UNIWERSYTET GDAŃSKI

WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI

Mikołaj Błaszczyk

Kierunek: Informatyka

Specjalność: Tester programista

Numer albumu: 244509

Gra muzyczno-rytmiczna Rythm Code

Praca licencjacka napisana

pod kierunkiem **dr inż. Jerzego Skurczyńskiego**

Gdańsk 2019

Spis treści

1. Wstęp

1.1 Dlaczego gra rytmiczna? – Czyli krótka historia gier video.

Gry video to jedna z najmłodszych branż elektronicznej rozrywki. Pierwsze tego typu produkcje były głównie tworzone na uczelniach przez studentów głównie w celach dążenia do rozwoju osobistego w raczkującej jeszcze branży IT i daleko im było do produkcji komercyjnych jakie możemy zauważyć teraz na rynku.

Lata 70. były wyjątkowe pod tym względem, bo właśnie wtedy różne firmy zauważyły ogromny potencjał w tej nowej gałęzi nowoczesnych technologii i to właśnie w tym okresie wraz z premierą *Ponga* w 1972 roku zaczęło powstawać ich coraz więcej. Produkcje zaczęły przynosić pieniądze, więc rynek został zalany pierwszymi automatami i konsolami do gier, które tylko ułatwiły twórcom tworzenie nowych gier.

Jedną z gier, która wykorzystała ten *boom* była właśnie produkcja rytmiczna stworzona w 1978 roku przez Ralpha Baera i Howarda Morrisona, nazwana po prostu: *Simon*. Założenie było proste – gra pokazywała graczowi sekwencje prostych dźwięków, z których każdy był symbolizowany innym kolorowym przyciskiem. Grający natomiast miał za zadanie powtórzyć daną sekwencję, a gdy to mu się udało, dostawał on coraz trudniejsze kombinacje. Taka zabawa okazała się niezwykle popularna, ponieważ nie tylko była dobrą rozrywką, przeważnie dla dzieci, ale także ćwiczyła pamięć grających łącząc bodźce audio-wizualne, przez co stała się niezwykle popularną i pożyteczną zabawką lat 70. i 80.

To właśnie dzięki grze Simon w roku 1996 powstała gra rytmiczna, która na zawsze wpłynęła na przyszłość tego typu produkcji – *PaRappa the Rapper*. Kontynuowała ona pomysł Baera i Morrisona, jednocześnie udoskonalając go za pomocą ówczesnych technologii. W ten sposób zamiast nudnych, prostych dźwięków tym razem powtarzaliśmy sekwencje piosenki gdzie gracz musiał dopasować sekwencje naciskanych przycisków do rytmu oraz muzyki sprawiając, że przy udanej kombinacji postać na ekranie śpiewała.

To właśnie *Parappa the Rapper* stał się grą, która wyznaczyła podstawy współczesnych gier rytmicznych, a więc program, którego opis tworzenia będę przedstawiał w kolejnych rozdziałach, nawiązuje podstawami do tego produktu.

1.2 Inspiracje, które zaważyły na efekcie końcowym

W dzisiejszych czasach nie możemy narzekać na brak gier rytmicznych na rynku, a więc i tworów do inspirowania się jest wiele. Przy tworzeniu projektu najwięcej inspiracji czerpałem z dwóch tytułów, których stylistyka, jak i rozwiązania techniczne zostały przeze mnie wykorzystane.

Jeżeli chodzi o interfejs użytkownika czy styl rozgrywki, to zastosowane zostało tutaj rozwiązanie z serii gier *Yakuza*, gdzie gracz ma dostępne do „wystukiwania" 4 przyciski, każdy symbolizujący inną poziomą linię, po której poruszają się odpowiednie nuty, wskazujące, w którym momencie trzeba odegrać odpowiedni rytm.

Rysunek 1 - Przykładowy zrzut ekranu z gry Yakuza 6: Song of Life

Nuty natomiast są przechowywane w odpowiednim pliku, wskazującym, do której piosenki odnosi się dany ich zestaw oraz w której sekundzie utworu powinny się pojawić. Czas pojawienia się jest ustalany ręcznie, czyli tak, jak było to robione w grach *PaRappa the Rapper* czy też popularnym *Dance Dance Revolution*. W ten sposób możemy uzyskać wysoką precyzje rytmu.

1.3 Tematyka

Stworzona przeze mnie praca koncentruje się przede wszystkim na procesie tworzenia gry video, a więc od procesu planowania po projektowanie i wdrażanie funkcjonalności. Skupiam się tutaj głównie na problemach, jakie mogą wystąpić podczas programowania oraz nad sposobem implementowania wcześniej wymyślonych pomysłów.

Głównym zadaniem pracy jest przedstawienie czytelnikowi na przykładzie gry rytmicznej, jak wygląda tworzenie tego typu projektu. W rezultacie ma to stanowić solidne oparcie dla każdego, kto także chciałby tworzyć podobne oprogramowanie.

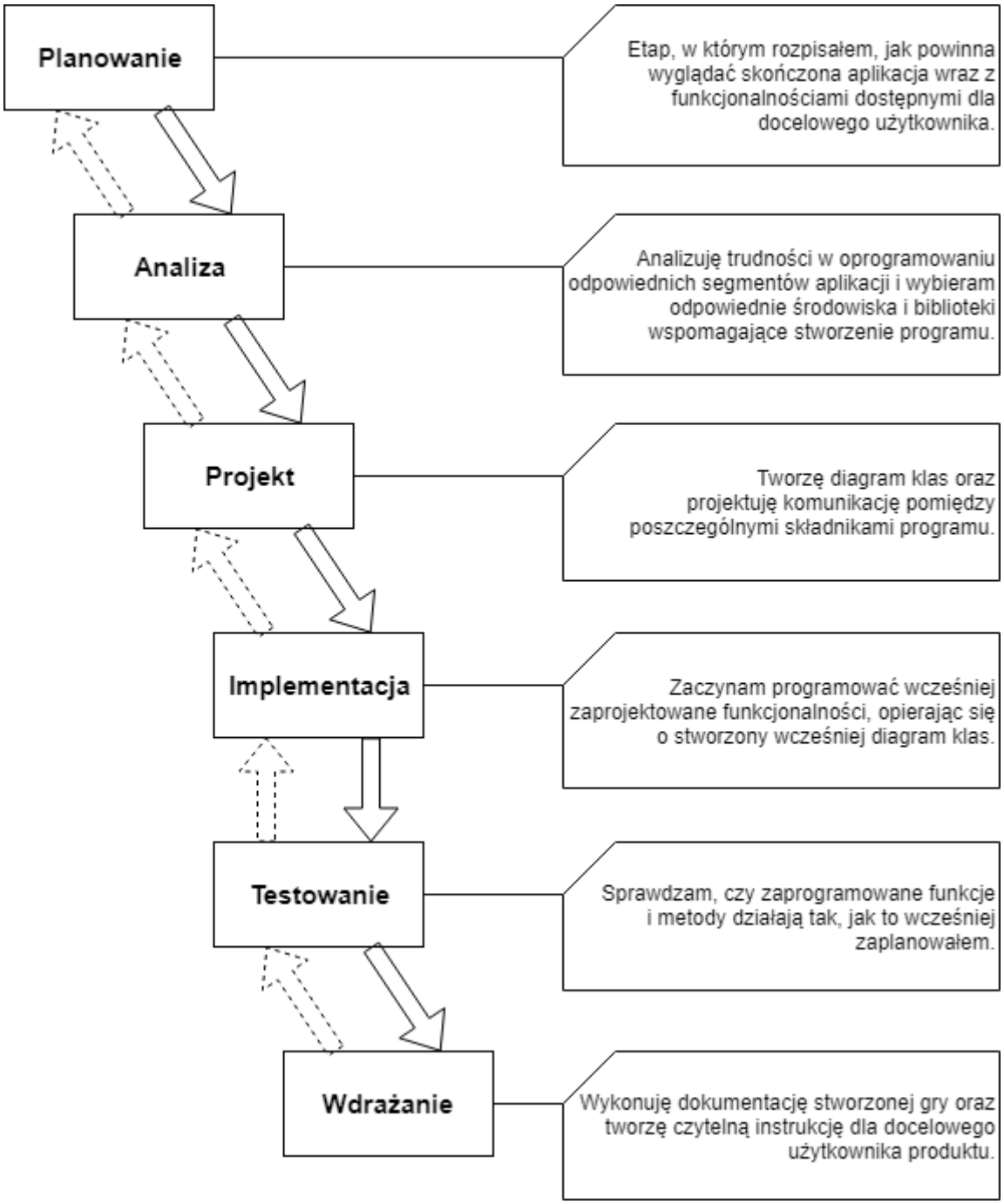
Skończona aplikacja umożliwia odegranie wybranych przez użytkownika poziomów muzycznych, a także zmianę prostych opcji takich jak rozdzielczość okna programu czy zmianę ustawień sterowania.

Cały projekt tworzony był w oparciu o model kaskadowym – pierwszym etapem było zaplanowanie, co powinno znaleźć się w skończonym programie (stworzenie diagramu przypadków użycia), następnie przeanalizowanie jakie zasoby będą potrzebne do ukończenia poszczególnych wymagań oraz jakie mogą wystąpić trudności implementacyjne. Po tym wszystkim zaprojektowany został cały system działania i modele używane w aplikacji klas (stworzenie diagramu klas).

Następnymi etapami tworzenia gry były: implementacja rozwiązań, czyli zaprogramowanie wszystkiego na podstawie wykonanych wcześniej etapów modelu kaskadowego oraz przetestowanie działania każdej z zaimplementowanych funkcji.

Ostatnim krok polegał na stworzeniu czytelnej dokumentacji programu oraz instrukcji dla docelowego użytkownika, które są częścią tej pracy i znajdują się w rozdziałach: „Wyjaśnienie zasad działania programu” oraz „Techniczne wyjaśnienie kwestii funkcjonalności gry”.

Poniżej zaprezentowałem diagram, który przedstawia każdy etap w odniesieniu do wykonanych przeze mnie czynności.



Rysunek 2 - diagram przedstawiający proces tworzenia gry

W dalszych działach znajduje się także opis ważnych dla funkcjonowania programu fragmentów kodu oraz ogólny opis poszczególnych segmentów wraz z wyjaśnieniem w jaki sposób wpływają one na efekt końcowy widoczny po uruchomieniu aplikacji.

Na końcu postanowiłem także napisać, które pomysły i funkcjonalności nie znalazły się w wersji końcowej i dlaczego, oraz opisuję plany rozwoju stworzonej produkcji.

2. Proces twórczy

2.1 Jak się zabrać za tworzenie gry?

Stworzenie gry video nie jest łatwym zadaniem. Aby wszystko udało się pomyślnie, przed rozpoczęciem programowania należy zastanowić się, co właściwie chcemy zrobić? Pytanie może być banalne, ponieważ wydawałoby się, że każdy wie jak wygląda gra video. Nie wszystko jednak jest takie proste, a więc powinniśmy się głębiej zastanowić co rozumiemy poprzez wyrażenie „gra rytmiczna" i czego potrzebujemy, by pomyślnie zrealizować nasz projekt.

Najpierw musimy postawić sobie założenia, które powiedzą nam, jak ten projekt ma wyglądać. W moim wypadku założenie wygląda następująco – gra rytmiczna będzie pozwalała użytkownikowi na wybór danej piosenki, a następnie rozegranie partii, która przyznaje odpowiednią liczbą punktów za jego zmagania. Podczas danej partii będziemy odtwarzać wybraną przez użytkownika muzykę oraz wyświetlać na ekranie odpowiednie grafiki, które ułatwią graczowi wybijanie rytmu oraz urozmaicą wygląd gry, czy to przez pokazywanie odpowiedniego różnorodnego tła dla każdego utworu, czy to wieńcząc starania naszego użytkownika poprzez wyświetlanie odpowiedniego napisu czy obrazu. Oczywiście użytkownik także musi mieć możliwość zmiany ustawień gry na takie, aby odpowiadały jego preferencjom.

Teraz, gdy wiemy, co dokładnie musimy oprogramować, powinniśmy zastanowić się jakie komponenty ułatwią nam to zadanie. Ważnym wyborem jest nie tylko język programowania ale także używane środowiska czy biblioteki, które umożliwią nam prostsze stworzenie tego co sobie założyliśmy na początku.

Jeżeli chodzi o język, to do tworzenia gier bardzo popularnym wyborem profesjonalistów jest C++ oraz C#. Tutaj natomiast wybrałem język Java. Jest on nie tylko niezwykle popularny przez co możemy znaleźć wiele odpowiednich książek czy też artykułów, które w dokładny sposób opisują pewne problemy, ale także samo pisanie od podstaw w Javie jest dużo łatwiejsze, a multiplatformowść tego języka jest tylko kolejną zaletą, gdy tworzymy grę video.

Teraz w kolejnym kroku postępowania, musimy wybrać środowisko programistyczne. Do wyboru dostępnych mamy wiele rozwiązań, poczynając od „podstawowego" pisania w języku po korzystanie ze zintegrowanych środowisk stworzonych z myślą o tworzeniu wyłącznie gier, takich jak *GameMaker*, *Unity* czy *Unreal Engine 4*.

W tym wypadku naszym celem było stworzenie gry od podstaw w języku *Java*, nie korzystając z żadnych uproszczeń. Oczywiste jest, że nawet nie wykorzystując dedykowanych produktów ułatwiających programowanie gier, potrzebujemy czegoś, co pomoże nam zarządzać naszym projektem. Tutaj został wybrany bardzo popularny *IDE – Eclipse*.

Po wyborze środowiska programistycznego musimy pomyśleć o innych sprawach. Gra musi posiadać grafikę. Java posiada wiele bibliotek stworzonych z myślą o programowaniu grafiki do gier, chociażby *Lightweight Java Game Library*. W naszym wypadku, jako że podstawowym założeniem była gra rytmiczna wykorzystująca prostą grafikę 2D, wystarczą nam wbudowane biblioteki *AWT* i *Swing*[1]. Można je w łatwy sposób wykorzystać do wyświetlania na ekranie prostych kształtów oraz *sprite'ów*.

Jako że tworzymy grę rytmiczną, to istotnym elementem jest obsługa muzyki. Tutaj już nie możemy użyć wbudowanych bibliotek, bo niestety nie zawierają one bardziej zaawansowanej obsługi plików czy też dostosowywania muzyki. Do rozwiązania tych problemów wybrałem odpowiednią bibliotekę *JLayer*, która oferuje nam więcej możliwości niż podstawowy *Swing*, a z tego powodu, że podlega licencji wolnego oprogramowania, jest także popularna i możemy znaleźć wiele źródeł, które ułatwią nam korzystanie z niej[2].

Mając takie podstawy za nami możemy w końcu zacząć tworzyć naszą grę w odpowiednich warunkach.

[1] Yang, H. (2018). *Java Swing Tutorials - Herong's Tutorial Examples* –
zasady operowania na bibliotece Swing. [5]
[2] JavaZoom. *JLayer API V1.0.1* - dokumentacja jest głównym źródłem informacji, na którym wzorowałem się operując na bibliotece *JLayer.* [4]

2.2 Problemy przewidziane i nieprzewidziane

Przed przystąpieniem do pracy ważnym elementem jest rozpisanie sobie problemów, które mogą wystąpić w czasie programowania. Pierwsze przeszkoda, którą musimy przemyśleć, znajduje się już w naszym założeniu, a mianowicie stworzenie możliwości dostosowywania ustawień gry.

Oczywistym jest, że informacji o rozdzielczości okna programu czy też zmianie sterowania nie możemy przechowywać w kodzie, ponieważ nie dałoby się ich zmieniać. Po każdym zamknięciu naszej gry wszystkie zapamiętane informacje byłby usuwane z pamięci RAM, a więc gra po ponownym uruchomieniu wracałaby do ustawień domyślnych, czego byśmy nie chcieli.

Dobrym pomysłem było zapisywanie tego typu danych na dysku twardym komputera, więc zdecydowałem się na przetrzymywanie wszystkich ustawień w odpowiednim pliku formatu *.txt* dzięki czemu nawet gdy wyłączymy grę, ta podczas kolejnej sesji będzie mogła wczytać sobie informacje o potrzebnych ustawieniach.

Drugi problem wynika z gatunku do jakiego należy tworzona produkcja. Jako że jest to gra rytmiczna, trzeba przemyśleć sposób przechowywania informacji w taki sposób, żeby gracz mógł łatwo wybrać z menu rozgrywki utwór, który chce odegrać.

Dlatego warto zdefiniować poziom rozgrywki jako obiekt z informacjami o tym jaką piosenkę, motyw graficzny czy zestaw nut zawiera. Później w łatwy sposób można taką listę przechowywać w pliku tekstowym, a nasz użytkownik dostanie możliwość wyboru interesującej go pozycji z tej listy.

Skoro jesteśmy już przy samej rozgrywce, to warto było także przemyśleć, jak przechowywać wyżej wymienione zestawy nut do każdego poziomu. Dobrze jest trzymać je także jako listę, tak jak informacje o poziomach. Lista ta mówiłaby, jaki typ nuty i w jakim czasie pokazać na ekranie. Jako że sama lista zawiera setki pozycji, to umieszczenie jej w kodzie jest bardzo złym pomysłem. Tutaj ponownie, tak jak ustawienia, możemy wszystko zapisać w odpowiednim pliku *.txt*.

Aby kod źródłowy programu był bardziej przejrzysty, warto też było wymyśleć pewną konwencję nazewniczą, czyli zasady, w ramach których będziemy nazywać nasze pliki tak aby zarówno sam program mógł szybko i wydajnie na nich operować, ale także aby cały projekt był łatwy do zrozumienia dla osoby postronnej, która nie ma z nim nic wspólnego.

W tym wypadku przyjąłem prostą zasadę – pliki, które zawiera jeden poziom, zawsze zaczynają się od tytułu piosenki, a potem zawierają w nazwie dodatkowe słowo, które definiuje to, czy zawartość danego pliku jest np. nutami, zestawem grafik albo muzyką.

Niestety zawsze przy tworzeniu programu pojawiają się pewne problemy, których nie przewidzieliśmy. Najczęściej wychodzą dopiero wtedy, gdy już jesteśmy na dalszym etapie prac.

W tym wypadku nie było inaczej. Pierwszy problem pojawił się już przy obsłudze plików dźwiękowych. Okazało się bowiem, że niestety wybrana biblioteka nie pozwala na zmianę głośności piosenki. W tym wypadku pierwszym pomysłem było zmienienie biblioteki do obsługi plików muzycznych. Jednak zamiast tego postanowiłem naprawić ten problem ręcznie i samemu dodać do bibliotek odpowiednie metody wykorzystując te, które już zostały zaimplementowane przez autora. Dzięki temu wystarczyło dodać kilka metod do wybranej biblioteki *.jar* i nasz program otrzymał możliwość zwiększania ilości decybeli w odtwarzanym pliku muzycznym.

Drugim nieprzewidzianym problemem była obsługa wątków. W tym wypadku problem wystąpił zarówno przy samej grze, jak i w samym menu głównym. W pierwszym wypadku rozwiązaniem okazało się napisanie odpowiedniej funkcji, która automatycznie zwalnia niepotrzebny wątek wcześniej używany do wyświetlania grafiki, dzięki czemu może dalej być użyty do kolejnych operacji.

W drugim przypadku problem dotyczył przycisku wyciszania muzyki w menu głównym. Przy naciśnięciu tego przycisku wątek odpowiedzialny za odtwarzanie muzyki zatrzymywał ją, tak aby później przy powtórnym wciśnięciu mogła być odtwarzana od tego samego momentu, w którym została zatrzymana. Przeszkoda wystąpiła, gdy użytkownik przechodził do kolejnej zakładki w menu, która zmieniała muzykę. Wątek muzyczny otrzymywał nowe zadanie, ale nie ukończywszy wcześniejszego nie mógł kontynuować pracy, co skutkowało błędem.

Rozwiązaniem była uważniejsza kontrola przy zwalnianiu wątku z powierzonego zadania przy próbie zmiany muzyki tak, by przed przyjęciem kolejnego polecenia wcześniejsze zostało pomyślnie wykonane. Wszystko to wystarczyło dodać do akcji wykonywanej przy każdorazowej zmianie w menu.

Bez względu na to, jak dobrze przemyślimy wszystko przed podjęciem pracy, musimy przygotować się na to, że zawsze możemy napotkać na drodze nieprzewidziany problem. W takim wypadku dobrym rozwiązaniem zawsze jest sięgnięcie do literatury czy tej książkowej, czy też internetowej.

2.3 Schemat kolejności prac

We wcześniejszych rozdziałach przemyśleliśmy założenia projektu, wybraliśmy środowisko, biblioteki, a także spróbowaliśmy przewidzieć, jakie problemy mogą wystąpić w trakcie tworzenia gry i jak je rozwiązać. Teraz pozostało nam ustalenie od czego zacząć programowanie. Pierwszy pomysł jest taki, żeby zacząć od razu od samej gry, a potem wszystko dopasować do niej. Ja natomiast postanowiłem rozpocząć proces twórczy od napisania elementów w kolejności takiej, w jakiej zobaczy je użytkownik.

Pierwszym zadaniem było więc napisanie menu gry. Do utworzonego panelu zawierającego wszystkie potrzebne przyciski – graj, opcje i wyjście, dodałem elementy graficzne i całość rozmieściłem w estetyczny sposób.

Każdy przycisk otrzymał funkcję, która decydowała, jaka czynność zostanie wykonana po jego naciśnięciu, tak więc opcja graj przenosi nas do nowego miejsca z innym tłem oraz przyciskiem powrotu do menu głównego – tutaj pozostawiłem miejsce na moduł wyboru piosenki, który został zaimplementowany później. Podobnie powstał przycisk o nazwie opcje, natomiast przycisk wyjście otrzymał oddzielną metodę, która odpowiadała za domknięcie wszystkich okien i wątków i wyłączenie programu.

Na taki gotowy już szkielet postanowiłem także nanieść możliwość wyciszenia muzyki w menu symbolizowane przyciskiem głośniczka w prawym górnym rogu ekranu.

Po tym wszystkim kolejnym krokiem było ustawienie piosenki odtwarzanej przy uruchomieniu gry i w ten sposób otrzymaliśmy trochę puste, ale już działające menu.

Kolejnym krokiem jest stworzenie modułu wyboru piosenki przez gracza, który po kliknięciu w wybraną pozycje przeniesie go do trybu rozgrywki. Tak jak pisałem w założeniach, najlepiej będzie stworzyć listę poziomów, która będzie zawierać informacje o muzyce, grafikach i innych istotnych elementach. W ten sposób powstała oddzielna klasa przechowująca obiekt ze wszystkimi tymi informacjami. Dzięki temu można szybko stworzyć listę tych obiektów. Teraz pozostało nam tylko dodanie dwóch przycisków, które pozwolą przewijać taką listę do kolejnej lub poprzedniej pozycji oraz stworzenie metody rysującej odpowiedni obrazek, informujący o tym, który poziom został przez nas wybrany. Ostatnim etapem jest teraz tylko dodanie akcji po naciśnięciu wybranej pozycji. Akcja ta ustawia w programie ekran na tryb rozgrywki, a poziom zostaje ustalony z wcześniej wybranej pozycji na liście.

Gdy to wszystko zostało wykonane, trzeba zaimplementować okno samej gry, czyli najtrudniejszy element tego projektu. Pierwszym zadaniem jest stworzenie odpowiedniego interfejsu, sprite'ów i grafik. Do tego wykorzystany został odpowiedni darmowy program graficzny – *Gimp*.

Kiedy już to zostało zaimplementowane, to kolejnym etapem prac było stworzenie odpowiedniej klasy, która będzie odpowiadała za fizykę stworzonych nut, tak aby poruszały się one zawsze według danej zasady kontrolowanej przez grę. W tym wypadku tak, jak to zostało napisane w dziale „Inspiracje, które zaważyły na efekcie końcowym", cały interfejs graficzny, czy fizyka spadającej nut jest wzorowana na tej zaprezentowanej w grze *Yakuza 6*.

Tutaj należy wymyślić parę zasad dotyczących naszych spadających nut. W tym wypadku odpowiedni wątek będzie miał za zadanie stworzenie potrzebnego nam obiektu, którego pozycja zmieni się na podstawie parametru określającego jego prędkość. Obiekt ten będzie tworzony po prawej strony i zmierzał w lewą, a gdy zniknie nam z pola widzenia, zostanie usunięty.

Każda nuta przy tworzeniu będzie miała także z góry określony typ, który będzie mówił grze, na której z czterech linii powinien ją umieścić. Warto także na koniec dodać funkcję, która będzie zwiększać punktację w zależności od tego, czy udało nam się odtworzyć odpowiedni rytm.

W tym momencie program potrzebuje jakiegoś mechanizmu, który będzie przechwytywał informację, czy użytkownik nacisnął prawidłowy przycisk w odpowiednim czasie. W ten sposób utworzona klasa będzie przechowywać informacje o klawiszach, jakie mogą być naciśnięte i co ma być wtedy wykonane. Jako że stworzyłem cztery linie, po których mogą poruszać się nuty to liczba klawiszy będzie wynosić tyle samo. Dodatkowo trzeba pamiętać, żeby określić dla każdego z nich akcje po wciśnięciu i po puszczeniu. Tutaj odpowiednie akcje utworzyłem w stworzonej wcześniej klasie samej gry muzycznej i odwołałem się do nich z klasy informującej, który klawisz został naciśnięty. Akcje te, po wciśnięciu pokazują na ekranie odpowiednią grafikę zaznaczającą, jedną z czterech dostępnych dróg spadania nut.

W ten sposób można najprościej opisać schemat, według, którego cały projekt został stworzony. Na tym etapie stworzona gra potrafi już liczyć i pokazywać liczbę zdobytych punktów oraz przypisuje muzykę i nuty do odpowiedniego poziomu. Dodatkowo poziomy możemy w prosty sposób dodawać poprzez edycję pliku z ustawieniami oraz dodając odpowiednie pliki do folderu z muzyką.

Cała metodyka polegała na tworzeniu komponentów w kolejności takiej, w której będzie widział je użytkownik, a więc pierwsze co powstało to ekran z menu i ustawieniami, a następnie okno wyboru poziomów i sama gra.

3. Wyjaśnienie zasad działania programu

3.1 Ogólna reguła funkcjonowania gry

Aby uporządkować stworzoną grę oraz spowodować, by miała ona bardziej przejrzysty kod, zastosowałem tutaj taki podział na klasy, aby każda z nich odpowiadała za inny segment programu. Sposób ten jest bardzo podobny do prezentowanego w książce *„Killer Game Programming in Java”* autorstwa *Andrew Davisona*[3]. To oznacza, że sama gra została podzielona na różne segmenty – grafikę, muzykę, fizykę i okna główne. Dzięki temu możemy utworzyć odpowiednią komunikację pomiędzy każdymi składnikami nie obciążając przy tym komputera, aby nie wykonywał niepotrzebnych operacji w tym samym czasie, a zamiast tego korzystał z danej klasy tylko wtedy, kiedy tego potrzebuje.

Najbardziej obszerna objętościowo klasa – *Rythm Code*, odpowiada tylko i wyłącznie za obsługę okienek w samym programie. To tutaj użytkownik może wejść w ustawienia, by zmienić rozdzielczość czy sterowanie w grze. Klasa ta również odpowiada za ekran wyboru piosenek, wczytując ich listę oraz odpowiednie zasoby z folderu z muzyką. Oprócz tego odpowiada za zapisywanie odpowiednich ustawień do pliku i wczytywanie ich przed samym uruchomieniem programu.

Inną istotną klasą w owym programie jest *Game*. Odpowiada ona, jak sama nazwa wskazuje, za obsługę samej gry, a więc obsługuje grafikę okna rozgrywki, uruchamia silnik fizyczny oraz podlicza punkty za poprawne odtworzenie rytmu.

Sama klasa *Note* służy do ustalenia zasad opadania nut i określania ich pozycji. Sprawdza ona również, jak daleko nuta została „wystukana” przez użytkownika od prawidłowej pozycji i na podstawie tego informuje, ile punktów należy dodać graczowi. Kolejnym ważnym zadaniem jest także zwalnianie wątków, kiedy nie musimy pokazywać danych nut na ekranie.

[3] Davison, A. (2005). *Killer Game Programming in Java* - książka opisująca dokładnie jak projektować i tworzyć gry video przy pomocy języka Java. [2]

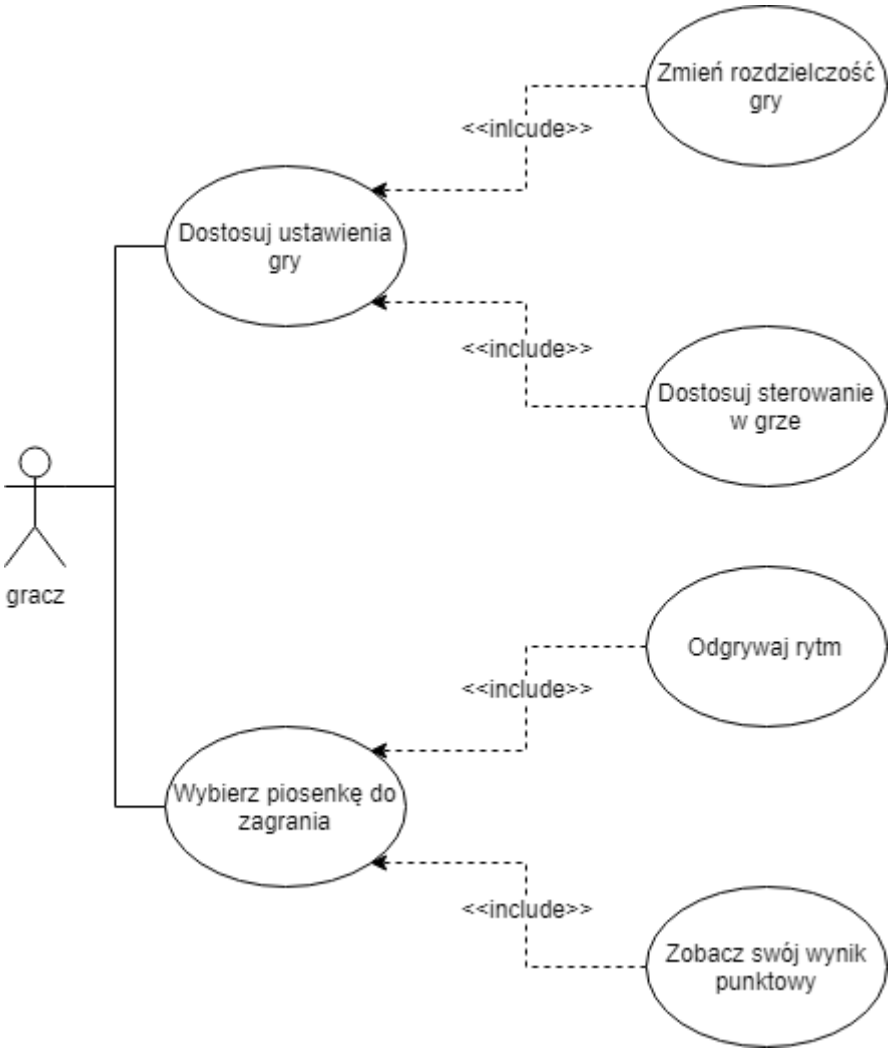
Do odczytywania kolejności, typu oraz czasu pojawienia się odpowiednich nut pomaga natomiast klasa *Beat*. Jest ona bardzo krótka i właściwie służy jako model przechowujący ważne informacje samej grze o tym kiedy powinna narysować dany sprite. Dzięki temu możemy szybko stworzyć listę takich obiektów, a informacje do niej wczytać z plików.

Obsługa plików muzycznych następuje w specjalnie dostosowanej do tego klasie *Music*. Tutaj, dzięki wykorzystywanej bibliotece *JLayer*, stworzony program może uruchamiać wybrane przez nas dźwięki, ustawiać ich głośność, sprawdzić czas odtwarzania, a na koniec, gdy jest to potrzebne – wyłączyć muzykę już nas nie interesującą.

Ostatnią klasą wartą uwagi jest *KeyListener*. Jedynym jej zadaniem jest sprawdzanie, czy został naciśnięty odpowiedni przycisk, jeśli tak, to przesłanie tej informacji do klas, które mają przypisane odpowiednie akcje do wciśnięcia danego klawisza.

3.2 Obsługa gry

Ogólny diagram przypadków użycia dla użytkownika programu wygląda następująco:

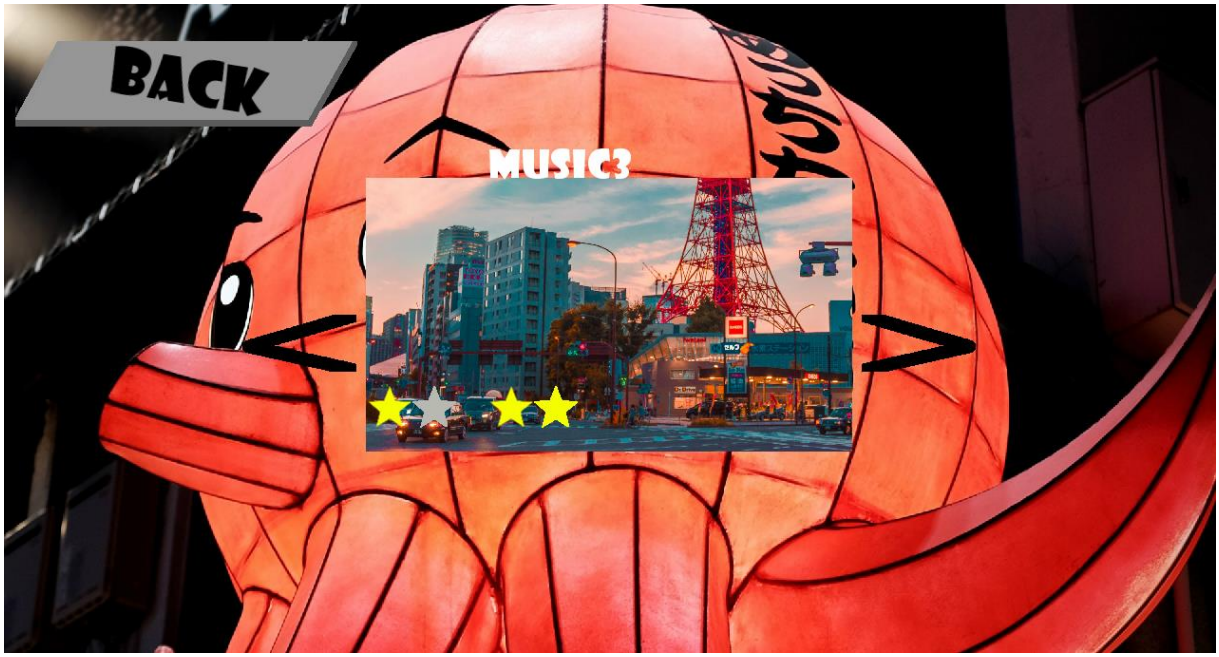


Rysunek 3 - diagram przypadków użycia dla gracza

Po uruchomieniu gry użytkownikowi pokaże się menu główne[4] wyglądające następująco:

Rysunek 4 - menu główne gry

Tutaj gracz będzie mógł wybrać pomiędzy trzema opcjami – zagraniem w grę, dostosowaniem ustawień i wyjściem z programu. Po naciśnięciu przycisku *PLAY* użytkownik zostanie przeniesiony do ekranu wyboru piosenki.



Rysunek 5 - menu wyboru utworu muzycznego

[4] Design Doc. (2017-2019). *Good Design, Bad Design - The Best & Worst of Graphic Design in Games* - wykłady na temat projektowania gier video, dostępne na platformie You Tube (wszystkie menu dostępne w grze oraz sam interfejs rozgrywki zostały stworzony na ich podstawie). [4]

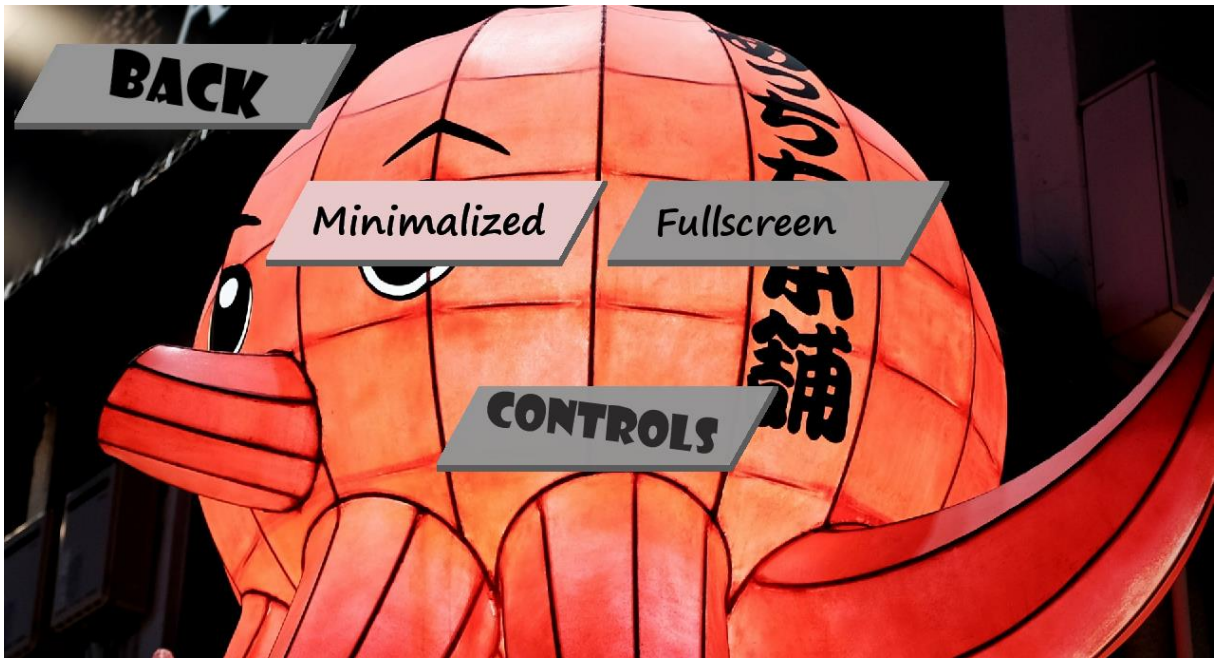
Strzałki wskazujące w lewo i prawo służą do wyboru utworu. Jego nazwa jest wyświetlana nad zdjęciem poglądowym, a gwiazdki symbolizują poziom trudności.

Dodatkowo dzięki przyciskowi *BACK* zawsze można cofnąć się do poprzedniego menu. Kiedy użytkownik wybierze interesujący go utwór, to zostanie przeniesiony do właściwej rozgrywki.

Rysunek 6 - przykładowy zrzut ekranu wykonany podczas właściwego grania

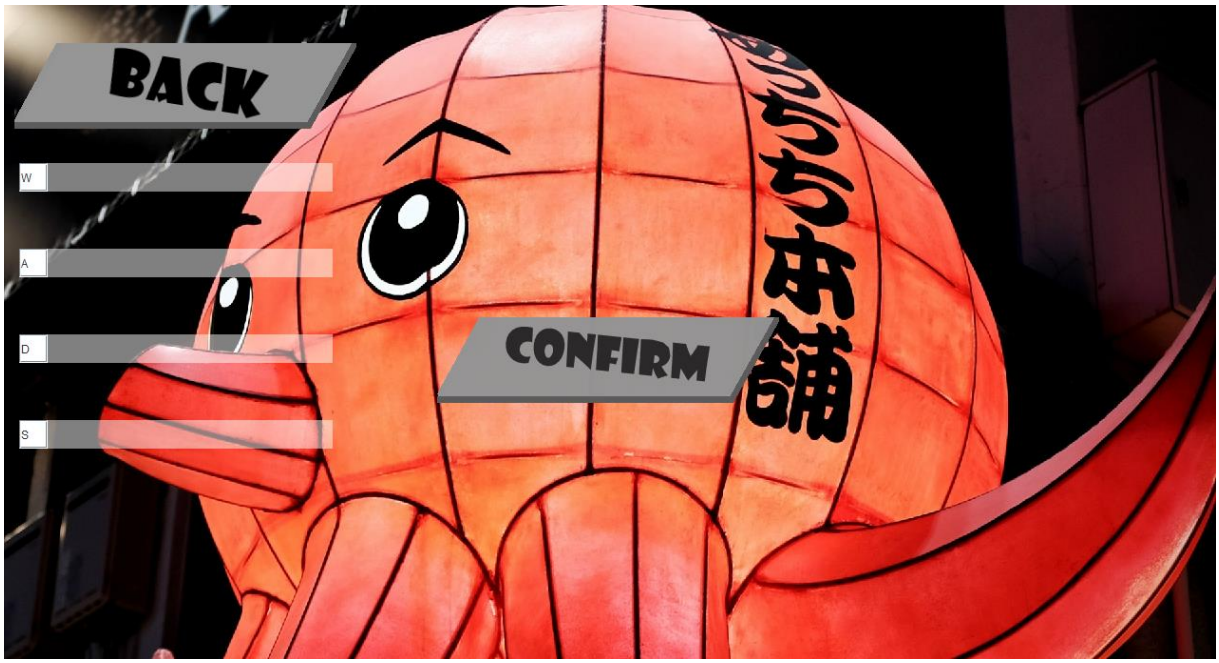
Na tym ekranie gracz będzie musiał naciskać odpowiednie klawisze w rytm odtwarzanej piosenki. Po lewej stronie zaznaczone są odpowiednie wiersze symbolizowane przez odpowiednie klawisze, które mogą zostać dostosowane w menu ustawień. Oprócz tego w dolnej części okna wyświetlany jest tytuł utworu, liczba aktualnie zdobytych punktów oraz poziom trudności.

Będąc w menu głównym, użytkownik może także kliknąć w przycisk *OPTIONS*, który przeniesie go do konfiguracji gry.



Rysunek 7 - menu ustawień gry

Ten panel umożliwi mu zmianę ustawień aplikacji. Oprócz przełączania pomiędzy rozdzielczościami (pełny ekran lub okno), gracz otrzymuje tutaj także opcję zmiany sterowania w grze. Po naciśnięciu przycisku *CONTROLS* użytkownik zostanie przeniesiony do ekranu zmiany ustawień sterowania.



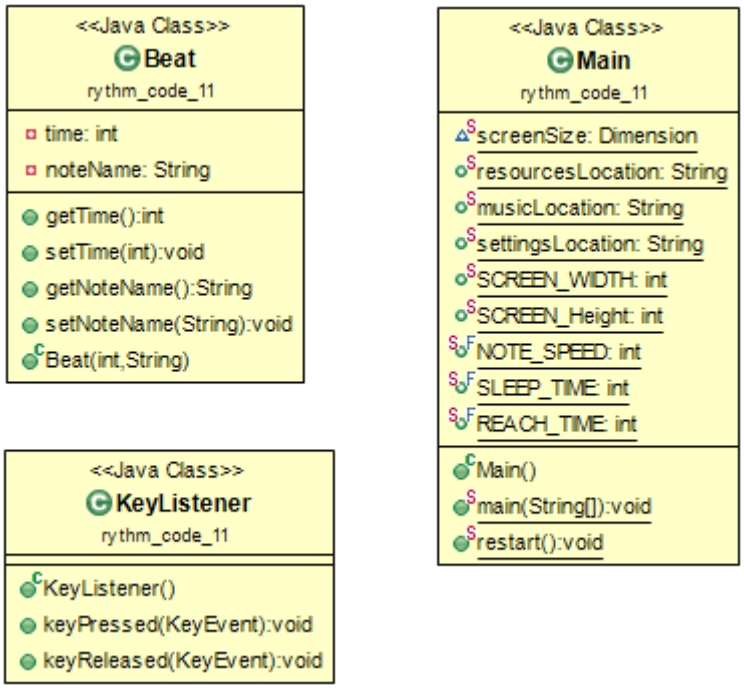
Rysunek 8 - menu konfiguracji sterowania gry

Po lewej stronie pojawia się nowa możliwość – zmiany wszystkich czterech używanych w grze klawiszy. Kiedy gracz zakończy proces zmiany ustawień sterowania, może on zaakceptować zmiany naciskając przycisk *CONFIRM*.

4. Techniczne wyjaśnienie kwestii funkcjonalności gry

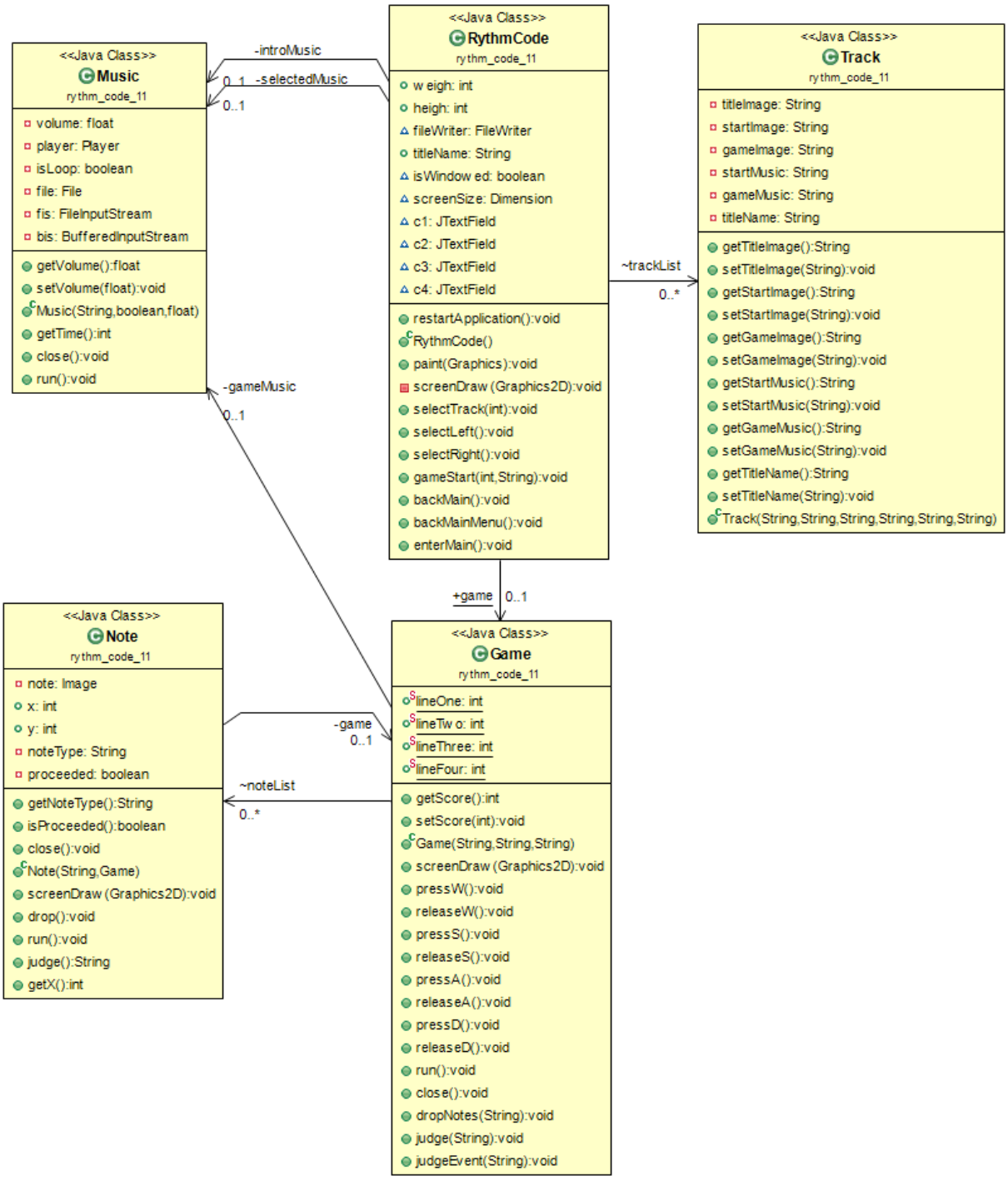
4.1 Diagramy klas projektu

Diagram klas użytkowych:



Rysunek 9 - Diagram przedstawiający klasy użytkowe, nie mające relacji z innymi klasami

Diagram klas przedstawiający relacje pomiędzy nimi:



Rysunek 10 - Diagram klas przedstawiający istotne klasy i ich relacje

4.2 Obsługa menu oraz interfejsu gracza

RythmCode

Pierwszą klasą, która jest uruchamiana zawsze przy starcie gry, jest *RythmCode*. Odpowiada ona za wyświetlanie graczowi menu głównego wraz z możliwościami zmiany ustawień oraz przejścia do samej rozgrywki. Wykorzystuje ona zarówno biblioteki AWT, jak i SWING[5] do wczytania, a następnie wyświetlenia odpowiednich obrazów, np: tła czy przycisków.

Klasa ta posiada dwie ważne zmienne mówiące programowi, jakie okno powinno być wyświetlane - są to ***isMainScreen*** i ***isGameScreen***.

```
1.  private boolean isMainScreen = false;
2.  private boolean isGameScreen = false;
```

IsMainScreen jest ustawiane na wartość ***true*** po kliknięciu opcji „graj”. Program wtedy otrzymuje informację że może uruchomić rysowanie grafik 2D, które będą nam pokazywać dostępne poziomy do wyboru.

IsGameScreen jest natomiast ustawiane na **true** wtedy, kiedy gracz wybierze interesujący go poziom. Wtedy uruchamiana jest klasa *Game*, a wraz z nią odpowiednie okno i rysowanie grafik 2D przypisanych do wybranego poziomu.

Przechowujemy tutaj również zmienne dotyczące muzyki, która ma być uruchomiona przy otwarciu menu głównego.

```
1.  private Music introMusic = new Music("intro.mp3", true, -20);
```

IntroMusic jest zmienną typu *Music*, czyli stworzonej przeze mnie klasy przechowywującej informacje o konkretnym utworze. Na wejściu przyjmuje parametry takie jak nazwa, czy ma być zapętlony oraz o ile należy zwiększyć poziom głośności oryginalnego utworu. W tym wypadku program będzie szukał w folderze *music* pliku muzycznego o nazwie *intro.mp3* oraz będzie odtwarzał daną muzykę w trybie zapętlonym i zmniejszy jej poziom dźwięku o 20 decybeli. Bodnar, J. (2007 - 2019). *Java 2D games tutorial.* www.zetcode.com

[5] Bodnar, J. (2007-2019). *Java 2D games tutorial*. www.zetcode.com - Obsługa obu bibliotek AWT oraz Swing została wykonana na podstawie omówionych przykładów gier, dostępnych na stronie autora. [1]

Klasa ***RythmCode*** od razu po uruchomieniu wczytuje także listę poziomów do wyboru z odpowiedniego pliku i przyporządkowuje z niej informacje potrzebne do uruchomienia odpowiedniej rozgrywki, takie jak ścieżka dźwiękowa, grafiki czy nuty.

Wykorzystujemy tutaj zmienną *csvFile*, która mówi aplikacji o lokalizacji pliku z zawierającego wszystkie potrzebne informacje do wczytania poziomu wybranego przez gracza. W ten sposób do listy typu zdefiniowanego przez stworzoną klasę ***Track*** możemy dodać kolejne poziomy, które program będzie wyświetlał użytkownikowi w trybie wyboru piosenki.

W tej klasie znajduje się także metoda odpowiedzialna za narysowanie interfejsu menu - jest to ***screenDraw***. Funkcja ta nie tylko odpowiada za obsługę grafik w menu, ale także wykonuje takie czynności jak wyświetlanie dostępnych poziomów, a także przekierowuje do innej metody – obsługującej już samą rozgrywkę.

```
1.  private void screenDraw(Graphics2D g) {
2.      g.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING, RenderingHints.VALUE_TE
    XT_ANTIALIAS_ON);
3.      g.drawImage(introBackground, 0, 0, null);
4.      if(isMainScreen)
5.      {
6.          g.drawImage(selectedImage, (int) (Main.SCREEN_WIDTH*0.5)-
    (int) (Main.SCREEN_WIDTH*0.2),  (int) (Main.SCREEN_Height*0.5)-
    (int) (Main.SCREEN_Height*0.2), null);
7.          g.drawImage(titleImage,(int) (Main.SCREEN_WIDTH*0.5)-
    (int) (Main.SCREEN_WIDTH*0.2),  (int) (Main.SCREEN_Height*0.5)-
    (int) (Main.SCREEN_Height*0.2), null);
8.      }
9.      if(isGameScreen){
10.         game.screenDraw(g);
11.     }
12.     paintComponents(g);
13.     try {
14.         Thread.sleep(5);
15.     } catch(Exception e) {
16.         e.printStackTrace();
17.     }
18.     this.repaint();
19. }
```

4.3 Kontrola zasobów graficznych i muzyki

Music

Jak sama nazwa wskazuje, jest to klasa, której zadaniem jest obsługa plików muzycznych (w przypadku mojej gry są to głównie pliki z rozszerzeniem .mp3).

Posiada ona zmienną globalną o nazwie ***volume***, która oznacza poziom głośności danej ścieżki dźwiękowej. Dodatkowo ***isLoop*** określa nam, czy program powinien odtwarzać dany utwór w zapętleniu.

```
1.  private float volume = 0;
2.  private boolean isLoop;
```

Najważniejsze w tej klasie natomiast są cztery zmienne: ***player***, ***file***, ***fileInput*** i ***bufferedInput***.

```
1.  private Player player;
2.  private File file;
3.  private FileInputStream fis;
4.  private BufferedInputStream bis;
```

File ma za zadanie informować program o ścieżce interesującego nas pliku z odpowiednią muzyką, dzięki czemu może on bez problemu odnaleźć potrzebną mu w danym momencie ścieżkę dźwiękową.

FileInput odpowiada za przechowywanie ciągu bajtów pliku określonego przez ścieżkę w zmiennej ***file***.

BufferedInput jest potrzebna w programie, by dodać możliwość buforowania odczytanego wcześniej ciągu bajtów tak, aby program mógł po zatrzymaniu się w danym momencie, wrócić do niego, a nie zaczynać odczyt od początku.

Tak odpowiednio przygotowane pliki są już gotowe dla zmiennej ***player***, aby je odtworzyć. Jest to zmienna klasy o tej samej nazwie zaimportowanej z pakietu *JLayer*. W uproszczeniu, pakiet ten jest odpowiedzialny za dekodowanie ciągów bajtów w taki sposób, aby komputer potrafił odtworzyć je jako szeroko rozumianą muzykę. Dodatek ten zawiera wiele pomocnych metod, które potrafią zmieniać głośność danej ścieżki dźwiękowej, zatrzymać odtwarzanie i zwrócić informacje o tym, w której sekundzie muzyka została przerwana.

Cały proces odczytywania muzyki, wykorzystujący powyższe zmienne jest wykonywany zawsze przy wywoływaniu klasy ***Music*** i wygląda następująco:

```
1.  public Music(String name, boolean isLoop, float volume)
2.  {
3.      try {
4.          this.volume = volume;
5.          this.isLoop = isLoop;
6.          file = new File(Main.class.getResource("../music/" + name).toURI());
7.          fis = new FileInputStream(file);
8.          bis = new BufferedInputStream(fis);
9.          player = new Player(bis);
10.     }catch (Exception e) {
11.         System.out.println(e.getMessage());
12.     }
13. }
```

Klasa ***Music*** zawiera także metody, które pozwalają na podstawowe operacje na wczytanych, plikach takie jak uruchomienie piosenki, wyłączenie odtwarzania piosenki i odczyt ile czasu upłynęło od rozpoczęcia operacji odtwarzania. Odpowiadają za to kolejno metody: ***run***, ***close*** i ***getTime***.

Warto wspomnieć, że funkcja ***close***, oprócz wyłączenie utworu, zwalnia także wątek, który jest używany do przetrzymywania informacji o muzyce.

Track

Jest klasą modelową, która służy temu, aby pomóc grze w przechowywaniu danych o każdym poziomie. To tutaj przetrzymujemy nazwy plików potrzebnych do uruchomienia każdego z poziomów. Dzięki temu możemy utworzyć listę takich modeli, która po wybraniu odpowiedniej pozycji zwróci programowi, jakie pliki powinien wczytać, aby odtworzyć dany poziom gry.

Wywołanie tej klasy polega na zapisaniu informacji o każdym poziomie z argumentów przekazywanych w metodzie wywoławczej.

```
1.  public Track(String titleImage, String startImage,
2.           String gameImage, String startMusic,
3.           String gameMusic, String titleName) {
4.      super();
5.      this.titleImage = titleImage;
6.      this.startImage = startImage;
7.      this.gameImage = gameImage;
8.      this.startMusic = startMusic;
9.      this.gameMusic = gameMusic;
10.     this.titleName = titleName;
11. }
```

Program wczytuje tutaj takie rzeczy, jak muzyka poziomu, kilkusekundowa próbka muzyki (potrzebna do odtworzenia przy ekranie wyboru piosenki), tytuł utworu, logo utworu, obrazek widoczny na menu wyboru piosenek oraz tło poziomu.

4.4 Fizyka i sposób działania gry

Note

W tej klasie program wykonuje wszystkie czynności związane z zachowaniem pojawiających się na ekranie nut. Do rysowania nut w odpowiednim miejscu ekranu gry wykorzystywana jest funkcja ***screenDraw()***.

Oprócz tego do zmieniania pozycji nut, tak aby poruszały się one w prawą stronę, a następnie zostawały usuwane w przypadku przekroczenia okna gry odpowiada metoda ***drop()***.

```
1.  public void drop() {
2.      x -= Main.NOTE_SPEED;
3.      if(x < 0) {
4.          close();
5.      }
6.  }
```

Funkcja ***close()***, używana w powyższym kodzie, informuje tylko program o tym, że dana nuta znalazła się poza ekranem. Robi to poprzez zmianę wartości zmiennej ***proceeded*** na wartość *false*, dzięki czemu stworzony wcześniej wątek otrzymuje informację, że może zakończyć swoją pracę.

```
1.  @Override
2.  public void run() {
3.      try {
4.          while(true) {
5.              drop();
6.              if(proceeded) {
7.                  Thread.sleep(Main.SLEEP_TIME);
8.
9.              }
10.             else {
11.                 interrupt();
12.                 break;
13.             }
14.         }
15.     }catch(Exception e){
16.         System.err.println(e.getMessage());
17.     }
18. }
```

Działanie tego wątku obrazuje powyższy fragment kodu – komputer przesuwa nuty dopóki zmienna **proceeded** nie posiada wartości *false*.

Model zawiera też funkcje ***judge()*** podliczającą punkty w zależności od momentu, w którym dana nuta została usunięta z ekranu. Działa ona w prosty sposób – sprawdza

współrzędną *x* i w zależności od tego czy jest ona większa, czy mniejsza od tej z góry ustalonej, przyznaje odpowiednią ocenę.

```
1.  public String judge() {
2.      if(x<35) {
3.          close();
4.          return "Late";
5.      }
6.      else if(x<50) {
7.          game.setScore(game.getScore()+1000);
8.          close();
9.          return "Perfect";
10.     }
11.     else if(x<60) {
12.         game.setScore(game.getScore()+500);
13.         close();
14.         return "Great";
15.     }
16.     else if(x<80) {
17.         game.setScore(game.getScore()+100);
18.         close();
19.         return "Good";
20.     }
21.     else if(x<2) {
22.         close();
23.         return "Miss";
24.     }
25.     return "null";
26. }
```

Zwrócona w ten sposób wartość informuje również samą grę o tym jaki komentarz wyświetlić na ekranie.

Game

Jest klasą obsługującą tylko i wyłącznie okno samej rozgrywki. To tutaj program sprawdza, jaka piosenka powinna być teraz obsłużona na podstawie otrzymanych informacji od klasy ***RythmCode,*** oraz dzięki tym danym pobiera odpowiednie pliki graficzne i muzyczne z zasobów.

Funkcja ***screenDraw()*** oprócz tworzenia całego interfejsu graficznego ma także za zadanie rysowanie cały czas nut w prawidłowej pozycji.

```
1.  public void screenDraw(Graphics2D g) {
2.      ...
3.      for(int i=0; i<noteList.size();i++) {
4.          Note note = noteList.get(i);
5.      ...
6.          if(!note.isProceeded()) {
7.              noteList.remove(i);
8.              i--;
9.          }else {
10.             note.screenDraw(g);
11.         }
12.     }
13.     ...
14. }
```

Sprawdzamy w niej w pętli czy odpowiednia nutka zniknęła poza ekranem i jeżeli tak było – usuwamy ją z listy. Natomiast dopóki dana lista zawiera jakieś pozycje, to rysujemy je według schematu zawartego w klasie ***Note***.

Oprócz tego w klasie ***Game*** program posiada odpowiednie akcje mówiące co należy zrobić po naciśnięciu danego przycisku na klawiaturze. Każdy przycisk posiada dwa takie schematy – po wciśnięciu i po puszczeniu.

```
1.  public void pressW() {
2.      judge("1");
3.      notesRoute1 = new ImageIcon(
4.      Main.class.getResource("../resources/notesRoutePressed.png")
5.      ).getImage();
6.      gameCheckHit1 = new ImageIcon(
7.      Main.class.getResource("../resources/gameChceckHit.png")
8.      ).getImage();
9.      new Music("noteBass.mp3", false, -5).start();
10. }
11. public void releaseW() {
12.     notesRoute1 = new ImageIcon(
13.     Main.class.getResource("../resources/notesRoute.png")
14.     ).getImage();
15.     gameCheckHit1 = null;
16. }
```

Powyższy kod pokazuje przykładowe czynności wykonywane dla jednego z klawiszy sterowania. Pierwsza metoda po naciśnięciu przycisku sprawdza pierwszą linię w poszukiwaniu, w jakiej odległości od punktu docelowego użytkownik „wystukał” nutę, aby przyznać odpowiednią liczbę punktów. Później pokazuje na planszy gry

odpowiednią grafikę informującą o tym, która linia została wybrana przez użytkownika oraz odtwarza symboliczny dźwięk.

Druga akcja po puszczeniu klawisza przywraca poprzedni stan planszy poprzez usunięcie z ekranu pokazywanej wcześniej grafiki.

Wykorzystywana w poprzednich metodach funkcja ***judge()*** pobiera z klasy **Note** informacje o tym jak daleko od prawidłowej pozycji została usunięta dana nuta.

```
1.  public void judge(String input) {
2.      for(int i=0; i<noteList.size(); i++) {
3.          Note note = noteList.get(i);
4.          if(input.equals(note.getNoteType())) {
5.              judgeEvent(note.judge());
6.              break;
7.          }
8.      }
9.  }
```

Wykonywana w tej metodzie procedura ***judgeEvent()*** odpowiada tylko za rysowanie odpowiedniej grafiki w zależności od wystawionej oceny.

Najważniejsza jednak w tej klasie jest metoda ***dropNotes()***, ponieważ to ona wczytuje z odpowiedniego pliku informacje o tym, w której sekundzie piosenki i na jakiej linii planszy utworzyć nową nutę.

```
1.  public void dropNotes(String titleName) {
2.      ...
3.      Beat[] beats = new Beat[];
4.      int i=0;
5.      gameMusic.start();
6.      while(i < beats.length && !isInterrupted()) {
7.          if(beats[i].getTime() <= gameMusic.getTime()) {
8.              Note note = new Note(beats[i].getNoteName(), this);
9.              note.start();
10.             noteList.add(note);
11.             i++;
12.         }
13.     }
14. }
```

Bardzo ważna jest zawarta w niej pętla, która uruchamia ruch danej nutki oraz dodaje ją do listy pozostałych nut, dzięki której funkcja rysująca otrzymuje dane o pozostałych obiektach do narysowanie na planszy gry.

Wątek kontrolujący okno rozgrywki ma dwie akcje: ***run()***, informująca o czynnościach wykonywanych podczas jego działania, oraz ***close()***, który odpowiada za prawidłowe zamknięcie pozostałych wątków wykorzystywanych w grze.

```
1.  @Override
2.  public void run() {
3.      dropNotes(this.titleName);
4.  }
5.  public void close() {
6.      gameMusic.close();
7.      this.interrupt();
8.  }
```

Warto dodać, że przy kończeniu rozgrywki należy pamiętać o wyłączeniu wątku odtwarzającego muzykę, ponieważ w innym wypadku nawet gdy dana ścieżka dźwiękowa przestanie być odtwarzana, to program dalej będzie trzymał ją w pamięci podręcznej co przy próbie wczytania nowego utworu wywoła błąd. Tutaj jest to wykonywane w szóstej linijce powyższego kodu, metodą **close()** dotyczącą klasy ***Music***.

4.5 Klasy funkcjonalne

Beat

Aby zapewnić szybkie wczytywanie informacji o czasie w jakim, stworzony program powinien utworzyć w oknie rozgrywki odpowiednią nutę , należy stworzyć odpowiedni model.

```
1.  public class Beat {
2.      private int time;
3.      private String noteName;
4.
5.      public int getTime() {
6.          return time;
7.      }
8.      public void setTime(int time) {
9.          this.time = time;
10.     }
11.     public String getNoteName() {
12.         return noteName;
13.     }
14.     public void setNoteName(String noteName) {
15.         this.noteName = noteName;
16.     }
17.
18.     public Beat(int time, String noteName) {
19.         super();
20.         this.time = time;
21.         this.noteName = noteName;
22.     }
23. }
```

W ten sposób możemy zapisać nazwę nuty i czas, który będzie mówił kiedy powinna ona się pojawić podczas rozgrywki. Nazwa ta będzie informowała program do jakiej drogi powinien on przypisać daną nutę. Dzięki takiemu modelowi można teraz w klasie ***Game*** utworzyć listę obiektów typu ***Beat***, wczytując dane do tej listy z odpowiedniego pliku.

KeyListener

Odpowiada za obsługę przycisków klawiatury. Jest to model, w którym z góry ustalamy zasady postępowania w przypadku, gdy komputer wykryje zmianę stanu danego klawisza. W tym wypadku odwołujemy się do metod zapisanych wcześniej w klasie ***Game***.

Poniższy fragment kodu prezentuje przykładowe akcje dla klawisza „strzałki w górę”.

```
1.  @Override
2.  public void keyPressed(KeyEvent e) {
3.      ...
4.      if(e.getKeyCode() == KeyEvent.VK_UP) {
5.          RythmCode.game.pressW();
6.      }
7.      ...
8.  }
9.
10. @Override
11. public void keyReleased(KeyEvent e) {
12.     ...
13.     if(e.getKeyCode() == KeyEvent.VK_UP) {
14.         RythmCode.game.releaseW();
15.     }
16.     ...
17. }
```

5. Plany rozwoju na przyszłość

5.1 Pomysły niewykorzystane

Tworząc projekt zawsze trzeba liczyć się z tym, że nie wszystkie nasze pomysły zostaną zaimplementowane. Przeszkodą może być niewystarczająca ilość czasu lub trudności w zaprogramowaniu odpowiedniego elementu. Nie inaczej było przy tworzeniu tego projektu. O ile większość podstawowych założeń została zrealizowana pomyślnie, to niestety po drodze trzeba było pójść na parę kompromisów.

Jednym z nich było całkowite ominięcie wsparcia kontrolerów do gier. Był to pomysł, który pojawił się na początku podczas planowania gry. Niestety przeszkodą okazał się tutaj czas. Biorąc pod uwagę, że jest to dosyć ograniczony zasób, oznaczałoby to, że tworząc wsparcie dla różnego rodzaju kontrolerów, musielibyśmy zaniedbać aspekty optymalizacyjne stworzonego programu.

Kolejnym pomysłem niezrealizowanym było stworzenie serwera, który przechowywałby informacje o punktacji różnych graczy za dany poziom. W ten sposób gracze z całego świata mogliby konkurować o najlepszy wynik przejścia danego poziomu. Tutaj problemem było to, że w takim wypadku należałoby przeprogramować całą grę dodając do niej tryb online. Aby w skuteczny sposób pokazywać wyniki, każdy użytkownik musiałby także mieć odpowiednie konto, aby przypisywać jego login do odpowiedniego wyniku punktowego. To tworzy kolejne moduły do stworzenia – logowanie i rejestrację. W dodatku trzeba w takim wypadku także utrzymywać bazę danych przechowującą wszystkie te informacje. Przy tak małym projekcie jest to zwyczajnie nieopłacalne.

Tryb fabularny także był jednym z pomysłów, które nie znalazły się w wersji końcowej gry. Zamierzenia były takie, aby dodać silnie fabularyzowany tryb dla pojedynczego gracza z odpowiednimi animacjami i nowymi mechanikami. Prawdopodobnie gdyby przy tym projekcie pracowało więcej osób, ten pomysł mógłby zakończyć się powodzeniem. O ile zaprogramowanie takiego trybu nie byłoby dużym problemem, ponieważ działałby on na utworzonym już wcześniej szkielecie gry, o tyle tworzenie odpowiednich animacji, grafik czy innych zasobów byłoby bardzo trudne. W tym przypadku do pomocy przy projekcie potrzebny byłby grafik. Cały projekt był robiony tylko przeze mnie, więc taki pomysł nie mógł zostać zrealizowany.

5.2 Perspektywy na przyszłość

Po zakończeniu prac nad projektem pozostało do poruszenia kilka kwestii odnośnie dalszego rozwoju stworzonej gry. Sprawą priorytetową jest dodanie wszystkich pomysłów niewykorzystanych. Po wydaniu wersji finalnej nie trzeba już się martwić o brak czasu, dlatego też można powoli szlifować poszczególne elementy, dopóki nie będą one gotowe.

Po dodaniu obsługi kontrolera, bardzo dobrym pomysłem byłoby zaimplementowanie wsparcia mat do tańczenia, aby w pełni wykorzystać potencjał gry. W ten sposób otrzymany produkt byłby jeszcze bardziej interaktywny. Dodało by to możliwość sprawdzania i rozwijania koordynacji ruchowo-słuchowej graczy, a także sprawiłoby, że gra mogłaby znaleźć swoje miejsce w ośrodkach zdrowia i służyć jako sposób rehabilitacji dla dzieci i osób starszych.

Po takiej rozbudowie produktu można już rozpocząć proces wydania gry na rynku komputerowym i konsolowym. Zamierzonymi platformami docelowymi byłyby komputery osobiste z systemem Windows oraz konsola przenośna *Nintendo Switch*. Do tego trzeba byłoby przeportować grę na odpowiedni silnik wspierany przez konsole do gier. Tutaj najlepszy wyborem byłby *Unity* lub *Unreal Engine 4*, ponieważ wspierają one tworzenie produktów multiplatformowych. Cała operacja byłaby dosyć skomplikowana, biorąc pod uwagę, że aby stworzony projekt działał na wyżej wymienionych silnikach, to musiałby być przepisany na inny język akceptowany przez te środowiska. Akurat dla tych dwóch opcji można wybrać język *C++* lub *C#*.

Tym oto sposobem produkt stałby się oprogramowaniem komercyjnym, gotowym do ukazania się na rynku konsumenckim.

Spis ilustracji

Bibliografia

[1] Bodnar, J. (2007 - 2019). *Java 2D games tutorial.* www.zetcode.com. Data pobrania - 02.2019

[2] Davison, A. (2005). *Killer Game Programming in Java.* O'Reilly Media.

[3] Design Doc. (2017-2019). *Good Design, Bad Design - The Best & Worst of Graphic Design in Games.* Seria filmów na platformie YouTube. Data pobrania - 04.2019

[4] JavaZoom. (1999-2008). *JLayer API V1.0.1 - dokumentacja.* www.javazoom.net. Data pobrania - 02.2019

[5] Yang, H. (2018). *Java Swing Tutorials - Herong's Tutorial Examples.* Wydane niezależnie.

Załącznik nr 3 do zarządzenia Rektora UG nr 70/R/15

OŚWIADCZENIE

Wyrażam zgodę / nie wyrażam zgody* na udostępnienie osobom zainteresowanym mojej pracy dyplomowej dla celów naukowo-badawczych.
Zgoda na udostępnienie pracy dyplomowej nie oznacza wyrażenia zgody na kopiowanie pracy dyplomowej w całości lub w części.

* *niepotrzebne skreślić*

…… ……

data *podpis*

OŚWIADCZENIE

Ja, niżej podpisana(y) oświadczam, że przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

....

data podpis