WWSIS@

Wrocławska Wyższa Szkoła

Informatyki Stosowanej

Wydział Informatyki

Jakub Bożek

Nr albumu: 4984

Stworzenie gry mobilnej na system Android z wykorzystaniem biblioteki LibGDX

Praca inżynierska

Kierunek: Informatyka

Specjalność/specjalizacja: Programowanie

Praca wykonana pod kierunkiem:

mgra Pawła Rzechonka

Wrocław 2019

Spis treści

Wstęp

Wybór tematu pracy nie był dla autora przypadkowy. Zdecydował się on na wykorzystanie biblioteki LibGDX w swojej pracy, ponieważ wymagało to od niego jeszcze lepszego poznania Javy, jako języka programowania, z którym wiąże swoją przyszłość. Zainteresowania autora różnego rodzaju grami mobilnymi i komputerowymi było kolejnym powodem, dla którego podjął się on zaprojektowania własnej gry mobilnej. Stworzenie takiego projektu wymaga nie tylko znajomości programowania, lecz pozwala też dokształcać swój warsztat w zakresie obróbki graficznej i fonicznej.

Celem pracy jest zaprogramowanie gry mobilnej na system Android z wykorzystaniem biblioteki LibGDX. Biblioteka ta jest rozszerzeniem podstawowych bibliotek Javy i służy przede wszystkim do projektowania gier i rozbudowanych interfejsów graficznych.

Zaprojektowana gra osadzona jest w świecie fantasy. Zastosowana zostanie grafiki 2D w stylu PixelArt. Projekt zakłada stworzenie gry RPG (*role playing game*), w którym użytkownicy będą mogli się wcielać w wykreowane przez siebie postaci. Zadaniem użytkownika będzie penetracja lochów w poszukiwaniu przejścia do kolejnego poziomu oraz walka z potworami - przeciwnikami sterowanymi przez AI.

Gra opierać się będzie o system turowy, w którym gracz wykonuje ruchy w ramach dostępnych punktów akcji. Każdy użytkownik będzie mógł rozwijać swoją postać zwiększając jej umiejętności i atrybuty. Z założenia zaprojektowania gra nie posiada jednego, konkretnego zakończenia. Celem gry jest rozwijanie postaci przez użytkownika, który wraz z osiąganiem kolejnych poziomów swojego zaawansowania, będzie otrzymywał większe wyzwania. Dlatego też prezentowana gra nie ma jednego, scenariuszowego zakończenia.

W kolejnej części pracy opisane zostaną technologie i narzędzia, z których korzystał autor. Do wykorzystanych technologii zalicza się do nich Java, LibGDX, Android, a także PHP i MySQL. Natomiast narzędziami, z których korzystał autor są Android Studio, Adobe Photoshop CS6 oraz ChipTone by SFB Games, XAMMP i MySQL Workbench.

W kolejnym rozdziale znajduje się opis projektu, czyli wymagania funkcjonalne i niefunkcjonalnej projektu oraz pakiety i klasy. Czwarty rozdział to z kolei opis implementacji najważniejszych elementów programu tj. zastosowane algorytmy, system graficznego generowania mapy, system turowy, system sterowania, a także opis zastosowanej bazy danych.

Ostatnią częścią prezentowane pracy będzie zakończenie, które stanowi podsumowanie tego, co udało się autorowi osiągnąć w zakładanym projekcie.

Stworzenie gry mobilnej było przede wszystkim ogromnym wyzwaniem dla jej autora, który w swoim projekcie musiał wykazać się nie tylko znajomością programowania, lecz także podstaw grafiki komputerowej czy obróbki dźwięków. Taka wielopłaszczyznowość projektu była pierwszym, tak dużym sprawdzianem zdobytych umiejętności i posiadanych kompetencji.

1. Tematyka projektu

Rynek gier mobilnych zarówno w Polsce jak i na świecie rozwija się bardzo dynamicznie. Spowodowane jest to przede wszystkim coraz większą dostępnością do narzędzi, jakimi są smartfony oraz tablety obsługujące aplikacje mobilne. Szacuje się, że w 2010 r. przemysł gier mobilnych wart był 33 miliardów dolarów. Już w 2016 r. rynek gier mobilnych wygenerował łącznie 40,6 miliarda dolarów. Kwotę tą można porównać do łącznej sumy wydanej przez całe społeczeństwo na świecie w kasach biletowych w kinach. Z prezentowanych przez Newzoo danych wynika, że w samym 2017 r. rynek gier mobilnych wygenerował przychód na poziomie 110 miliardów dolarów. Szacuje się, że do 2020 r. przychód z gier mobilnych na świecie osiągnie poziom prawie 129 miliardów dolarów.

Jeszcze inny raport zakładał, że w 2018 r. przychód z gier mobilnych na świecie osiągnie poziom 137,9 miliarda dolarów, a jego wartość do 2021 roku osiągnie poziom 180 miliardów dolarów. Bez względu na prezentowane dane i ich rozbieżność wynikającą z różnych dat publikowania niniejszych raportów, warto zauważyć, że rynek gier mobilnych jest chyba najprężniej rozwijającym się rynkiem na świecie. Dlatego też aplikacje mobilne, a właściwie ich coraz większa popularność jest szansą dla młodych programistów, dopiero co wchodzących na rynek pracy, ponieważ pracy w tej dziedzinie z miesiąca na miesiąc będzie przybywać.

Jednakże, aby lepiej zrozumieć cały projekt, należy zacząć od scharakteryzowania podstawowego pojęcia, jakim jest „gra komputerowa” oraz gra typu „dungeon crawler” oraz gra typu RPG.

1.1. Pojęcie gry komputerowej

Samo pojęcie gry jest bardzo szeroki i niejednoznaczne. Najprościej mówią, za grę rozumieć możemy pewnego rodzaju rozrywkę, w którą zaangażowana może być jedna lub więcej osób. W większości przypadków, kiedy zapytamy kogoś czym, jest dla niego gra, zapewne opowie nam o tzw. „planszówkach” lub też właśnie grach komputerowych, które od czasu pojawienia się komputerów i rozwoju sieci, stały się bardzo popularną formą rozrywki dla milionów ludzi na całym świecie.

Analizując literaturę dotyczącą gier komputerowych nie znajdziemy jednej, dokładnej definicji tego pojęcia. Oczywiście na samym początku należy zajrzeć do Encyklopedii PWN, z której dowiadujemy się, że gra komputerowa jest to gra, która wymaga programu

komputerowego do przeprowadzenia rozgrywki, w której udział może brać jeden lub więcej graczy. Nieco inną definicję prezentuje Słownik Język Polskiego, z którego dowiadujemy się, że oprócz tego, iż jest to program komputerowy, za pomocą którego rozgrywana jest gra, to sama gra komputerowa stanowi formę rozrywki, do której niezbędny jest komputer, a właściwie ekran komputera, na którym wyświetlana jest gra.

Jednakże w dalszym ciągu nie są to definicje kompleksowe, które brałyby pod uwagę poziom złożoności tego pojęcia. Jedno z bardziej obszernych wyjaśnień znaleźć można w książce *Obrót dobrami wirtualnymi w grach komputerowych. Studium cywilnoprawne* autorstwa dra Kamila Szpyta, w której autor wskazuje, iż sama *gra* stanowi zabawę towarzyską prowadzoną według pewnych zasad lub też rozgrywkę prowadzoną między zawodnikami lub zespołami według zasad określonych regulaminy danej dyscypliny. Może to być także przedmiot służący do grania, czyli prowadzenia rozgrywki. Autor w swojej monografii wskazuj, iż pod pojęciem gier komputerowych może kryć się sama rozgrywka - np. gram w grę komputerową, a także wytwór artystyczny, np. kiedy powiemy „kupiłem sobie grę komputerową" jako przedmiot sam w sobie.

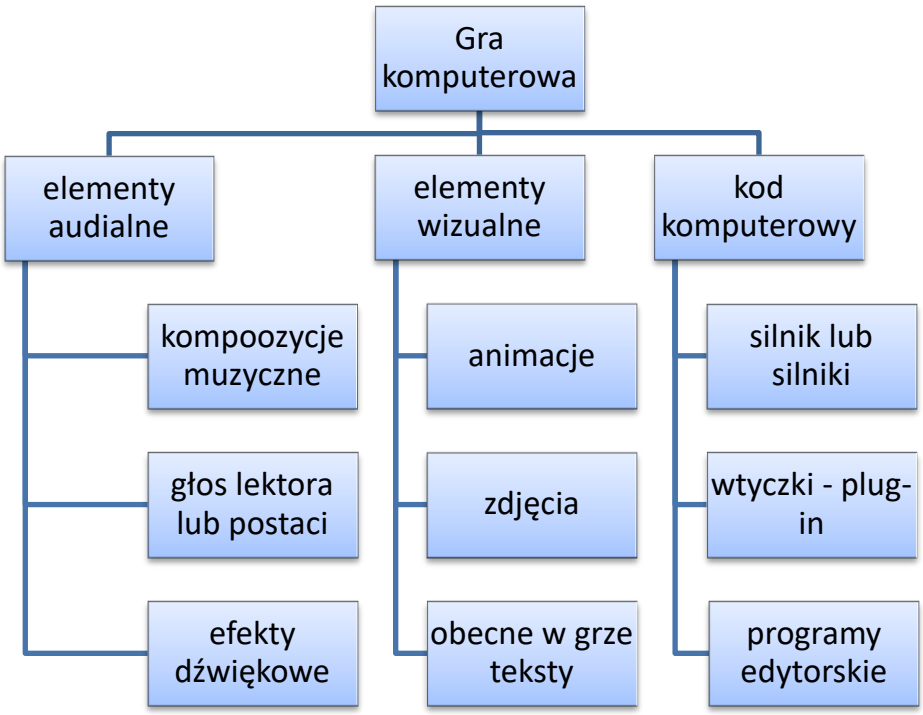
Autor wskazuje ponadto, iż w polskim prawodawstwie nie istnieje jedna, spójna definicja gry komputerowej, dlatego też w dalszej części swojej pracy odwołuje się do wyjaśnień tego pojęcia użytych w aktach prawnych i monografiach z tego zakresu. Wskazuje m.in. że według Marii Wąsowskiej *„pod pojęciem gry komputerowej, z punktu widzenia techniki, rozumiany jest program komputerowy, który po uruchomieniu na ekranie komputera lub przeznaczonej do tego innej maszyny (konsoli, telewizora, automatu zręcznościowego) prezentuje dane, m.in. dźwięk i grafikę, jednocześnie komunikując się z użytkownikiem, gwarantuje mu wpływ na przebieg jej wyświetlania w myśl ustalonych pierwotnie reguł i zasad".*

Dr Kamil Szpyta wskazuje jednak, iż wszystkie definicje, które spotkać można w polskich monografiach i aktach prawnych są niekompletne lub też w niektórych przypadkach błędne, ponieważ ich autorzy utożsamiają pojęcie gry komputerowej z programami komputerowymi, przez co stają się one synonimem programu komputerowego, a to nie jest do końca prawidłowe. Oczywiście program komputerowy stanowi bardzo ważną część gry komputerowej, jednak nie jest on jedyną składową całego pojęcia.

Bardzo ważną część gry komputerowej stanowi program sterujący czyli tzw. silnik, który zbudowany jest z wielu instrukcji napisanych w języku zrozumiałym dla komputera

lub też innej maszyny, na której działa gra komputerowa i za jego pomocą komunikuje się z graczem. Autor jasno podkreśla, iż w dzisiejszym świecie sam kod programu sterującego nie zawiera instrukcji dotyczących generowania elementów, które wyświetlają się na ekranie komputera. Wszystkie elementy tworzone są oddzielnie i dopiero na etapie implementacji łączone w jedną całość.

Aby ułatwić zrozumienie złożoności budowy gry komputerowej, dr Szpyt przedstawił ogólny schemat budowy każdej gry komputerowej, która powinna składać się z 3 elementów - audialnych, wizualnych i programowych.



Schemat 1.1 Ogólne elementy każdej gry komputerowej

opracowanie własne na podstawie K. Szpyt, Obrót dobrami wirtualnymi w grach komputerowych. Studium cywilnoprawne, s.5.

W prezentowanej pracy opisywana będzie gra mobilna, która to stanowi rodzaj gier komputerowych. - z jedną różnicą - została ona stworzona na urządzenia mobilne - przenośne tj. smartfony czy tablety. Warto zaznaczyć, że już sama gra komputerowa jest wymieniana w literaturze jako jeden z rodzajów aplikacji mobilnych.

W stworzonej przez autora grze przeplatają się dwa typy gier, na których opary został cały projekt. Pierwszy z nich to gra typu „dungeon crawler", a druga to RPG, które to zostaną omówione poniżej.

1.2. Gra typu RPG

RPG - *role play games* jest to gra fabularna, w której gracze wcielają się w role fikcyjnych postaci. W grach tego typu gracze biorą udział w opowiadanej przez narratora historii, która stanowi tło dla całej gry. Gry RPG są rodzajem gier towarzyskich, w które można grać samodzielnie lub też w grupie. Celem gry typu RPG jest po pierwsze rozegranie gry według założonego przez autora scenariusza, osiągnięcie celów grupowych lub też indywidualnych, a także rozwój postaci.

Gry RPG są często wielowątkowe i charakteryzują się bardzo dużą ilością zadań (*questów*), które gracze muszą wykonać. Warto zauważyć, że na samym początku RPG były grami papierowymi - tzw. planszówki lub gry karciane, w których oprócz graczy, występował także narrator - *game master*, który prowadził całą rozgrywkę. Po raz kolejny, wraz z rozwojem sieci komputerowej oraz dostępnych technologii, zaczęto tworzyć komputerowe wersje gier RPG.

Dungeon & Dragons to pierwsza komercyjna wersja gry fabularnej, która wydana została w 1974 roku w wersji miniaturowej gry figurkowej. Nie bez znaczenia pozostaje data opublikowania tej gry - przypada ona na rok po śmierci R.R Tolkiena, w którego świecie fantasy została właściwie osadzona pierwsza wersja tej gry. Jednakże sami autorzy *D&D* twierdzą, iż większość inspiracji czerpali z mitologii greckiej i egipskiej, a świat Tolkiena był dla nich inspiracją w bardzo niewielkim stopniu. Jednakże w pierwotnej wersji gry pojawiły się charakterystyczne dla Tolkiena pojęcia i postaci tj. *hobbit, enta czy balroga*, które z czasem musiały zostać zmienione dzięki staraniom właścicieli praw autorskich - spadkobierców Tolkiena.

1.3. Gra typu „dungeon crawler”

Gry typu *dungeon crawler* jak łatwo się domyśleć wywodzą się od wspomnianej powyżej *Dungeon & Dragons,* czyli od prekursora gier fabularnych na świecie i stanowią rodzaj przygody w grach fabularnych, w której bohaterowie penetrują lochy lub inne miejsca - zamki czy jaskinie, w których mapa przypomina labirynt, walczą z potworami - smokami czy orkami, pokonują pułapki zastawione w grze, a także zbierają ukryte skarby.

Samo pojęcie *dungeon crawler* bardzo często używane jest w znaczeniu pejoratywnym, ponieważ przygody w grach tego typu nie mają spójnego scenariusza, niektórzy uważają, że są po prostu nielogiczne, przez co stają się łatwiejsze

w przeprowadzeniu dla *game master'a.* Fabuła walki ze smokami i innymi potworami w lochach lub innych labiryntach jest bardzo często wykorzystywana w grach *roguelike*.

1.4. Gra typu „roguelike"

Gra typu *roguelike* cechuje się przede wszystkim losowością świata. Każdorazowo w rozgrywkach lochy, potwory, skarby, miasta i inne elementy składowe gry generowane są losowo. Gry tego typu prezentowane są w postaci znaków ASCII w trybie tekstowym lub za pomocą bardzo prostej grafiki kafelkowej. W grach tego typu rozgrywki zazwyczaj toczą się w trybie turowym, a śmierć gracza kończy daną rozgrywkę bez możliwości powrotu do wcześniej zapisanego etapu gry. Obecnie gry tego typu nie należą do najbardziej popularnych, ponieważ poziom skomplikowania gry, który nie idzie w parze z poziomem graficznym sprawie, że duża część osób przenosi się na gry z dużo lepszą jakością przede wszystkim graficzną. Nie oznacza to jednak, że *roguelike* nie ma swoich fanów - każda gra komputerowa posiada swoje grono odbiorców - swoich zwolenników i zapalonych fanów.

Prezentowana w poniższej części praca będzie mieć charakter gry RPG, w której wykorzystano elementy gry *dungeon crawler* - będą elementy walki z potworami i penetrowanie lochów, a także elementy gry *roguelike* - lochy oraz pojawiające się potwory będą generowane losowo.

2. Wykorzystane technologie

Założeniem projektu jest stworzenie gry mobilnej z wykorzystaniem języka Java. Autor doskonale zdaje sobie sprawę, z tego, że gry można stworzyć za pomocą specjalnie dedykowanych silników do gier tj. Unity, Unreal Engine oraz tym podobnych. Jednakże autor zdecydował się stworzyć swoją grę właśnie w Javie z uwagi na chęć rozwoju oraz zainteresowania tym właśnie językiem. Aby ułatwić sobie pracę, autor zdecydował się skorzystać z rozszerzenia do Javy jakim jest biblioteka LibGDX. W rozdziale tym oprócz wskazanych powyżej zostanie także opisany system Android jako środowisko, na które została zaprogramowana gra. Wskazane poniżej technologie są niezbędne do stworzenia szkieletu gry, czyli kodu źródłowego projektu.

2.1. Java

Java jest to technologia, którą wykorzystuje się do tworzenia różnego rodzaju aplikacji sieciowych. Jest ot ogólnoświatowy standard wykorzystywany do tworzenia zarówno aplikacji na urządzenia mobilne, aplikacji wbudowanych, gier, a także zawartości i treści internetowych. Java jest także podstawową technologią do tworzenia wszelkiego rodzaju oprogramowania dla przedsiębiorców.

Środowisko programistyczne Java swoją premierę miało pod koniec lat 90. ubiegłego wieku. Oprócz samej Javy wyróżniamy także jej środowisko pomocnicze, którym jest JVM - *Jave Virtual Machine,* czyli nic innego jak wirtualna maszyna Javy. Tak wielkiego sukcesu Javy możemy upatrywać głównie w tym, że została ona ustandaryzowana. Stworzono specjalne dokumentacje techniczne, które dają zarówno programiście jak i użytkownikowi gwarancję zgodności pomiędzy poszczególnymi składnikami tego środowiska.

Wspomniano powyżej, że Java swoją premierę miała pod koniec lat 90. ubiegłego wieku. Jednak warto cofnąć się o kilka lat wstecz, aby poznać początki jej powstawania. Wszystko zaczęło się w 1991 r., kiedy to zespół inżynierów jednej z największych w tamtym czasie amerykańskich firm informatycznych, firmę Sun postanowił stworzyć niewielki język programowania, za pomocą którego można byłoby tworzyć zwięzłe kody wykorzystywane przez m.in. tunery telewizji kablowej. Przewodniczącymi zespołu byli Patric Naughton i James Gosling, a ich projekt otrzymał kryptonim *Green.* Nazwa Java także wzięła się właściwie z przypadku, ponieważ pierwotnie Gosling chciał nazwać język Oak, jednak okazało się, że już istnieje język programistyczny o takiej nazwie. Wtedy wymyślono Java, co okazało się idealną nazwą.

W 1996 r. firma Sun po raz pierwszy oficjalnie zaprezentowała światu nowy język programowania jakim była Java. Niedługo po premierze okazało się, że wersja 1.0 jest po prostu za słaba do tworzenia dużych projektów. W 1997 r. powstała wersja 1.1, jednak pomimo ulepszeń nadal jej możliwości były ograniczone. W 1998 r. wydano koleją wersję - 1.2, której po kilku dniach zmieniono nazwę na Java 2 Standard Edition Software Development Kit Version 1.2. Przez kolejne lata powstawały kolejne wersje Javy, rozrastała się biblioteka standardowa, a sama Java zwiększała swoją wydajność. Na bieżąco poprawiane były też błędy i dlatego z czasem Java stała się najpopularniejszą platformą do tworzenia aplikacji.

Największe zmiany w Javie wprowadziła wersja 1.5 z 2004 r., którą przemianowano później na Java 5.0. Nowością były Klasy sparametryzowane, pętla *for each,* atrybuty o zmiennej liczbie argumentów (varargs), enumeracje oraz statyczny import. W 2006 r. pokazała się na rynku kolejna wersja - Java 6, jednak nie wprowadzono w tym czasie żadnych nowości, jedynie usprawniono jej działanie oraz rozbudowano bibliotekę. Największe zmiany nastąpił w 2009 r., kiedy to firma Sun popadła w problemy finansowe i została wykupiona przez firmę Oracle. Na kolejnych kilka lat wstrzymano proace rozwojowe nad Javą. Dopiero w 2011 r. wypuszczono na rynek jej kolejną wersję - Javę 7, która posiadała takie ulepszenia jak instrukcja switch z łańcuchami, operator diamentowy, literały binarne oraz udoskonalenia mechanizmu obsługi wyjątków.

Przełomowym był rok 2014, kiedy to ukazało się kolejne wydanie Javy - wersja 8. Wprowadzono m.in. wyrażenia lambda, które miały zdecydowanie usprawnić pracę programistów. Do innych zmian należą implementacja Java Script oraz nowość w obsłudze daty, czasu i profilu Java. Aktualizacja Javy w 2014 r. było ostatnim tak dużym działaniem usprawniającym funkcjonalność całego środowiska.

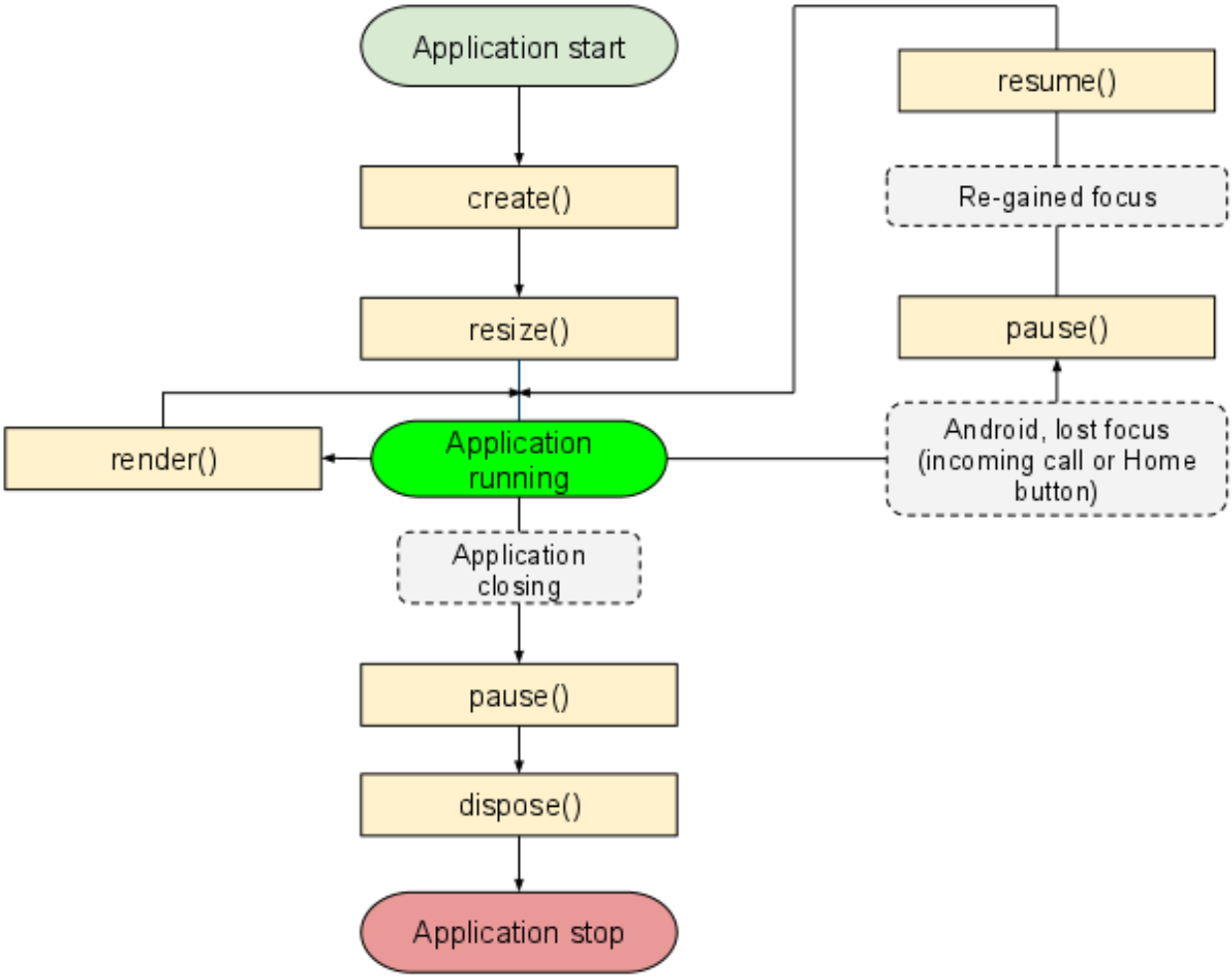
Warto zauważyć, że architektura Javy zmienia się bardzo powoli i jest dość tradycyjna. Już od samego początku jej twórcy zakładali kompatybilność nowych wersji języka z poprzednimi, co oznacza, że nawet programy napisane w początkowych wersjach, powinny działać bez problemu w tych najnowszych. Jednak zmiany z 2014 r. wprowadziły nie tylko ulepszenia całego środowiska, lecz także dokonały „kapitalnego remontu" jej kodu źródłowego, co zmieniło całkowicie sposób programowania w tym języku. Java jest wszędzie - statystki jasno wskazują, że jest to najpopularniejsza technologia wykorzystywana do tworzenia wszelkiego rodzaju aplikacji - korzysta z niej blisko 9 milionów programistów na całym świecie.

2.2.LibGDX

Biblioteka LibGDX jest to freamworkowa niskopoziomowa platforma ułatwiająca programowanie gier komputerowych na wielu platformach. Oznacza to, że można napisać jeden kod źródłowy gry i za pomocą tej platformy uruchomić ją na systemach Windows, Linux, Mac OS X, Android, iOS oraz HTML5. W przypadku prezentowanej pracy będzie to system Android.

Biblioteka ta jest darmowa i działa na licencji Open Source Apach2, co oznacza, że każdy ma prawo do korzystania z jej zasobów podczas tworzeniach swoich projektów. LibGDX zasadniczo składa się z 6 modułów - *application, files, input, net, audio oraz graphics. Application* pozwala na uruchomienie biblioteki i informuje klienta API o zdarzeniach na poziomie aplikacji - m.in. zmianie rozmiaru okna. Ten interfejs pozwala na rejestrację użytkownika oraz działania tzw. wsparcia, czyli wyszukiwanie w pamięci zapytań na dany temat. *Files* ukazuje podstawowy system plików omawianej platformy. Zapewnia dostęp do plików abstrakcyjnych na wszystkich platformach niezależnie od nośnika. Moduł i*nput* zapewnia ujednolicony model wejścia i procedurę obsługi dla wszystkich platform. Moduł *net (networking)* zapewnia dostęp do zasobów internetowych podczas pracy nad projektem. *Audio* z kolei daje możliwość odtwarzania dźwięków i efektów muzyki strumieniowej. *Graphics* pozwala na odczytywanie i ustawiania trybów wideo oraz zapewnia możliwość generowania map oraz tekstur. LibGDX obsługuje zarówno grafikę 2D, jak i 3D oraz obsługuje OpenGL ES.

W bibliotece LibGDX zostały stworzone zestawy API, które są niezwykle pomocne przy m.in. renderowaniu sprajtów, tekstu oraz interfejsu; odtwarzaniu efektów dźwiękowych czy zadań z zakresu algebry liniowej i trygonometrii.

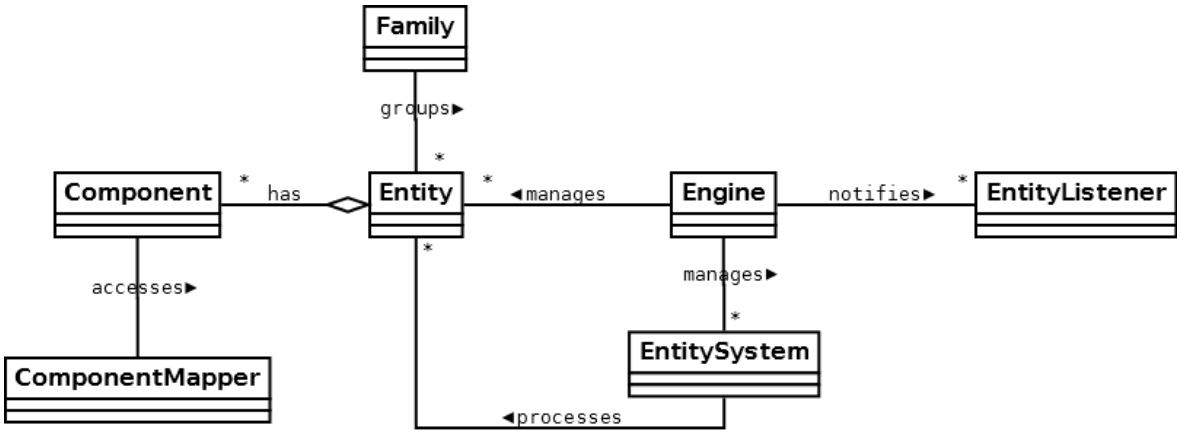


Schemat 2.1 Cykl życia aplikacji LibGDX

Źródło: *https://github.com/libgdx/libgdx/wiki/The-life-cycle*

2.3. Rozszerzenie Ashley

Ashley to rozszerzenie biblioteki LibGDX niezbędne do tego, aby w odpowiedni sposób zarządzać całym cyklem projektowym. Rozszerzenie to wykorzystuje się do dzielenie funkcji na niezależne systemy, które stają się systemami uniwersalnymi, przez co z łatwością można je wykorzystywać w kolejnych projektach. Takie działania przyspieszają proces tworzenia aplikacji.



Schemat 2.2 Schemat zależności w rozszerzeniu Ashley

Źródło: http://silversableprog.blogspot.com/2015/04/libgdx-i-ashley-wstep.html

2.3.1. Component

Component (komponent) jest klasą, która opisuje daną cechę np. pozycję - zdefiniowaną za pomocą *x* i *y* lub też wielkość - określającą *width* oraz *height*. Komponenty stanowią najważniejsze elementy całego programu, ponieważ są wykorzystywane przy tworzeniu obiektów - *entity*, a także w niektórych systemach poprzez wykorzystanie klasy *family*.

ComponentMapper (mapowanie komponentów) stanowi pewnego rodzaju wyszukiwarkę komponentów z danego obiektu. Jest to najszybsza metoda dostępu do komponentów, za pomocą której możemy z nich korzystać i je modyfikować. Zazwyczaj przechowuje się wszystkie ComponentMappers w jednej klasie - w formie zmiennych statycznych.

2.3.2. Entity

Entity (obiekt) jest to zbiór komponentów określających jego zachowanie lub przeznaczenie. W odróżnieniu od obiektów w klasycznym programowaniu obiektowym, w którym zmienne są przypisane na stałe do klasy, w przypadku *entity* możemy w każdym momencie dodawać i usuwać komponenty zmieniając zachowanie danego obiektu. Aby przyspieszyć proces powielania danego *entity*, np. kafli tworzących mapę, stosuje się *EntityFactory (*tzw. fabrykę), która automatyzuje dany proces.

2.3.3. Family

Family, podobnie jak *ComponentMapper*, jest wyszukiwarką, za pomocą której możemy wyszukiwać obiekty. Istnieją trzy sposoby wyszukiwania obiektów w klasie *Family*:

- *all* - obiekty muszą posiadać wszystkie wymienione cechy,
- *one* - obiekty muszą posiadać minimum jeden wskazany komponent,
- *exculde* - obiekty nie mogą posiadać żadnej z wymienionych cech.

2.3.4. EntitySystem

EntitySystems są to klasy odpowiedzialne za działanie naszej aplikacji - tworzą tzw. logikę programu. Do działania wykorzystują główną pętlę programu przy pomocy zmiennej **(float deltaTime).** Większość systemów operuje na obiektach, które wcześniej wyszukiwane są za pomocą *Family*. Z klasy *EntitySystem* dziedziczą inne systemy tj.:

- *IteratingSystem* - ten system posiada dwie wyróżniające go cechy. Pierwszą z nich jest informacja o obiekcie *Family* zawarta w konstruktorze. Druga z nich to metoda *processEntity*, która w trakcie jednej iteracji głównej pętli programu, wykonuje daną czynność dla wszystkich obiektów podanych w konstruktorze.
- *IntervalSystem* - jak sama nazwa wskazuje, jest to system, który uruchamiany jest co pewien czas, zgodnie z określoną w konstruktorze częstotliwością.

2.3.5. Engine

Engine to kluczowy obiekt rozszerzeniu Ashley, który odpowiada za zarządzanie wszystkimi *entities* oraz *EntitySystems*. W każdym momencie działania aplikacji można dodać lub usunąć wyżej wymienione elementy.

2.4. System Android

Android jest obecnie najpopularniejszą platformą systemową dedykowana urządzeniom mobilnym takim jak telefony, smartfony czy tablety. Cała platforma składa się z systemu operacyjnego, oprogramowania pośredniego tzw. middleware, interfejsu oraz dodatkowych aplikacji. Cały system oparty został na jądrze Linuxa i oprogramowaniu działającym na licencji Apache License, która jest kompatybilna z GNU. Największym znaczeniem dla rozwoju całej technologii jaką jest Android miał fakt, że platforma ta nie jest związana żadnymi prawami autorskimi. Forma licencji tego systemu jak i darmowy do niego dostęp skutkował bardzo szybkim rozwojem całego projektu.

Historia Androida sięga 2003 r., kiedy to w Palo Alto (siedzibie wielu międzynarodowych korporacji m.in. HP oraz Facebooka) powstała niewielka spółka - Android Inc. Od samego początku działalność spółki wzbudzała duże zainteresowanie wśród globalnych gracz - w tym Google. W 2005 r. Android Inc. Został wykupiony przez Google za ogromną sumę 130 mln dolarów. To z kolei skutkowało poruszeniem na całym świecie - podejrzewano, że zakup systemu Android związany jest z tym, że Google chce wyprodukować swojego smartfona, który będzie konkurencją dla telefonów Apple. Jednakże już w 2007 r. ogłoszono, że stworzenie jednego smartfona nie jest dla Google atrakcyjne - ich planem jest wyposażyć wszystkie urządzenia mobilne na świecie właśnie w ten system. Słowa wypowiedziane przez Eric'a Schimdta w 2007 r. stały się prorocze, ponieważ już w 2011 r. Android stał się najpopularniejszym systemem na urządzenia mobilne.

Pierwotnie wersje Androida zostały nazwane Astro i Bender, jednak ostatecznie nie udało się ich wprowadzić, ponieważ trwały dyskusje i spory z właścicielami owych znaków towarowych. W 2009 r. podjęto decyzję, że każda kolejne wersja systemu będzie rozpoczynała się od kolejnej litery alfabetu i nawiązywała nazwą do jakiegoś deseru. Tabela poniżej prezentuje ewolucję systemu Android oraz jego wersji.

System Android został wybrany przez autora jako środowisko docelowe swojej gry, ponieważ jak wskazano powyżej, cieszy się on największą popularnością. Praktycznie większość smartfonów oraz tabletów obsługiwana jest przez Android, dlatego też stworzenie gry na ten system pozwala jednocześnie zapewnić sobie szersze grono potencjalnych odbiorców już w pierwszych dniach po premierze danego projektu.

Do obsługi prezentowanej gry mobilnej użytkownik potrzebuje posiadać urządzenie z systemem android w wersji nie starszej niż ***Nougat.***

NAZWA KODOWA	NUMER WERSJI	DATA PIERWSZEGO WYDANIA	POZIOMY API
Apple Pie	1.0	23 września 2008	1
Banana Bread	1.1	9 lutego 2009	2
Cupcake	1.5	27 kwietnia 2009	3
Donut	1.6	15 września 2009	4
Eclair	2.0 - 2.1	26 października 2009	5-7
Froyo	2.2 - 2.2.3	20 maja 2010	8
Gingerbread	2.3 - 2.3.7	6 grudnia 2010	9-10
Honeycomb	3.0 - 3.2.6	22 lutego 2011	11-13
Ice Cream Sandwich	4.0 - 4.0.4	18 października 2011	14-15
Jelly Beans	4.1 - 4.3.1	9 lutego 2012	16-18
KitKat	4.4- 4.4.4	31 października 2013	19-20
Lillipop	5.0 - 5.1.1	12 listopada 2014	21-22
Marshmallow	6.0 - 6.0.1	5 października 2015	23
Nougat	7.0 - 7.1.2	22 sierpnia 2016	24-25
Oreo	8.0 - 8.1	21 sierpnia 2017	26-27
Pie	9.0	6 sierpnia 2018	28

Tabela 2.1 Lista wersji systemu Android

Opracowanie własne na podstawie: https://www.android.com/intl/pl_pl/history

2.5. PHP

PHP jest to skryptowy jeżyk programowania, który używany jest najczęściej do tworzenia stron internetowych oraz aplikacji webowych działających w czasie rzeczywistym. Na potrzeby prezentowanej pracy został on użyty w kontekście stworzonej

bazy danych. PHP wykonywany jest po stronie serwera, co oznacza, że nie jest on widoczny dla użytkownika.

Zastosowanie PHP pozwala m.in. na dynamiczne generowanie strony, tworzenie i modyfikowanie plików na serwerze, a także ograniczenie dostępu do danych podstron, szyfrowanie danych czy stosowanie popularnych *cookies.* Innym zastosowaniem PHP jest obsługa baz danych, lecz to z kolei wymaga znajomości język SQL, który służy do tworzenia i modyfikowania baz danych, a także pobierania danych z istniejących już baz danych.

2.6. MySQL

Sam SQL to język zapytań, który umożliwia komunikację z bazami danych. Z kolei wykorzystana w pracy MySQL to najpopularniejsza na świecie baza danych typu *open source,* która przechowuje dane niezbędne do prawidłowego działania aplikacji.

3. Opis projektu

3.1. Wymagania niefunkcjonalne

Podstawowym założeniem realizowanego projektu było stworzenie prostego programu opartego na nieskomplikowanym kodzie, który zapewniałby przejrzystość i prostotę modyfikacji. Takie rozwiązanie miałoby w przyszłości uchronić przed modyfikacja kodu źródłowego w sytuacji, kiedy chcielibyśmy wprowadzić jakiekolwiek rozszerzenia w programie. Poniżej zaprezentowana zostanie szczegółowa lista wymagań - założeń, które postawiono sobie na etapie pisania projektu.

- Program powinien być odseparowany od plików zewnętrznych, pomijając te, które mogą znajdować się na serwerze. Przykładem takiej sytuacji może być informacja o danym graczu znajdująca się w bazie danych na serwerze.
- Parametry odpowiedzialne za działanie programu powinny być zapisane w jednym miejscu, np. w klasie *GameConfig.*
- Klasy w aplikacji powinny między ze sobą jak najmniej powiązań, nie licząc tych, które wynikają z dziedziczenia. Dodatkowo każda klasa w programie powinna odpowiadać za jedno, przypisane zadanie.
- Stałe nazewnictwo klas i obiektów - podczas tworzenia programu przyjęto za język użytkowy język angielski, a dalsza praca nad programem wymaga konsekwentnego stosowania się do raz przyjętej nomenklatury.
- Każdy element programu charakteryzujący się jakąkolwiek zmiennością w czasie, w swoich obliczeniach powinien wykorzystywać zmienne czasowe, które dostarczono do funkcji **update(float delta).**
- Wymogiem koniecznym do prawidłowego funkcjonowania aplikacji jest skalowalność oraz dostosowywanie się do każdego możliwego urządzenia mobilnego.
- Minimalizowanie zajmowanej pamięci operacyjnej poprzez usuwanie niepotrzebnych plików z pamięci, a także brak konieczności wczytywania wszystkich plików, które nie są wykorzystywane w danym momencie w programie.

3.2. Wymagania funkcjonalne

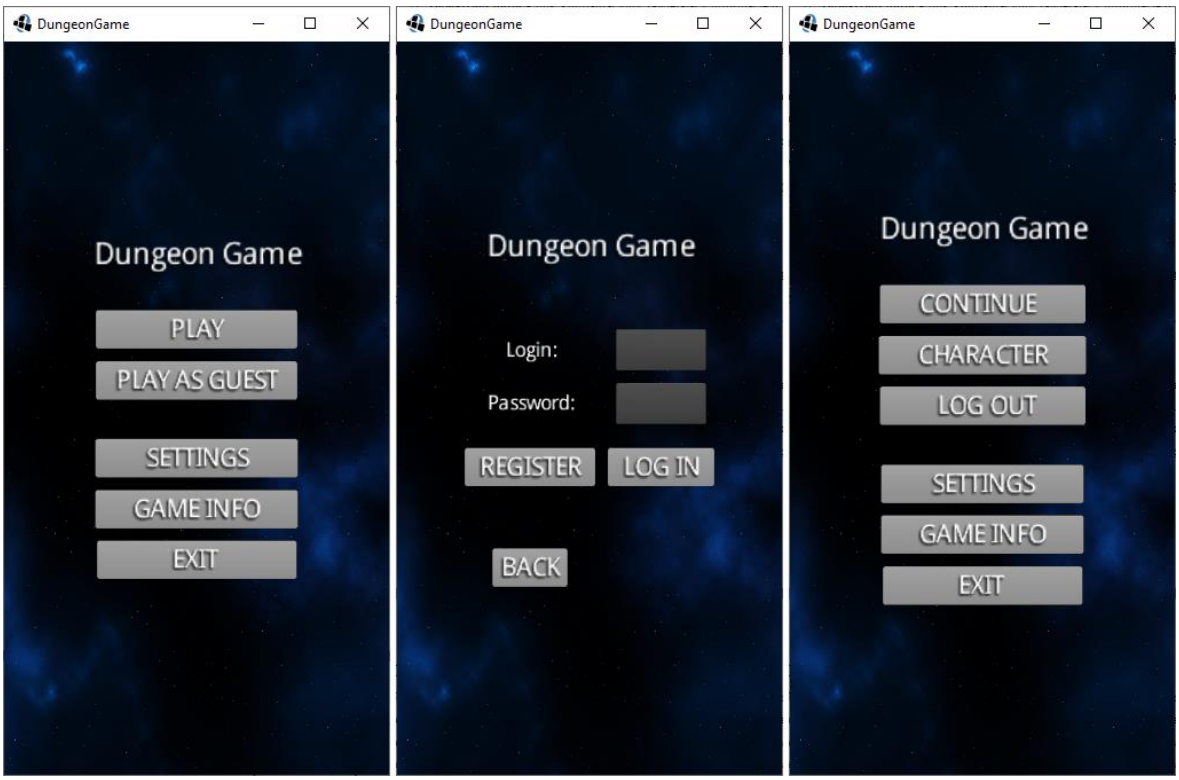
Wymagania (założenia) funkcjonalne definiują możliwości programu i z punktu widzenia użytkownika są najważniejszymi jego elementami. Wymagania funkcjonalne definiują m.in. zakres czynności gracza, czyli *de facto* to, co użytkownik będzie mógł robić w danej grze. Założenia te są o tyle ważnym elementem projektu, iż to na ich podstawie użytkownik końcowy będzie decydował o atrakcyjności danego projektu, a niespełnienie choćby jednego z nich, może zaważyć na ostatecznej decyzji potencjalnego użytkownika w kwestii wyboru tego właśnie programu. W trakcie tworzenia aplikacji lista wymagań funkcjonalnych ulegała zmianom, a właściwie rozszerzała się, aby ostatecznie wszystkie niezbędne funkcjonalności zostały w projekcie spełnione. Założenia funkcjonalne w projekcie można podzielić na dwie kategorie - pierwsza z nich odnosi się do działań podejmowanych na poziomie menu gry, a druga podczas właściwej rozgrywki.

Pełną listę wymagań funkcjonalnych na poziomie menu możemy podzielić na trzy poziomy:

- **Odpalenie aplikacji -** podczas pierwszego uruchomienia aplikacji, użytkownik ma do wyboru dwie opcje:
    - **PLAY AS GUEST -** gracz ma możliwość rozpoczęcia gry jako gość, bez konieczności tworzenia konta lub logowania się. Otrzymana w tym trybie postać posiada podstawowe statystyki.
        - **CONTIUNE -** kontynuacja gry
        - **SAVE PROGRESS -** z założenia gry jakoś gość postać przepada w momencie wyłączenia aplikacji, chyba że gracz zdecyduje się utworzyć konto.
    - **PLAY -** klikając w PLAY gracz ma możliwość zalogowania się lub stworzenia nowego konta. Po zalogowaniu lub założeniu konta, użytkownik ma do wyboru:
        - **CONTIUNUE -** kontynuacja gry
        - **CHARACTER -** zarządzanie postacią
        - **LOG OUT -** wylogowanie z gry.

Ponadto menu zawsze zawiera trzy elementy, które są stałe:

- **SETTINGS -** ustawienia aplikacji
- **GAME INFO -** instrukcja gry
- **EXIT -** wyłączenie aplikacji.



Rysunek 3.1 Zrzuty ekranu z menu aplikacji

opracowanie własne

Pełna lista wymagań funkcjonalnych na poziomie gry prezentuje się następująco:

- **Ruch postaci poza walką -** użytkownik klikając w ekran wskazuje miejsce, do którego przechodzi postać. Ruch postaci kończy się w momencie, kiedy została ona zauważona przez przeciwnika lub do miejsca wskazanego przez gracza nie prowadzi żadna droga.
- **Punkty akcji -** punkty posiadane przez gracza niezbędne do wykonaniu ruchu w walce lub podczas samej walki. Ich ilość odnawia się co każdą turę.
- **Ruch postaci w walce** różni się od ruchu poza nią tym, że po zauważeniu postaci przez przeciwnika, poruszając się gracz zaczyna zużywać dostępne punkty akcji. Punkty akcji używane są w grze zarówno do poruszania się jak i atakowania.
- **Walka:**
    - **zwykły atak -** atak zużywający jeden punkt akcji,
    - **precyzyjny atak -** atak zużywający 3 punkty akcji, który musi zostać użyty w celu pokonania jednego z rodzaju przeciwników, który posiada 95% szansy na uniknięcie zwykłego ataku, dlatego nie można go uniknąć. Ponadto precyzyjny atak przebija połowę pancerze przeciwnika.
    - **obrona -** zużywa jeden punkt akcji i kończy turę w zamian za zwiększenie pancerza.

- **Przejście na następny poziom lochu** - w zależności od tego, czy użytkownik gra jako gość, czy jako zalogowany użytkownik, następuje zapis gry w aplikacji lub do bazy danych, a następnie przejście na kolejny poziom gry. Aby przejść na następny poziom, gracz musi znaleźć przejście znajdującego się na mapie, którego lokalizacja jest zależna od początkowej lokalizacji gracza. Przejście zawsze znajduje się w najbardziej odległym pokoju na mapie względem pozycji początkowej gracza.
- **Powrót do menu / śmierć postaci** - w zależności od tego, czy użytkownik gra jako gość, czy jako zalogowany użytkownik, następuje zapis gry w aplikacji lub do bazy danych, a następnie powrót do menu głównego.

Rysunek 3.2 Interfejs gry - po lewej ruch postaci poza walką, po prawej ruch postaci w walce.

opracowanie własne

3.3.Pakiety i klasy

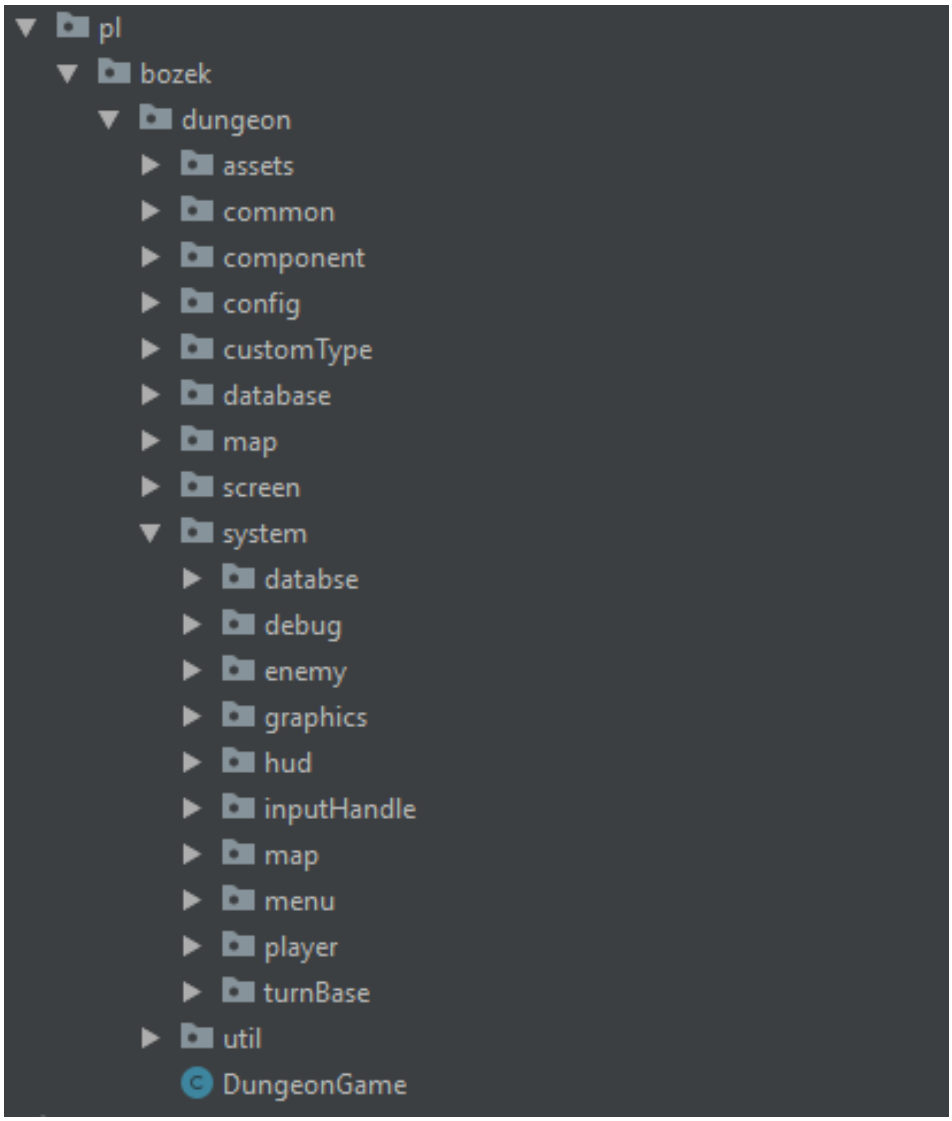
Biorąc pod uwagę poziom rozbudowania aplikacji, do jej efektywnego zarządzania, koniecznym było stworzenie odpowiednich pakietów dzielących ją na mniejsze, prostsze w obsłudze części. Każdy pakiet z kolei składa się z kolejnych pakietów lub klas.

Prezentowana praca składa się z pakietu nadrzędnego *pl.bozek.dungeon*, który z kolei składa się z takich pakietów jak:

- **assets -** pakiet ten zawiera klasy odpowiedzialne za obsługę multimediów,
- **common -** pakiet ten zawiera trzy rodzaje klas:
    - **factory** służące do dodawania obiektów *entity* do silnika gry - elementów mapy, gracza oraz przeciwników, a także elementów interfejsu,
    - ***CurentData,*** która służy do przechowywania bieżących danych aplikacji,
    - ***PlayerData,*** która służy do przechowywania informacji o graczu.
- **component -** pakiet przechowujący komponenty obiektów *entity,*
- **config -** składa się z trzech klas:
    - ***DungeonConfig*** - plik konfiguracyjny generatora map,
    - ***GameConfig*** - plik konfiguracyjny zawierający informacje o trybie debugowania, wymiarach okna aplikacji, a także skalowalności aplikacji.
    - ***GameHudConfig*** - plik konfiguracyjny zawierający informacje o wymiarach wszystkich elementów interfejsu użytkownika.
- **database** - zawiera jedną klasę ***DataBase,*** która służy do wysyłania zapytań do serwera,
- **map** - pakiet ten zawiera klasy odpowiedzialne za generowanie matematycznego modelu mapy,
- **screen** - pakiet ten przechowuje klasy, które odpowiedzialne są za prawidłowe wyświetlanie aplikacji, należą do nich:
    - ***BasicScreen*** - klasa służąca jako baza dla pozostałych klas w tym pakiecie,
    - ***MainLoadingScreen*** - klasa wyświetlająca czarny ekran podczas ładowania *assetów,*
    - ***MainManuScreen*** - służąca do wyświetlania menu głównego,
    - ***DungeonScreen*** - klasa odpowiadająca za wyświetlanie właściwej gry,
- **system:**
    - **database -** zawiera klasy służące do nasłuchiwania odpowiedzi od serwera,
    - **debug -** zawiera klasy pomocne w testowaniu aplikacji,

- **enemy -** zbiór klas opisujących zachowanie przeciwników w grze,
- **graphics -** zbiór klas służących do wyświetlania elementów graficznych,
- **hud -** zbiór klas służących do wyświetlania elementów graficznych interfejsu,
- **inputHandle -** zawiera klasy reagujące na interakcje użytkownika z aplikacją,
- **map -** zawiera klasy odpowiedzialne za graficzne generowanie mapy,
- **player -** zawiera klasy opisujące możliwości gracza,
- **turnbase -** zawiera klasy odpowiedzialne za system turowy gry,

- **util** zawiera klasy i pakiety pomocnicze:
  - **pakiet astar -** zawiera klasy algorytmu szukania drogi
  - ***PathUtils* -** klasa pomocnicza systemów ruchu,



Rysunek 3.3 Pakiety

opracowanie własne

4. Implementacja

W prezentowanym rozdziale przedstawione zostaną elementy, które były najbardziej pracochłonne w całym projekcie oraz charakteryzują się cechami autorskimi, zaprojektowanymi lub zmodyfikowanymi przez autora. Ze względu na objętość prezentowanej pracy oraz jej zakres, nie ma potrzeby prezentowania każdego zaprogramowanego elementu projektu, skupiono się natomiast na pokazaniu jego najciekawszych elementów.

4.1. Algorytmy

W tej części pracy zaprezentowane zostaną wybrane algorytmy zastosowane w projekcie. Ze względu na objętość pracy, algorytmy te nie są w 100% autorskimi rozwiązaniami, autor skorzystał z gotowych rozwiązań znalezionych w Internecie, które udoskonalił na potrzeby swojego programu, zgodnie z dołączonymi do nich licencjami.

4.1.1. Algorytm tworzenia mapy

Algorytm tworzenia mapy, z którego skorzystał autor, został zaprojektowany przez Boba Nystroma, autora książki *Game Programming Patterns*, w języku Dart. Najważniejszym założeniem stworzonego przez Nystroma algorytmu jest jego wydajność - ma on działać na tyle szybko, aby gracz nie musiał czekać bardzo dużo czasu na załadowanie się kolejnych poziomów.

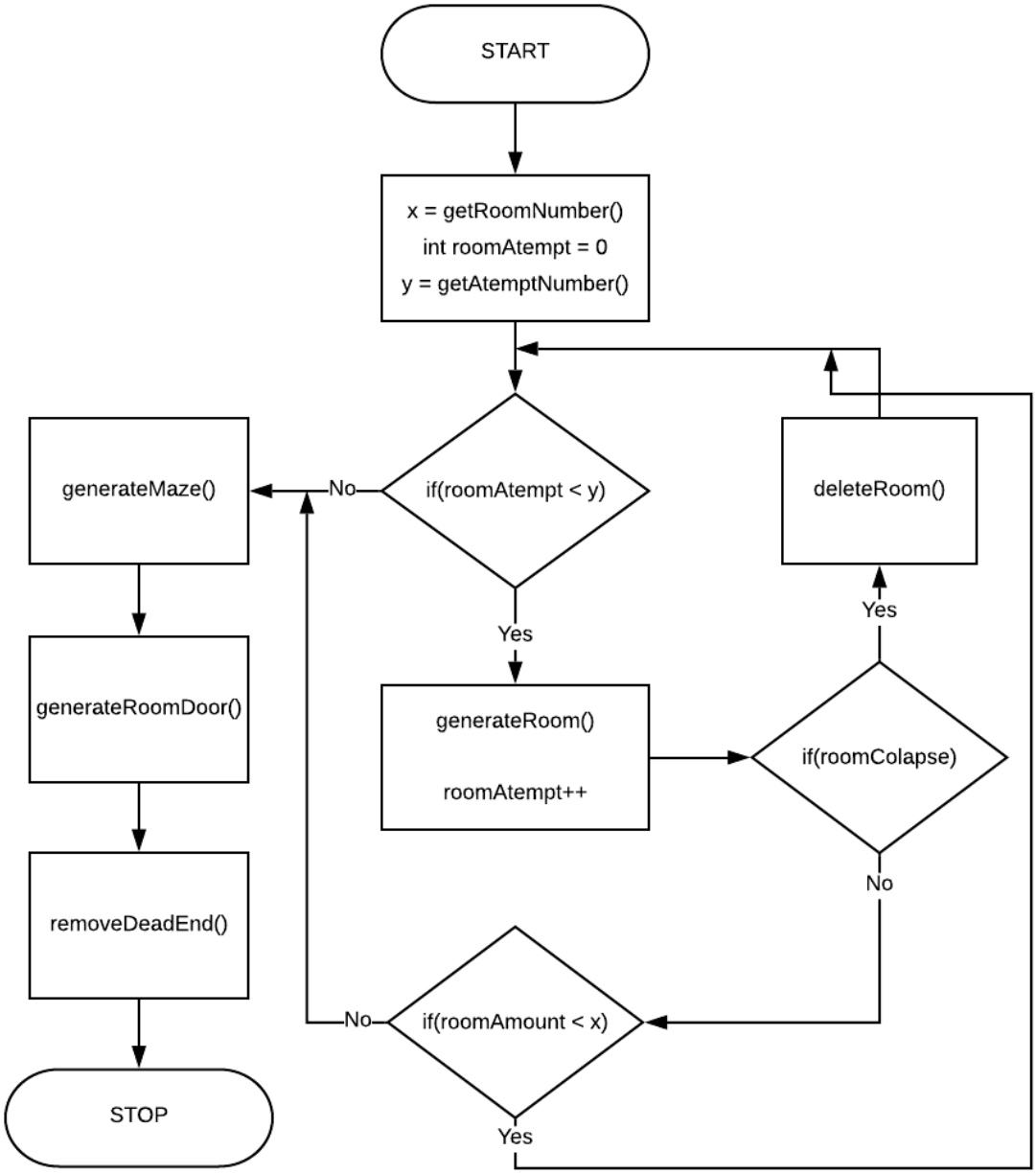
Inne założenia algorytmu odnoszą się do tego, jak ostatecznie będzie się prezentować wygenerowana mapa. Zaliczamy do nich:

- mapa to zbiór pokoi i prowadzących do nich korytarzy, nie jest to więc tradycyjny labirynt, w którym gracz przemiesza się z punktu A do punktu B tylko za pomocą wąskich ścieżek,
- labirynt nie może posiadać zamkniętych pokoi, co oznacza, że do każdego pomieszczenia gracz musi mieć dostęp,
- labirynt nie jest doskonały, to znaczy, że nie posiada jednego rozwiązania, nie istnieje jedna, konkretna ścieżka z punktu A do punktu B,
- labirynt posiada korytarze służące do przemieszczania się między pokojami, jednak niektóre z nich mogą stanowić ślepą uliczkę.

Użytkownik serwisu GitHub.com o nick'u *czyzby*. stworzył swój program zawierający różne generatory, a wśród nich znalazł się także generator lochów, działający na podstawie

zmodyfikowanego algorytmu Nystroma. Użytkownik ten przede wszystkim przetłumaczył algorytm z języka Dart na Java, udoskonalając funkcje odpowiedzialne za usuwanie ślepych ścieżek.

Działanie algorytmu zostanie zaprezentowane na poniższym schemacie blokowym.



Schemat 4.1 Schemat blokowy algorytmu generowania mapy

opracowanie własne

Z kolei najważniejszą modyfikacją stworzoną przez autora pracy jest dodanie nowej klasy *RoomInfo* odpowiedzialnej za przechowywanie informacji o wygenerowanych pokojach. W dalszej części pracy zostanie przedstawione wykorzystanie macierzy reprezentującej labirynt wygenerowany przez algorytm użytkownika *czyzby*.

4.1.2. Algorytm wyszukiwania drogi

Autor w swojej pracy korzysta z algorytmu A*, który po raz pierwszy został opisany w 1968 r. przez Petera Harta, Nilsa Nilssona oraz Bertrama Raphaela. Jest to algorytm heurystyczny, który służy do znajdywania najkrótszej ścieżki i jest on pełny i optymalny, co oznacza, że znajduje daną ścieżkę jeśli taka w ogóle istnieje i jest najkrótsza.

Działanie algorytmu opiera się na minimalizacji funkcji celu *f(x),* która zdefiniowana jest jako suma funkcji kosztu *(g(x)),* a także funkcji haurystycznej *(h(x)).*

$$f(x) = g(x) + h(x)$$

W każdym kolejnym kroku algorytm przedłuża utworzoną drogę o kolejny punkt, wybierając go tak, aby wartość funkcji *f* była jak najmniejsza. Funkcji *g(x)* definiuje koszt dojścia do danego punktu, z kolei funkcja *h(x)* oblicza koszt dotarcia z danego punktu do punktu docelowego.

Warunki jakie muszą zostać spełnione aby funkcja heurystyczna była prawidłowa to:

- Warunek dopuszczalności: $g(x) + h(x) \leq g(x_t)$
- Warunek monotoniczności: $g(x_j) + h(x_j) \geq g(x_i) + h(x_i)$
  gdzie $j > i$, a x_t oznacza punkt końcowy

Algorytm A* jest na tyle popularny, że autor w swojej pracy skorzystał z gotowej implementacji tego algorytmu, stworzonej przez użytkownika serwisu GitHub.com o nick'u *marcelo-s,* nie widząc potrzeby modyfikowania go - wolał skupić się na innych elementach pracy.

4.2. System graficznego generowania mapy

Zastosowany w pracy generator mapy składa się z dwóch klas - ***GenerateRandomMathDungeonSystem*** oraz ***GenerateMapTilesSystem***.

Pierwsza z nich - ***GenerateRandomMathDungeonSystem*** - korzysta z algorytmu przedstawionego w punkcie 4.1.1. prezentowanej pracy. Algorytm ten umożliwia dobór parametrów wejściowych tj.:

- *MaxRoomSize* i *MinRoomSize*, które odpowiadają za minimalną i maksymalną wysokość oraz szerokość wygenerowanych pokoi,
- *Tolerance* - parametr ten odpowiedzialny jest za zachowanie proporcji pomiędzy wysokością, a szerokością generowanych pokoi. Przykładowo generator w pierwszej kolejności losuje szerokość pokoju, a następnie jego wysokość z zachowaniem określonej przez nas tolerancji, np. T=5,
- *GeneratoinAttempts* - parametr ten odpowiada za gęstość wygenerowanych na mapie pokoi, określamy go, aby maksymalnie wykorzystać dostępną przestrzeń. Założeniem jest, aby wygenerowana mapa składała się w większości z pokoi, a korytarze służyły tylko jako ścieżka do przemieszczania się pomiędzy pomieszczeniami,
- *WindingChance* - parametr ten określa zawiłość korytarzy łączących pokoje, określenie go na poziomie 0 oznacza, że generowane połączenia będą jak najprostsze,
- *RandomConnectorChance* - parametr ten określony na poziomie 0 oznacza, że labirynt będzie doskonałym - będzie jedna możliwość dojścia z punktu A do punktu B. Natomiast jego zwiększenie dodaje szanse na to, że będzie więcej niż jedna możliwość dojścia z punktu A do punktu B,
- *DeadEndRemovalIterations* - parametr ten określa ilość iteracji potrzebnych do wyeliminowania długich, ślepych zaułków. W zależności od wielkości mapy musimy wziąć pod uwagę wartość, którą przypiszemy do tego parametru - im większa liczba, tym dłuższy czas generowania mapy, jednak za niska wartość może skutkować nadmierną ilością ślepych korytarzy.

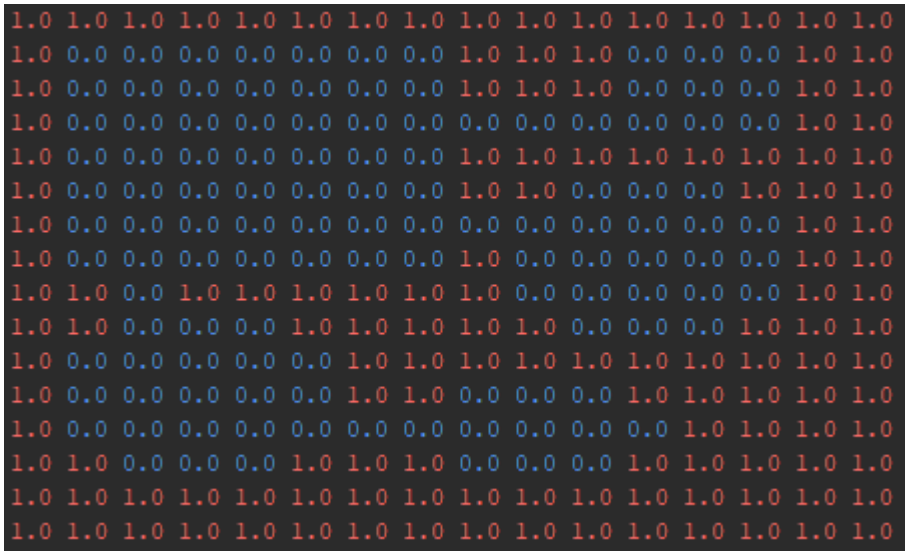
Po zdefiniowaniu tych parametrów, następuje wygenerowanie macierzy wynikowej składającej się z 0, które odpowiadają pokojom i korytarzom oraz 1, które wskazują gdzie będą puste, niezagospodarowane przestrzenie.

```
final DungeonGenerator dungeonGenerator = new DungeonGenerator();
dungeonGenerator.setMaxRoomSize(DungeonConfig.MAX_ROOM_SIZE);
dungeonGenerator.setTolerance(DungeonConfig.ROOM_SIZE_TOLERANCE);
dungeonGenerator.setMinRoomSize(DungeonConfig.MIN_ROOM_SIZE);
dungeonGenerator.setRoomGenerationAttempts(DungeonConfig.ATTEMPTS);
dungeonGenerator.setWindingChance(DungeonConfig.WIND_CHANCE);
dungeonGenerator.setRandomConnectorChance(DungeonConfig.CONNECT_CHANCE);
dungeonGenerator.setDeadEndRemovalIterations(DungeonConfig.DEAD_END);
dungeonGenerator.addRoomTypes(RoomType.DefaultRoomType.values());
dungeonGenerator.generate(dungeonLayout);
```

Rysunek 4.1 Konfiguracja generatora mapy

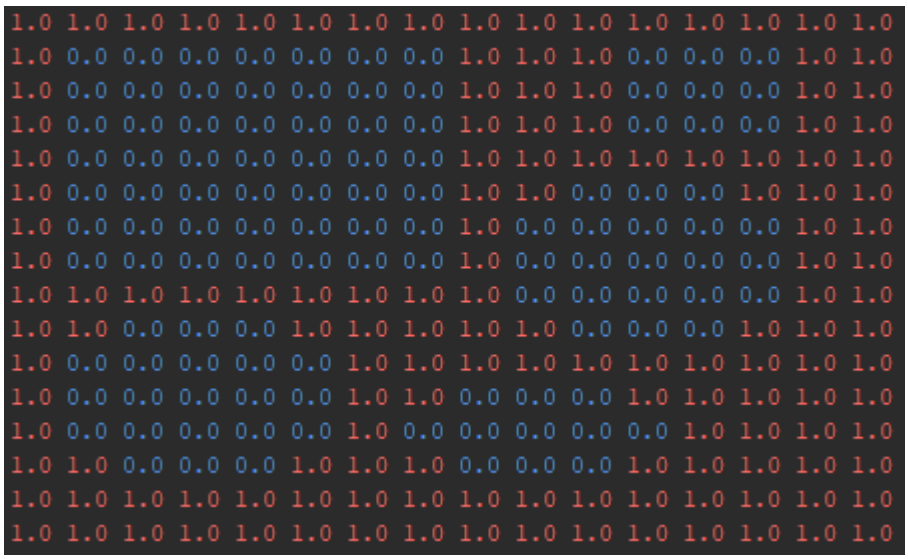
opracowanie własne

W tej klasie znajdują się także dwie funkcje. Pierwsza z nich ***printDungeon()*** odpowiedzialna jest za wyświetlanie kolumn i wierszy wygenerowanej wcześniej macierzy. Aby ułatwić proces debudowania, autor zastosował technikę kolorowania tekstu w konsoli, przypisując kolor czerwony 1 oraz kolor niebieski 0. Druga z funkcji - ***printRooms()*** - działa tak samo jak wskazana powyżej funkcja, z tą różnicą, że wyświetla same pokoje bez korytarzy w oparciu o autorską klasę ***RoomInfo***.



Rysunek 4.2 Efekt użycia funkcji *printDungeon()* w konsoli

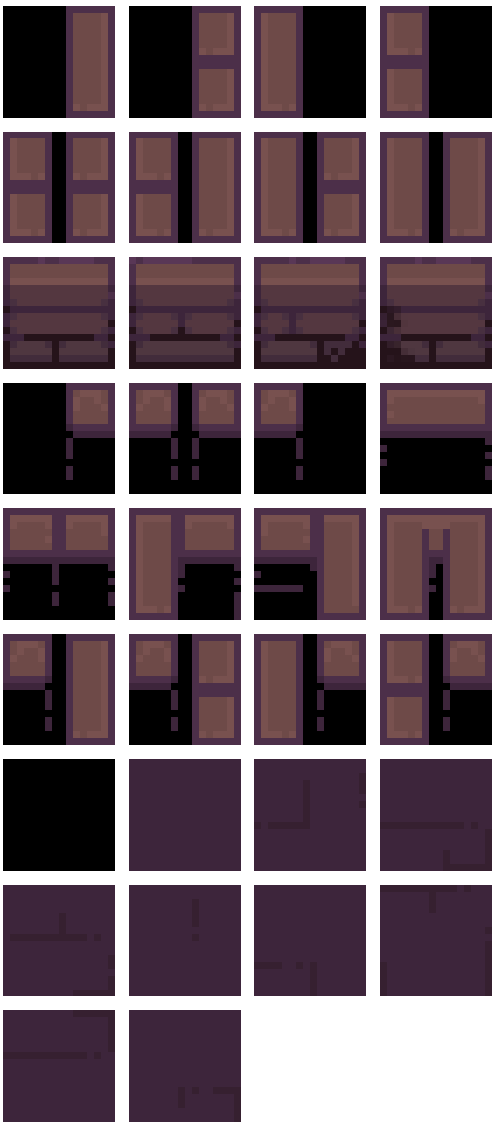
opracowanie własne



Rysunek 4.3 Efekt użycia funkcji *printRooms()* w konsoli

opracowanie własne

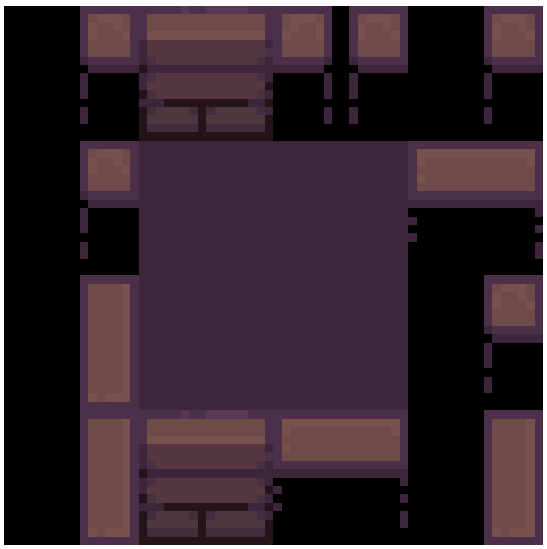
Druga z klas - *GenerateMapTilesSystem* - dziedziczy z poprzednio opisanej klasy i odpowiada za przekształcenie modelu matematycznego na wersję graficzną, którą widzi użytkownik gry. Pojedyncze elementy macierzy - 0 i 1 są losowo zastępowane jednym, kwadratowym elementem graficznym - zdefiniowanym jako ściana lub podłoga. Poniżej zaprezentowano wszystkie dostępne elementy graficzne zwane potocznie puzzlami.



Rysunek 4.4 Wszystkie dostępne puzzle

opracowanie własne

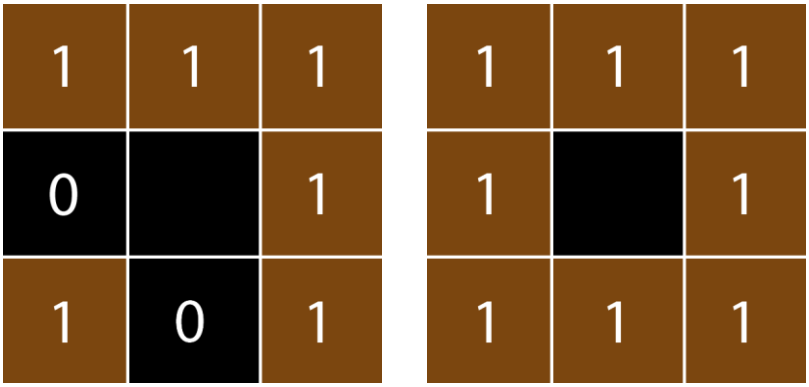
Takie rozwiązanie okazało się wadliwe z punktu widzenia użytkownika. O ile podłogi zawsze generowały się prawidłowo, o tyle w przypadku ścian, generator nie mógł określić kompatybilności pomiędzy poszczególnymi ścianami. Przykład złego ułożenia puzzli zaprezentowano poniżej.



Rysunek 4.5 Błędnie wygenerowana mapa

opracowanie własne

Ze względu na wadliwość tego rozwiązania, autor analizował inne możliwości. Mapa, która miała się generować skojarzyła mu się z puzzlami, dlatego też postanowił zdefiniować możliwe połączenia między klockami. Pierwszym etapem udoskonalania całego systemu było nałożenie wirtualnej siatki 3x3, która dzieliła każdego puzzla na 9 mniejszych elementów. W ten sposób mógł doprecyzować i zdefiniować jakie puzzle mogą ze sobą graniczyć.



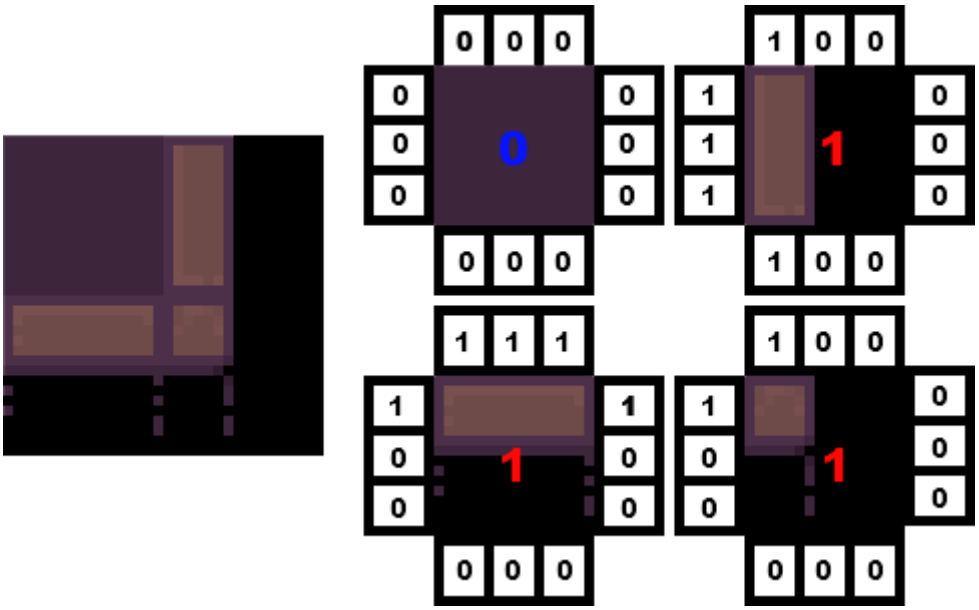
Rysunek 4.6 Schemat siatki 3x3

opracowanie własne

To rozwiązanie było poprawne, działało w przypadku łączenia prostych puzzli, jednakże problemem, który się pojawił było łączenie puzzli bardziej skomplikowanych, głównie w narożnikach. Pozostając przy takim rozwiązaniu, należałoby określić

szczegółowo, które puzzle - ściany do siebie pasują, a to wydłużyłoby ilość linijek kodu, co z kolei kłóciłoby się z założeniem niefunkcjonalnym dotyczącym nieskomplikowania kodu.

Ostatecznym pomysłem implementacji graficznego generatora mapy, była modyfikacja wirtualnej siatki zaprezentowanej powyżej. Nowy system zakładał nałożenie siatki po zewnętrznych stronach każdego puzzla, co pozwoliło na ich „podział" na 12, a nie 9 elementów. Schemat tego rozwiązania zaprezentowany został poniżej.



Rysunek 4.7 Przykładowy narożnik pokoju i schemat jego generowania

opracowanie własne

Takie rozwiązanie pozwoliło wyeliminować problem połączeń między puzzlami reprezentującymi ściany. Należy podkreślić, że podłogi w macierzy głównej zawsze zdefiniowane są jako 0 - kolor niebieski, a siatka ta została nałożona na nie tylko w celu zachowania spójności. Natomiast puzzle odpowiadające ścianom - oznaczone kolorem czerwonym zawierają 1 w miejscach, gdzie znajduje się ściana i 0 w miejscach, gdzie są puste przestrzenie.

Poniżej został zaprezentowany sposób definiowania dwóch ścian w klasie *TileList.*

```
/**
 *  Do konstruktora podajemy:
 *      4 parametry int[3], odpowiadające kolejno górnej, dolnej, lewej i prawej krawędzi
 *      rodzaj puzzla, 1 - ściana, 0 - podłoga
 *      nazwe tekstury
 * */
private final static BasicTile UP_WALL_1 =
        new BasicTile(new int[]{1, 1, 1}, new int[]{1, 1, 1},
                new int[]{1, 1, 1}, new int[]{1, 1, 1},
                 type: 1, RegionNames.WALL_9);

private final static BasicTile UP_WALL_2 =
        new BasicTile(new int[]{1, 1, 1}, new int[]{1, 1, 1},
                new int[]{1, 1, 1}, new int[]{1, 1, 1},
                 type: 1, RegionNames.WALL_10);
```

Rysunek 4.8 Deklaracja pojedynczego puzzla w programie zapisana za pomocą zer i jedynek.

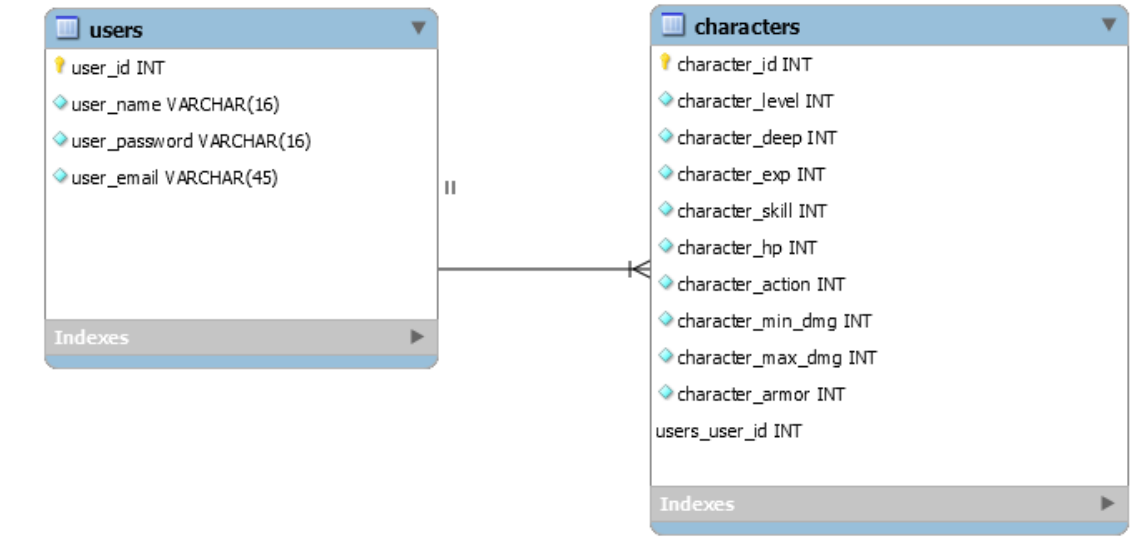
opracowanie własne

4.3. Baza danych

Stworzenie bazy danych to kolejny element prezentowanej pracy inżynierskiej, bez której gra nie mogłaby poprawnie działać. Baza danych została stworzona w celu przechowywania informacji o użytkownikach oraz postaciach.

Baza danych składa się z dwóch tabeli - users i characters, które są połączone relacją jeden do wielu. Pierwsza z nich przechowuje nazwę użytkownika, jego hasło oraz e-mail, a także unikalny numer każdego z graczy, który automatycznie jest generowany podczas tworzenia nowego użytkownika.

Druga z tabel przechowuje wszystkie informacje o postaciach - poziom postaci, głębokość lochu, punkty doświadczenia, dostępny punkty umiejętności, maksymalna ilość punktów życia, ilość pancerza, a także referencyjny zakres obrażeń. Każda z postaci ma przypisany swój unikalnym numer ID, a elementem łączącym użytkowników z postaciami jest ID gracza.



Rysunek 4.9 Diagram ERD bazy danych.

opracowanie własne

Na potrzeby pracy inżynierskiej w środowisku testowym został zainstalowany pakiet XAMMP, który posłużył jako lokalny serwer przechowujący bazę danych oraz interpreter skryptów PHP. Pakiet XAMMP posiada narzędzie phpMyAdmin, które ułatwia zarządzanie stworzoną bazą danych. W celu dodania tabel do bazy danych zostały wykorzystane skrypty wygenerowane w oparciu o model przedstawiony na schemacie powyżej.

```
CREATE TABLE IF NOT EXISTS `dungeon_data`.`users` (
  `user_id` INT NOT NULL AUTO_INCREMENT,
  `user_name` VARCHAR(16) NOT NULL,
  `user_password` VARCHAR(16) NOT NULL,
  `user_email` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE INDEX `name_UNIQUE` (`user_name` ASC),
  UNIQUE INDEX `user_id_UNIQUE` (`user_id` ASC),
  UNIQUE INDEX `user_email_UNIQUE` (`user_email` ASC))
ENGINE = InnoDB
```

Rysunek 4.10 Skrypt tworzący tabelę usere.

opracowanie własne

```
CREATE TABLE IF NOT EXISTS `dungeon_data`.`characters` (
  `character_id` INT NOT NULL AUTO_INCREMENT,
  `character_level` INT NOT NULL,
  `character_deep` INT NOT NULL,
  `character_exp` INT NOT NULL,
  `character_skill` INT NOT NULL,
  `character_hp` INT NOT NULL,
  `character_action` INT NOT NULL,
  `character_min_dmg` INT NOT NULL,
  `character_max_dmg` INT NOT NULL,
  `character_armor` INT NOT NULL,
  `users_user_id` INT NOT NULL,
  PRIMARY KEY (`character_id`, `users_user_id`),
  UNIQUE INDEX `characters_id_UNIQUE` (`character_id` ASC),
  INDEX `fk_characters_users_idx` (`users_user_id` ASC),
  CONSTRAINT `fk_characters_users`
    FOREIGN KEY (`users_user_id`)
    REFERENCES `dungeon_data`.`users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
```

Rysunek 4.11 Skrypt tworzący tabelę characters.

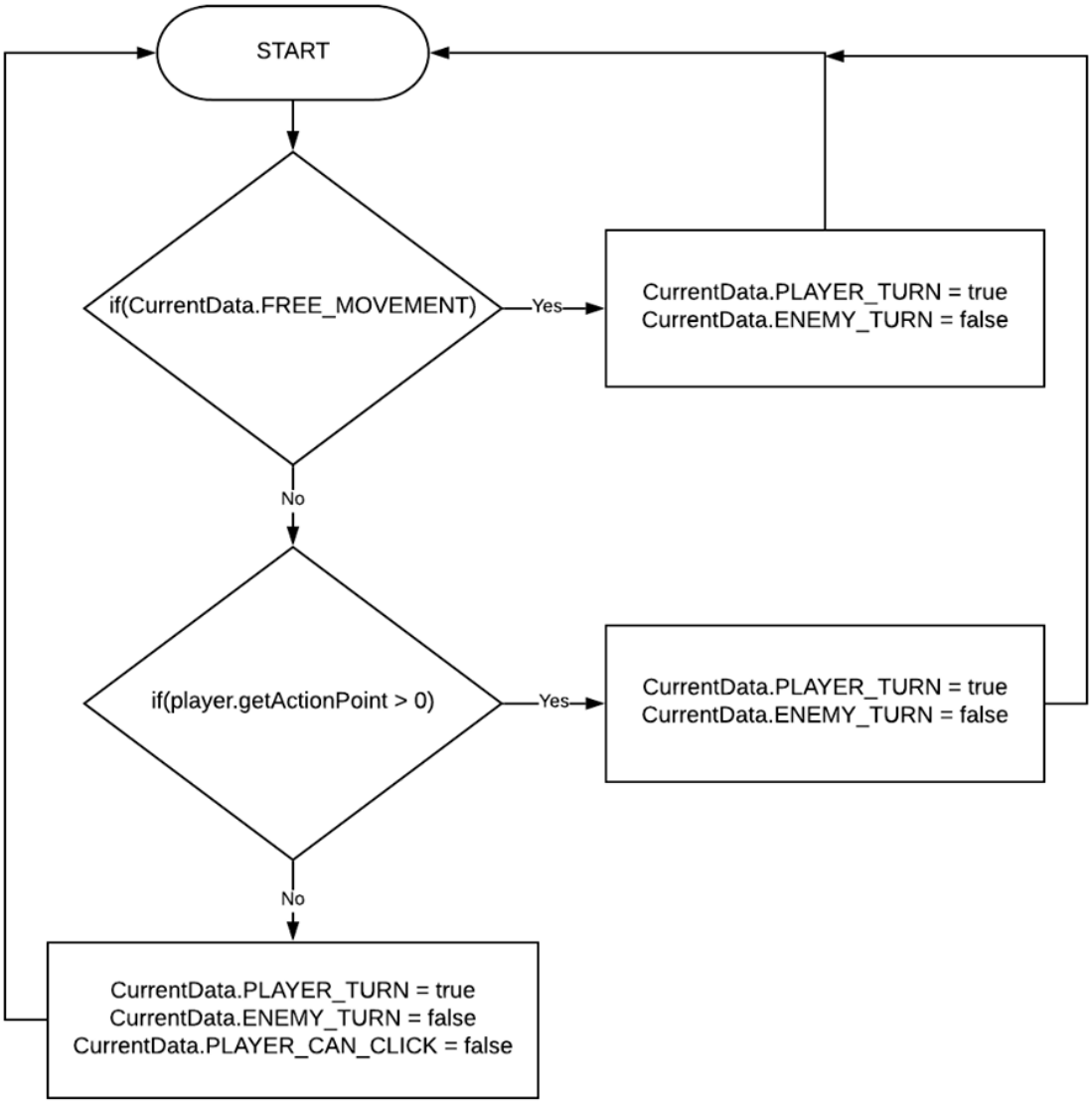
opracowanie własne

Program w celu komunikacji z bazą danych wysyła zapytanie pod wskazanych adres na serwerze lokalnym, gdzie znajduje się skrypt PHP zawierający potrzebne instrukcje. Na podstawie informacji zwrotnej program decyduje o kolejnych działaniach.

4.4. System turowy

System turowy zastosowany w aplikacji w głównej mierze działa w oparciu o klasę ***CheckTurnSystem,*** która zawiera kilka warunków logicznych, na podstawie których ustawiane są flagi - statyczne pola typu boolean z klasy ***CurrentData***, które decydują, czyja tura powinna nastąpić. Flagi te pozwalają na kontrolowanie kolejność aktywacji innych klas powiązanych z systemem turowym.

System ten powiązany jest z większością klas w programie, dlatego zostanie zaprezentowany schemat blokowy tylko klasy ***ChcekTurnSystem.***



Schemat 4.2 Schemat blokowy klasy *CheckTurnSystem.*

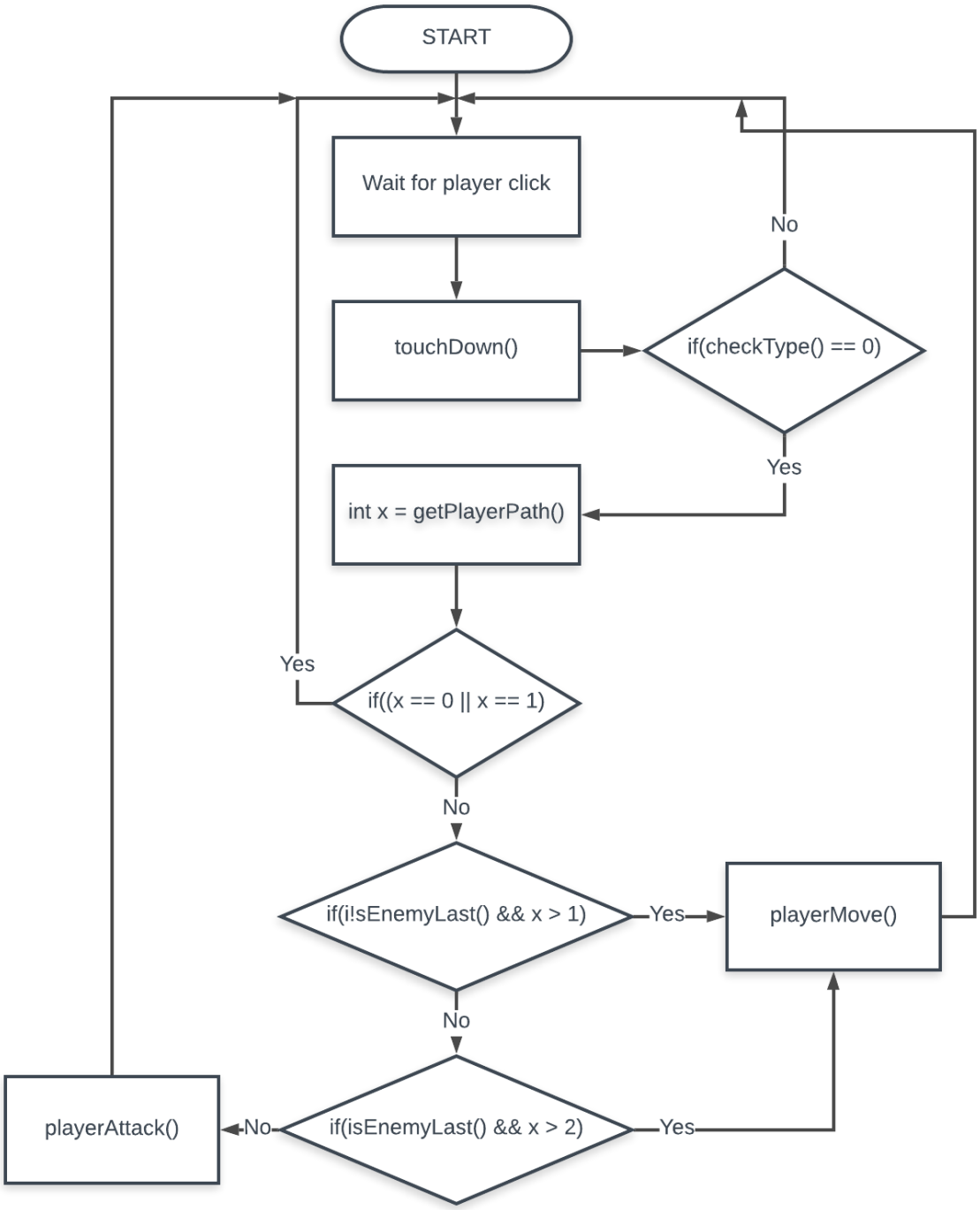
opracowanie własne

4.5. System sterowania

Omawiając system sterowania należy rozpocząć od opisu interakcji użytkownika aplikacji z urządzeniem - tzw. system wykrywania dotyku ***GameInputHandler.*** Klasa ta za pośrednictwem funkcji **touchDown()** odczytuje współrzędne kliknięcia, a następnie sprawdza czy fragment mapy, w który kliknęliśmy jest korytarzem, czy ścianą. Na potrzeby opisu systemu sterowania pominięte zostały kwestie utraty punktów akcji przez gracza.

- Jeśli dany punkt jest ścianą lub pozycją gracza, wówczas algorytm wraca na początek i czeka na kolejne kliknięcie użytkownika.

- Jeśli współrzędne kliknięcia wskazują na część korytarza, wtedy program zapisuje je w klasie ***CurrentData.*** Następnie za pomocą funkcji **getPlayerPath()** znajdującej się w klasie ***PathUtils,*** program sprawdza czy gracz może wykonać ruch, a także czy na wskazanej pozycji znajduje się przeciwnik. Jeśli na wskazanej pozycji znajduje się przeciwnik, dane o tym obiekcie zapisywane są w klasie ***CurrentData.***
    - jeśli na wskazanej pozycji znajdował się przeciwnik, a droga wyznaczona przez program wynosiła 2 pola - pozycja gracza graniczy z pozycją przeciwnika, wtedy następuje atak,
    - jeśli na wskazanej pozycji znajdował się przeciwnik, a droga wyznaczona przez program była większa niż 2, następuje ruch do pozycji graniczącej z pozycją przeciwnika,
    - jeśli na wskazanej pozycji nie znajdował się przeciwnik, a droga była większa od 1, wtedy następuje ruch we wskazane miejsce,
    - jeśli algorytm nie znalazł drogi - nic się nie dzieje i użytkownik ponownie musi kliknąć i wskazać pozycję,



Schemat 4.3 Schemat blokowy systemu sterowania

opracowanie własne

4.6. Kompilacja oraz konfiguracja aplikacji

Dla osób zainteresowanych modyfikacją parametrów aplikacji bądź jej rozwojem, koniecznym będzie zaimportowanie kodu źródłowego dostarczonego wraz prezentowanym projektem, za pomocą ***Gradle,*** czyli narzędzia służącego do budowania projektów. Najlepszym IDE do tego celu jest Android Studio.

Elementami projektu, które w łatwy sposób można poddać modyfikacji są: baza danych, tekstury oraz pliki konfiguracyjne.

W celu podpięcia własnej bazy danych należy:

- w klasie ***DataBase*** należy zmodyfikować pole **prefixHost,** podając w nim adres swojej domeny,
- pod wskazanym adresem w katalogu **dungeon_game** należy umieścić skrypty PHP dostarczone wraz z projektem,
- utworzyć bazę danych - najlepiej o nazwie **dungeon_data,** a następnie wykorzystać skrypty SQL **users** i **characters** w celu dodania wymaganych tabel do bazy.

W celu modyfikacji tekstur należy:

- umieścić nowe tekstury w katalogu **desktop\raw**\tekstury (przykładowy folder - należy utworzyć folder, do którego umieszczone zostaną nowe tekstury)
- zmienić parametry wejściowe funkcji **process()** w klasie ***AssetPacker*** podając nazwę utworzonego folderu. Następnie należy uruchomić tę klasę w celu utworzenia pliku z rozszerzeniem .atlas w katalogu z multimediami,
- w klasie ***AssetPaths*** utworzyć statyczną zmienną tekstową zawierającą ścieżkę dostępu do utworzonego pliku .atlas,
- w klasie ***RegionNames*** utworzyć statyczne zmienne tekstowe zawierające nazwy regionów w nowo utworzonej teksturze (nazwy regionów to nazwy tekstur przed połączeniem w jedną dużą teksturę),
- w klasie ***AssetDescriptors*** utworzyć statyczną zmienną **AssetDescriptor** podając jako jej parametr zmienną z klasy ***AssetPaths,***
- w miejscach, gdzie chcemy zmienić tekstury, zmienić parametr przechowujący informacje o starych teksturach na nowy z pomocą nowo utworzonego pola z klasy ***AssetDescriptors***.

5. Podsumowanie

Gry mobilne od zawsze znajdowały się w kręgu zainteresowań autora. Stojąc przed wyborem tematu do swojej pracy inżynierskiej, zdecydował się on na stworzenie swojej gry mobilnej, która działać będzie w środowisku Android.

Wykorzystanie w projekcie JAVA pozwoliło autorowi na doszkolenie się w tym konkretnym języku programowania, z którym autor planuje związać swoją przyszłość.

Pierwotnymi założeniami projektu było zaprogramowanie gry mobilnej z wykorzystaniem biblioteki LibGDX. Wszystkie zakładane przez autora elementy programu zostały stworzone - w mniejszym lub większym stopniu. Biorąc pod uwagę złożoność projektu w przyszłości w łatwy sposób można go rozbudować. W prezentowanym projekcie autor działał samodzielnie, dlatego też musiał korzystać z darmowych, legalnie dostępnych grafik, czy dźwięków. W sytuacji, kiedy w przyszłości znajdzie się zespół ludzi - grafików, dźwiękowców itd., którzy chcieliby rozwinąć program razem z autorem, wówczas jest ogromna szansa na to, aby gra wizualnie prezentowała się dużo lepiej.

Działania, które można podjąć w przyszłości, aby rozwinąć grę to:

- wprowadzenie systemu zadań i nagród,
- wprowadzenie ekwipunku,
- dodanie nowych klas postaci np. łuczników lub magów,
- zwiększenie poziomu zabezpieczeń bazy danych,
- synchronizacja aplikacji z platformą Google Games i Google Play.

W ramach podsumowania warto zaznaczyć, iż prezentowana gra nie opiera się na gotowym silniku, który zapewniłby wiele funkcji, które autor musiał samodzielnie zaprogramować. Jednym z takich elementów jest generator mapy. Korzystając z gotowych rozwiązań, autor musiałby po prostu dodać wybrane funkcje do programu, jednak w przypadku prezentowanej gry - system generowania mapy jest autorskim rozwiązaniem, a prace nad nim zajęły dobrych kilka tygodni. Na tym przykładzie można stwierdzić, że wybór biblioteki LibGDX nie był najlepszym rozwiązaniem, dlatego też w przyszłości kiedy autor po raz kolejny podejmie się pracy nad grą mobilną, wówczas na pewno będzie korzystał z gotowych silników udoskonalając wybrane funkcje.

Spis źródeł i literatury

Monografie i teksty zwarte

[1] Evans B.J., Flanagan D., *Java w pigułce 6 edycja,* Gliwice 2015, s. 19.

[2] Horstmann C.S., Cornell G., *Java. Podstawy wydanie IX*, Gliwice 2013, s. 30

[3] Szpyt K., *Obrót dobrami wirtualnymi w grach komputerowych. Studium cywilnoprawne*, s.2: https://www.ksiegarnia.beck.pl/media/product_custom_files/1/7/17055-obrot-dobrami-wirtualnymi-w-grach-komputerowych-studium-cywilnoprawne-kamil-szpyt-fragment.pdf (dostęp 16.06.2019).

[4] Wąsowska M., *Umowa o stworzenie gry komputerowej*, w: K. Grzybczyk (red.), Prawo w wirtualnych światach, Warszawa 2013, s. 36.

Netografia

[1] Algorytm A* https://github.com/marcelo-s/A-Star-Java-Implementation?fbclid=IwAR2WLmMU950JIKxw8PFKXz3YQuq9T1ozBTFk3sgp9UsaICWcw_h5H257Pas

[2] Algorytm generowania mapy stworzony przez Boba Nystroma http://journal.stuffwithstuff.com/2014/12/21/rooms-and-mazes/?fbclid=IwAR3woQTyC2OS7zGbMb9DssIcQOpa5Ey14-6ZMRH77hxa-QrPAyCMpg88viU

[3] Android - historia najpopularniejszego mobilnego systemu operacyjnego: https://www.komputerswiat.pl/artykuly/redakcyjne/android-historia-najpopularniejszego-mobilnego-systemu-operacyjnego/eplhj12 (dostęp 21.01.2019)

[4] Czym jest Android?: https://android.com.pl/artykuly/3608-czym-jest-android/ (dostęp 21.01.2019).

[5] Dowiedz się więcej o technologii Java: https://www.java.com/pl/about/ (dostęp 20.01.2019).

[6] Encyklopedia PWN, *gry komputerowe*: https://encyklopedia.pwn.pl/haslo/gry-komputerowe;3908365.html (dostęp 16.06.2019).
https://jaki-jezyk-programowania.pl/technologie/php/ (dostęp 13.10.2019)

[7] Jak prezentuje się rynek gier mobilnych [infografika]: https://komorkomania.pl/10641,jak-prezentuje-sie-rynek-gier-mobilnych-infografika (dostęp 20.01.2019)

[8] LibGDX.info&tutorial: https://libgdx.info/author/libgdxinfo/ (dostęp 22.01.2019).
[9] Przegląd modułów: https://github.com/libgdx/libgdx/wiki/Modules-overview (dostęp 22.01.2019)
[10[Ramy aplikacji: https://github.com/libgdx/libgdx/wiki/The-application-framework (dostęp 22.01.2019)
[11] Słownik Język Polskiego, *gra komputerowa:* https://sjp.pwn.pl/sjp/gra-komputerowa;2462665.html (dostęp 16.06.2019).
[12] To maszyna, która nie zwalnia tempa: https://techcity.pl/2018/02/rynek-gier-mobilnych.html (dostęp 20.01.2019)
[13] W 2018 r. gry mobilne wygenerują 51% przychodów na rynku gier: https://mobirank.pl/2018/05/02/w-2018-r-gry-mobilne-wygeneruja-51-proc-przychodow-na-rynku-gier/ (dostęp 20.01.2019)
[14] Wprowadzenie do LibGDX: https://libgdx.badlogicgames.com/documentation/ (dostęp 22.01.2019)
[15] Zmodyfikowana wersja algorytmu generowania mapy: https://github.com/czyzby/noise4j?fbclid=IwAR31Mc6aKAQxKE7Rz_VazsYeB3sYCjDpUak518bIOk19k5BpK5cQYDT8hAk

Spis elementów graficznych

Schematy

Tabele

Rysunki