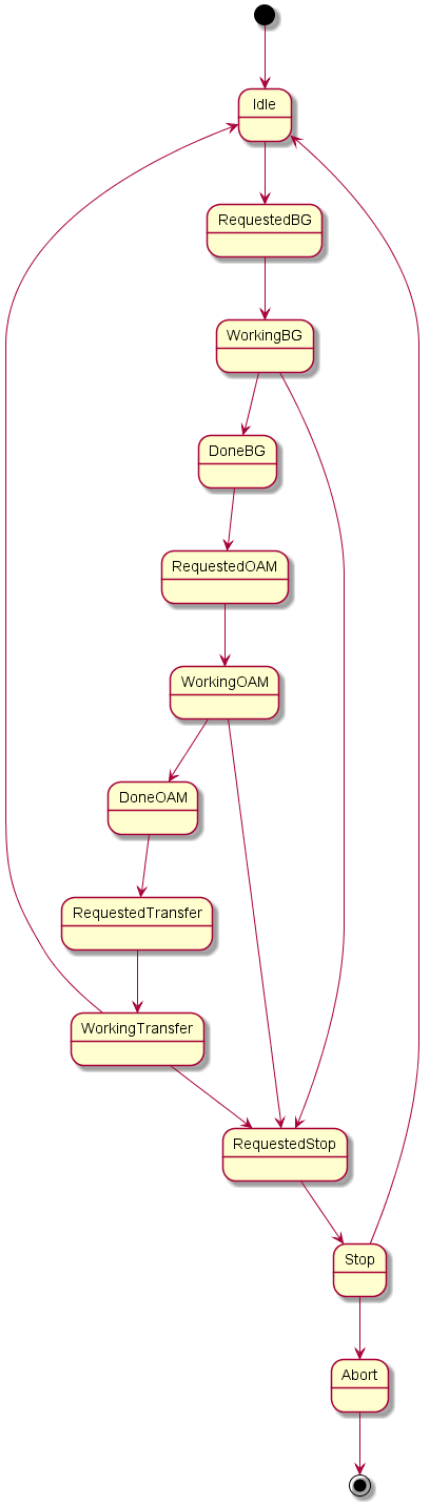
- Interfejs publiczny - pozwalający sterować zachowaniem podsystemu synchronizując go z emulacją całego systemu
- Maszyna stanowa - implementująca w wątku roboczym rzeczywistą funkcjonalność podsystemu graficznego.

Interfejs publiczny tej klasy podobnie jak w przypadku klasy CpuCore, pozwala na wykonanie kolejnej akcji podsystemu, podając jako informację zwrotną ilość cykli procesora potrzebną na wykonanie tej właśnie akcji.



Rysunek 4.10 Model klasy GpuCore

Maszyna stanowa implementuje podział działania podsystemu graficznego na atomowe etapy, w celu jak najlepszej synchronizacji z resztą systemu (rys. 4.11).



Rysunek 4.11 Maszyna stanowa wątku wykonawczego Gpu

Stany:

- Idle - Wątek ukończył ostatnie zlecenie (jeśli nastąpiło) i oczekuje na nowe zlecenie pracy
- RequestedBG - Wątek otrzymał zlecenie aktualizacji tła dla danej linii ekranu
- WorkingBG - Wątek pracuje nad aktualizacją tła dla danej linii ekranu ekranu
- DoneBG - Wątek ukończył pracę nad aktualizacją tła dla danej linii ekranu

- RequestedOAM - Wątek otrzymał zlecenie aktualizacji sprite-ów na danej linii ekranu
- WorkingOAM - Wątek pracuje nad aktualizacją sprite-ów na danej linii ekranu
- DoneOAM - Wątek ukończył pracę nad aktualizacją sprite’4-ów na danej linii ekrany
- RequestedTransfer - Wątek otrzymał zlecenie przeniesienia wewnętrznego bufora grafiki do bufora zewnętrznego używanego przez aplikacje
- WorkingTransfer - Wątek pracuje nad przeniesieniem wewnętrznego bufora grafiki do bufora zewnętrznego używanego przez aplikacje
- RequestedStop - Wątek otrzymał zlecenie zatrzymania się po zakończeniu aktualnej operacji
- Stop - Wątek został zatrzymany i oczekuje na wznowienie
- Abort - Wątek otrzymał zlecenie zakończenia pracy

5. Opis Implementacji

5.1. Wybrane technologie implementacji

5.1.1. C#

Obiektowy język programowania zaprojektowany przez firmę Microsoft [8]. Kod stworzony w tym języku nie jest kompilowany do kodu maszynowego, a do kodu pośredniego uruchamianego w środowisku .NET.

Język ten został wybrany ze względu na kilka aspektów:

- Łatwość projektowania oprogramowania w tym języku z uwagi na jego obiektowość
- Wieloplatformowość - kod napisany w tym języku możliwy jest do łatwego przeniesienia na inne platformy: GNU/Linux, Android.
- Małą popularność wśród twórców emulatorów - jako że język ten nie jest kompilowany bezpośrednio do kodu maszynowego, programy w nim napisane mają pewien narzut wydajnościowy wynikający ze środowiska uruchomieniowego. Jego użycie miało pokazać, że implementacja emulatora w tym języku jest możliwa i sensowna.
- Mechanizm refleksji - Język ten wspiera refleksje [9] dzięki czemu można zaimplementować w nim mechanizm dynamicznej rekompilacji z kodu procesora z80 na kod pośredni środowiska .NET.

5.1.2. WPF

WPF - Windows Presentation Foundation [10] - Framework pozwalający na szybką implementację interfejsu okienkowego aplikacji z użyciem języka znaczników XAML. Został wybrany w celu implementacji interfejsu użytkownika emulatora na platformie Windows, ze względu na łatwość użycia i dobrą integrację z dedykowanym środowiskiem programistycznym.

5.2. Napotkane problemy

5.2.1. Błędy dokumentacji

Dokumentacja Game Boy-a nie jest ogólnodostępna. Z tego powodu całość dokumentacji dostępnej w internecie jest efektem zastosowania inżynierii odwrotnej - procesu dokumentacji urządzenia, lub oprogramowania na podstawie obserwacji jego działania. Rezultatem są błędy merytoryczne w różnych źródłach. Projekt i implementacja niniejszego emulatora bazują w większości na dokumencie [5]. Dokument ten stara się kompleksowo opisać działanie Game Boy-a jako systemu, jednak jak dowiódł proces implementacji, a później weryfikacji, zawiera on pewne niejasności i błędy. Rozwiązanie tego problemu wymagało implementacji debuggera, mechanizmu logowania, analizy porównawczej działania programów czy zagłębienia się w sposób działania ROM-ów weryfikacyjnych.

5.2.2. Skomplikowany podsystem graficzny

Podsystem graficzny Game Boy-a działa równolegle do procesora. Z punktu widzenia emulacji na jego działanie składają się dość skomplikowane procesy, z dekodowaniem sprite-ów na czele. Początkowa, dość naiwna implementacja tego procesu była bardzo obciążająca dla procesora systemu gospodarza, przez co wymagała kilku optymalizacji. Działanie podsystemu grafiki miało wpływ na dostęp do pamięci OAM i pamięci Video. Pociąga to ze sobą spory wpływ na powiązanie z rejestrami statusowymi i przerwaniami, przez co finalnie podsystem grafiki, jest dość silnie powiązany z warstwą emulującą przestrzeń adresową konsoli, jak i z samym systemem. Niektóre programy na Game Boy-a wykorzystują zależności czasowe pomiędzy kolejnymi etapami pracy podsystemu

graficznego i o ile z punktu widzenia tej części emulatora udało się te relacje zachować, tak ze względu na problem opisany w podrozdziale 10.1.3, nie wszędzie działa to bezproblemowo.

5.2.3. Emulacja z dokładnością do cyklu procesora

Część programów pisanych na Game Boy-a silnie bazuje na zależnościach czasowych wynikających z czasu wykonywania rozkazów procesora. Wprowadza to wymaganie, aby podsystemy emulatora były ze sobą zsynchronizowane co do cyklu działania wirtualnego procesora. Wymaganie to komplikuje działanie systemu, spowalnia jego działanie i uodparnia go na wprowadzenie pewnych prostych optymalizacji, jak np. asynchronicznie działające podsystemy.

Efektem opisanych w rozdziale 10.1.1 problemów z dokumentacją jest nieuwzględnienie w projekcie faktu, że instrukcje warunkowe, zależnie od obranej ścieżki wykonania, mogą trwać różną ilość cykli procesora. Naprawa tego problemu wymaga przeprojektowania sporej części emulującej procesor Game Boy-a, jednak z powodu późnego jego wykrycia, nie zostało to dokonane do momentu ukończenia tego dokumentu. Problem ten udało się w większości przypadków obejść, jednak nadal nie można mówić o 100% dokładności. Z tego samego powodu emulator nie przechodzi weryfikacji działania ROM-em “instr_timing” opisanym w rozdziale 11.1. Testy wykazały, że obejście to jest wystarczające dla znakomitej większości programów stworzonych dla tej platformy.

5.3. Zastosowane ulepszenia

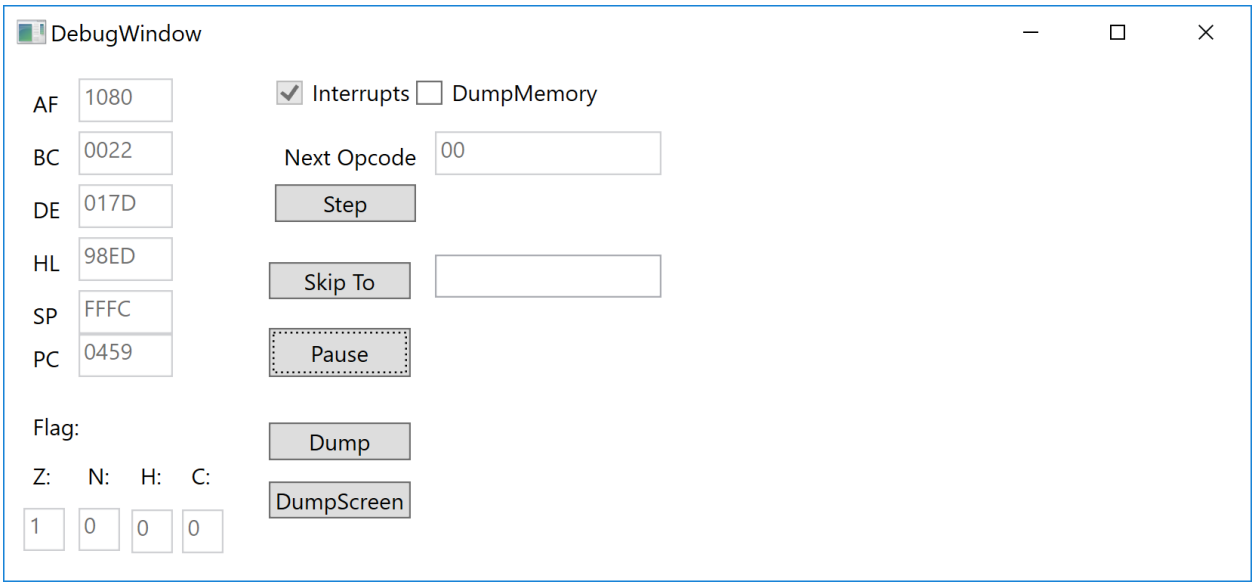
W fazie weryfikacji działania emulatora, zostało dodanych do niego kilka ulepszeń, wpływających na jego prędkość działania jak i możliwości analizy obserwowanych błędów.

5.3.1. Debugger

W toku weryfikacji działania emulatora z rzeczywistymi grami, powstało zapotrzebowanie na stworzenie debuggera, który pozwoliłby na analizę przypadków nieprawidłowego działania emulatora.

Został on wyposażony w następujące funkcje (rys. 5.1):

- Wykonywanie kodu programu rozkaz za rozkazem
- Wykonywanie kodu programu do momentu osiągnięcia zadanego adresu
- Zatrzymanie i wznowienie wykonania programu w dowolnym momencie
- Zapisanie zawartości pamięci emulowanego systemu do pliku
- Odświeżenie zawartości bufora grafiki bez wykonywania dalszych instrukcji procesora.
- Wyświetlanie stanu rejestrów procesora



Rysunek 5.1 Okno debuggera

5.3.2. Mechanizm logowania

Równolegle do debuggera, w emulatorze został zaimplementowany również mechanizm logowania. Kiedy jest on aktywny, zapisuje do pliku kolejne wykonywane rozkazy, razem z aktualnym adresem licznika programowego oraz rejestrami i adresami pamięci dotyczącymi danego rozkazu. Logi te są bardzo przydatne przy wykrywaniu nieprawidłowości w działaniu programu, jak nieskończone pętle czy wyjście poza zakres pamięci w którym znajduje się kod wykonywalny.

Początkowa implementacja mechanizmu logowania, powodowała blisko 20-krotny spadek prędkości emulacji, z powodu zapisywania do pliku przy wywołaniu każdej komendy. Pomyślnie udała się optymalizacja tego zachowania, poprzez dodanie bufora który zapisuje się do pliku po przepełnieniu, albo w razie nagłego zakończenia działania emulatora. Po zastosowaniu tej optymalizacji, spadek prędkości działania emulator jest już około 3-krotny. Z tego powodu mechanizm logowania jest użyty tylko w debugowej wersji emulatora.

5.3.3. Ulepszona reprezentacja przestrzeni adresowej

Badania z użyciem profilera wydajności wskazały, że kluczowymi dla wydajności operacjami w emulatorze jest dostęp do pamięci i funkcjonalność podsystemu graficznego. Z tego powodu podjęto próbę przyspieszenia działania odczytów i zapisów do pamięci. Reprezentacja przestrzeni adresowej, będąca wcześniej listą kolejnych akcesorów, została zamieniona na tablicę wiążącą adres pamięci, z akcesorem któremu on podlega. Dzięki tej zmianie, zużycie procesora w czasie emulacji spadło o ⅓, kosztem nieznacznego wzrostu zużycia pamięci.

5.3.4. Trzymanie przetworzonych kafelków w pamięci podręcznej

Kolejnym krokiem mającym na celu przyspieszenie działania emulatora, było wprowadzenie do podsystemu graficznego mechanizmu cache-owania. Najbardziej wymagającą funkcjonalnością jest tutaj metoda odpowiedzialna za odczytanie kafelka z pamięci wideo, a następnie przetworzenie go do postaci, która może być zapisana do bufora używanego przez emulator. Dodanie mechanizmu pozwalającego na zachowywanie przetworzonych kafelków w pamięci podręcznej, który odświeża je w przypadku

wystąpienia zmiany w pamięci wideo, pozwoliło na zmniejszenie obciążenia procesora pracą wątku graficznego o 50%.

5.3.5. Przeniesienie emulacji podsystemu graficznego do osobnego wątku.

Wstępny projekt emulatora zakładał działanie całego systemu w jednym wątku. Uruchamianie funkcji związanych z obsługą podsystemu graficznego, pomiędzy wykonywaniem kolejnych rozkazów procesora, było przyczyną sporego spowolnienia działania emulatora. Dopiero przeniesienie tej funkcjonalności do osobnego wątku i sprawienie, aby podsystem graficzny działał równolegle do obsługiwanych rozkazów (z synchronizacją wykonania kodu w celu zapewnienia dokładności emulacji), pozwoliło na osiągnięcie właściwej dla Game Boy-a szybkości emulacji poprzez 2-krotne przyspieszenie pracy całego emulatora.

5.3.6. Ograniczenie wpływu części służącej logowaniu na działanie wersji produkcyjnej

Pierwotna implementacja systemu logowania odcinała logowanie w wersji produkcyjnej na poziomie samego loggera. Użycie profilera wydajności wskazało jednak, że użycie mechanizmu zamieniającego typ wyliczeniowy na typ napisowy, na poziomie samego wywołania interfejsu loggera, również ma spory wpływ na wydajność. Z tego powodu, zamiast blokować działanie loggera w wersji produkcyjnej wewnątrz jego implementacji, zastosowano dyrektywę warunkową preprocesora w celu wyłączenia wszystkich wywołań z kompilacji w przypadku budowania produkcyjnej wersji emulatora. Pozwoliło to uzyskać przyrost wydajności na poziomie 20%.

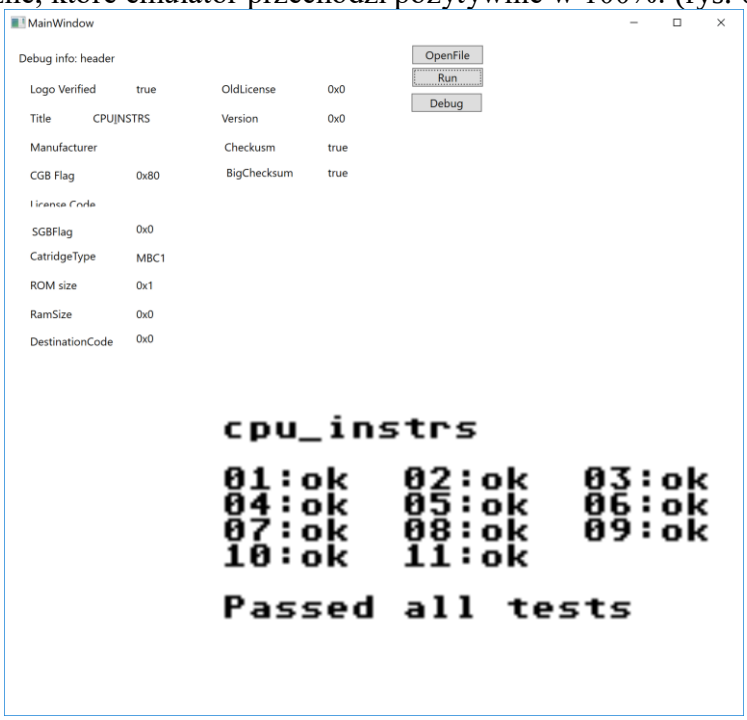
6. Weryfikacja działania

Zaimplementowany emulator wymaga weryfikacji poprawności jego działania. Wykorzystano do niej dwa sposoby: ROM-y testowe i oprogramowanie homebrew.

6.1. ROM-y testowe

W internecie dostępne są ROM-y, które zakładając poprawne działanie kilku podstawowych rozkazów procesora i obsługi wyświetlacza, pozwalają zweryfikować działanie emulatora w dwóch dostępnych scenariuszach:

a) “cpu_instrs” - Weryfikuje poprawność wykonywania instrukcji obsługiwanych przez procesor. Sprawdza kombinacje użycia rozkazów we wszystkich możliwych wariacjach rejestrów ze skrajnymi wartościami obsługiwanych przedziałów. W oparciu o wykorzystanie tego ROM-u testowego zostały utworzone testy automatyczne, które emulator przechodzi pozytywnie w 100%. (rys. 6.1)



Rysunek 6.1 Rezultat działania ROM-u cpu_instrs

b) “instr_timing” - Weryfikuje dokładność emulacji co do liczby cykli zużytych na obsługę poszczególnych rozkazów procesora. Testy te wykazały (rys. 6.2), że błędy co do liczby cykli procesora potrzebnych na wykonanie niektórych grup rozkazów. Więcej informacji na ten temat można znaleźć w rozdziale 13.2.

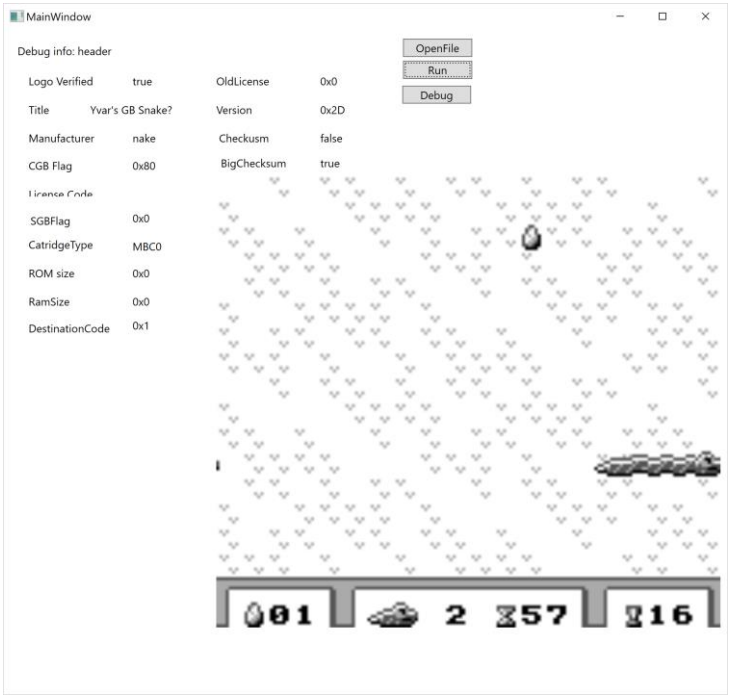
Rysunek 6.2 Rezultat działania ROM-u intr_timing

6.2. Programy homebrew

W celu weryfikacji działania na emulatorze uruchomiłem 2 gry typu homebrew (amatorskie produkcje dostępne za darmo)

6.2.1. Yvar's GB snake

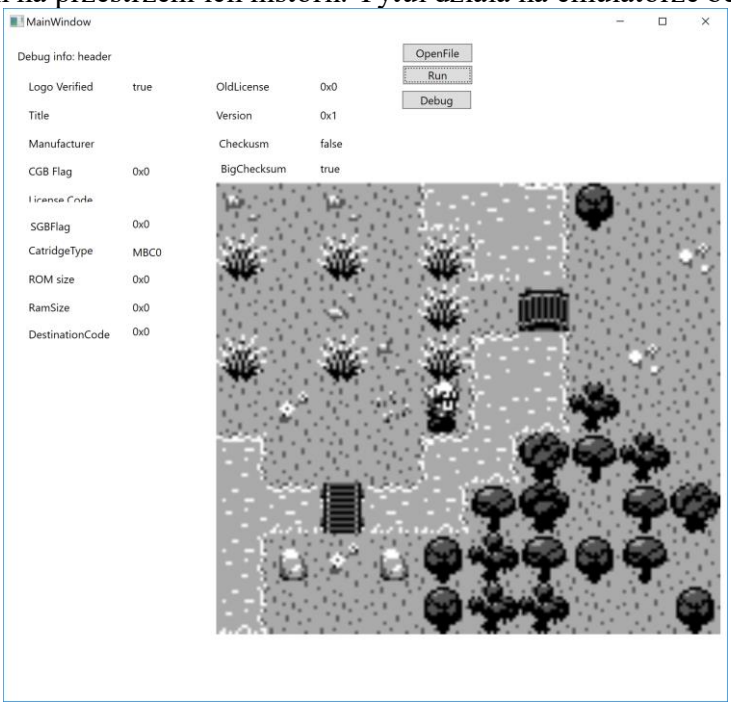
Amatorska gra (rys. 6.3) stworzona przez Yvar'a de Goffau w 2017 roku. Jest to implementacja gry zręcznościowej typu snake. W czasie rozgrywki nie zauważono żadnych problemów, program zachowuje się tak jak powinien.



Rysunek 6.3 Gra Yvar's GB Snake uruchomiana w emulatorze

6.2.2. Evoland

Evoland (rys. 6.4) to stworzony przez fana - Fabien'a Loison'a - port gry komputerowej to tym samym tytule. Gra z gatunku action-RPG ilustruje rozwój mechanik rozgrywki używanych w grach na przestrzeni ich historii. Tytuł działa na emulatorze bezproblemowo.



Rysunek 6.4 Gra Evoland uruchomiona w emulatorze

7. Możliwości rozwoju

W procesie tworzenia emulatora, wskazano następujące potencjalne możliwości jego rozwoju w przyszłości:

7.1. Zbadanie i poprawna implementacja emulacji z dokładnością do cyklu procesora

Jak zostało już wspomniane, z powodu błędów w dokumentacji, emulator nie osiągnął pełnej dokładności emulacji co do cyklu procesora. Stan ten możliwy jest do naprawy. Wymaga to przeprojektowania części odpowiedzialnej za reprezentacje rozkazu procesora, w ten sposób, aby pozwalała w dynamiczny sposób wyliczać ilość cykli potrzebnych na wykonanie danego rozkazu. Poprawa implementacji pod tym kątem powinna naprawić rzadkie błędy w niektórych komercyjnych grach, jak “BombJack”, czy “Pokemon”.

7.2. Implementacja dynamicznej rekompilacji

Dynamiczna rekompilacja, opisana co do działania w rozdziale 6.2.2, jest kolejnym sposobem na przyspieszenie działania emulatora. W ramach sprawdzenia zastosowania języka C# do stworzenia emulatora, powstał eksperymentalny kod, tłumaczący kilka rozkazów procesora z80 na kod pośredni środowiska .NET, potwierdzając możliwość zastosowania dynamicznej rekompilacji w tym emulatorze. Wykorzystanie kodu pośredniego środowiska .NET pozwala w dodatku na zachowanie przenaszalności implementacji emulatora pomiędzy platformami, coś co jest niemożliwe w przypadku implementacji mechanizmu dynarec w języku C++. Zmiana ta wymaga solidnego przeprojektowania części wykonującej rozkazy procesora i dobrego projektu mechanizmu pozwalającego zsynchronizować działanie procesora z innymi podsystemami.

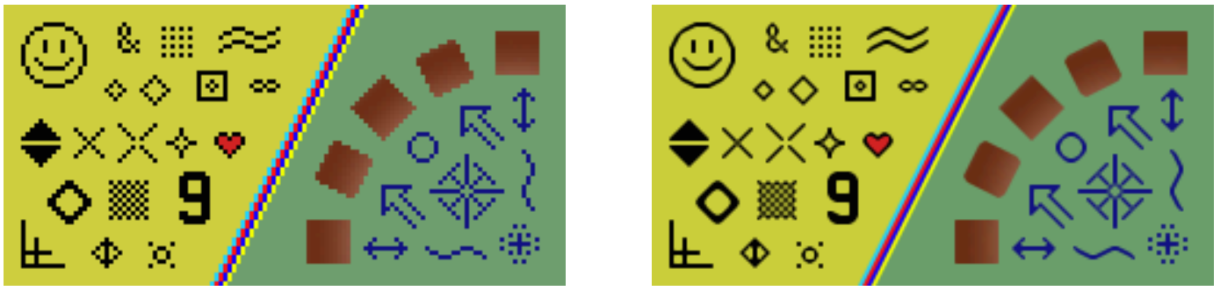
7.3. Poprawa wyglądu i funkcjonalności GbEmuApp

GbEmuApp - to aplikacja implementująca prosty interfejs użytkownika. Nie pozwala ona na żadną możliwość konfigurację działania emulatora. W celu uczynienia jej bardziej przyjazną użytkownikowi, należy przeprojektować interfejs, wg. następujących wytycznych:

- Przeniesienie informacji odczytanych z nagłówka do osobnego, domyślnie ukrytego okna.
- Dodanie panelu pozwalającego na konfigurację emulatora: sterowania, szybkości emulacji, czy tego jak emulator ma się identyfikować dla emulowanych programów.
- Dodanie możliwości skalowania okna, w tym skalowania obszaru wyświetlacza.

7.4. Zastosowanie filtrów graficznych w celu poprawy jakości generowanego obrazu

Rozdzielczość ekranu Game Boy-a to 160 na 144 piksele. Wymiar obszaru wyświetlacza w oknie emulatora na wielu współczesnych komputerach jest większy niż ta rozdzielczość. WPF oferuje domyślne włączone skalowanie liniowe, pozwalające na skalowanie rozdzielczości bitmapy. Możliwe jednak jest zaimplementowanie filtrów, takich jak HQ2X [11] które podnoszą jakość wyświetlanego obrazu, w stosunku np. do metody najbliższego sąsiada [12] (rys. 7.1)



Rysunek 7.1 Różnica pomiędzu metodami skalowania - najbliższego sąsiada i hq2x (źródło: [13])

7.5. Implementacja obsługi dźwięku

Jedną z najważniejszych funkcjonalności, która została pominięta w projekcie emulatora, ze względu na założenia, jest emulacja dźwięku. Podsystem ten może być łatwo zaimplementowany, poprzez podmianę akcesora obsługującego adresy związane z rejestrami dźwięku.

7.6. Rozszerzenie i reorganizacja funkcjonalności debuggera

Elementem który został późno zaimplementowany jest debugger. Z racji, że początkowy projekt w ogóle go nie przewidywał, funkcjonalność debuggera została dodana w kilku miejscach. Jako możliwe ulepszenie tego stanu rzeczy, można refaktoryzować debugger, umieścić go w projekcie emulatora oraz możliwie dodać tam funkcje jak np. wyświetlanie obserwowanych adresów w pamięci.

7.7. Dodanie wsparcie dla Super Game Boy-a (SGB) i Game Boy-a Color (GBC).

Super Game Boy i Game Boy Color to urządzenia niewiele różniące się od Game Boy-a pod względem sprzętowym. Zawierały ten sam procesor, jednak tym razem wprowadzono obsługę trybu działania o 2 razy wyższym taktowaniu. Game Boy Color wprowadza do tego kolorowy ekran, zmieniając sposób kodowania sprite-ów. Obie te zmiany nie wymagają drastycznych zmian w projekcie emulatora.

7.8. Zapisywanie stanu

Jedną z najpopularniejszych funkcjonalności oferowanych przez współczesne emulatory jest możliwość zapisu stanu gry w dowolnym momencie. Jest ona realizowana poprzez dokonanie zrzutu zawartości pamięci konsoli, jak i jej rejestrów procesora.

8. Podsumowanie

W ramach postawionego celu udało się zaimplementować emulator będący w stanie uruchamiać oprogramowanie stworzone dla konsoli Game Boy. Emulator nie jest kompletny – posiada ograniczenia zgodne z założeniami tej pracy. Możliwa jest bezproblemowa rozgrywka w kilka tytułów typu homebrew, jak i uruchomienie zrzutów oryginalnych gier z Game Boy-a.

Oryginalny projekt, który był punktem wyjścia do implementacji okazał się nie uwzględniać pewnych zawiłości działania tej konsoli. Z tego powodu, w celu poprawy działania emulatora sugeruje się jego dalszy rozwój, zgodnie z kierunkiem opisanym w rozdziale 11.

Około 40% czasu spędzonego nad tą pracą zabrało zbieranie wymagań - wyszukiwanie w sieci internetowej co dokładniejszych opisów działania podsystemów konsoli, czytanie dokumentacji i łączenie tej wiedzy w całość. Proces implementacji i weryfikacji pokazał jednak, że projekt ten nadal wymaga spędzenia większej ilości czasu nad analizą działania konsoli, w celu zaimplementowania dokładniejszego emulatora.

Technologie użyte w projekcie sprawdziły się bardzo dobrze. WPF pozwolił na bardzo szybkie stworzenie użytecznego interfejsu. C# pozwolił na łatwe stworzenie kodu pasującego do projektu, szybkie prototypowanie rozwiązań. Pomimo narzutu wprowadzonego przez środowisko uruchomieniowe platformy .NET, emulator działa płynnie, nie obciążając komputera w nadmierny sposób. Środowisko programistyczne Visual Studio jako środowisko dedykowane dla tworzenia oprogramowania z użyciem C#, ułatwiło pracę udostępniając przydatne narzędzia do refaktoryzacji kodu i analizy kodu. Jednym z najbardziej przydatnych tego typu narzędzi był wbudowany profiler kodu, który pozwolił na analizę wydajności aplikacji w trakcie działania emulacji i wyróżnienie kluczowych obszarów, których optymalizacja pozwoliła na osiągnięcie najlepszych skutków, jeśli chodzi o poprawę wydajności emulacji.

9. Bibliografia

[1] „Emulator - Wikipedia” - https://en.wikipedia.org/wiki/Emulator. [Data uzyskania dostępu: 31.05.2019].

[2] „Just-in-time compilation – Wikipedia” - https://en.wikipedia.org/wiki/Just-in-time_compilation . [Data uzyskania dostępu: 31.05.2019].

[3] „Ahead-of-time compilation - Wikipedia” - https://en.wikipedia.org/wiki/Ahead-of-time_compilation. [Data uzyskania dostępu: 31.05.2019].

[4] „Game Boy - Wikipedia” - https://en.wikipedia.org/wiki/Game_Boy. [Data uzyskania dostępu: 31.05.2019].

[5] „Game Boy CPU Manual” - http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf. [Data uzyskania dostępu: 31.05.2019].

[6] „Proxy pattern - Wikipedia” - https://en.wikipedia.org/wiki/Proxy_pattern. [Data uzyskania dostępu: 31.05.2019].

[7] „Thread pool - Wikipedia” - https://en.wikipedia.org/wiki/Thread_pool . [Data uzyskania dostępu: 31.05.2019].

[8] „Przewodnik po języku C# | Microsoft Docs” - https://docs.microsoft.com/pl-pl/dotnet/csharp/. [Data uzyskania dostępu: 31.05.2019].

[9] „Mechanizm refleksji - Wikipedia” - https://pl.wikipedia.org/wiki/Mechanizm_refleksji. [Data uzyskania dostępu: 31.05.2019].

[10] „Windows Presentation Foundation | Microsoft Docs” - https://docs.microsoft.com/en-us/dotnet/framework/wpf/. [Data uzyskania dostępu: 31.05.2019].

[11] „Hqx - Wikipedia” - https://en.wikipedia.org/wiki/Hqx. [Data uzyskania dostępu: 31.05.2019].

[12] „Tech-Algorithm.com ~ Nearest Neighbor Image Scaling” - http://tech-algorithm.com/articles/nearest-neighbor-image-scaling/. [Data uzyskania dostępu: 31.05.2019].

[13] „HiEnd3D” - https://web.archive.org/web/20131205091805/http://www.hiend3d.com/hq2x.html. [Data uzyskania dostępu: 31.05.2019].

[14] „List of best selling game consoles - Wikipedia” - https://en.wikipedia.org/wiki/List_of_best-selling_game_consoles#cite_note-GB_and_GBC-32. [Data uzyskania dostępu: 31.05.2019].

10. Spis rysunków

Rysunek 4.1 Podział emulatora na artefakty....12
Rysunek 4.2 Podział emulatora na wątki z uwagi na ich artefakt pochodzenia 12
Rysunek 4.3 Model klasy dla klasy GbSystem....13
Rysunek 4.4 Model klasy Catridge 14
Rysunek 4.5 Model klasy Input 14
Rysunek 4.6 Model klasy MemProxy 15
Rysunek 4.7 Model klasy MBCMemAccess 15
Rysunek 4.8 Model klasy CpuCore....16
Rysunek 4.9 Sekwencja przetwarzania instrukcji przez procesor 16
Rysunek 4.10 Model klasy GpuCore....17
Rysunek 4.11 Maszyna stanowa wątku wykonawczego Gpu 18
Rysunek 5.1 Okno debuggera 22
Rysunek 6.1 Rezultat działania ROM-u cpu_instrs 24
Rysunek 6.2 Rezultat działania ROM-u intr_timing....25
Rysunek 6.3 Gra Yvar's GB Snake uruchomiana w emulatorze....26
Rysunek 6.4 Gra Evoland uruchomiona w emulatorze 26
Rysunek 7.1 Różnica pomiędzu metodami skalowania - najbliższego sąsiada i hq2x (źródło: [13]) 28