**GRAFICZNY AUTOMAT PERKUSYJNY**

**Michał KĘDZIA**1**, Adrian KRĘCIKOWSKI**2**, Tomasz WOJTKIEWICZ**3

1. nr albumu: 222463 e-mail: 222463@edu.p.lodz.pl
2. nr albumu: 222482 e-mail: 222482@edu.p.lodz.pl
3. nr albumu: 222616 e-mail: 222616@edu.p.lodz.pl

**Streszczenie:** Program przekształcający obraz wejściowy na pętlę perkusyjną zapisaną do pliku z rozszerzeniem .wav oraz wyświetlający daną pętlę jako reprezentację graficzną sygnału dźwiękowego w postaci wykresu funkcji sygnału audio od czasu.

**Słowa kluczowe:** muzyka, automat perkusyjny, fala dźwiękowa.

**1. WSTĘP**

Celem projektu jest utworzenie automatu perkusyjnego, który na wejściu otrzymuje prosty obrazek utworzony przez użytkownika (zgodnie z pewnymi ustalonymi przez nas założeniami), a następnie na jego podstawie odtwarza i/lub zapisuje ścieżkę dźwiękową do pliku z rozszerzeniem .wav. Ma to być nasza próba połączenia pasji do muzyki z programowaniem, a także szansa na poznanie bibliotek obsługujących dźwięk. Przede wszystkim chcemy jednak dostarczyć użytkownikom tego programu trochę zabawy z muzyką i rozbudzić w nich w mniejszym lub większym stopniu zdolności muzyczne. Jako ciekawostkę warto wspomnieć, że w trakcie tworzenia projektu w serwisie Youtube pojawił się film użytkownika Dylan Tallchief przedstawiający podobny koncept zrealizowany w programie Microsoft Excel. Link do filmu:

<https://www.youtube.com/watch?v=To2JIXGoYzA>

**2. MATERIAŁY I METODY**

**2.1. Dane**

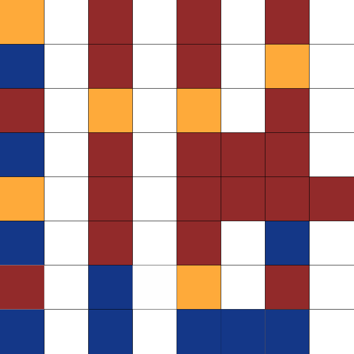
Dane obrazowe wykorzystane w realizacji projektu można podzielić na dwie kategorie: te, które zostały przez nas stworzone na potrzeby programu w celu ułatwienia użytkownikowi pracy z nim, oraz te, które nie tworzą przyjemnego dla ucha rezultatu i są rozpatrywane jedynie doświadczalnie z ciekawości do reakcji programu na ich użycie.

Stworzone przez nas obrazy bazują na szablonie(1) przedstawiającym czarne linie siatki dzielące białe tło o wymiarach 800x800 pikseli na 8 kolumn i 8 wierszy.

.

*(1)Omawiany szablon*

Szablon ten został stworzony, aby użytkownikowi łatwiej było narzędziem wypełnienia ( wiadro z farbą ) pokolorować obraz wg uznania. Takie przykładowo pokolorowane obrazy(2) również znajdują się w dołączonym do projektu folderze.



*(2)Przykładowo wypełniony szablon*

Rzucającą się w oczy właściwością tych obrazów jest powtarzający się zestaw kolorów. Jest to cecha kluczowa dla poprawnego działania programu i zostanie ona omówiona w późniejszej części sprawozdania.  
Druga kategoria danych zawiera obrazy nie spełniające wytycznych naszego szablonu. Jest to np. obraz ‘Mona Lisa’, zdjęcie motocykla czy wykonana w programie Paint tęcza. Te obrazy pochodzą (za wyjątkiem tęczy) ze strony przedmiotu Przetwarzanie sygnałów i obrazów w serwisie Wikamp i są dostępne po zalogowaniu się.

**2.2. Metody**

Interfejs użytkownika zrealizowany jest w okienku konsoli. Program na wstępie pyta nas o nazwę pliku, który ma odczytać, informując jednocześnie o konieczności umieszczenia go w folderze Pliki\_projektu. Po upewnieniu się, że podana nazwa wskazuje na istniejący plik program otwiera go i prosi o podanie tempa w standardowej jednostce przemysłu muzycznego BPM – Beats Per Minute. Jeżeli użytkownik poda tempo mniejsze niż 30 BPM lub większe niż 220 BPM program zignoruje podaną wartość i będzie prosić do skutku (wyjaśnienie narzuconego ograniczenia w części 2.4 Eksperymenty).

Z podanego tempa program obliczy czas w milisekundach, jaki powinien dzielić odtworzenie dwóch pojedynczych dźwięków, zgodnie z poniższym wzorem(3), w którym delay to odstęp pomiędzy dźwiękami, 60000 to jedna minuta w milisekundach, BPM to podane przez użytkownika tempo a dzielenie przez cztery odbywa się aby uzyskać odstęp między szesnastkami.

*(3)Wzór na odstęp między dźwiękami*

Kolejnym krokiem jest zapytanie o ilość powtórzeń danej pętli. Ma to na celu ułatwić tworzenie dłuższych utworów składających się z powtarzającej się pętli.

Po zebraniu danych od użytkownika zaczyna się właściwa część programu. Wywołuje się funkcja create\_output(), która jest głównym algorytmem projektu. Zwracaną wartością jest gotowy do wyeksportowania w formacie .wav ciąg danych tworzących pętlę perkusyjną. Poniżej algorytm opisujący jej działanie:

1. Utwórz pustą tablicę *table*
2. Znajdź uśrednioną wartość RGB obszaru dookoła wybranego/pierwszego piksela obrazu
3. Porównaj wartość z pkt 2 i przypisz ją do najbliższej wartości ze słownika *colorVal*
4. Do tablicy *table* dodaj dźwięk w zależności od koloru uzyskanego w pkt 3.
5. Do tablicy *table* dodaj kawałek ciszy tak, aby:

*długość sampla + kawałek ciszy = delay*

1. Przeskocz w prawo o zadaną liczbę pikseli
2. Jeżeli nie dotarłeś do prawej krawędzi idź do pkt 2
3. Dotarłeś do prawej krawędzi. Przeskocz w dół o zadaną liczbę pikseli, a następnie do lewej krawędzi.
4. Jeżeli nie dotarłeś do dolnej krawędzi, idź do pkt 2
5. Dotarłeś do dolnej krawędzi. Powtórz skanowanie obrazu od początku (pkt 2, pierwszy piksel) tyle razy, ile powtórzeń zażądał użytkownik – 1.
6. Zainicjuj nową zmienną *output* i przypisz 1 ms ciszy.
7. Przejdź po tablicy *table* i po kolei dodawaj jej wartości do zmiennej *output*
8. Na koniec dodaj do zmiennej *output* 200 ms ciszy, aby podczas odtwarzania w programach takich jak Windows Media Player dźwięk na końcu nie urywał się niespodziewanie
9. Zwróć zmienną *output*

Po zwróceniu zmiennej output program odtwarza uzyskany dźwięk i wyświetla go jako funkcję sygnału dźwiękowego od czasu na wykresie. Po jego zamknięciu użytkownik dostaje możliwość zapisania utworzonego pliku pod wybraną przez siebie nazwą.

**2.3. Technologie i narzędzia**

Język programowania: Python 3.7

Wybór języka Python nie jest przypadkowy. Jest to język skryptowy posiadający wiele potężnych bibliotek do obsługi obrazów, plików dźwiękowych i nie tylko. Jest językiem wysokopoziomowym, co wpływa na krótszy i czytelniejszy kod. Pozwala to skupić pracę na części umysłowo-teoretycznej zamiast, jak to jest w przypadku języków niskiego poziomu, na samym przetwarzaniu pomysłu w długi, skomplikowany kod.

Biblioteki:

* PIL (obsługa obrazów),
* Wave (obsługa plików .wav),
* Pydub (uproszczone poruszanie się po bibliotece Wave),
* Numpy (operowanie sygnałem audio jako tablicą wartości liczbowych),
* Math (funkcje matematyczne),
* Os (funkcje systemu operacyjnego, np. usuwanie pliku),
* Time (obsługa czasu),
* Matplotlib (obsługa wykresów).

Programy graficzne: MS Paint, SketchBook.

Komputery i ich zasoby obliczeniowe:

* MacBook Pro 13, procesor 2,3 GHz dwurdzeniowy Intel® Core™ i5, RAM 8 GB 2113 MHz LPDDR3, OS macOs Catalina 10.15.12
* Dell precision M4800, procesor czterordzeniowy Intel® Core™ i7-4900QM, RAM 16 GB, OS Windows 10 Pro x64
* PC, procesor Intel® Core™ i3-3220 3.3 GHz, RAM 8 GB, OS Windows 10 Pro x64
* Dell Inspiron 15 5000 series, procesor Intel® Core™ i7 czterordzeniowy, RAM 16 GB, OS Windows 10 Pro x64

Link do repozytorium Github:

**! WKLEIĆ LINK DO PROJEKTU NA GITHUB !**

**2.4. Eksperymenty**

**2.4.1 Eksperyment pierwszy: ustalenie dolnego i górnego ograniczenia tempa podawanego przez użytkownika.**

Przewidywane rezultaty: przy zbyt wolnym tempie odstęp pomiędzy dźwiękami będzie zbyt duży, co spowoduje niezdolność słuchacza do poczucia rytmu danej pętli. Przy zbyt szybkim tempie program nie będzie w stanie odtworzyć dźwięków z oczekiwaną prędkością, ponieważ musiałby zacząć odtwarzać następny sampel zanim skończy odtwarzać pierwszy. Program nie obsługuje odtwarzania kilku dźwięków na raz, więc nie poradzi sobie z zawrotnymi prędkościami.

Przebieg eksperymentu: uruchomić program i jako obraz podać do wczytania ‘obraz\_2\_projekt.png’. Nie ma na nim białych kwadratów, co zmusi program do ciągłego odtwarzania dźwięków jeden po drugim, bez dodawania przerwy. Zacząć od BPM = 0 i iść w górę ze skokiem 30 BPM lub mniejszym. Zanotować obserwacje do tabeli.

Tabela 2.4.1.

|  |  |
| --- | --- |
| Tempo (BPM) | Obserwacje |
| 0 | Dzielenie przez zero (patrz wzór(3)) |
| 30 | Powolny, odczuwalny rytm. Długość > 60s |
| 60 | Żwawy rytm. |
| 90 | Szybki rytm. |
| 120 | Bardzo szybki rytm. |
| 150 | Ciąg następujących po sobie dźwięków. |
| 180 | Jak wyżej, ciężko zauważyć różnicę. |
| 210 | Dźwięki prawie nachodzą na siebie. |
| 220 | Jak wyżej. |
| -20 | Jak wyżej. |

Wnioski: przy takim rozstawieniu dźwięków, nawet niewielkie tempa potrafią sprawiać wrażenie szybkich utworów. Warunki powyższego eksperymentu były ekstremalne. Umiejętne rozmieszczanie dźwięków na szablonie zmniejszy tempo i pozwoli na dodawanie dynamiki do tworzonych dźwięków. Przy tempie ujemnym delay również staje się ujemny. Program został zabezpieczony wcześniej przed ujemną wartością delay i w takim przypadku traktuje delay jak 0. Dlatego przy tempie   
-20 BPM program zachowuje się jak przy 210 BPM. Eksperyment należy powtórzyć dla umiejętnie przygotowanego obrazu.

**2.4.2 Eksperyment drugi: ciąg dalszy ustalania granic tempa dla mniej ekstremalnych danych**

Przewidywane rezultaty: tym razem przewidujemy, że do uzyskania szybszego rytmu trzeba będzie użyć szybszego tempa.

Przebieg eksperymentu: uruchomić program i jako obraz podać do wczytania ‘obraz\_trap\_projekt.png’. Zostały na nim umieszczone pauzy tak, aby w niektórych momentach móc je wypełnić tworząc tzw. przejścia perkusyjne. Powtórzyć wypełnianie tabelki z eksperymentu 2.4.1.

Tabela 2.4.2.

|  |  |
| --- | --- |
| Tempo (BPM) | Obserwacje |
| 0 | Dzielenie przez zero (patrz wzór(3)) |
| 30 | Rytm nieodczuwalny. |
| 60 | Odczuwalny rytm, kolejność dźwięków wydaje się przypadkowa. |
| 90 | Bardzo powolny rytm. |
| 120 | Wolny rytm. |
| 150 | Żwawy rytm. |
| 180 | Szybki rytm. |
| 210 | Bardzo szybki rytm. |
| 240 | Ciężko zauważyć różnicę. |

Wnioski: Przy umiejętnym rozmieszczeniu dźwięków na szablonie, zmiany tempa przesunęły się na dół względem tabeli. Wyniki zgadzają się z rzeczywistością. Większość radiowych hitów jest nagrywana właśnie w przedziale 80 – 160 BPM, gdzie 80 BPM to często bardzo powolne melodie jazzowe a 140 BPM to szybkie, nowoczesne brzmienia. Na podstawie tego jak i poprzedniego eksperymentu narzucamy w programie dolne ograniczenie 30 BPM oraz górne ograniczenie 220 BPM.

**2.4.3 Eksperyment trzeci: reakcja programu na obrazy nieszablonowe**

Przewidywane rezultaty: wynikowy plik dźwiękowy programu będzie zlepkiem przypadkowo dodawanych dźwięków. Jeżeli obraz ma wyraźną charakterystykę barwową, np. większość obrazu to odcienie żółtego, to te same dźwięki będą powtarzały się po sobie przez większość czasu.

Przebieg eksperymentu: uruchomić program. Jako obrazy podać do wczytania w kolejności: ‘mona\_lisa.jpg’, ‘lena.png’, ‘motorcycle\_left.png’ oraz ‘rainbow.jpg’. Zapisać obserwacje do tabeli. Odtwarzać w prędkości 220 BPM.

Tabela 2.4.3.

|  |  |
| --- | --- |
| Obraz | Obserwacje |
| mona\_lisa.jpg | Na początku długa cisza, potem na zmianę grający werbel i talerz. |
| lena.png | Talerz, czasami przerywany ciszą i uderzeniami werbla. |
| motorcycle\_left.png | Pół na pół talerz i werbel. |
| rainbow.jpg | Po kolei seria talerza, tzw. stopy oraz werbla. |

Wnioski: tylko tęcza wywołała pojawienie się stopy. Świadczy to o przeważających barwach czerwonych i niebieskich na obrazach. Ciekawe rezultaty mogłoby dać dodanie koloru zielonego do programu. Cisza w obrazie Mona Lisa spowodowana była prawdopodobnie czasem działania programu, a nie jak wpierw pomyśleliśmy dużą ilością koloru białego u góry obrazu.

**2.4.4 Eksperyment czwarty: badanie wydajności na różnych urządzeniach**

Przebieg eksperymentu: można powiedzieć, że ten eksperyment był prowadzony podczas samego pisania programu. Do testów zostały użyte urządzenia wymienione w pkt 2.3. Opiszemy więc po prostu nasze obserwacje podczas pracy z programem na różnych typach komputerów.

Wnioski: Program działa sprawnie na wszystkich komputerach. Przy uruchamianiu pojawia się warning:

*„RuntimeWarning: Couldn't find ffmpeg or avconv - defaulting to ffmpeg, but may not work*

*warn("Couldn't find ffmpeg or avconv - defaulting to ffmpeg, but may not work", RuntimeWarning)”.*

Ignorujemy jednak ten błąd, ponieważ nie zauważyliśmy jego negatywnego wpływu na działanie programu. Warto wspomnieć tutaj o ciekawej zmianie podejścia do odtwarzania pliku wynikowego. W pierwszej wersji chcieliśmy odtwarzać pojedyncze dźwięki na żywo jeden po drugim tak, aby nigdzie nie trzeba było tworzyć pliku tymczasowego. Takie podejście sprawiało jednak problemy- pojawiało się opóźnienie spowodowane samym działaniem programu i wykonywaniem kodu. Dlatego w finalnej wersji zdecydowaliśmy się na eksport danych do tymczasowego pliku .wav i późniejsze go odtworzenie. Wpłynęło to na zniwelowanie jakichkolwiek dodatkowych opóźnień między dźwiękami.

**2.4.5 Eksperyment piąty: badanie reakcji użytkowników na program**

Przebieg eksperymentu: przedstawić program testerom

i zanotować opinie.

Opinia #1: „Jako tester mogę stwierdzić że od strony obsługi program jest zrobiony dość intuicyjnie. Korzystnie z niego nie sprawia żadnych problemów. Jest to program bazowy można do niego dodać kilka dodatkowych elementów np. wysokość dźwięku mogłaby być zależna od stopnia jasności koloru (ciemniejszy kolor niższe dźwięki, jaśniejszy wyższe). Generalnie program jest dość ciekawy i podoba mi się. „

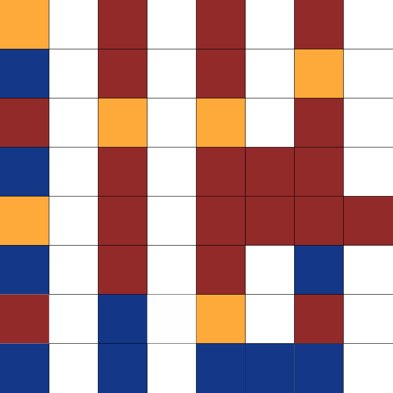
Opinia #2: „Program sam w sobie jest łatwy i przyjemny w obsłudze, poprzez szybkie kolorowanie w Paincie można stworzyć fajny beat, co sprawia frajdę.”

Opinia #3: „Program jest prosty ale daje dużo frajdy. Jako muzyk cenie sobie wszystkie kreatywne podejścia do mojej pasji. Jest to w zasadzie całkiem niezły i ciekawy początek pełnoprawnego DAW-a (Digital Audio Interface).”

Wnioski: obsługa programu nie sprawia użytkownikom trudności. Reakcje są pozytywne, a testerzy wydają się mieć frajdę z własnoręcznego tworzenia własnych pętli perkusyjnych.

**3. REZULTATY**

Plikiem wynikowym naszego programu jest przede wszystkim obraz zawierający wykres sygnału od czasu naszej pętli oraz plik audio również zawierający stworzoną pętlę. Oto dwa przykładowe porównania obrazów przed i po działaniu programu:



*3.1 Przed*

Obraz zawierający zrzut ekranu

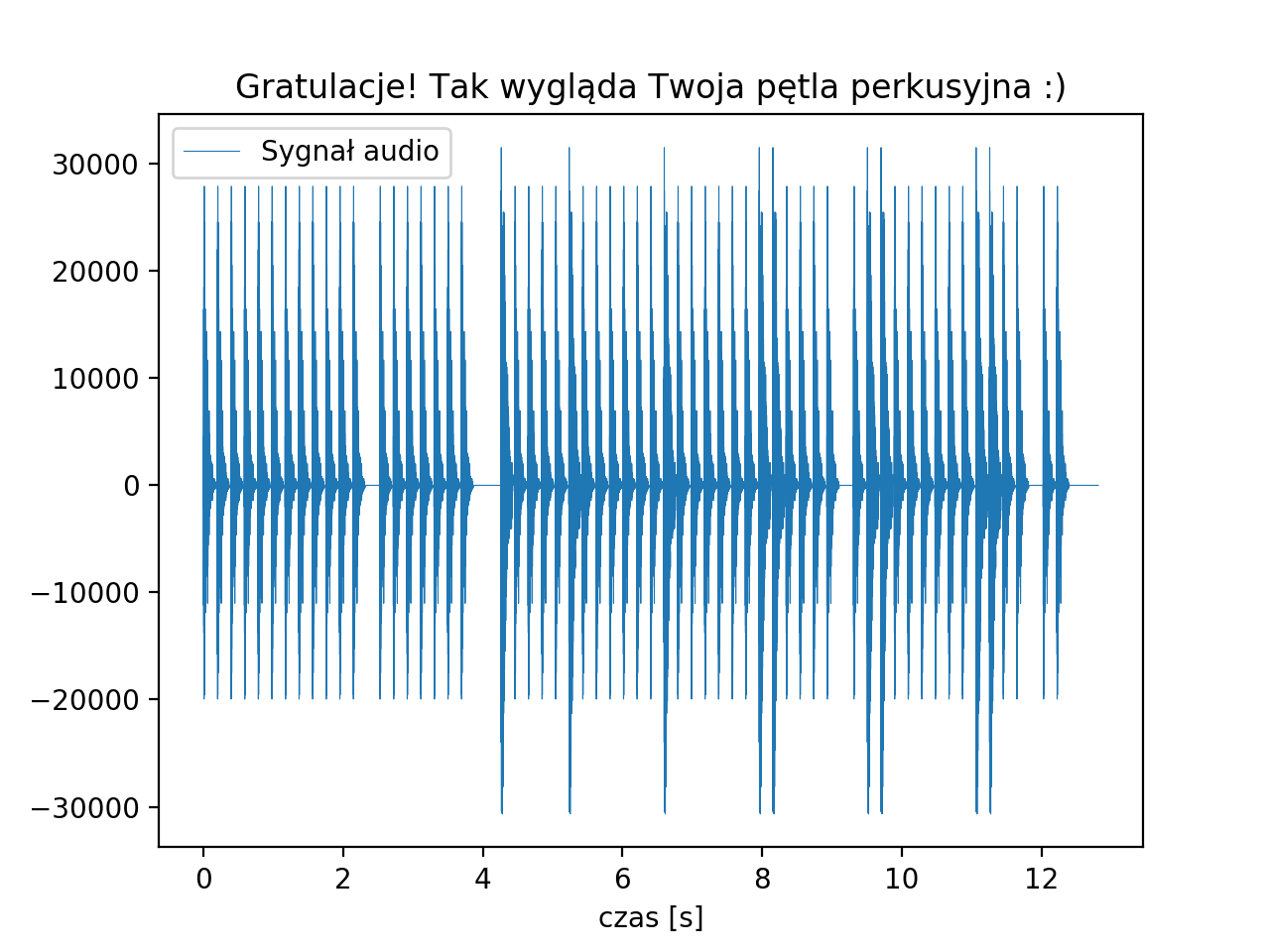
Opis wygenerowany automatycznie

*3.1 Po*

Obraz zawierający osoba, kobieta, odzież, przybranie głowy

Opis wygenerowany automatycznie

*3.2 Przed*

**

*3.2 Po*

Pliki wynikowe w formacie .wav znajdują się w folderze z projektem pod odpowiadającą im nazwą 3.1.wav oraz 3.2.wav.

**4. DYSKUSJA**

Wyniki działania naszego programu nie zaskakują nas i zostały już omówione w części 2.4 (eksperymenty). Jesteśmy natomiast bardzo pozytywnie zaskoczeni jakością i rytmiką pliku 3.1.wav. Taka pętlą perkusyjna mogła być częścią jakiegoś popularnego utworu. Jak wspomniano w części 2.4, obraz nie spełniający wymagań naszego szablonu nie jest przyjemny do słuchania. Można jednak określić barwy występujące w obrazie słuchając jakie elementy perkusyjne pojawiają się w pliku i mając na uwadze jakim kolorom odpowiadają.

Wady i zalety korzystania z naszego programu:

1. prosty w obsłudze
2. darmowy
3. plik wynikowy nie zawiera opóźnień
4. umiejscowienie dźwięków jest precyzyjne co do milisekundy
5. mało dźwięków
6. brak możliwości wgrania własnych brzmień
7. brak możliwości modulacji wysokości sampli
8. do obsługi niezbędny jest program graficzny

**5. PODSUMOWANIE I WNIOSKI**

Jesteśmy zadowoleni z naszego programu. Jest w nim wiele miejsca na dodanie ulepszeń i poprawienie wad z pkt 4, ale wierzymy, że sama baza jest solidna i zbudowana na dobrych algorytmach i wzorach. Podczas testów mieliśmy też okazję obserwować naszych bliskich bawiących się w producentów muzycznych, co sprawiło frajdę im jak i nam. A taki w końcu był główny cel naszego projektu. Dlatego każda osoba, która sięgnie po nasz program i pobawi się z nim chociaż przez   
10- 15 minut, będzie dla nas sukcesem. Chcemy rozwijać nasz projekt dalej jeżeli tylko będzie to w naszym zasięgu wiedzy i umiejętności.