

POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRYCZNY
INSTYTUT AUTOMATYKI I INŻYNIERII INFORMATYCZNEJ

PRACA DYPLOMOWA INŻYNIERSKA

**Wieloplatformowa gra z wykorzystaniem
generowania proceduralnego**

Autor:
Grzegorz OJDANA

Promotor:
Dr inż. Izabela JANICKA-LIPSKA

Poznań, styczeń 2015



Temat
pracy dyplomowej inżynierskiej
nr 029/Z3/2015

Politechnika Poznańska
Wydział Elektryczny
Instytut Automatyki i Inżynierii Informatycznej

Studia stacjonarne I stopnia
Kierunek: Informatyka
Specjalność: Bezpieczeństwo systemów informatycznych

Zobowiązuję/zobowiązujemy się samodzielnie wykonać pracę w zakresie wyspecyfikowanym poniżej. Wszystkie elementy (m.in. rysunki, tabele, cytaty, programy komputerowe, urządzenia itp.), które zostaną wykorzystane w pracy, a nie będą mojego/naszego autorstwa, będą w odpowiedni sposób zaznaczone i będzie podane źródło ich pochodzenia.

	Imię i nazwisko	Nr albumu	Data i podpis
Student:	Grzegorz OJDANA	106073	25.09.2014 Grzegorz Ojdana
Student:			

Tytuł pracy: Wieloplatformowa gra z wykorzystaniem generowania proceduralnego

Wersja angielska tytułu: Cross-platform game with the use of procedural generation

Dane wyjściowe: Literatura przedmiotu.

1. Technika generowania proceduralnego.

2. Opracowanie logiki gry.

Zakres pracy: 3. Implementacja gry z wykorzystaniem wybranych elementów generowania proceduralnego.

4. Testowanie i wnioski

Miejsce prowadzenia prac: Laboratorium IAiII

Termin oddania pracy: 31 stycznia 2015

Promotor: dr inż. Izabela Janicka-Lipska

Z-ca DYREKTORA INSTYTUTU
Automatyki i Inżynierii Informatycznej

dr Jerzy Bartoszek

Dyrektor Instytutu

PRODZIEKAN
Wydziału Elektrycznego PP

dr inż. Krzysztof Sroka

Dziekan

Poznań, 25 września 2014 r.

Miejscowość, data

Streszczenie

Celem niniejszej pracy jest przygotowanie gry komputerowej działającej na wielu platformach systemowych, wykorzystującej technikę generowania proceduralnego do programowego tworzenia typowych elementów świata dwuwymiarowej gry. Autor w swojej pracy przybliża czytelnikowi zagadnienie generowania proceduralnego oraz jego zastosowanie w grach.

Istotną częścią pracy jest opisanie technik i algorytmów zastosowanych do realizacji tworzenia proceduralnego wybranych elementów gry oraz nakreślenie i wyjaśnienie założeń, którymi kierował się autor. W pracy zawarto przedstawienie procesu powstawania mapy gry oraz pozostałych elementów tworzonych omawianym podejściem.

Część pracy stanowi omówienie logiki gry oraz sposobu implementacji najważniejszych elementów gry. Przedstawiono także metodykę testowania oraz uzyskane wyniki. Dokonano oceny zysku wynikającego z użycia techniki generowania proceduralnego w opracowanym projekcie gry komputerowej.

Abstract

The aim of this thesis is to prepare the computer game that runs on a variety of computer platforms and uses procedural content generation technique to programmatically create typical elements of two-dimensional game world. Author introduces the reader to procedural content generation topic and its application in games.

An important part of thesis is to describe the techniques and algorithms used to implement procedural creation of chosen game elements and to explain assumptions, which led author. The paper contains a representation of game map creation process and other elements created using discussed approach.

Part of this thesis is explanation of game logic and how important game aspects are implemented. Also presents the methodology and results of testing. An assessment of profit resulting from use procedural generation technique in game project is made.

Spis treści

Wstęp	vii
1 Charakterystyka projektu	1
1.1 Charakterystyka gry	1
1.2 Wieloplatformowość	3
1.3 Wykorzystane narzędzia i biblioteki	5
2 Generowanie proceduralne	7
2.1 Charakterystyka generowania proceduralnego	7
2.2 Generowanie proceduralne w grach komputerowych	10
2.3 Obszary wykorzystania generowania proceduralnego w projekcie	11
3 Implementacja elementów gry wykorzystujących generowanie proceduralne	13
3.1 Proces tworzenia świata gry	13
3.2 Mapa świata	15
3.3 Miasta	21
3.4 Drogi	26
3.5 Autobusy	30
4 Logika gry	31
4.1 Obsługa podróżnych	31
4.2 Zarządzanie przedsiębiorstwem	39
4.3 Zdarzenia losowe	41
5 Testy aplikacji	45
5.1 Metodyka testowa	45
5.2 Testy funkcjonalne	46
5.3 Testy wydajnościowe	52
5.4 Analiza rozmiaru aplikacji	55
6 Podsumowanie	57
7 Instrukcja dla użytkownika	61
7.1 Instalacja aplikacji	61
7.2 Główne ekrany gry	62

7.3 Sterowanie	64
Bibliografia	65
Załącznik	67

Wstęp

Uzasadnienie wyboru tematu

Generowanie proceduralne jest techniką szeroko wykorzystywaną w różnych formach rozrywki cyfrowej i treści tworzonych przy pomocy komputerów. Zaliczyć można do nich między innymi filmy, gry, muzykę czy grafikę komputerową. W grach komputerowych generowanie proceduralne wykorzystywane jest do programowego generowania elementów gry, którym twórcy chcą zapewnić unikalność i oryginalność oraz takich, których ręczne tworzenie byłoby zbyt czasochłonne lub zbyt kosztowne. Wybór tematu pracy podyktowany był chęcią zgłębienia tematyki generowania proceduralnego oraz praktycznego wykorzystania tej techniki.

Cel i zakres pracy

Celem niniejszej pracy jest opracowanie gry komputerowej wykorzystującej technikę generowania proceduralnego. Autor przybliża temat wykorzystania generowania proceduralnego w grach komputerowych oraz odnosi się do przykładów. Praca ta przedstawia także opis obszarów gry wykorzystujących generowanie proceduralne, omawia użyte w projekcie algorytmy oraz prezentuje przykładowe wyniki.

Tematem tworzonej gry jest prowadzenie przez gracza przedsiębiorstwa transportowego specjalizującego się w międzymiastowej komunikacji autobusowej. Istotą rozgrywki jest tworzenie linii autobusowych przechodzących przez wybrane przez gracza miasta w taki sposób, by pasażerowie byli w stanie dotrzeć do swoich miast docelowych oraz aby utrzymywane linie były rentowne.

Generowanie proceduralne zostało wykorzystane w grze do uzyskiwania dla każdej nowej rozgrywki odmiennego świata gry, ale spełniającego pewne założenia. Tworzone proceduralnie są elementy takie jak teren (mapa gry), uwzględniając typ terenu, zbiorniki wodne i zalesienie, a także miasta — ich cechy mające wpływ na rozgrywkę oraz elementy

czysto wizualne jak struktura miast i poszczególne budynki. Celem było uatrakcyjnienie rozgrywki, jako że dla każdej rozpoczynanej gry można uzyskać niepowtarzalny świat gry, mogący w wyniku różnić się poziomem trudności oraz odborem od innych światów z poprzednio przeprowadzonych rozgrywek.

Dodatkowo przyjętym założeniem było przygotowanie gry na wiele platform systemowych. Dostępne są narzędzia służące programistom do tworzenia oprogramowania działającego i wyglądającego tak samo lub podobnie na różnych systemach, jednocześnie minimalizując nakład pracy włożony w utrzymanie kompatybilności. Do przygotowania aplikacji wykorzystano bibliotekę programistyczną, której użycie ułatwiło przygotowanie gry działającej na wielu systemach i urządzeniach z niewielkim dodatkowym nakładem pracy związany ze szczegółami związanymi ze specyfiką poszczególnych platform.

Struktura pracy

Rozdział 1. przedstawia szerszy opis gry oraz przyjętych założeń projektowych. Wysegregowane zostały cechy gry i możliwości gracza. Przedstawione zostały narzędzia i biblioteki użyte do przygotowania aplikacji.

Rozdział 2. przybliża zagadnienie generowania proceduralnego oraz podaje przykłady wykorzystania tej techniki w rozrywce cyfrowej. Ponadto nawiąsa obszary wykorzystania generowania proceduralnego w projekcie.

Rozdział 3. stanowi szczegółowy opis implementacji elementów gry wykorzystujących generowanie proceduralne. Przedstawione zostały w nim użyte algorytmy oraz zastosowane rozwiązania. Opisy uzupełniają zrzuty ekranu obrazujące przykłady używanych wyników.

Rozdział 4. przybliża logikę i zasady gry oraz założenia i mechanizmy prowadzonej rozgrywki. Omawia sposób realizacji wybranych istotnych elementów gry.

Rozdział 5. skupia się na przedstawieniu metod testowania i wyników przeprowadzonych testów aplikacji.

Rozdział 6. zawiera podsumowanie wyników prac. Przedstawia ocenę dokonań i wnioski płynące z realizacji projektu, zawiera wyszczególnienie napotkanych problemów oraz wymienia korzyści uzyskane dzięki wykorzystaniu generowania proceduralnego.

Rozdział 7. stanowi krótką instrukcję dla użytkownika.

Do pracy dołączono załącznik w postaci płyty CD z elektroniczną wersją tekstu pracy, kodem źródłowym programu oraz plikami gotowych aplikacji możliwych do uruchomienia na systemach, dla których opracowano grę. Ponadto na płycie umieszczono także instrukcję do gry wraz z poradami dla gracza, w formie pliku HTML.

Rozdział 1

Charakterystyka projektu

Poniższy rozdział stanowi omówienie tematyki gry oraz możliwości gracza. Przybliżony zostaje sposób prezentacji użytkownikowi świata gry. Opisane jest także zagadnienie wieloplatformowości gry oraz wyszczególnione są technologie użyte w trakcie realizacji projektu.

1.1 Charakterystyka gry

Gra opracowana na potrzeby pracy jest jednoosobową grą strategiczną czasu rzeczywistego z elementami ekonomii. Tematem przewodnim gry jest zarządzanie przedsiębiorstwem autobusowym. Głównym zadaniem gracza jest tworzenie i utrzymywanie połączeń (linii) autobusowych łączących wybrane miasta. Wytyczonymi trasami poruszają się autobusy rozwożące pasażerów pomiędzy miastami.

1.1.1 Opis rozgrywki

Mapa świata przedstawia wygenerowany programowo teren, na którym umieszczonych jest pewna liczba miast. Miasta połączone są ze sobą siecią dróg, która tworzy spójny graf (z każdego miasta można dojechać do dowolnego innego miasta, być może przejeżdżając po drodze przez inne miasta). Tworzone przez gracza linie autobusowe prowadzą z wybranego miasta do innego (bądź tego samego — pętla), przechodząc przez kolejne miasta połączone bezpośrednio drogą.

Miejscowości różnią się między sobą wielkością (która ma wpływ na liczbę generowanych pasażerów wyjeżdżających z miasta oraz chcących przybyć do niego) oraz charakterystyką (miejscowości leżące nad jeziorem lub morzem albo w pobliżu gór są miejscowościami turystycznymi) i wyglądem budynków.

Wraz z upływem czasu gry w miastach pojawiają się potencjalni pasażerowie. Wybór celu podróży przez pasażera nie jest zupełnie losowy i zależy od kilku czynników, takich jak wielkość miasta docelowego (do większych miejscowości podąża więcej ludzi niż do mniejszych), odległość między miastami (pomiędzy sąsiednimi miejscowościami zaobserwować można większy ruch niż między miastami oddalonymi od siebie), typ miasta (kurorty letnie i zimowe odnotowują w sezonie zwiększoną liczbę przyjezdnych) oraz popularność. Zmiana popularności wraz z upływem czasu sprawia, że wytyczona intratna trasa będzie rentowna tylko przez pewien czas. Ponadto na wybór miast docelowych przez pasażerów wpływ mają występujące w grze wydarzenia losowe, np. rozgrywany w danym mieście ważny mecz piłkarski skutkuje wybrianiem się do tego miasta w krótkim czasie dużej liczby osób.

Aby wytyczyć trasę przebiegającą przez wybrane miasto, musi być w nim wybudowany przystanek. Początkowo gracz ma do dyspozycji kilka sąsiadujących ze sobą miast posiadających przystanki. W trakcie rozgrywki gracz powinien dążyć do rozszerzenia sieci komunikacyjnej poprzez budowanie przystanków w wybranych miastach oraz łączenie przystanków liniami autobusowymi tak, aby zaspokoić potrzeby podróżnych i w efekcie maksymalizować swoje zyski.

Gracz powinien zatem wyznaczać trasy w taki sposób, aby pasażerowie mogli w możliwie najkrótszym czasie dotrzeć do swojego miasta docelowego. Jeśli to możliwe, pasażerowie wybierają bezpośrednie połączenie, unikając przesiadek. Gdy pasażer nie może w żaden sposób dostać się do celu, traci cierpliwość, przestaje czekać i opuszcza przystanek (zostaje usunięty z mapy).

Po wytyczonych trasach poruszają się autobusy. Gracz dostaje początkowo kilka autobusów o słabych parametrach, a w razie potrzeby może dokupić kolejne. Autobusy różnią się charakterystyką (liczba miejsc, prędkość maksymalna, spalanie paliwa, komfort), która ma wpływ na ich cenę. Cechy te mają realny wpływ na rozgrywkę.

Przystanki mają ograniczoną pojemność. Na pełnych przystankach nie pojawią się nowi pasażerowie. Aby uniknąć tłoku na przystankach, gracz może zwiększać ich pojemność poprzez rozbudowę. Ważne jest dobre planowanie tras tworzonych linii i umiejętne dobieranie liczby autobusów przypisanych do nich.

Gracz odpowiedzialny jest za dbanie o dobrą kondycję finansową swojego przedsiębiorstwa. Główny przychód stanowi opłaty pasażerów za przejazd. Przewiezieni podróżni płacą za podróż po dotarciu do celu. Gracz nie ustala cen biletów, a wpływy z przewozu pasażerów zależą od szybkości transportu, komfortu przejazdu, długości trasy podróży oraz porównania faktycznie przebytej odległości z optymalną. Stan konta pomniejszany jest o koszty utrzymania przystanków i autobusów. Zarobione pieniądze można wydać na

powiększenie floty oraz zasięgu, a w celu oszczędności można sprzedać autobus. Możliwa jest także naprawa pojazdu (jako że stan autobusów pogarsza się z czasem i pokonaną odległością, zwiększaając ryzyko wypadku).

Gra nie posiada warunku zwycięstwa ani nie jest ograniczona czasowo — gracz sam ustala sobie cel, a może być nim objęcie w możliwie krótkim czasie całego obszaru mapy rentownymi liniami lub zdobycie jak największej sumy pieniędzy w określonym czasie. Warunkiem porażki jest przekroczenie określonego minimalnego limitu pieniędzy.

Aby dać graczowi większą kontrolę nad prowadzoną rozgrywką, umożliwiono sterowanie tempem upływu czasu. Użytkownik może zwiększać i zmniejszać szybkość, z jaką płynie czas w grze, a także wstrzymywać czas. Pozwala to graczowi dostosować dynamikę gry do panującej sytuacji.

Szczegółowy opis wybranych mechanizmów gry (wraz z wyjaśnieniem sposobu implementacji) wspomnianych w tym podrozdziale można znaleźć w rozdziale 4.

1.1.2 Prezentacja rozgrywki

Świat gry przedstawiony jest „z góry” (*z lotu ptaka*). Grafika w grze jest dwuwymiarowa. Zdecydowano się na taki sposób prezentacji świata gry głównie ze względu na specyfikę docelowych platform systemowych. Rysowanie trójwymiarowej, szczegółowej grafiki mogłoby być zbyt wymagające dla słabszych urządzeń mobilnych oraz przeglądarek internetowych. Poza tym przygotowanie bardziej realistycznej grafiki byłoby dla autora zbyt czasochłonne.

Teren świata gry reprezentowany jest w postaci macierzy kwadratowych pól tzw. kafli (*tiles*), której wymiary zależą od wybranego przez gracza rozmiaru świata gry. Taki sposób reprezentacji terenu (wraz z widokiem „z góry” lub w postaci rzutu izometrycznego) jest często wykorzystywany w grach strategicznych (serie *Civilization*, *SimCity*, *Heroes of Might and Magic*, *Age of Empires*), ale nie tylko (*Grand Theft Auto* cz. I i II, seria *Final Fantasy*, *The Sims*). Wybór sposobu reprezentacji terenu podykowany był łatwością z jaką można generować taki teren proceduralnie oraz faktem, iż część użytych w grze algorytmów przystosowanych jest do terenu reprezentowanego w postaci kafli (np. algorytm *A** użyty do wyznaczania optymalnych dróg między miastami).

1.2 Wieloplatformowość

Jednym z założeń projektu było przygotowanie gry na wiele platform systemowych. Współcześnie wiele aplikacji (nie tylko gier) przygotowywanych jest dla więcej niż jednego

systemu operacyjnego, co pozwala dotrzeć do większej grupy odbiorców. Do przygotowania aplikacji wieloplatformowej dobrym wyborem jest język Java, ponieważ dla większości liczących się systemów operacyjnych, zarówno przeznaczonych na komputery osobiste, jak i na urządzenia mobilne, istnieje środowisko uruchomieniowe dla aplikacji napisanych w tym języku¹. Z uwagi na powyższe cechy wybrano ten język do implementacji gry.

Aby ułatwić sobie opracowanie wieloplatformowej aplikacji zdecydowano się wykorzystać bibliotekę programistyczną przeznaczoną specjalnie do pisania gier działających na wielu systemach operacyjnych. Użycie takiej biblioteki całkowicie lub częściowo zwalnia programistę z dodatkowej pracy związanej z przygotowaniem kodu pod daną platformę, jako że szczegóły implementacyjne są przed nim ukryte. Ponadto część pracy zostaje zautomatyzowana poprzez dostarczenie predefiniowanych projektów do rozbudowy oraz skryptów komplikacji na dany system. Biblioteką, którą użyto w projekcie, jest otwartoźródłowa biblioteka *libGDX*², przeznaczona dla języka Java. Pozwala ona opracować grę działającą na komputerach osobistych (systemy z rodzin *Windows*, *Mac OS* i *Linux*), urządzeniach mobilnych (działających pod kontrolą systemów *Android*, *iOS* lub *BlackBerry*) oraz w dowolnej przeglądarce internetowej obsługującej technologię *HTML5*.

1.2.1 Przygotowane wersje gry

Grę przygotowano w następujących wariantach:

- w postaci pliku wykonywalnego dla komputerów osobistych z zainstalowanym środowiskiem uruchomieniowym Java w wersji 7 lub wyższej,
- w postaci aplikacji dla urządzeń mobilnych działających pod kontrolą systemu *Android* w wersji 2.2 (*Froyo*) lub wyższej,
- w postaci gry „przeglądarkowej” w technologii *HTML5* wykorzystującej *JavaScript* oraz *WebGL* (dzięki translacji kodu z języka Java do *JavaScript* przy użyciu technologii *GWT* (*Google Web Toolkit*)³), co zapewnia *libGDX*.

Gra została umieszczona w Internecie pod adresem www.g-ojdana.vxm.pl/busgame. Na stronie dostępna jest gra online (w przeglądarce internetowej) oraz instrukcja do gry w formie wypisanych cech poszczególnych elementów świata gry i podpowiedzi dla gracza. Omówione zostały również najważniejsze ekranы gry i elementy interfejsu użytkownika.

¹<https://www.java.com/pl/about/>

²<http://libgdx.badlogicgames.com/>

³<http://www.gwtproject.org/>

1.3 Wykorzystane narzędzia i biblioteki

Grę napisano w języku Java (w wersji 7) przy użyciu biblioteki libGDX, których wybór uzasadniono w poprzednim podrozdziale.

Środowiskiem programistycznym, w którym przygotowano aplikację, jest Eclipse⁴ w wersji 4.4 (*Luna*). Do przygotowania wersji aplikacji na system Android, niezbędne było użycie wtyczki ADT (*Android Development Tools*⁵) dla środowiska Eclipse. Ponadto w celu automatyzacji pewnych czynności związanych z komplikacją projektu skorzystano z narzędzia Gradle⁶.

W trakcie pracy nad projektem korzystano z otwartego systemu kontroli wersji git⁷. Repozytorium przechowywano w serwisie Bitbucket⁸, umożliwiającym nieodpłatne utrzymywanie nieograniczonej liczby prywatnych repozytoriów.

Diagramy UML załączone do pracy opracowano w programie Visual Paradigm for UML Community Edition⁹ w wersji 10.2, udostępnianym na licencji pozwalającej na darmowe wykorzystanie w niekomercyjnych projektach. Wykresy opracowano w darmowym narzędziu służącym do ich tworzenia, dostępnym w formie aplikacji internetowej, meta-chart.com¹⁰.

Na grafikę gry składają się tekstury i ikony przygotowane własnoręcznie lub pobrane z Internetu z serwisów udostępniających zasoby dla twórców gier na wolnych licencjach (freeware lub Creative Commons 3.0¹¹). Serwisy, z których wykorzystano grafiki, to: opengameart.org¹², icons8.com¹³ oraz fatcow.com¹⁴. Własne grafiki przygotowano w programach GIMP 2.8¹⁵ oraz Inkscape¹⁶, które dostępne są na licencji GNU.

⁴<https://www.eclipse.org/>

⁵<http://developer.android.com/tools/sdk/eclipse-adt.html>

⁶<http://www.gradle.org/overview>

⁷<http://git-scm.com/>

⁸<https://bitbucket.org/>

⁹<http://www.visual-paradigm.com/download/community.jsp>

¹⁰<http://www.meta-chart.com/>

¹¹<http://creativecommons.org/licenses/by/3.0/pl/legalcode>

¹²<http://opengameart.org/>

¹³<http://icons8.com>

¹⁴<http://www.fatcow.com/free-icons>

¹⁵<http://www.gimp.org/>

¹⁶<https://inkscape.org/en/>

Rozdział 2

Generowanie proceduralne

Generowanie proceduralne jest techniką szeroko wykorzystywaną w rozrywce cyfrowej, a najintensywniej w grach komputerowych. Algorytmy proceduralne posiadają szereg zalet i wad, które przesądzają o użyteczności tej techniki w danym zastosowaniu. Opracowany projekt prezentuje możliwości generowania proceduralnego na przykładzie wybranych elementów świata gry.

2.1 Charakterystyka generowania proceduralnego

Technika generowania proceduralnego obejmuje algorytmy i programy, które na podstawie ustalonych parametrów i przy ograniczonym udziale użytkownika dają w wyniku działania pewną treść (ang. *content*). Innymi słowy, generowanie proceduralne jest techniką pozwalającą tworzyć treść za pomocą komputerów, używając algorytmów opartych o wzory matematyczne, zastępując lub uzupełniając ludzką pracę [9, rozdz. 1].

2.1.1 Cechy algorytmów generowania proceduralnego

Algorytmy generowania proceduralnego działają na pewnym poziomie abstrakcji. Oznacza to, iż zamiast „ręcznie” określać szczegóły jakiegoś elementu, proces tworzenia tego elementu uogólniany jest do postaci algorytmu, który stworzy te detale w momencie wywołania procedury. Używając takiego podejścia, możliwe jest tworzenie zaawansowanych modeli lub tekstur bazując na prostszych, w szczególności na prymitywach.

Wałąną cechą algorytmów generowania proceduralnego jest możliwość kontroli uzyskiwanych rezultatów poprzez parametry wejściowe oraz przez ustalone w algorytmie ograniczenia [2]. Twórca procedury powinien określić zakres dopuszczalnych wartości cech generowanego elementu, jako że zwykle nie tworzy się w ten sposób całkowicie losowych treści. Jako przykład można wyobrazić sobie generator poziomu dla gry z gatunku FPS. Dobrze

przygotowana mapa powinna mieć odpowiednio zbilansowaną liczbę oraz umiejscowienie wygenerowanych przeciwników oraz przeszkode. Umiejscowienie ich na mapie całkowicie losowo z pewnością doprowadzi do generowania dużej liczby map niegrywalnych, w których bohater nie będzie miał realnych szans na pokonanie przeciwników. Ograniczenie liczby przeciwników generowanych na danym obszarze oraz rozrzucenie małych grup przeciwników po mapie poprawi jakość generowanych poziomów. Gdyby sumować poziom zagrożenia grupy przeciwników generowanych w pewnym obszarze mapy i ograniczyć go maksymalną wartością, można by taki parametr uznać za poziom trudności poziomu i uznać go za parametr wejściowy generatora. Szczególnym przypadkiem parametru procedury generowania jest *ziarno losowości* (ang. *seed*) używane przez generator liczb pseudolosowych stosowany w procesie generowania.

Wyniki generowania dla różnych parametrów wejściowych powinny zauważalnie różnić się od siebie. Generator terenu, który jedyne co zmienia przy kolejnych wywołaniach to położenie jednego drzewa na mapie, daje mało ciekawy efekt i zbyt przewidywalny dla użytkownika wynik. Nie można jednak przesadzić w drugą stronę — generowany rezultat nie może sprawiać wrażenia całkowitej losowości. Tworzony proceduralnie poziom gry musi spełniać założone przez twórców gry warunki, by był grywalny. Pewne elementy mogą być stałe lub być podobne w kolejnych rezultatach wywołania procedury generującej. Ocena ekspresywności może okazać się zadaniem nietrywialnym, tym bardziej, iż w większości przypadków treść tworzona proceduralnie ma z założenia nie dawać po sobie poznać, że nie została stworzona bezpośrednio przez człowieka.

Wiele z algorytmów generowania proceduralnego można nazwać elastycznymi, ponieważ używając ich, można uzyskać efekty o różnorodnym zastosowaniu. Dla przykładu, *szum Perlina* często używany jest do generowania tekstur skał, gór, ognia, marmuru, chmur, lawy, plam oleju [8], a także do generowania mapy wysokości w grach czy do podziału świata gry na biomy. Tak szerokie spektrum zastosowań powoduje, iż jest to algorytm powszechnie stosowany w aplikacjach korzystających z techniki generowania proceduralnego. Zatem ten sam generator może dawać zarówno praktyczne, jak i czysto artystyczne rezultaty, zależnie od użytych parametrów i implementacji.

Generatory mogą cechować się różną szybkością działania, zależnie od zastosowania i sytuacji. Jeśli generator używany w grze ma być stosowany w trakcie toczenia rozgrywki, musi działać wystarczająco szybko, aby użytkownik nie odczuwał spowolnienia programu. Z kolei od generatora wspomagającego tworzenie treści, np. w edytorze gry lub w programie graficznym (efekt szumu, efekt chmur lub fraktal — efekty dostępne w wielu programach do obróbki graficznej), zwykle nie wymaga się aż tak szybkiego działania, co pozwala wykorzystać inne, być może dokładniejsze i lepsze do danego zastosowania algorytmy.

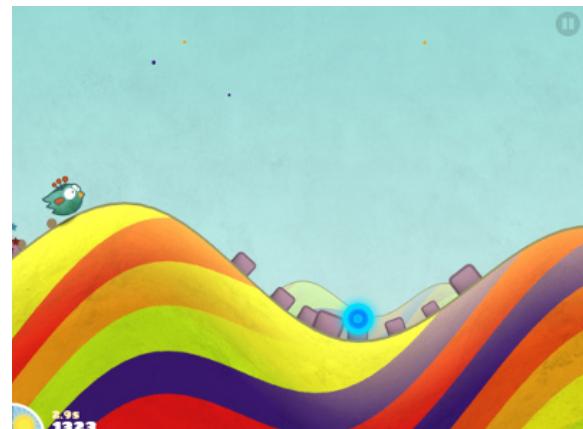
2.1.2 Zalety i wady podejścia proceduralnego

Podejście proceduralne stosowane jest często w celu oszczędności. Generowanie treści przez komputer za pomocą funkcji może zastąpić przygotowywanie tych samych elementów przez człowieka, co pozwala na zredukowanie liczby osób pracujących przy projekcie bądź zaoszczędzenie czasu twórców. Zależy to jednak od specyfiki tworzonego elementu. Może się okazać, że czas poświęcony na opracowanie procedury będzie niewspółmierny do otrzymywanych rezultatów i efektywniej byłoby przygotować dany element ręcznie.

Kolejną oszczędnością jest zmniejszenie rozmiaru opracowywanej aplikacji dzięki generowaniu zasobów (np. tekstur czy siatek modeli) poprzez program zamiast dołączania tych elementów do dystrybuowanej aplikacji. Kod procedury generującej teksturę zajmie zdecydowanie mniej pamięci niż plik graficzny.

Inną zaletą podejścia proceduralnego jest odtwarzalność uzyskiwanych rezultatów. Stosując te same parametry dla funkcji generowania, uzyskujemy w rezultacie zawsze ten sam wynik. Zatem znając wynik dla pewnych ustalonych danych wejściowych, możliwe jest uzyskanie w dowolnym momencie spodziewanego rezultatu. Może to być wykorzystane w grach do generowania poziomu gry zamiast przechowywania modelu poziomu w danych aplikacji.

Reprezentacja proceduralna nie posiada stałej rozdzielczości, tzn. jest nieograniczona w wymiarach generowanego wyniku. Tworzona proceduralnie tekstura może mieć dowolny poziom szczegółowości, a tworzony poziom gry mieć skalowalny poziom detali. Możliwe jest generowanie nieskończonego świata, który może być przedmierzany przez bohatera w wybranym kierunku bez końca. W takim wypadku program tworzy kolejne części świata i rozszerza mapę wtedy, gdy bohater zbliża się do jej dotychczasowego krańca.



RYSUNEK 2.1: Zrzut ekranu z gry mobilnej *Tiny Wings*, która używa techniki proceduralnej do tworzenia tekstur terenu.¹

Wynik generowania proceduralnego może być niespodzianką dla użytkownika. Bez znajomości mechanizmu generującego kontrola wyniku poprzez parametry może być trudna. Cechą ta, zależnie od sytuacji, może być wadą bądź zaletą. Generowanie kolejnych niespodziewanych, być może abstrakcyjnych rezultatów, może zaowocować inspiracją dla grafika

¹ Źródło obrazu: <http://www.apfellike.com/wp-content/uploads/2014/08/>

lub projektanta poziomów gry. Z drugiej strony, nieprzewidywalność wyniku znacząco utrudnia debugowanie procedury generującej.

Czas tworzenia elementu metodą proceduralną przeważnie jest dłuższy od czasu dostępu do zasobu zapisanego w pliku. Wymusza to na programiście szczególną uwagę na złożoność czasową tworzonej procedury generującej.

2.2 Generowanie proceduralne w grach komputerowych

W grach komputerowych do elementów (treści) możliwych do wygenerowania proceduralnie zaliczyć można niemal wszystkie elementy dowolnej gry:

- **Mapę świata gry** rozumianą jako teren po którym porusza się bohater. Tworzone proceduralnie mogą tu być takie szczegóły, jak: ukształtowanie terenu, pokrycie strefami klimatycznymi i biomami oraz roślinnością i akwenami wodnymi, rozmieszczenie przeszkód i wyposażenia możliwego do odnalezienia przez bohatera, położenie budynków na mapie, a także rozmieszczenie przeciwników czy postaci niegrywalnych (*NPC*). Ważnymi przykładami gier, które używały podejścia proceduralnego do generowania mapy świata gry, są *Minecraft*², *Dwarf Fortress*³, *Spore*⁴ oraz *Elite*⁵. W *Dwarf Fortress* generowana jest nawet historia świata, którą gracz może przeczytać, aby lepiej wczuć się w klimat gry.
- **Poziomy gry**, a zatem ich struktura, rozmieszczenie oraz liczba przeszkód i przeciwników, a także elementów wyposażenia i innych przedmiotów do znalezienia itp. Jako przykłady gier korzystających z takiego rozwiązania można wymienić takie tytuły jak: *Terraria*⁶, *Diablo*⁷ (cz. 1 i 2), *Rogue*⁸ (1980), *Cywylizacja IV*⁹ czy *Spelunky*¹⁰.
- **Elementy świata gry** do których zaliczyć można m.in. pojazdy, budynki (wygląd zewnętrzny, wnętrze), broń, przedmioty wyposażenia postaci. W tym zakresie z generowania proceduralnego korzystała gra *Galactic Arms Race*¹¹.
- **Postaci** — ich wygląd, cechy charakteru, wyposażenie, atrybuty, imię, historię itd. Jako przykłady wymienić można wspomniane wcześniej gry *Dwarf Fortress* i *Spore*.

²<https://minecraft.net/>

³<http://www.bay12games.com/dwarves/>

⁴<http://www.spore.com/>

⁵<http://www.iancgbell.clara.net/elite/>

⁶<http://terraria.org/>

⁷<http://eu.blizzard.com/pl-pl/games/legacy/>

⁸<https://yukkurigames.com/pphs/>

⁹<http://www.2kgames.com/civ4/home.htm>

¹⁰<http://spelunkeworld.com/>

¹¹<http://galacticarmsrace.blogspot.com/>

- **Elementy fabularne**, włączając w to misje i zadania dla bohatera (ang. *quests*). Z tego rozwiązania korzystają głównie gry typu cRPG, m.in. gry z serii *The Elder Scrolls*¹².
- **Siatki modeli trójwymiarowych**, a także **animacje** modeli, z czego korzysta przytoczona wcześniej gra *Spore* (zrzut ekranu przedstawiono na rysunku 2.2).
- **Tekstury**, zarówno wykorzystywane w grach 2D, jak i nakładane na modele w grach 3D. Generowania tekstur używa m.in. gry mobilna *Tiny Wings*¹³, której zrzut ekranu prezentuje rys. 2.1.
- **Dźwięki i muzykę** w grze (*Spore*).

2.3 Obszary wykorzystania generowania proceduralnego w projekcie

Wykorzystanie generowania proceduralnego w projekcie miało uczynić grę atrakcyjniejszą. W większości gier możliwe jest prowadzenie rozgrywki na mapach (poziomach) opracowanych przez twórców gry, co przy niedużej ich liczbie i niewielkim zróżnicowaniu, może prowadzić do szybkiego znudzenia gracza. Generowanie świata gry wprowadza do rozgrywki element nieprzewidywalności.

Poniżej wyszczególniono wszystkie elementy mapy gry tworzone przy użyciu generowania proceduralnego podczas przygotowywania nowej gry:

- teren:
 - typ terenu na danym obszarze mapy: ląd, woda oraz uszczegółowienie: nizina, wyżyna, obszar górski, jezioro bądź morze,
 - rzeki,
 - lasy — rozmieszczenie obszarów leśnych oraz pojedynczych drzew na mapie,



RYSUNEK 2.2: Wiele elementów w grze *Spore* powstawało z wykorzystaniem techniki generowania proceduralnego, m.in. modele i animacje postaci oraz tekstury.¹⁴

¹²<http://www.elderscrolls.com/>

¹³<http://www.andreasilliger.com/index.php>

¹⁴Źródło obrazu: <http://www.spore.com/what/spore>

- dobór tekstur terenu;
- miasta:
 - rozmieszczenie miast na mapie,
 - rozmiar oraz parametry miast,
 - rozmieszczenie budynków w miastach oraz ich bryły i reprezentacja graficzna;
- drogi.

Ponadto w czasie przygotowywania nowej gry generowane są także autobusy (marka i model, parametry pojazdu).

Do generowania świata gry używane są zarówno znane algorytmy generowania (np. *Simplex Noise*), jak i własne bądź zmodyfikowane istniejące algorytmy. Część z nich wykorzystuje generator liczb pseudolosowych. Na potrzeby gry wystarczający jest standaryzowany generator dostępny w języku Java (klasa `Random` zawarta w standardowym pakiecie `java.util.Random`). Niektóre z opisanych w dalszej części pracy algorytmów potrzebują do swego działania rezultatów innego algorytmu (np. aby wygenerować rzeki i wyznaczyć akweny, trzeba mieć wygenerowaną mapę wysokości terenu). Proces powstawania mapy świata gry przedstawiono szczegółowo w następnym rozdziale.

Rozdział 3

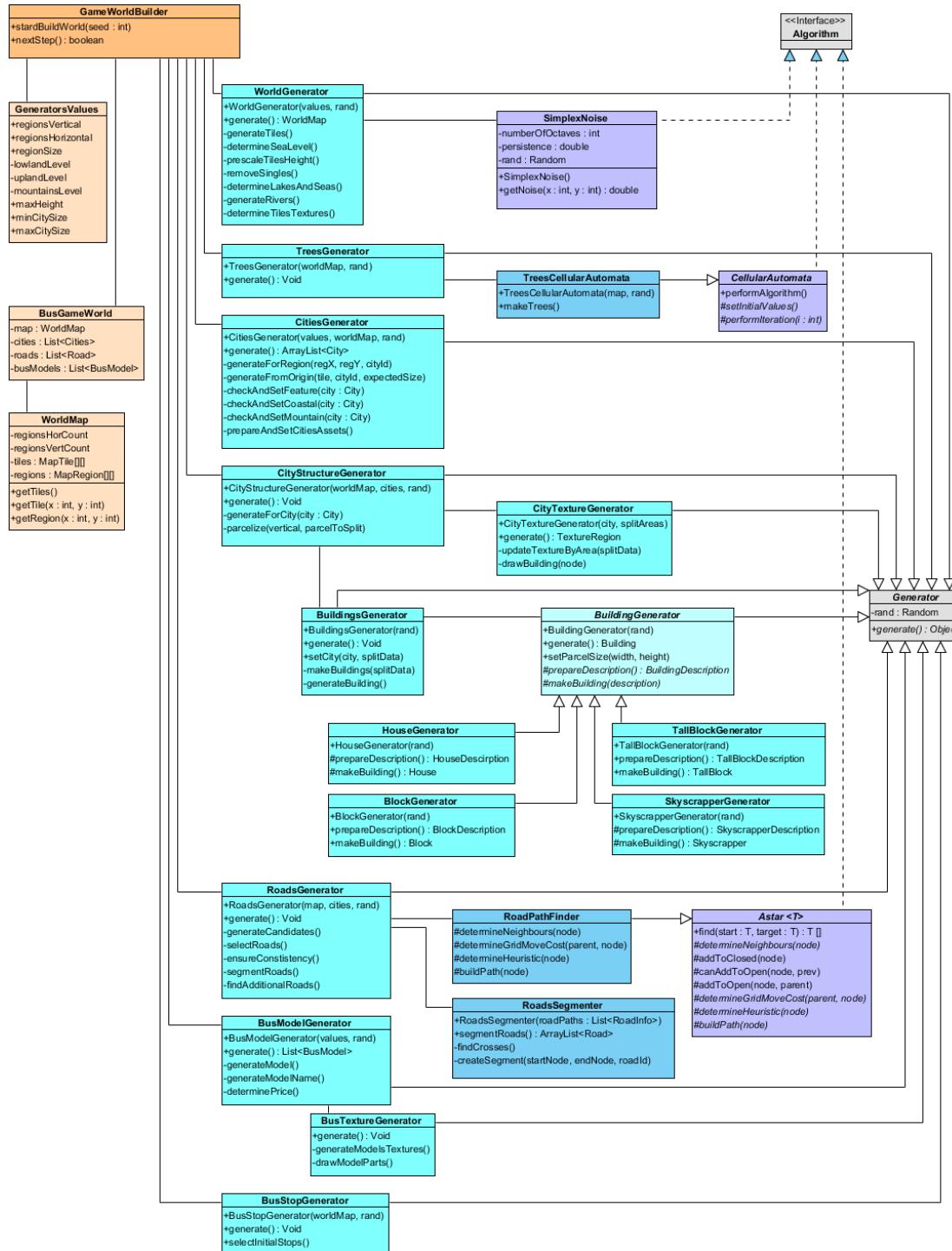
Implementacja elementów gry wykorzystujących generowanie proceduralne

Wybrane elementy świata gry tworzone są techniką proceduralną. Część z nich generowana jest powszechnie wykorzystywany w tym celu algorytmami, a część autorskimi. W procesie tworzenia świata gry używane jest ziarno generatora liczb pseudolosowych, od którego zależy końcowy rezultat. Daje to użytkownikowi wpływ na otoczenie, w którym będzie grał.

3.1 Proces tworzenia świata gry

Tworzenie poszczególnych elementów świata gry oddelegowano do wyspecjalizowanych klas, które przedstawiono na diagramie klas (rysunek 3.1). Główną klasą, której zadaniem jest przeprowadzenie procesu tworzenia środowiska gry, jest klasa `GameWorldBuilder`. Przyjmuje ona ziarno generatora liczb pseudolosowych, które użyte zostanie do zainicjowania generatora używanego dalej w procesie generacji. Kolorem jasnoniebieskim oznaczono klasy generatorów, wywodzące się ze wspólnej klasy bazowej `Generator`. Kolorem ciemnoniebieskim oznaczono pomocnicze klasy poszczególnych generatorów, a kolorem fioletowym wyróżniono klasy będące uogólnieniami lub implementacjami wybranych algorytmów.

Proces tworzenia elementów świata gry przeprowadzany jest krokowo. Odbywa się poprzez wielokrotne wywołanie metody `nextStep()` klasy `GameWorldBuilder`, w której za każdym razem wykonywana jest metoda `generate()` następnego w kolejności generatora. Trwa to tak długo, dopóki całe środowisko gry nie zostanie przygotowane. Generatory bezpośrednio powiązane z klasą `GameWorldBuilder` wywoływane są w tej samej kolejności,



RYSUNEK 3.1: Diagram klas generatorów elementów świata gry.

w jakiej przedstawiono je na diagramie, od góry do dołu. Końcowym rezultatem jest obiekt klasy `BusGameWorld`, który skupia w sobie elementy przygotowanego świata gry.

Krokowe generowanie kolejnych elementów umożliwia przedstawienie graczowi paska

postępu, co daje informację o stanie procesu. Proces przygotowywania nowej gry, w zależności od platformy i urządzenia, może trwać od kilku do kilkunastu sekund.

3.2 Mapa świata

Jak wyjaśniono w pierwszym rozdziale, mapa terenu świata reprezentowana jest w postaci macierzy kwadratowych pól zwanych kaflami. Wymiary mapy są stałe i wynoszą 288 pól w poziomie na 240 pól w pionie. Wymiary właściwego obszaru mapy, na którym umiejscawiane są miasta, wynoszą 240 na 192 pola, a pozostały obszar (24 pola dodane do długości każdego boku mapy po obu stronach właściwego obszaru znajdującego się w środku) dodane są po to, aby elementy interfejsu użytkownika nie zasłaniały miast usytuowanych przy granicach mapy. Każde pole posiada określony typ terenu. Na polach będących częścią lądu mogą być usytuowane drzewa, drogi oraz miasta. Przez wodę mogą być przeprowadzone mosty.

Generacja mapy świata podzielona została na etapy, które omówiono w kolejnych podrozdziałach. Warto zaznaczyć, iż w każdym z etapów, gdziekolwiek jest mowa o wylosowaniu wartości, używana jest ta sama instancja generatora liczb pseudolosowych, zainicjowanego wartością ziarna podaną przez użytkownika (bądź wylosowaną innym obiektem generatora, jeśli użytkownik nie podał wartości ziarna).

3.2.1 Mapa wysokości

Pierwszym krokiem przygotowywania mapy świata jest wygenerowanie mapy wysokości terenu. Jest ona podstawą dla dalszych etapów tworzenia mapy terenu. Najczęściej stosowanymi w tym celu algorytmami są *szum Perlina* [2, 7, 8] oraz *automat komórkowy* [4]. W tym projekcie wykorzystano algorytm *Simplex noise*, również opracowany przez Kena Perlina, w celu zastąpienia jego oryginalnego algorytmu generowania szumu [3].



RYSUNEK 3.2: Grafiki przedstawiają wynik nałożenia na siebie warstw szumu, od lewej: jednej, dwóch i sześciu warstw. Dla każdej kolejnej warstwy dwukrotnie mała wartość parametru częstotliwości.

Simplex noise (od słowa „sympleks”) cechuje się niższą złożonością obliczeniową w stosunku do klasycznego algorytmu szumu Perlina ($O(N^2)$ zamiast $O(2^N)$, gdzie N to liczba wymiarów), lepszą skalowalnością do wyższych wymiarów oraz łatwiejszą implementacją sprzętową [3].

W projekcie wykorzystano dwuwymiarowy szum Simplex, ponieważ potrzebne były wartości szumu dla dwuwymiarowej macierzy pól.

Najciekawsze rezultaty osiąga się poprzez nakładanie na siebie wyników generowania szumu dla odpowiednio zmanipulowanych wartości parametrów. Takie podejście określa się jako *synteza spektralna* (ang. *spectral synthesis* [5]). Przykładowe rezultaty w odniesieniu do tekstur pokazano na rysunku 3.2. W projekcie wstępna wartość wysokości terenu dla pola o współrzędnych x i y wyznaczano następującym wyrażeniem:

$$n(x, y) = \sum_{i=0}^{O-1} a_i \text{noise}(f_i x, f_i y) = \sum_{i=0}^{O-1} p^{O-i} \text{noise}\left(\frac{1}{2^i} x, \frac{1}{2^i} y\right)$$

gdzie: $n(x, y) \in [-1, 1]$ to wartość zsumowanych wartości szumu dla podanych współrzędnych pól (będących tutaj liczbami całkowitymi), O oznacza liczbę oktaw (liczbę nałożonych na siebie warstw szumu), f_i to częstotliwość (zależna od numeru oktawy), a_i oznacza amplitudę (która stanowi wagę wartości szumu i -tej oktawy), obliczaną na podstawie wartości p , która jest parametrem wejściowym. Zwiększenie liczby oktaw skutkuje uzyskaniem bardziej szczegółowej mapy, natomiast zwiększenie wartości parametru p prowadzi do uzyskania mniej regularnych kształtów. W drodze testów metodą prób i subiektywnej oceny uzyskiwanych wyników wybrano następujące parametry dla generowania mapy wysokości: $O = 9$, $p = 0,55$.

Uzyskane w ten sposób wartości dla każdego pola są skalowane i przybliżane do liczb całkowitych tak, by mieściły się odpowiednio w przedziale $[0, 255]$.

3.2.2 Ląd a woda

Na podstawie wygenerowanej mapy elewacji wyznaczana jest wysokość będąca granicą między lądem a wodą, którą można uznać za *poziom morza*. Wszystkie kafle o wysokości większej od tej wartości utworzą ląd, a pozostałe staną się obszarem wody. Wartość ta obliczana jest jako 67% średniej wysokości pól. Dzięki temu większość pól będzie stanowić ląd, natomiast liczba pól wody nie jest stała i zależy od tego, ile spośród kafli nie znajduje się wyżej od tej wysokości. Zbyt duże pokrycie mapy wodą powodowałoby marnowanie obszaru mapy (ponieważ na kaflach wody nie można umieścić miasta ani drogi), natomiast mapa słabo nawodniona mogłaby być uznana przez gracza za mało ciekawą. Przykładowe pokrycie mapy wodą ukazano na rysunku 3.3.

Sąsiadujące ze sobą pola wody tworzą *akwen*. Akweny można podzielić na morza i jeziora. Różnią się tym, że morza graniczą z dowolnym brzegiem mapy.

3.2.3 Ukształtowanie lądu

Aby urozmaicić wizualnie tworzony teren, polom lądu przypisane są bardziej szczegółowe typy terenu. Rozróżniane są: *plaża* (lub *piasek*, jeśli pole nie znajduje się w pobliżu wody), *nizina*, *wyżyna* oraz *obszar górski* (dzielony na *niskie partie góra* i *wysokie partie góra*). Przykładowe reprezentacje graficzne pól poszczególnych typów terenu przedstawiono na rysunku 3.4.

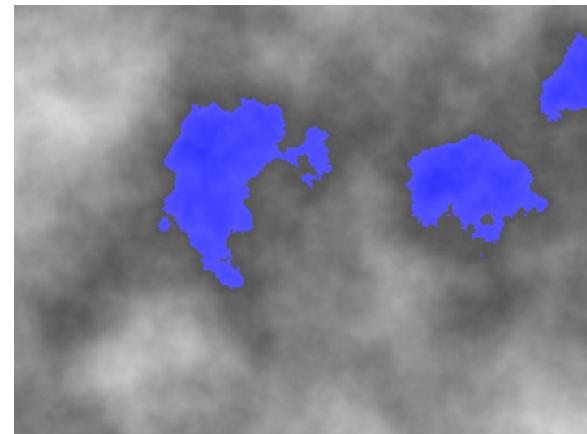
Przy tworzeniu mapy terenu w grach często używa się *diagramu Whittakera*¹, który określa biom (obszar charakterystycznej fauny i flory²) na danym obszarze w zależności od średniej temperatury oraz ilości opadów. Czasem diagram ten przedstawiany jest jako zależność biomu od temperatury i wysokości terenu. W tym projekcie przyjęto, iż cała mapa znajduje się w strefie o umiarkowanym klimacie, z roślinnością i ukształtowaniem terenu zbliżonymi do typowych dla tego klimatu.

Typy terenu w opracowanym projekcie określane są na podstawie wysokości pola oraz bliskości akwenu. Ilość obszaru danego terenu na mapie nie jest stała i zależy od pokrycia mapy wodą. Im więcej obszaru pokryte jest akwenami, tym niższa jest średnia wysokość lądu i mniej pól zaliczyć można do obszaru góra.

Przygotowana mapa terenu może zawierać pojedyncze kafle danego terenu sąsiadujące z każdej strony z terenem innego typu. Takie pola są eliminowane. Jeśli pole nie posiada bliskiego sąsiada o tym samym typie terenu, zostaje mu nadany typ wzięty z wylosowanego sąsiedniego kafla. Wyjątek stanowią kafle typu *plaża*, ponieważ te mogą występować pojedynczo, o ile sąsiadują z polem akwenu.

3.2.4 Rzeki

Rzeki w grze rozpoczynają swój bieg z jednego z krańców mapy i spływają z terenu wyżej położonego na niższy. Każda rzeka swoje ujście znajduje w jeziorze (morzu) lub innej rzece, tworząc dopływ. Kafle początkowe rzek wybierane są ze zbioru kafli lądu będących



RYSUNEK 3.3: Mapa wysokości przedstawiona jako obraz w skali szarości z naniesioną warstwą koloru niebieskiego wskazującą na pola wody. Kolor biały oznacza najwyższą wartość wysokości, natomiast ciemny niebieski wskazuje na głębiny.

¹<http://www.marietta.edu/%7Ebiol/biomes/biome%5Fmain.htm#Whittaker%20Diagrams> (dostęp: 20.11.2014)

²<http://sjp.pwn.pl/sjp/biom;2552225>



RYSUNEK 3.4: Przykładowe tekstury terenu nałożone na pola mapy. Od lewej strony: woda, plaża, nizina, wyżyna, obszar niskich partii gór oraz szczyty górskie (tj. wysokie partie gór). W narożnikach obszarów danego typu terenu nakładane są specjalnie przygotowane, ścięte ukośnie teksty, co ma na celu poprawienie estetyki przedstawianej graczowi mapy (dzięki temu zabiegowi siatka kwadratów pól jest mniej widoczna).

granicznymi polami mapy, których wysokość nad poziomem morza wynosi przynajmniej 15 jednostek. Pola początkowe nie są tu źródłami rzek, a raczej miejscami z których wpływają na widoczny dla gracza obszar mapy świata gry.



RYSUNEK 3.5: Mapa terenu świata gry z widocznymi wyznaczonymi rzekami. Biegą one od brzegów mapy w stronę jej środka, znajdując ujście w jeziorach.

Liczba generowanych rzek jest losowana z przedziału [2, 7]. Aby utworzyć rzekę, najpierw losowany jest kafel początkowy ze zbioru opisanego powyżej. Rozpoczynając od wybranego pola początkowego, program generujący rzekę porusza się w jednym z możliwych kierunków, odwiedzając kolejne pola mapy. Algorytm próbuje zachować kierunek biegu rzeki, o ile to możliwe. Rzeka nie może wpływać na pole o wyższej wartości wysokości niż poprzednie pole, zatem często należy zmienić kierunek lub wycofać się na poprzednie pole.

Aby kształty rzek były bardziej kręte, zmiana kierunku jest co kilka pól wymuszana.

Ogółem przedstawiony sposób wyznaczania ścieżki może przypominać algorytm błądzenia losowego, jednak w kolejnych krokach kierunek nie jest losowy. Bieg rzeki zmienia się zawsze łagodnie. Przykładowo, jeśli rzeka biegnie w kierunku północnym, to może skręcić tylko na wschód, północny wschód, północny zachód bądź zachód.

Algorytm kończy działanie gdy natrafi na pole wody. Stworzona rzeka wpływa zatem do jeziora bądź morza lub tworzy dopływ innej rzeki. Minimalna długość rzeki wynosi 15 pól i jeśli rzeka nie spełnia tego warunku, jest odrzucana i wyznaczana jest inna rzeka.

Rzeki rysowane są jako dwuwymiarowa siatka wierzchołków utworzona na podstawie krzywej. Krzywa jest interpolowana, używając współrzędnych środków pól z tablicy kafli

rzeki przy pomocy algorytmu funkcji sklejanej (ang. *spline*) *Catmulla-Roma* [10]. Dzięki temu graczowi prezentowana jest wygładzona krzywa przypominająca wijącą się rzekę, zamiast malowanych na niebiesko kwadratów pól, co podnosi walory estetyczne gry. Przykład mapy z nanesionymi wygenerowanymi rzekami przedstawiono na rysunku 3.5.

3.2.5 Lasy

W celu urozmaicenia wizualnego mapy gry pewna część lądu pokrywana jest obszarem leśnym. Wybór kafli, na których umieszczone zostaną drzewa, dokonywany jest przy użyciu dwuwymiarowego *automatu komórkowego* (ang. *cellular automata*, CA). Innym przykładem jego zastosowania w grach jest proceduralne tworzenie jaskiń [9, rozdz. 3].

Jest to rozwiązań pozwalające na dobrą kontrolę wyniku dzięki parametryzacji [4]. W opisywanym zastosowaniu wyróżnić można trzy parametry:

- n : liczba iteracji,
- m : odległość najdalszego sąsiada w *sąsiedztwie Moore'a*, tj. liczba kolejnych pól w linii prostej najbliższych do rozpatrywanego pola,
- t : próg akceptacji, tj. minimalna liczba sąsiadów w sąsiedztwie Moore'a w odległości m , która decyduje o tym, czy na rozpatrywanym polu umieszczone zostanie drzewo czy też nie.

W tematyce automatów komórkowych *sąsiedztwem Moore'a* nazywamy kwadratowy obszar pól otaczających pole w centrum obszaru takich, że mogą mieć wpływ na jego ewolucję w dwuwymiarowym automacie komórkowym³. Dla reprezentacji mapy w postaci siatki kwadratów pole nie leżące na brzegu mapy posiada ośmiu sąsiadów w sąsiedztwie Moore'a.

Zastosowany algorytm można podzielić na 3 fazy:

1. Pierwszą fazą jest ustalenie stanu początkowego automatu. Wszystkim polom brzegowym mapy na bokach północnym i zachodnim przypisywane są wartości 0 lub 1, z prawdopodobieństwem $\frac{1}{2}$. Następnie pozostałym polom mapy, iterując po wierszach i kolumnach, od najmniejszych do największych współrzędnych, przypisywana jest wartość $tiles[y - 1][x] \text{ XOR } tiles[y][x - 1]$, gdzie *tiles* oznacza dwuwymiarową tablicę pól mapy odpowiadającą stanowi automatu. Stan początkowy większości pól zależy zatem od stanu początkowego dwóch pól sąsiednich. Innym podejściem jest przypisanie wszystkim polom wartości 0 bądź 1 z prawdopodobieństwem $\frac{1}{2}$, jednak zastosowane podejście skutkuje utworzeniem bardziej rozległych obszarów leśnych.

³<http://mathworld.wolfram.com/MooreNeighborhood.html> (dostęp: 26.11.2014)

2. Następna faza wykonywana jest tyle razy, ile określa liczba iteracji. W każdej iteracji wszystkie komórki automatu (pole mapy) przechodzą ewolucję. Jeżeli liczba sąsiadów w rozpatrywanym sąsiedztwie Moore'a, która posiada wartość 1 w danym stanie, jest większa bądź równa wartości progu (parametr t), to w następnym stanie rozpatrywane pole otrzyma wartość 1. W przeciwnym wypadku przypisana zostanie mu wartość 0.
3. Ostatnim krokiem jest umieszczenie drzew na polach mapy, które w ostatnim ustalonym stanie posiadają wartość 1 oraz są polami lądu o typie innym niż plaża.

Aby tworzone obszary leśne przypominały te ze świata rzeczywistego, ustalonono metodą prób i błędów następujące parametry: $n = 2$, $m = 3$, $t = 25$. Dzięki takiemu rozwiązaniu lasem pokryta większość pól lądu. Eksperymentując z parametrami, otrzymywano obszary leśne o różnych kształtach, bardziej lub mniej poszarpanych krawędziach oraz różnych rozmiarach i gestszym bądź rzadszym pokryciu mapy. Przykładowe rezultaty przedstawiono na rysunku 3.6.

Na danym polu może znajdować się maksymalnie jedno drzewo. Drzewa umieszczane są na polach w pobliżu ich centrów (w kwadracie o boku $\frac{1}{3}$ szerokości pola). Pozwala to zachować pewną odległość od sąsiadnych drzew oraz zapobiega wyrastaniu drzew na przebiegających na sąsiadnych polach drogach. Tworzone lasy są typu mieszanego, tj. mogą występować w nich zarówno drzewa liściaste jak i iglaste, z pewną przewagą tych pierwszych. Przyjęto, iż nie są umieszczane drzewa na polach plaży ani wysokich partii górskich. Lasy w obszarze niskich partii górzystych są zawsze typu iglastego. Tekstury drzew nakładane na pola wybierane są pseudolosowo z przygotowanego zbioru, zależnie od typu lasu.



RYSUNEK 3.6: Lasy uzyskane dla tego samego ziarna generatora świata gry, ale z różnymi wartościami parametrów automatu komórkowego. Ciemnozielone obszary to skupiska drzew. Parametry użyte do uzyskania wyniku jak na obrazku po lewej stronie: $n = 2$, $m = 3$, $t = 25$, w środku: $n = 3$, $m = 2$, $t = 13$, po prawej: $n = 4$, $m = 3$, $t = 25$.

3.3 Miasta

Miasta stanowią istotny element świata gry. Od ich rozmiaru i położenia zależy, ilu podróżnych będzie się w nich pojawiać oraz ilu pasażerów będzie do tych miast podążać. Ich graficzna prezentacja ma pomóc graczowi w rozpoznaniu charakterystyki poszczególnych miejscowości.

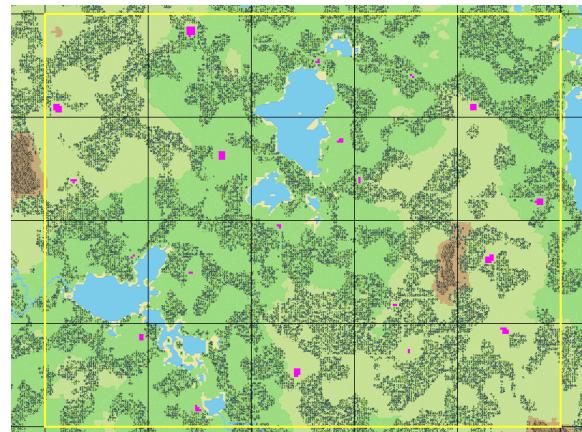
3.3.1 Położenie i rozmiar miejscowości

Aby zapewnić względnie równomierne rozmieszczenie miast zdecydowano się podzielić mapę na kwadratowe, równoliczne obszary pól nazwane *regionami*. Wymiary regionu to 48 na 48 kafli. Podziałowi poddawany jest wewnętrzny obszar mapy, z pominięciem dodatkowo wygenerowanych obszarów przy bokach mapy, opisanych na początku rozdziału. Teren dzielony jest zatem na 20 regionów (5 kolumn, 4 rzędy). W każdym regionie może znaleźć się co najwyżej jedno miasto, zatem maksymalna liczba miast na mapie wynosi 20. Takie rozwiązańe eliminuje sytuację, w której wiele miast znalazło by się w skupisku w pewnym rogu mapy, pozostawiając pozostały obszar terenu niezaludnionym i właściwie bezużytecznym w grze. Przykładowe rozmieszczenie miast na mapie przedstawiono na rysunku 3.7.

Jeśli w regionie jest zbyt mało sąsiadujących ze sobą pól lądu takich, na których można umieścić miasto, wtedy w regionie nie zostanie wygenerowane miasto, w związku z czym liczba miast na mapie będzie odpowiednio mniejsza. Polami należącymi do obszaru miasta mogą być tylko pola lądu nie będące typu *wysokie partie gór* (tzn. miasta nie są umieszczane na wodzie ani na szczytach górskich). Aby miasta nie znajdowały się zbyt blisko siebie, pola stanowiące brzeg regionu oraz bezpośrednio z nimi sąsiadujące nie są rozpatrywane jako potencjalne

TABLICA 3.1: Typy miejscowości występujące w grze. Zakres możliwych rozmiarów podano w liczbie pól.

Typ	Rozmiar	Prawdopod.
Wieś	2 - 3	0,2
Miasteczko	4 - 7	0,3
Miasto	8 - 13	0,3
Metropolia	14 - 20	0,2



RYSUNEK 3.7: Rozmieszczenie miast na mapie. Obszary miejscowości zaznaczone różowym kolorem. Czarne linie wyznaczają granice regionów. Żółtym obramowaniem zaznaczono wewnętrzny obszar mapy, w którym umiejscawiane są miasta.

pola miasta.

Algorytm wybierający pola należące do miasta najpierw ustala pole początkowe, dookoła którego rozrastać się będzie miejscowości, zależnie od jej spodziewanego rozmiaru i dostępności pól. Rozmiar miasta zależy od typu miejscowości (przedstawiono je w tablicy 3.1). Typ miasta ma wpływ na liczbę przyjezdnych oraz pojawiających się w mieście podróżnych a także na wygląd miasta, tj. na typ umieszczanych w mieście budynków. Typy miast dobierane są z pewnym ustaloną prawdopodobieństwem, jednak rzeczywisty ich rozkład może się różnić, ponieważ jeśli w danym miejscu nie można wygenerować miasta danego typu z powodu zbyt małej liczby dostępnych pól, zostanie utworzona największa możliwa miejscowości. Pełną procedurę przedstawia algorytm 1.

Algorytm 1 Algorytm wyboru pól dla miejscowości w danym regionie.

Wejście: region pól *reg* dla którego należy umieścić miasto, liczba *minCitySize* określająca najmniejszy możliwy rozmiar miejscowości (w liczbie pól), wartość *cityType* określająca pożądany typ miejscowości (typy wyszczególniono w tablicy 3.1).

Wyjście: obiekt miasta *city* utworzony z wyznaczonego zbioru pól lub *null*, jeśli nie można stworzyć miejscowości w regionie *reg*.

Metoda:

- 1: Wyznacz pożądany rozmiar miasta *expectedSize* poprzez wylosowanie wartości z przedziału rozmiarów miejscowości typu *cityType* (tablica 3.1).
 - 2: Jako zbiór *cset* przyjmij podzbiór wszystkich pól regionu, które spełniają warunki dla pola miejscowości.
 - 3: **Jeżeli** zbiór *cset* jest pusty **to** zakończ algorytm zwracając *null*.
 - 4: Za *origin* przyjmij losowy element usunięty ze zbioru *cset*.
 - 5: Utwórz kolejkę pól *candidates* i wstaw do niej element *origin*.
 - 6: Utwórz pusty zbiór pól *cityTilesSet*.
 - 7: **Dopóki** *candidates.size > 0* **wykonaj**
 - 8: *tile* \leftarrow *candidates.dequeue()*
 - 9: *cityTilesSet.add(tile)*
 - 10: **Jeżeli** *cityTilesSet.size = expectedSize* **to** przerwij pętlę.
 - 11: Do kolejki *candidates* dodaj tych sąsiadów *tile*, którzy spełniają warunki dla pól miasta oraz nie były jeszcze rozpatrywane.
 - 12: **KoniecDopóki**
 - 13: **Jeżeli** *cityTilesSet.size < minCitySize* **to**
 - 14: Wróć do kroku 4.
 - 15: **w p. p.**
 - 16: *city* \leftarrow *newCity(cityTilesSet)*
 - 17: Na podstawie liczby pól miejscowości wyznacz i zapamiętaj jej faktyczny typ.
 - 18: **Zwróć** *city*.
 - 19: **KoniecJeżeli**
-

3.3.2 Kurorty

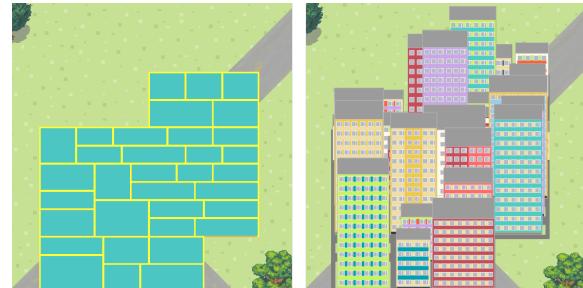
Ze względu na położenie miejscowości może pełnić funkcję kurortu turystycznego. Miasto leżące nad morzem bądź dużym jeziorem staje się *kurortem letnim*, natomiast miejscowości leżąca w obszarze górskim bądź u podnóża gór pełni funkcję *kurortu zimowego*. Miejscowości turystyczne są częściej odwiedzane przez podróżnych w sezonie, zatem latem wzrośnie liczba chętnych do odwiedzenia kurortu letniego, a w miesiącach zimowych wzrośnie ruch do miast położonych w górach. Pora roku ma realny wpływ na rozgrywkę i gracz powinien wykorzystać ten fakt przy ustalaniu tras autobusów.

Miejscowość staje się kurortem letnim, jeśli w odległości do trzech pól od dowolnego granicznego pola obszaru miasta znajdzie się pole akwenu, którego obszar wynosi przynajmniej 20 pól (zatem miasto nie musi leżeć bezpośrednio nad wodą — między miastem a wodą może znajdować się plaża). Podobnie ma się rzecz z kurortami zimowymi, z tą różnicą, że zamiast pól wody pożądane są pola typu *obszar górska*.

Możliwe jest, że jedna miejscowości przyjąć może charakterystykę kurortu letniego i zimowego. W takim wypadku miejscowości staje się *kurortem całorocznym* i odwiedzana jest częściej zarówno latem, jak i zimą.

3.3.3 Podział przestrzeni miejscowości

W obszarze każdego miasta umiejscawiany jest zbiór budynków, które nadają miejscowości unikalny charakter oraz ułatwiają graczowi wizualną ocenę rozmiaru i typu miasta. W celu rozumieszczenia budynków w mieście, zbiór pól mapy należących do obszaru danego miasta dzielony jest na prostokątne obszary, które następnie dzielone są algorytmem *podziału binarnego* (ang. *binary space partitioning, BSP*). Algorytm ten jest często używany w grach do podziału przestrzeni m.in. do tworzenia struktury miasta czy do programowego uzyskania systemu jaskiń [9, rozdz. 3].



RYSUNEK 3.8: Przykładowy podział obszaru miejscowości na parcele (na lewym obrazku) oraz umiejscowienie budynków (na prawym obrazku). Granice parcel zaznaczono żółtym kolorem. Przedstawiona miejscowości jest typu *miasto*, a widoczne budynki to *niskie bloki* oraz *wysokie bloki*.

Wejściem dla algorytmu jest prostokątny obszar o znanych wymiarach. Przestrzeń ta dzielona jest na dwa mniejsze prostokąty, w pionie bądź poziomie. Następnie wyznaczone podobszary dzielone są rekurencyjnie tym samym algorytmem. Podział trwa tak długo, aż żaden z obszarów nie może zostać podzielony na mniejszy (znane są minimalne wymiary

obszaru). Do przedstawionego algorytmu wprowadzono modyfikację, która polega na tym, że jeśli obszar do podziału ma wymiary mniejsze bądź równe maksymalnym dopuszczalnym wymiarom, to z prawdopodobieństwem $\frac{1}{2}$ nie zostanie podzielony.

Najmniejsze uzyskane obszary nazywane są dalej *parcelami*. Na każdej parceli postawiony będzie budynek. Minimalne oraz maksymalne wymiary parceli zależą od typu miasta (im większe miasto tym większe budynki mogą się w nim pojawić). Parcyla składa się z *jednostek* będących kwadratami o boku $\frac{1}{6}$ szerokości kafla mapy. Zatem na jednym kaflu umiejscowionych może być kilka budynków.

Przykładowy podział przestrzeni miasta na parcele oraz umiejscowienie na nich budynków przedstawiono na rysunku 3.8.

3.3.4 Budynki

Na przygotowanych parcelach umieszczane są budynki. Poniżej scharakteryzowano wszystkie typy budowli występujących w grze.

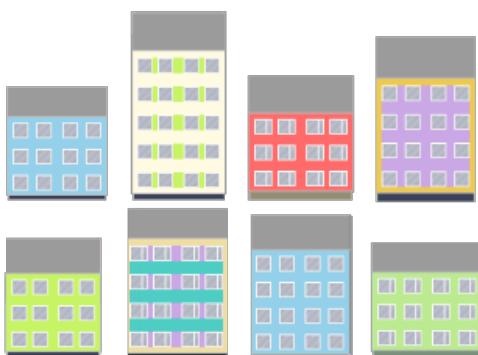
Dom



Dom (rys. 3.9) to najmniejsza rozmiaru i zajmowaną powierzchnią budowla. Może posiadać jedną lub dwie kondygnacje. Występuje w miastach dowolnego typu prócz metropolii.

RYSUNEK 3.9: Przykładowe reprezentacje domów.

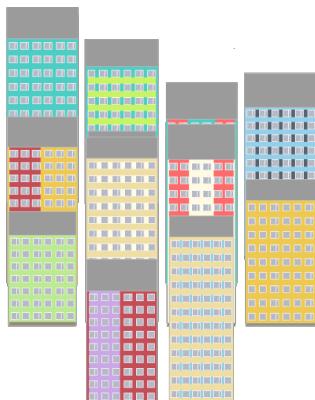
Niski blok



Niskie bloki (rys. 3.10) mogą mieć od 3 do 5 kondygnacji. Zajmują stosunkowo nie dużo przestrzeni. Dzięki różnym wariantom malowania są bardzo zróżnicowane kolorystycznie. Umieszczane są w miejscowościach dowolnego typu.

RYSUNEK 3.10: Niskie bloki.

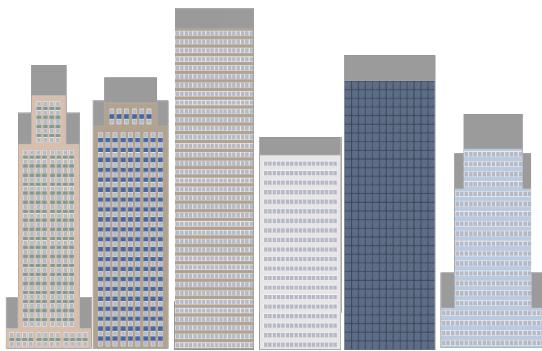
Wysoki blok



Wysokie bloki mieszkalne (rys. 3.11) mogą posiadać od 7 do 12 kondygnacji. Posiadają więcej mieszkań na piętrze od małych bloków. Pojawiają się w każdej miejscowości z wyjątkiem wsi.

RYSUNEK 3.11: Wysokie bloki.

Wieżowiec



RYSUNEK 3.12: Wieżowce.

Wieżowce (rys. 3.12) to najwyższe i zazwyczaj zajmujące największe parcele budynki. Mogą posiadać od 16 do 40 kondygnacji. Fasady niektórych z nich pokryte są w całości szkłem, inne posiadają przestrzeń między oknami. Ich bryły mogą składać się z jednego, dwóch lub trzech elementów: bryły głównej (centralnej, występuje zawsze), bryły dolnej (szerszej od bryły głównej) oraz bryły górnej (najwęższej i najniższej). Wieżowce występują jedynie w metropolach.

Budynki charakteryzują się wieloma parametrami uszczegóławiającymi ich wygląd, dzięki czemu bardzo mało prawdopodobnym jest otrzymanie dwóch takich samych miast na mapie. Procedury tworzące struktury poszczególnych budynków na pewnym poziomie abstrakcji działają tak samo: wyznaczana jest liczba pięter budynku z pewnego przedziału (zależnego od typu budynku) oraz liczba lokali na piętrze, liczba okien w lokalu, tekstury okien, kolor ścian budynku, ewentualnie kolor i kształt dodatkowych elementów budynku. Wyznaczana jest także przestrzeń między oknami lokalu, szerokość odstępu między lokalam oraz wysokość kondygnacji.

W celu uproszczenia rysowania budynki przedstawiane są zawsze od strony nie zawiązującej drzwi. Budowle rysowane są symbolicznie, bez zachowania dokładnej skali w stosunku do pozostałych obiektów gry. Także liczba budynków w mieście jest dalece niższa od realistycznej, aby miasta nie zajmowały niepotrzebnie dużo przestrzeni mapy.

3.3.5 Identyfikacja miejscowości

Każdej z miejscowości przypisywana jest nazwa w postaci unikalnej w skali mapy litery alfabetu angielskiego. Kolejne symbole alfabetu przypisywane są kolejnym regionom mapy, poruszając się od lewej do prawej, od góry do dołu. Dzięki temu miasto w regionie leżącym w lewym górnym rogu zawsze będzie miało nazwę A, jego prawy sąsiad — nazwę B itd. Ułatwia to graczowi nawigację na mapie gry.

Ponadto każdej miejscowości przypisana jest unikalna kombinacja figury geometrycznej i koloru. Figura zależy od typu miejscowości. Wsi przypisany jest zawsze trójkąt, miasteczku — koło, miastu odpowiada kwadrat, a do oznaczenia metropolii używany jest pięciokąt. Taka identyfikacja ma pomóc graczowi w szybkim rozumnianiu celów podróżnych, jako że pasażerowie autobusów oraz ci oczekujący na przystankach przedstawiani są właśnie jako ikonki w postaci kolorowej figury odpowiadającej znakowi graficznemu miejscowości docelowej podróżnego. Przykład prezentacji graficznej pasażerów można zobaczyć w podrozdziale opisującym generowanie autobusów.

3.4 Drogi

Miasta połączone są siecią dróg, która może być rozważana jako graf spójny nieskierowany, nie pozbawiony cykli, o wierzchołkach w miejscowościach bądź w skrzyżowaniach i drogach jako krawędziach. Zatem autobus wyruszający z dowolnego miasta może dojechać do każdego innego, być może przejeżdżając przez pewną liczbę innych miejscowości. Przykładową mapę z naniesioną siatką dróg przedstawiono na rysunku 3.13.

W dalszej części tego podrozdziału pod pojęciem droga rozumie się bezpośrednie połączenie drogowe między parą miast, tj. nie przechodzące przez żadne inne miasto, ale być może przechodzące przez skrzyżowania. Zaznaczyć należy, że każda droga jest dwukierunkowa, zatem droga określona jako prowadząca do danego miasta jest jednocześnie drogą wychodzącą z tej miejscowości.

3.4.1 Wyznaczenie zbioru dróg

Generację sieci dróg podzielić można na cztery etapy. Pierwszym z nich jest stworzenie pomocniczego zbioru ścieżek dróg pomiędzy każdą parą miast, których regiony sąsiadują ze sobą w pionie bądź w poziomie. Do wyznaczenia ścieżek dróg używany jest algorytm A^* . Jest to często używany algorytm szukania najkrótszej ścieżki między punktami na mapie, bądź ogólnie — najkrótszej drogi w grafie ważonym. Charakteryzuje się tym, iż jest zupełny i optymalny [1], tzn. zawsze znajduje najkrótszą ścieżkę, o ile taka istnieje. Jego dużą zaletą jest szybkość działania, ponieważ dzięki wykorzystywaniu heurystyki algorytm

próbuje przewidzieć kierunek, w którym należy podążać, aby uzyskać optymalną ścieżkę. Ponieważ w opracowywanym projekcie drogi mogą być prowadzone w ośmiu kierunkach, odpowiednią heurystyką w tym przypadku [6] będzie wartość wyznaczana w przedstawiony sposób (pseudokod):

```
C0 = 10 // koszt ruchu w pionie lub w poziomie
CD = 14 // koszt ruchu ukośnie, tj.  $10 * \sqrt{2}$ 

int heuristic(node) {
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return C0 * (dx + dy) + (CD - 2 * C0) * min(dx, dy)
}
```

Algorytm startuje w polu będącym polem głównym jednego z miast, a celem jest osiągnięcie pola głównego drugiego miasta z rozpatrywanej pary. Poruszać można się po polach dowolnego typu z wyłączeniem pól wysokich partii górskich (drogi w grze omijają szczyty górskie). Koszt przeprowadzenia drogi przez pole wody jest trzykrotnie większy od kosztu ruchu na pole lądu. Skutkuje to tym, że algorytm próbuje omijać jeziora i unikać przeprawy przez wodę. Czasem jednak opłacalne jest wytyczenie mostu.

Mosty tworzone są zawsze jako proste odcinki drogi (w dowolnym z ośmiu rozpatrywanych kierunków). Mosty nie mogą się krzyżować, zatem jeśli algorytm napotka istniejący most, wycofa się i spróbuje innej ścieżki.

Drugim krokiem procesu generacji siatki dróg jest utworzenie początkowego zbioru dróg, wybierając drogi ze zbioru pomocniczego utworzonego w pierwszym kroku. Przyjęto, że do mniejszych miejscowości powinno prowadzić mniej dróg, niż do większych. Program wybiera do zbioru początkowego takie najkrótsze drogi ze zbioru pomocniczego, by do każdej miejscowości doprowadzonych było maksymalnie tyle dróg, ile wynika z typu miejscowości (odpowiednio dla wsi, miasteczek, miast i metropolii: 1, 2, 3 i 4 drogi). To założenie może jednak stracić na znaczeniu w następnych krokach.

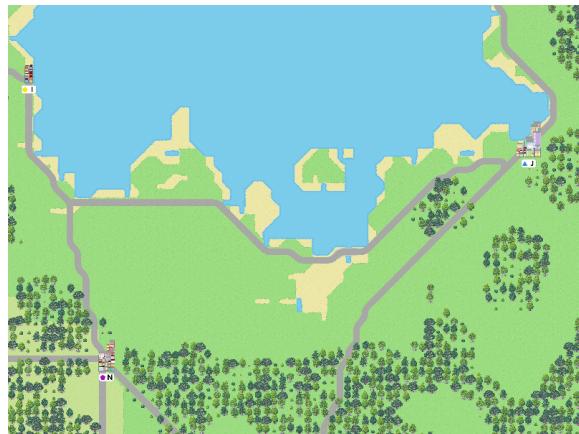


RYSUNEK 3.13: Mapa z wyznaczoną siatką dróg. Duże obszary wody oraz górzyste tereny utrudniły połączenie miast drogami, wymuszając prowadzenie dróg dookoła tych obszarów. U dołu rysunku widać długą linię, która niewątpliwie skróci czas podróży między miejscowościami oddzielonymi akwenem.

Trzecim etapem jest zapewnienie spójności grafu sieci dróg poprzez rozszerzenie tworzonego zbioru dróg, o ile zajdzie taka potrzeba. Sprawdzane jest, czy z każdego miasta można dotrzeć do dowolnego innego, przechodząc przez ustaloną maksymalną liczbę siedmiu dróg. Jeśli wygenerowania sieć dróg nie spełnia tego warunku dla rozpatrywanej pary miast, należy dodać drogę (bądź kilka) ze zbioru przygotowanego w pierwszym etapie generacji dróg. Dobierane jest możliwe mało dróg takich, których dodanie do wynikowego zbioru dróg spowoduje spełnienie założonego warunku spójności. Po tym kroku może okazać się, że do niektórych małych miejscowości wcale nie prowadzi mniej dróg, niż do większych.

3.4.2 Segmentacja

W ostatnim kroku odbywa się segmentacja dróg. Proces ten ma na celu znalezienie wspólnych części dróg oraz skrzyżowań, aby zminimalizowaćłączną liczbę pól dróg oraz aby uczynić strukturę siatki dróg bliższą realizmowi. Jeśli wiele dróg biegnie po tych samych lub sąsiednich polach, ich ścieżki powinny być przybliżone na pewnym odcinku tak, by współdzieliły rozważany odcinek drogi, a w miejscu, w którym drogi te się rozchodzą, powinno znaleźć się skrzyżowanie.



RYSUNEK 3.14: Przykład wyznaczonych skrzyżowań dróg. Droga biegnąca z miasta I (po lewej stronie obrazka) do miasta J (po prawej) krzyżuje się z drogą I-N, dzięki czemu miasto N (na dole po lewej) zyskało bezpośrednią drogę do miasta J. Na grafice widać także inne skrzyżowanie, w pobliżu miasta J.

Po procesie segmentacji drogi nie są już rozważane jako tablice pól, a jako listy segmentów, którymi są poprowadzone. W obiekcie segmentu pamiętane będą pola mapy, na których biegnie droga (bądź wiele dróg). Warto zaznaczyć, że ścieżki dróg mogą się nieznacznie zmienić od tych wyznaczonych w kroku pierwszym.

Jako *segment* rozumie się odcinek drogi, którego końcem jest pole miasta stanowiące jeden z końców drogi, bądź skrzyżowanie dróg. Program przegląda pola kolejnych dróg ze zbioru wyznaczonego w kroku trzecim oraz ich pola sąsiednie w celu znalezienia dróg biegących po tych samych polach bądź bardzo blisko siebie

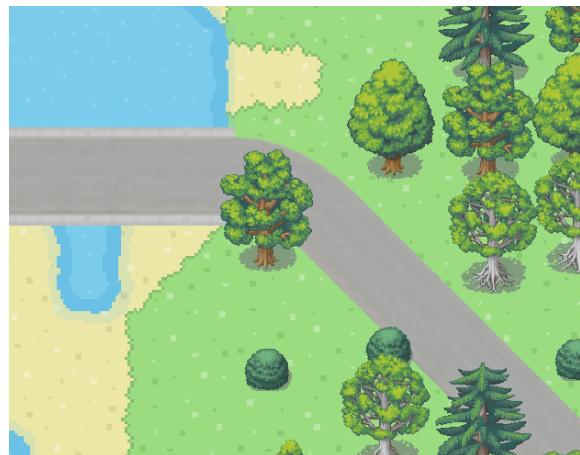
(w obiektach pól mapy pamiętane są drogi, które są nimi wytyczone). Wyróżnić można trzy sytuacje. W pierwszej, dwie drogi biegnące blisko siebie mają wspólny koniec (tj.

wychodzą z tego samego miasta). Wtedy znajdują się najdalsze takie pole (rozpatrując pola obu dróg, rozpoczynając od wspólnego końca), w którym drogi te biegną blisko siebie. Na tym polu umiejscawiane jest skrzyżowanie. Odcinek jednej z rozważanych dróg (krótszy) od wspólnego końca do pola skrzyżowania stanie się segmentem wspólnym dla obu dróg przez te drogi. Od skrzyżowania do miast docelowych obu dróg biegną osobne segmenty, wykorzystywane od teraz przez rozpatrywane drogi. Sytuację przedstawia rysunek 3.14. Drogi między miastami I-N oraz I-J biegną od miasta I po wspólnym odcinku (segmencie) do skrzyżowania, w którym się rozchodzą.

W drugiej sytuacji dwie drogi, które biegną na pewnym odcinku blisko siebie, nie posiadają wspólnego końca (nie prowadzą do tego samego miasta). W tej sytuacji program stara się wyznaczyć możliwie najdłuższy odcinek pól, po którego obu końcach rozpatrywane drogi biegną na sąsiednich polach. Na tych końcach umiejscowione zostaną skrzyżowania, a między nimi wybrana zostanie krótsza ścieżka pól jednej z tych dróg, która stanie się ich wspólnym segmentem. Od skrzyżowań do miast, które są końcami rozpatrywanych dróg, utworzone zostaną niezależne segmenty.

Trzecia sytuacja zakłada, że droga nie krzyżuje się z żadną inną drogą na odcinku pól nienależących do żadnej miejscowości (skrzyżowania nie są tworzone w obrębie obszarów miejscowości). Wtedy cała droga staje się jednym niezależnym segmentem.

Podczas procesu segmentacji drogi możliwe jest napotkanie skrzyżowania z utworzonym już segmentem (a nie z inną nierożpatrywaną drogą, jak opisano w powyżej wyszczególnionych przypadkach). Wtedy proces przebiega podobnie, tylko zamiast pary dróg rozpatruje się drogę oraz segment. Możliwe jest zatem podzielenie istniejącego segmentu na krótsze i wstawienie między nimi skrzyżowania, do którego przyłączają się segmenty rozpatrywanej drogi. Listy segmentów należących do dróg, które biegną po podzielonym segmencie, zostaną uaktualnione.



RYSUNEK 3.15: Wygląd mostu oraz drogi na lądzie.

Dzięki wyznaczonym skrzyżowaniom może się okazać, że z pewnego miasta można teraz dojechać bezpośrednią drogą do innego miasta, z którym to wcześniej nie było bezpośredniego połączenia drogowego. Zwiększa to możliwości gracza w zakresie wyznaczania tras linii autobusowych.

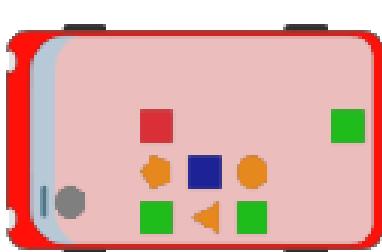
Droga rysowana jest w podobny sposób, co rzeka, tj. jako siatka trójkątów z nałożoną tekstonią, utworzona wzdłuż krzywej przebiegu ścieżki drogi. Przykład wyglądu drogi na lądzie oraz mostu przedstawiono na rysunku 3.15.

3.5 Autobusy

Autobusy w grze charakteryzują się kilkoma parametrami mającymi wpływ na rozgrywkę. Parametry te to:

- *pojemność* (maksymalna liczba pasażerów mogących znaleźć się w autobusie),
- *prędkość* (szybsze autobusy poruszają się między miastami w krótszym czasie),
- *spalanie* (wpływa na koszt związany z pokonaniem trasy przez autobus; wysokie wartości parametrów *prędkość* oraz *pojemność* przekładają się na zwiększenie spalania),
- *wygoda* (pasażerowie zapłacą więcej za przejazd w komfortowych warunkach).

Powyższe parametry zależą od modelu pojazdu. Modele posiadają także nazwę producenta (wylosowaną spośród przygotowanego zbioru) oraz nazwę modelu (w postaci konkatenacji wielkiej litery z alfabetu angielskiego i kilku cyfr). Na podstawie parametrów wyznaczana jest cena kupna autobusu. Im lepsze parametry posiada dany model, tym więcej gracza będzie musiał za niego zapłacić.



RYSUNEK 3.16: Autobus z pasażerami w środku.

Przykładową reprezentację graficzną autobusu przedstawiono na rysunku 3.16. Rozmiar tekstuury autobusu zależy od liczby miejsc dla pasażerów. Dostępne są pojazdy o trzech oraz o czterech kolumnach siedzeń. Liczba rzędów siedzeń wybierana jest z zakresu od 4 do 10 (choć jest to zależne od liczby kolumn). Zatem autobusy posiadają różne liczby miejsc, co w połączeniu z pozostałymi parametrami pojazdu pozwala graczowi dobrać pojazd do potrzeb konkretnej trasy. Autobus rysowany na mapie podczas pokonywania trasy jest kolorowany barwą przypisaną do linii, którą obsługuje.

Rozdział 4

Logika gry

Na wygenerowanym terenie gracz ma za zadanie zbudować i rozwijać intratną sieć komunikacyjną. Aby tego dokonać, musi przyjrzeć się potrzebom pojawiających się podróżnych i dostosować do nich tworzone trasy. W celu uczynienia rozgrywki ciekawszą i wymagającą, przygotowano szereg mechanizmów wzbogacających świat gry.

4.1 Obsługa podróżnych

Istotą prowadzonej rozgrywki jest sprostanie wymogom podróżnych. Gracz tworzy połączenia autobusowe poprzez wytyczenie tras łączących poszczególne miejscowości. Do przygotowanych linii przypisywane są posiadane autobusy w celu rozwożenia podróżnych po mapie.

4.1.1 Pojawianie się podróżnych na przystankach

Wraz z upływem czasu na przystankach należących do gracza pojawiają się podróżni z zamiarem przemieszczenia się do wybranej miejscowości. Liczba podróżnych pojawiających się w ciągu doby zależy od rozmiaru (typu) miasta. Z dużych miejscowości chce odjechać więcej pasażerów, niż z małych. Ponadto na liczbę potencjalnych pasażerów wpływa też liczba posiadanych przez gracza przystanków. Wraz ze wzrostem liczby przystanków rośnie liczba pojawiających się podróżnych, ponieważ rośnie zasięg siatki komunikacyjnej.

Przystanki mają ograniczoną pojemność i żaden nowy podróżny nie pojawi się na pełnym przystanku. Wyjątkiem jest sytuacja gdy pasażerowie przesiadają się w danym mieście — wtedy wysiądą z autobusu na zatłoczonym przystanku, jednak nie zostanie tu wygenerowany żaden nowy podróżny, dopóki nie zwolni się miejsce. Przystanek, który jest zatłoczony (tzn. jest prawie pełen), rysowany jest z użyciem pomarańczowego koloru tła,

natomiast zapełnienie przystanku sygnalizowane jest graczowi czerwonym kolorem pola przystanku.

4.1.2 Wybór celu podróży

Celem podróży pasażera może być tylko miejscowości, w której gracz posiada przystanek. Miejscowość ta wybierana jest na podstawie prawdopodobieństwa, na które wpływ mają:

- odległość „Manhattan” między regionami miast (suma wartości różnic współrzędnych regionów, w których leżą miasta; między pobliskimi miastami podróżuje więcej pasażerów niż między odległymi),
- typ potencjalnej miejscowości docelowej (do większych miejscowości podróżuje więcej pasażerów),
- popularność miasta docelowego.

Popularność miejscowości wynika z typu kurortu w danej miejscowości (jeśli jest kurortem) oraz aktualnej pory roku w grze. W miesiącach letnich miejscowości będące kurortami letnimi są częściej odwiedzane przez turystów. Analogicznie w porze zimowej podróżni częściej wybierają miejscowości będące kurortami zimowymi.

Na popularność mają też wpływ wydarzenia losowe, opisane na końcu tego rozdziału, które krótkotrwale, ale intensywnie zwiększą zainteresowanie daną miejscowością.

4.1.3 Tworzenie linii autobusowych

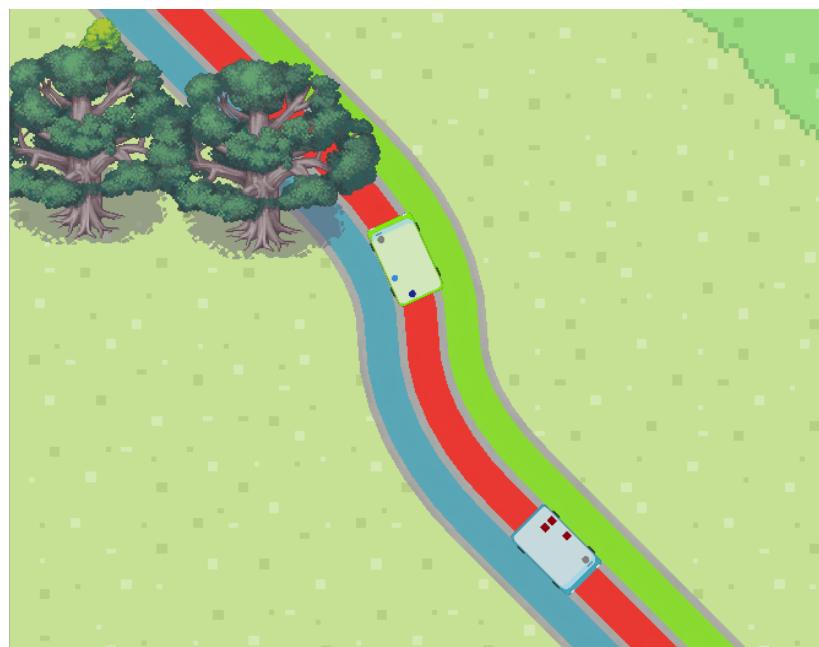
Aby utworzyć linię autobusową, gracz otwiera ekran tworzenia linii, a następnie wybiera przystanki, które mają znaleźć się na trasie linii. Trasa linii musi składać się z przynajmniej dwóch miejscowości. Gracz „klikając” kolejne miasta na mapie, aby włączyć je do trasy. Następujące po sobie miasta muszą być połączone bezpośrednią drogą (tj. nie przechodzącą przez żadne inne miasto) oraz muszą być w nich wybudowane przystanki. Miejscowość może znaleźć się na trasie linii tylko raz (wyjątkiem jest utworzenie pętli, kiedy jako ostatni przystanek gracz wybierze ten sam przystanek co pierwszy dodany).

Każda linia posiada swój unikalny kolor (przypisany przez program) oraz unikalny numer linii. Numer może zostać wybrany przez gracza, jednak musi się mieścić w zakresie [1, 99].

Graczowi udostępniony jest ekran, na którym przedstawione są informacje dotyczące przygotowanych linii. Informacje te to: przystanek początkowy i końcowy, liczba przystanków na trasie oraz liczba kursujących na trasie autobusów. Na tym ekranie linia może zostać usunięta.

Gracz może w dowolnym momencie rozgrywki przypisać autobus do wybranej linii bądź przypisać pojazdowi brak linii. Jeśli linia autobusu zostanie zmieniona podczas jego jazdy, autobus dojedzie do najbliższego przystanku na swojej trasie, pozwoli wysiąść wszystkim pasażerom, a następnie zostanie przeniesiony na odpowiedni przystanek nowo przypisanej linii.

Aby wyróżnić przebieg tras poszczególnych linii, na drogi łączące poszczególne miasta tras nakładane są ścieżki (krzywe) w kolorze przypisanym do linii. Pokazano to na rysunku 4.1. Ponadto autobusy jeżdżące w danej linii też są barwione przypisanym jej kolorem.



RYSUNEK 4.1: Przez przedstawiony na rysunku odcinek drogi przebiegają trzy linie autobusowe: czerwona, zielona i niebieska. Mijające się autobusy przypisane są do linii zielonej i niebieskiej, co można wynioskować z kolorów pojazdów.

4.1.4 Autobusy na trasie

Po przypisaniu autobusu do linii pojazd pojawia się na pierwszym bądź ostatnim przystanku trasy. Jest to zależne od liczby autobusów przypisanych do danej linii — jeśli liczba ta jest nieparzysta, autobus wyrusza z pierwszego przystanku i zmierza w stronę ostatniego miasta trasy. W przeciwnym wypadku autobus rozpoczyna swoją trasę z ostatniego przystanku i zmierza w przeciwnym kierunku.

Wjazd autobusu na przystanek oraz wyjazd z przystanku są animowane. Gdy pojazd zatrzyma się na przystanku, wpierw opuszczają go pasażerowie w nim przebywający, którzy na danym przystanku kończą swoją podróż bądź przesiadają się do autobusu innej

linii. Pasażerowie wysiadają pojedynczo, z krótkim odstępem czasowym i animacją przesuwania się z pojazdu na pole przystanku. Jeśli pasażer dotarł do przystanku docelowego swej podróży, nad jego ikoną pojawia się kwota, którą zapłacił za podróż. Następnie wysiadają do pojazdu pasażerowie oczekujący na autobus wybranej linii. Pasażerowie wysiadają pojedynczo, wybierając miejsce w sposób losowy spośród wolnych. Jeśli liczba podróżnych oczekujących na ten autobus przekracza dostępną liczbę wolnych miejsc, wybór podróżnych, którzy wsiadą do pojazdu, odbywa się drogą losowania.

Autobusy przemierzające odcinki dróg między przystankami poruszają się ze stałą prędkością, która wynika z prędkości przypisanej modelowi pojazdu. Pojazdy swobodnie mijają się na drogach, nie powodując kolizji.

Autobus zatrzymuje się na każdym przystanku. Gdy pojazd dotrze do ostatniego przystanku trasy, zawraca i przemierza trasę w przeciwnym kierunku, odwiedzając przystanki w odwrotnej kolejności od wyznaczonej przez gracza przy tworzeniu trasy. Wyjątkiem jest trasa utworzona jako pętla — wówczas autobus nie zawraca, lecz rozpoczyna swoją trasę od nowa (ponieważ znajduje się w mieście, z którego wyruszył).

4.1.5 Wybór autobusu przez podróżnego

Kiedy pasażer pojawi się na przystanku, wyznaczana jest dla niego lista linii, którymi może dotrzeć do swojego miasta docelowego. Odbywa się to poprzez przeszukanie grafu tras (którego wierzchołkami są przystanki a krawędziami najkrótsze bezpośrednie drogi między miastami, którymi biegną trasy linii). Przeszukiwanie rozpoczyna się w miejscowości, w której podróżny się znajduje, a kończy w miejscowości docelowej jego podróży. Ogólny algorytm szukania najlepszych tras dla pasażera przedstawia algorytm 2. Jego wynikiem jest lista struktur w postaci $\{linia, kierunek, przystanek\}$ *wysiadkowy*, *liczba przesiadek*, gdzie *przystanek wysiadkowy* oznacza miejscowości, w których pasażer musi wysiąść, jeśli zdecyduje się na podróż linią *linia* w określonym kierunku (autobusy na trasach linii jeżdżą w dwóch kierunkach). Lista ta posortowana jest niemalejąco wg liczby przesiadek.

Gdy na przystanku, na którym oczekuje podróżny, pojawi się autobus, sprawdzane jest czy linia, do której przypisany jest dany autobus, znajduje się na liście linii rozpatrywanych przez podróżnego oraz czy kierunek jazdy autobusu jest zgodny z oczekiwaniem. Jeśli pasażer nie jest zniecierpliwiony, wsiadzie tylko do autobusu linii, której pamiętała znaleziona konfiguracja posiada minimalną liczbę przesiadek. Zatem jeśli do wyboru pasażer ma wiele możliwości podróży o takiej samej długości liczonej w liczbie przystanków na trasie, to preferuje te o minimalnej liczbie przesiadek. Gdy pasażer jest zniecierpliwiony,

staje się mniej wybredny i wsiądzie do autobusu z dowolnej linii spośród pamiętanych konfiguracji przejazdu, pod warunkiem, że autobus jedzie w odpowiednim kierunku (w stronę miejscowości docelowej).

Algorytm 2 Ogólny algorytm wyznaczania najlepszych możliwości podróży dla pasażera.

Wejście: Przystanek początkowy S , przystanek docelowy D .

Wyjście: Lista konfiguracji podróży $tracks$ w postaci opisanej w algorytmie.

Metoda:

- 1: Jeśli nie zrobiono tego wcześniej, wyznacz i zapamiętaj wszystkie ścieżki między przystankami S i D o minimalnej długości. Ścieżka rozumiana tu jest jako lista przystanków (wierzchołków) spełniających warunek: z przystanku i można dojechać bezpośrednio do przystanku $i + 1$, tzn. są one na trasie choćby jednej linii autobusowej i na drodze między nimi nie stoi inny przystanek (zatem: w grafie tras linii autobusowych są połączone krawędziami).
 - 2: Dla każdej znalezionej ścieżki wyznacz wszystkie możliwe konfiguracje tras w postaci:
 { *linia autobusowa, do której należy wsiąść na przystanku S,*
 kierunek jazdy autobusu z danej linii,
 przystanek, w którym należy wysiąść (przystanek D lub przesiadkowy),
 liczba przesiadek na trasie }.
 - 3: Za $tracks$ przyjmij listę wyznaczonych konfiguracji podróży posortowaną niemalejąco wg liczby przesiadek.
 - 4: **Zwróć** $tracks$.
-

Przygotowane konfiguracje przejazdów są wyznaczane na żądanie, tj. gdy zachodzi taka potrzeba. Raz wyliczona konfiguracja dla pary miast (startowego i docelowego) jest pamiętaana i gdy kolejny pasażer będzie chciał dojechać z tego samego miasta startowego do tego samego przystanku docelowego, nie ma potrzeby wyznaczania tej listy ponownie.

Wyznaczana w punkcie pierwszym algorytmu 2 lista najkrótszych ścieżek między przystankami odbywa się przy pomocy algorytmu przeszukiwania grafu opartego o przeszukiwanie wszerz. Wyznaczane są niejako przy okazji ścieżki z wierzchołka startowego do wszystkich pozostałych wierzchołków (przystanków), a wynik jest zapamiętywany i używany ponownie w razie potrzeby.

Pamiętane ścieżki oraz konfiguracje najlepszych tras tracą ważność gdy gracz doda nową lub usunie istniejącą linię autobusową. Wtedy przechowywane wyniki są usuwane i gdy zajdzie potrzeba, wyznaczane ponownie. Z tego powodu podróżnemu poszukiwana jest konfiguracja możliwych tras za każdym razem, gdy pojawi się na przystanku, a nie tylko wtedy, gdy zostanie wygenerowany. Gdy podróżny wsiądzie na przystanku przesiadkowym, „rozkład jazdy” obowiązujący w momencie wejścia do autobusu może być już nieaktualny i konieczne może okazać się ponowne „przemyślenie” trasy swojej podróży przez podróżnego.

4.1.6 Płatność za podróż

Przedsiębiorstwo prowadzone przez gracza zarabia pieniądze na przewiezionych pasażerach, którzy inaczej niż w prawdziwym świecie, płacą za podróż dopiero po dotarciu do przystanku docelowego. Na wysokość zapłaty za podróż mają wpływ:

- czas podróży (każdy chce możliwie szybko dostać się do celu),
- pokonana odległość (opisano poniżej),
- komfort podróży (autobusy różnią się oferowaną wygodą oraz stanem technicznym).

Dłuższa trasa wiąże się z wyższą ceną za przejazd, jednak niepotrzebnie wydłużona trasa wpływa negatywnie na cenę biletu. Pasażerowie orientują się w terenie świata, w którym żyją i są w stanie wyznaczyć najkrótszą możliwą trasę do swojego miasta docelowego. Podróżni nie chcą obejodzić mapy dookoła, aby dotrzeć do sąsiedniego miasta. Brany jest zatem pod uwagę stosunek odległości przebytej przez pasażera do odległości spodziewanej (najkrótszej) między miastem, z którego wyruszył a miastem docelowym podróży (liczonej jako liczba pól należących do dróg, po których autobus może się poruszać).

4.1.7 Cierpliwość oczekujących podróżnych

Pasażerowie nie czekają na autobus w nieskończoność. Każdy podróżny charakteryzuje się *cierpliwością* wyrażoną liczbą godzin, jakie może sumarycznie spędzić na przystanku. Liczba ta jest losowana z ustalonego zakresu. Kiedy łączny czas oczekiwania pasażera na przystanku zbliża się do tej wartości, pasażer zaczyna się denerwować, co jest sygnalizowane graczowi animacją „trzęsienia się” pasażera. Kiedy suma czasów oczekiwania przekroczy wartość cierpliwości, pasażer opuszcza przystanek. Oznacza to brak zysku dla gracza. Na wyznaczonych trasach powinna kursować regularnie odpowiednia liczba autobusów, by oczekивание na kolejny autobus danej linii nie trwało zbyt długo.

4.1.8 Implementacja

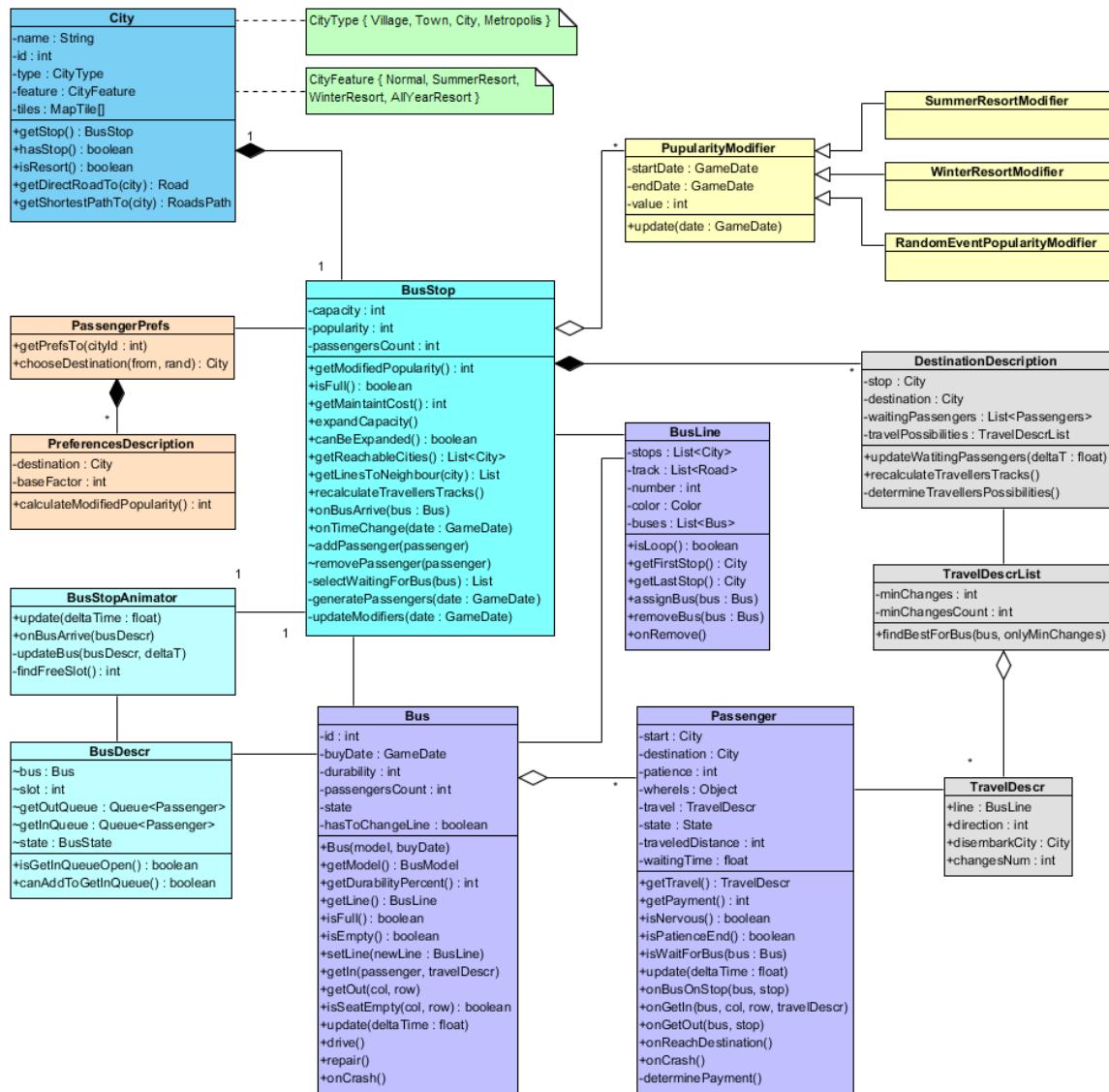
Diagram klas przedstawiony na rysunku 4.2 zawiera zbiór klas związanych z obsługą przystanku autobusowego. W poszczególnych klasach wymieniono tylko najważniejsze pola i metody, pominięto przede wszystkim większość metod pobierających i ustawiających dane, aby nie powiększać diagramu. Główną klasą jest klasa reprezentująca przystanek, `BusStop`. Instancja przystanku jest ściśle powiązana z miastem. Dla ułatwienia implementacji, miasto zawsze posiada przystanek, jednak, jeśli gracz nie posiada przystanku w danym mieście, przystanek jest ustawiony jako nieaktywny i nie jest dostępny ani widoczny dla gracza.

Popularność przystanku wśród podróżnych zdefiniowana jest jako suma wartości modyfikatorów, przechowywanych w obiekcie przystanku w postaci listy. Suma ta jest przycinana do zakresu [0, 5], gdzie 5 oznacza maksymalną popularność. Popularność ma zasadniczy wpływ na liczbę generowanych pasażerów chcących dotrzeć do danego miasta. Na popularność składają się modyfikatory związane z kurortami (letnim, zimowym lub obu, w przypadku kurortu całorocznego) oraz modyfikatory związane ze zdarzeniami losowymi. Każdy modyfikator posiada okres ważności, po którym jest usuwany z listy. Na początku sezonu tworzone i dodawane do listy są modyfikatory kurortów. Wartości poszczególnych modyfikatorów są aktualizowane wraz ze zmianą daty. Dla przykładu, kurort letni jest najbardziej popularny w środku lata, zatem wartość modyfikatora przyjmuje wtedy większą wartość, niż na początku czy pod koniec sezonu.

Obiekt klasy `PassengerPrefs`, związany z konkretnym przystankiem, skupia preferencje podróżnych generowanych na danym przystanku związane z wyborem przystanku docelowego. Wyjaśniano już, że na wybór przystanku docelowego podróżnego mają wpływ takie czynniki, jak bliskość miast i ich wielkość, zdarzenia losowe czy atrakcyjne położenie geograficzne. Tutaj gromadzone są dla każdego możliwego miasta docelowego bazowe wartości popularności, obliczone na podstawie niezmiennych czynników, takich jak odległość między miastami oraz ich typy. Ta bazowa wartość popularności będzie modyfikowana popularnością zmienną w czasie, pobraną z obiektu przystanku rozpatrywanego miasta docelowego, wyznaczoną na podstawie listy modyfikatorów danego miasta.

Pasażerowie oczekujący na przystanku są grupowani względem miast docelowych ich podróży. Jest to podyktowane kwestiami optymalizacyjnymi. Z uwagi, że podróżni wybierają optymalne ścieżki podróży, mogą oni współdzielić informacje dotyczące wyboru linii i oczekiwanych autobusów, ponieważ zamierzają odjechać z tego samego miejsca i dotrzeć do tego samego celu. Obiekt klasy `TravelDescr` przechowuje informacje o możliwości podróży pasażera, takie jak:

- linia, której autobus pasażer powinien wybrać,
- kierunek, w którym autobus powinien jechać (pasażer nie chce wsiąść do autobusu jadącego w przeciwnym kierunku, niż kierunek celu jego podróży),
- miasto, w którym należy wysiąść (to może być miasto przesiadkowe bądź docelowe, jeśli z obecnego miejsca można daną linią w określonym kierunku jazdy dojechać bezpośrednio do celu),
- liczba przesiadek, które pasażer będzie musiał wykonać, jeśli wybierze taką konfigurację podróży.



RYSUNEK 4.2: Diagram klas związanych z przystankiem autobusowym oraz obsługą autobusów i pasażerów.

Klasa **TravelDescrList** ma za zadanie gromadzić możliwe konfiguracje podróży. Wewnątrz klasy są one sortowane wg niemalejącej liczby przesiadek. Liczba pamiętanych konfiguracji podróży o minimalnej liczbie przesiadek jest pamiętana, co ułatwia decyzję, czy pasażer powinien wsiąść do autobusu danej linii. Pasażer, który nie jest zniecierpliwiony, wsiądzie tylko do autobusu, który dowiezie go do celu możliwie krótką trasą o możliwie najmniejszej liczbie przesiadek. Sprawdzenie, czy autobus, który właśnie zatrzymał się na przystanku jest tym, do którego warto wsiąść, jest w takiej reprezentacji danych zadaniem łatwym. Listy najlepszych możliwości podróży między parą miast są wyznaczane tylko wtedy, gdy są potrzebne, ponadto są pamiętane i współdzielone przez pasażerów. Wyznaczane są zatem wtedy, gdy pojawi się pierwszy pasażer chcący dojechać do danego

miasta z rozpatrywanego przystanku. Gdy gracz doda nową trasę lub usunie istniejącą, może zajść potrzeba wyznaczenia nowych list najlepszych konfiguracji tras i aktualnienie stanu wiedzy oczekujących pasażerów.

Pasażerowie wsiadają i wysiadają z autobusów pojedynczo, płynnie, w sposób animowany. Także wjazd pojazdu na przystanek i jego odjazd są animowane. Za takie zachowanie odpowiedzialna jest klasa `BusStopAnimator`. Dla każdego autobusu na przystanku tworzony jest opis (obiekt klasy `BusDescr`) zawierający takie dane dotyczące pojazdu, jak stanowisko, na którym stoi, stan autobusu (np. podjeżdża na stanowisko, wysadza pasażerów, oczekuje na wejście wszystkich wsiadających podróżnych), kolejki wsiadających oraz wsiadających pasażerów. Gdy autobus pojawia się na przystanku, w kolejce wsiadających umieszczeni są pasażerowie, dla których jest to przystanek docelowy lub przesiadkowy, by pojedynczo opuszczać pojazd. Podobnie z pasażerami wsiadającymi do pojazdu, z tym, że nie wszyscy chętni podróżni mogą zmieścić się w autobusie (sytuacja ta została wyjaśniona wcześniej w tym rozdziale). Gdy w trakcie obsługi autobusu na przystanku pojawi się nowy podróżny, dla którego rozważany autobus zawiera się na liście najlepszych konfiguracji podróży, to zostanie dołączony do kolejki wsiadających, pod warunkiem, że autobus będzie w stanie pozwalającym na wejście jeszcze jednego pasażera. Zatem tak zaimplementowana obsługa autobusów pozwala na symulację sytuacji z życia wziętych, takich jak wejście podróżnego do autobusu w ostatniej chwili przed odjazdem.

4.2 Zarządzanie przedsiębiorstwem

Głównym zadaniem gracza jest zarządzanie tworzonymi liniami autobusowymi tak, by przynosiły zyski. Podejmowane decyzje takie jak wybór przystanków, przez które przebiega trasa, dobór autobusów do linii czy decyzja o naprawie, sprzedaży lub kupnie pojazdu powinny być przemyślane, ponieważ jedna zła decyzja może doprowadzić do poważnych strat finansowych i w konsekwencji do porażki.

4.2.1 Finanse

Pieniądze zarabia się z przewozu pasażerów. Dodatkowym źródłem dochodu może być sprzedaż niepotrzebnych autobusów. Ponadto część przychodów może wynikać ze zdarzeń losowych. Zarobione pieniądze można przeznaczać na rozbudowę swojej sieci poprzez budowanie przystanków w miastach, do których przewiduje się duży ruch. Możliwe jest też kupno nowych oraz naprawa posiadanych autobusów.

Każdy autobus oraz przystanek wymaga utrzymania. 4 razy w ciągu miesiąca pewna kwota pieniędzy jest przeznaczana na każdy z nich. Kwoty odprowadzane za utrzymanie

przystanków uzależnione są od ich rozmiaru oraz typu miasta, w którym się znajdują. Kwota utrzymania jednego autobusu jest stała. Ponadto autobusy przemierzające trasę spalają paliwo, z czym wiąże się koszt jego zakupu. Gracz powinien dostosować liczbę autobusów przypisanych do danej linii tak, by nie kursowały puste, co byłoby nieopłacalne.

Dbanie o dodatni stan finansów prowadzonego przedsiębiorstwa jest ważne. Gdy na koncie firmy zabraknie środków, gracz nie może wykonać żadnego działania, które wymaga zapłaty. Wciąż jednak dokonywane są automatyczne opłaty związane z utrzymywaniem autobusów i przystanków. Ponadto stan konta może być pomniejszony o straty wynikające ze zdarzeń losowych (także na debecie), na które gracz nie ma wpływu. Ujemny stan konta jest sygnalizowany czerwonym kolorem tekstu drukowanej na ekranie liczby pieniędzy. Osiągnięcie przez stan konta wartości równej lub niższej -20 tys. skutkuje porażką i zakończeniem gry.

4.2.2 Autobusy

Stan posiadanych przez gracza autobusów wraz z wiekiem oraz przejechaną odlegością ulega stopniowemu pogorszeniu. Kiepski stan pojazdu zwiększa ryzyko wypadku. Autobus, który ulegnie wypadkowi, jest tracony, a pasażerowie nim podróżujący nie dojadą do celu. Stan autobusu przedstawiany jest graczowi w skali procentowej. Stan 100% oznacza idealny stan i cechuje nowy autobus. Stan 0% określa autobus niezdolny do jazdy.

Gracz może naprawiać swoje pojazdy. Cena naprawy zależy od stanu autobusu oraz kwoty zakupu (naprawa droższych modeli kosztuje odpowiednio więcej).

4.2.3 Przystanki

Początkowo gracz ma do dyspozycji kilka (od 4 do 6) przystanków, położonych blisko siebie. Gracz może postawić przystanek w wybranym mieście, w którym jeszcze nie posiada przystanku. Budowa jest jednorazowa, a z posiadanej przystanku nie można zrezygnować. Postawienie przystanku kosztuje, a kwota uzależniona jest od wielkości miejscowości (większe miasta są atrakcyjniejsze dla podróżnych oraz generują więcej pasażerów, zatem postawienie w nich przystanku kosztuje więcej).

Przystanki mogą pomieścić ograniczoną liczbę oczekujących podróżnych. Początkowa pojemność każdego przystanku wynosi 20 miejsc. Wyjątkiem są przystanki w metropolach, które gracz posiada od początku gry — ich pojemność początkowa wynosi 40 pasażerów. Gracz może płatnie rozbudować przystanek, zwiększając jego pojemność o 20 miejsc. Maksymalna pojemność przystanku wynosi 100 miejsc, zatem liczba inwestycji w przystanek jest ograniczona. Rozbudowa jest nieodwracalna, a większy przystanek jest droższy w utrzymaniu.

4.2.4 Statystyki

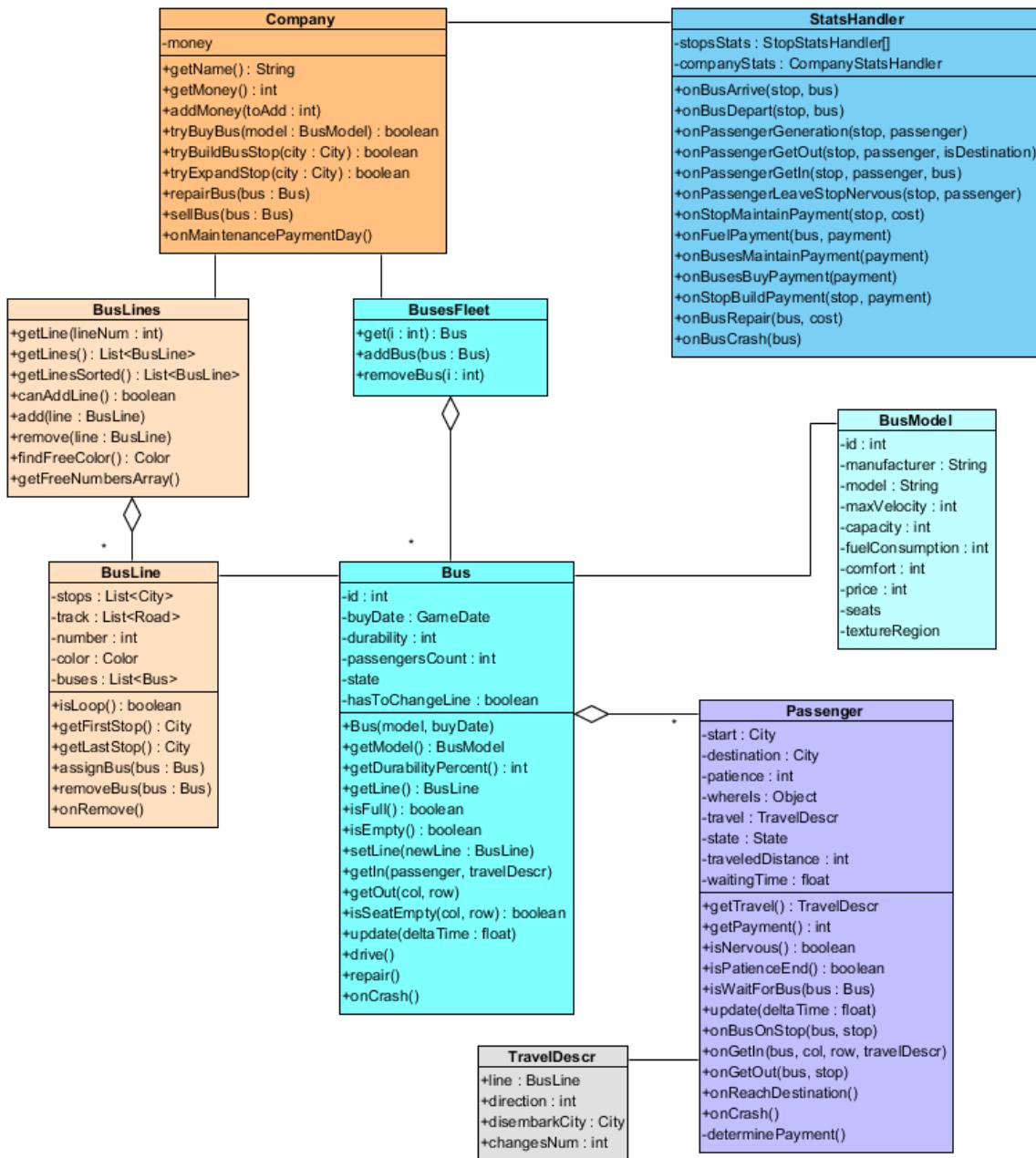
W celu ułatwienia zarządzania udostępniono graczowi szereg statystyk. Dostępne są statystyki dotyczące przewiezionych pasażerów i wykorzystania autobusów, liczby zakupionych i sprzedanych pojazdów oraz postawionych przystanków, poniesionych kosztów oraz osiągniętych przychodów, a także liczby wypadków. Gracz ma możliwość wyboru miast oraz okresu, którego dotyczą statystyki (w formie przedziału miesięcy, od początku gry do aktualnego miesiąca w grze). Przygotowane zostały predefiniowane zakresy czasu (np. bieżący miesiąc, poprzedni miesiąc, bieżący rok) oraz możliwość wyboru wszystkich miast jednym kliknięciem. Miasta posiadające przystanki wyróżnione są ciemniejszym kolorem tekstu. Statystyki przedstawiane są w formie tekstowej.

4.2.5 Implementacja

Na diagramie klas przedstawionym na rysunku 4.3 zamieszczono najważniejsze klasy związane z zarządzaniem przedsiębiorstwem autobusowym. Klasa `Company` udostępnia metody wykonywania działań, takich jak kupno i sprzedaż autobusu, postawienie przystanku czy rozbudowa przystanku. Klasy `BusLines` i `BusesFleet` są kontenerami skupiającymi utworzone przez gracza linie oraz posiadane autobusy. Z autobusem związane są pasażerowie, którzy posiadają zestaw danych dotyczących swojej podróży. Nagłówki metod poszczególnych klas przedstawionych na diagramie wyszczególniają działania i zachowania związane z obiektami tych klas, omówione w tym rozdziale. Niekto z boku umieszczona jest klasa `StatsHandler`, której zadaniem jest gromadzenie danych statystycznych wykorzystywanych, gdy gracz zleci opracowanie raportu. Dane dzielone są na dotyczące poszczególnych przystanków oraz dotyczące ogólnego przedsiębiorstwa. Dane statystyczne przechowywane są z podziałem na miesiące, których dotyczą, co umożliwia przygotowanie raportu z zakresu wskazanych miesięcy.

4.3 Zdarzenia losowe

W toku gry, wraz z upływającym czasem, występować mogą zdarzenia, które mają wpływ na rozgrywkę. Są to zdarzenia, których konsekwencjami mogą być przychody bądź straty finansowe lub czasowa zmiana popularności miast wśród podróżnych. Przykładowo, w pewnym mieście w najbliższym czasie odbędzie się koncert znanego zespołu muzycznego, co skutkuje znaczącym zwiększeniem liczby podróżnych pragnących dostać się do tego miasta, ale tylko w ciągu dwóch najbliższych dni. Gracz może wykorzystać ten fakt do reorganizacji tras swoich linii, by zarobić na przewozie nieoczekiwanych podróżnych. Innym



RYSUNEK 4.3: Diagram klas związanych z prowadzeniem przedsiębiorstwa autobusowego.

przykładem wydarzenia losowego jest wymiana komputerów w biurze, która wiąże się z pewnym narzuconym kosztem (na który gracz nie ma wpływu).

Gracz informowany jest o wydarzeniu ikonką wiadomości pojawiającą się w rogu ekranu, gdy tylko nastąpi zdarzenie. Na niej wyświetlona jest także liczba nieprzeczytyanych wiadomości. Po kliknięciu ikony otwiera się okno z treścią najstarszej nieprzeczytanej wiadomości oraz informacją o skutkach.

Wydarzenia czasowo podnoszące popularność miejscowości wśród pasażerów, takie

jak koncert, mecz lub wyjątkowo dobra pogoda (możliwe w miesiącach letnich), mogą dotyczyć wszystkich miast na mapie, nie tylko tych, na których gracz posiada przystanek. Może to zachęcić gracza do postawienia przystanku w danym mieście.

Zdarzeniem losowym jest także wypadek autobusu, jednak w tym przypadku częstość występowania zdarzenia zależna jest od stanu pojazdów, na który gracz ma wpływ.

Rozdział 5

Testy aplikacji

W celu zbadania poprawności działania aplikacji oraz jej wydajności przeprowadzono szereg testów. Autorowi pracy pomagała w testach funkcjonalnych mała grupa testerów. Ponadto zbadano zysk osiągnięty dzięki użyciu generowania proceduralnego do programowego przygotowywania zasobów gry.

5.1 Metodyka testowa

Aby możliwie dobrze przetestować opracowywaną aplikację, zdecydowano się przeprowadzić wiele różnorodnych procedur testowych. Testy funkcjonalne, oparte o scenariusze napisane przez autora pracy, miały za zadanie sprawdzić, na ile udało się zrealizować wymagania wynikające z opisu gry. Wciąż możliwe było wykrycie i poprawienie znalezionych błędów.

Testy wydajnościowe miały pomóc ocenić, jak dobrze aplikacja jest zoptymalizowana pod kątem działania na docelowych systemach i klasach urządzeń. Ponadto dokonano analizy zawartości otrzymanych plików wynikowych oraz rozmiarów aplikacji dla każdej docelowej platformy.

Część kodu aplikacji pokryta została testami jednostkowymi. Poziom pokrycia testami okazał się jednak na tyle niski, iż autor zdecydował się nie opisywać ich w pracy, ponieważ ich wpływ był dla projektu marginalny. Wyniki generowania proceduralnego trudno jest przewidzieć, zatem testowanie tą metodą nie byłoby dobrym wyborem dla testów klas generatorów. Wiele z funkcji gry opiera się na wykorzystaniu losowości, zatem ich testowanie też nie było łatwe. Dla przykładu, sprawdzono, czy podróżni wybierają miejscowości docelowe swoich podróży zgodnie z ustalonymi zasadami (i wynikającym z nich prawdopodobieństwem), ale popularność miast wśród podróżnych mogła być modyfikowana przez zdarzenia losowe oraz upływ czasu, zatem trzeba było uważnie analizować wyniki, biorąc

wymienione czynniki pod uwagę. Głównym sposobem testowym dla generatorów oraz dla wspomnianych mechanizmów były zatem testy funkcjonalne oraz wnikliwe obserwowanie uzyskiwanych rezultatów i porównywanie z założeniami.

5.2 Testy funkcjonalne

Poniżej zamieszczono scenariusze testów przeprowadzonych przez trzy osoby z otoczenia autora pracy. Wszystkie testy wykonywane były pod nadzorem autora. Osoby testujące nie posiadały wiedzy na temat budowy programu. Testerzy byli zaznajomieni z instrukcją do gry. Punkty scenariuszy zawierają akcję, którą wykonuje użytkownik (gracz) oraz reakcję programu i spodziewany widok po wykonaniu akcji. Scenariusze zostały opracowane dla wersji gry uruchamianych na komputerze osobistym.

5.2.1 Rozpoczęcie gry

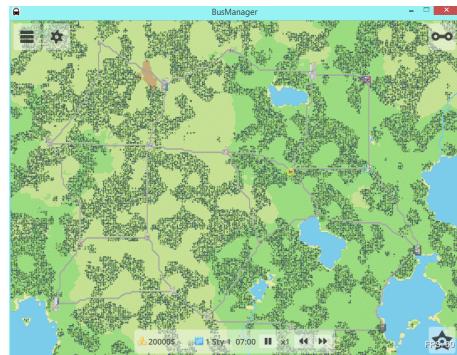
Cel testu: przygotowanie rozgrywki na mapie utworzonej z użyciem podanego przez gracza ziarna generatora świata gry.

Testowane zagadnienia: rozpoczęcie nowej gry, stworzenie świata gry z podaniem własnego ziarna dla generatora, zamiana ziarna w postaci tekstu na liczbę, odczytanie użytego ziarna i użycie go ponownie do wygenerowania takiej samej mapy, opuszczenie rozgrywki.

Warunki początkowe: otwartym ekranem jest ekran tworzenia nowej gry.

Scenariusz testu:

1. W polu tekstowym podpisany *Ziarno* gracz wpisuje ziarno generatora świata gry w postaci tekstu (np. **autobus**).
2. Gracz naciska przycisk *Stwórz świat*. Ekran zostaje zmieniony na ekran oczekiwania na przygotowanie gry. Widoczny jest aktualizowany pasek postępu.
3. Po przygotowaniu gry przez program graczowi ukazuje się ekran rozgrywki. Mapa widoczna po wpisaniu ziarna **autobus** przedstawiona została na rysunku 5.1.
4. Gracz naciska przycisk menu umieszczony w lewym górnym rogu ekranu. Otwiera się okno z dostępnymi pozycjami menu.



RYSUNEK 5.1: Mapa uzyskana używając ziarna generatora **autobus**.

5. Gracz wybiera pozycję *Ziarno* i odczytuje liczbę użytą jako ziarno generatora świata gry. Dla ziarna w postaci tekstu **autobus** wartość liczbową wynosi 7341.
6. Gracz naciska przycisk *OK*. Zamyka się okienko, w którym wyświetcone było ziarno generatora. Program wraca na ekran menu.
7. Gracz wybiera z otwartego menu gry pozycję *Opuść grę*. Graczowi ukazuje się okno dialogowe, na którym może potwierdzić chęć opuszczenia gry bądź wycofanie się ze swojej decyzji i powrót do menu.
8. Gracz naciska przycisk *Opuść grę*. Gra wraca na ekran tworzenia rozgrywki.
9. Gracz wpisuje jako wartość ziarna generatora liczbę odczytaną w punkcie 5, po czym naciska przycisk *Stwórz świat*. Gra przechodzi na ekran oczekiwania na przygotowanie gry.
10. Po stworzeniu gry graczowi ukazuje się dokładnie taka sama mapa, jak w punkcie 3.

5.2.2 Kupno autobusu

Cel testu: powiększenie zbioru posiadanych autobusów poprzez zakup pojazdu.

Testowane zagadnienia: panel zarządzania, okno floty (posiadanych autobusów), okno zakupu autobusu, mechanizmy zarządzania oknami, zakup autobusu, wydawanie pieniędzy.

Warunki początkowe: gracz znajduje się na ekranie rozgrywki, tuż po jej rozpoczęciu, i nie wykonał jeszcze żadnej akcji.

Scenariusz testu:

1. Gracz naciska przycisk otwierający panel zarządzania (tuż obok przycisku menu). Panel wysuwa się z góry ekranu.
2. Z otwartego panelu gracz wybiera pozycję *Posiadane busy*. Panel zarządzania zostaje zwinięty. Otwiera się okno floty przedstawiające autobusy posiadane przez gracza.
3. Gracz przewija zawartość okna oraz liczy dostępne autobusy.
4. Gracz otwiera ponownie panel zarządzania. Tym razem wybiera pozycję *Kupno busa*. Program zwija panel zarządzania i otwiera okno przedstawiające modele autobusów, które można zakupić. Otwarte okno przesłania okno floty (rys. 5.2).
5. Gracz chwyta za pasek tytułu okna kupna autobusu i przeciąga okno na prawą stronę ekranu, odsłaniając zakryte okno floty (rys. 5.2).



RYSUNEK 5.2: Otwarte okna posiadanych autobusów oraz kupna busa.

6. Gracz z okna kupna autobusu wybiera interesujący go model, po czym naciska zielony przycisk kupna, wyświetlający cenę zakupu. Zmienia się kwota posiadana przez gracza (wyświetlana na pasku informacyjnym, u dołu ekranu). Okno posiadanych autobusów zostaje odświeżone i do listy pojazdów zostaje dodany (jako ostatni element listy) zakupiony autobus. Gracz sprawdza to poprzez ponowne zliczenie autobusów przedstawionych w oknie floty.
7. Gracz zamyka okno kupna autobusu przyciskiem w prawym górnym rogu okna. Okno znika z ekranu.
8. Gracz otwiera panel zarządzania i otwiera okno zakupu autobusu. Otwiera się zamknięte przed chwilą okno i znika panel zarządzania.
9. Gracz ponownie otwiera panel zarządzania i znów wybiera opcję *Kupno busa*. Okno nie zostaje otwarte ponownie, jeśli wcześniej jest otwarte. Panel zarządzania zostaje zamknięty.

5.2.3 Tworzenie linii

Cel testu: przygotowanie linii autobusowej.

Testowane zagadnienia: ekran tworzenia linii, poruszanie się po mapie (przesuwanie, zoom), interakcja z elementami mapy (tutaj: miastami), sprawdzenie poprawności warunków dodawania miast do trasy tworzonej linii, ustawienie numeru linii.

Warunki początkowe: gra znajduje się w stanie tuż po rozpoczęciu rozgrywki, w której przygotowaniu wykorzystano ziarno autobus.

Scenariusz testu:

1. Gracz otwiera ekran tworzenia linii (przyciskiem znajdującym się w prawym dolnym rogu ekranu). Ukryte zostają wszystkie widoczne elementy interfejsu. U góry ekranu pojawia się panel tworzenia linii. Upływ czasu w grze zostaje wstrzymany.
2. Gracz przybliża kółkiem myszy (bądź klawiszem plus) mapę tak, by podpisy miast były dobrze widoczne. Mapa gry zostaje powiększona.
3. Gracz, używając klawiszy WSAD (bądź klawiszy strzałek), przesuwa mapę tak, by wyśrodkować widok na mieście J (miasto to posiada przystanek i ma niebieską obwódkę wokół swojego obszaru, a jego ikoną jest jasnoniebieski kwadrat).



RYSUNEK 5.3: Panel tworzenia linii. Dodano dwa przystanki do tworzonej trasy.

4. Gracz kliką na obszar miasta (np. na wybrany budynek). W panelu tworzenia linii pojawia się ikona miasta symbolizująca pierwszy przystanek tworzonej trasy.
5. Gracz przesuwa mapę trzymając klawisz A (lub strzałkę w lewo) tak dugo, aż na ekranie pojawi się miasto I . Mapa świata gry zostaje przesunięta.
6. Gracz dodaje miasto I do trasy tworzonej linii w podobny sposób, w jaki dodał do niej poprzednie miasto. Miasto to posiada przystanek, więc zostaje dodane do trasy. W panelu tworzenia linii pojawia się ikona wybranego miasta, tuż obok ikony poprzednio dodanego przystanku. Ponadto na mapie bezpośrednia droga pomiędzy miastami I oraz J zostaje zaznaczona kolorem ciemnoróżowym (tak, jak pokazano na rysunku 5.3).
7. Gracz znów przesuwa kamerę w lewo. Znajduje miasto H i próbuje dodać je do trasy linii, jednak nic się nie dzieje. Miasto H nie posiada przystanku, zatem nie może zostać włączone do trasy.
8. Gracz przesuwa mapę, odnajdując miasto N . Dodaje je do trasy, analogicznie jak poprzednie miasta. W panelu tworzenia linii widać już trzy ikony przystanków, a droga pomiędzy miastami J oraz N jest wyróżniona kolorem.
9. Gracz ponownie kliką na mapie na miasto N . Przystanek ten zostaje usunięty z tworzonej trasy. W panelu tworzenia linii pozostają dwa pierwsze dodane przystanki. Droga pomiędzy miastami J oraz N jest już rysowana bez kolorowania.
10. Gracz w panelu tworzenia linii rozwija pole wyboru oznaczone etykietą *Linia numer*. Pojawia się lista z numerami, którymi można oznaczyć trasę. Zawiera ona wszystkie liczby całkowite od 1 do 99 włącznie.
11. Gracz przydziela linii numer 7, klikając odpowiednią pozycję z rozwiniętej listy. Lista znika, a w polu, w którym wcześniej był wpisany numer 1, pojawia się liczba 7.
12. Gracz kliką zielony przycisk z napisem *Utwórz*. Linia zostaje utworzona. Zamknięta się panel tworzenia linii, pojawiają się ukryte w punkcie 1. elementy interfejsu, a upływ czasu gry zostaje przywrócony. Pojawia się też okno posiadanych autobusów, umożliwiając graczowi szybkie przypisanie autobusów do stworzonej linii.

5.2.4 Przewóz podróżnych

Cel testu: przewiezienie podróżnych między miejscowościami osiągalnymi przez utworzone trasy.

Testowane zagadnienia: zmiana tempa upływu czasu w grze, wstrzymywanie upływu czasu, panel linii, przypisanie autobusu do linii, wybór linii i autobusu przez pasażera, poruszanie się autobusu, wsiadanie i wysiadanie pasażerów, opłata za przejazd, przesiadki.

Warunki początkowe: gra znajduje się w stanie wynikającym z przeprowadzenia poprzedniego testu (*Tworzenie linii*). Otwarte jest okno autobusów gracza, tuż po utworzeniu linii.

Scenariusz testu:

1. Gracz przypisuje do linii 7 autobus *Sun U323* (drugi od góry na liście autobusów) poprzez kliknięcie przycisku z etykietą *brak* przy informacjach o tym pojeździe. Z rozwiniętego menu gracz wybiera pozycję *Linia 7*. Przycisk zmienia kolor na czerwony oraz etykietę na *Linia 7*, co oznacza poprawne przypisanie pojazdu do linii.
2. Gracz stopniowo zwiększa tempo upływu czasu, klikając na ikonę z dwoma trójkątami skierowanymi w prawą stronę, umieszczoną na panelu informacyjnym u dołu ekranu. Z każdym kliknięciem czas wyświetlany na tymże panelu zmienia się coraz szybciej. Gracz testuje kolejne prędkości aż do ustawienia x32. Po ponownej próbie przyspieszenia tempa upływu czasu gra powraca na tempo x1.
3. Gracz otwiera panel tworzenia linii. Zamknięte zostaje okno autobusów gracza. Gracz tworzy linię niebieską o dwóch przystankach, łączącą miasta O oraz J. Dokonuje tego w sposób wyjaśniony w poprzednim teście. Zmienia także numer linii na 2.
4. Gracz wstrzymuje upływ czasu w grze poprzez naciśnięcie przycisku pauzy na panelu u dołu ekranu. Przycisk ten zmienia wyświetlaną grafikę z dwóch pionowych prostokątów na trójkąt. Zaobserwować można, że data i czas przedstawiane na panelu informacyjnym przestały się zmieniać, a autobus kursujący po linii 7 przestał się poruszać.
5. W otwartym oknie floty gracz przypisuje autobus *Autosam Z531* (czwarty na liście, licząc od góry) do utworzonej linii 2, w analogiczny sposób, jak w punkcie 1.
6. Gracz zamyka okno floty i otwiera panel linii (przyciskiem w prawym górnym rogu ekranu gry). Z prawej strony ekranu wysuwa się panel linii. Gracz odczytuje z niego informacje dotyczące istniejących połączeń autobusowych, takie jak: numery i kolory



RYSUNEK 5.4: Obsługa pasażerów na przystanku po przyjeździe autobusu.

linii, miasta początkowe i końcowe tras, liczby przystanków oraz liczby przypisanych do linii autobusów. W tym momencie utworzone są dwie linie: linia 7 (czerwona) oraz linia 2 (niebieska), każda składająca się z dwóch przystanków na trasie oraz z jednym przypisanym autobusem.

7. Gracz zamyka panel linii poprzez ponowne naciśnięcie przycisku, którym go otworzył. Panel przesuwa się w prawą stronę, aż zostaje całkowicie schowany.
8. Gracz zbliża kamerę na miasto O. Powinien ujrzeć nad miastem biały prostokąt z ikonami miast oraz liczbami nad nimi. Liczby te określają liczbę podróżnych oczekujących do miasta wskazanego przez daną ikonkę miejscowości.
9. Gracz wznawia upływ czasu poprzez ponowne naciśnięcie przycisku, którym wstrzymał czas w punkcie 4. Gracz obserwuje wjazd busa na przystanek w mieście O.
10. Gracz obserwuje wsiadających pasażerów, przesuwających się kolejno z pola przystanku do autobusu. Wsiągają tylko podróżni chcący dotrzeć do miejscowości I lub J. Pozostali podróżni pozostaną na przystanku, ponieważ podstawionym autobusem nie dotrą do swoich miejscowości docelowych.
11. Gracz obserwuje odjazd autobusu z przystanku oraz śledzi jego dalszy ruch, do następnego przystanku, jakim jest miejscowości J. Autobus porusza się płynnie wzduż drogi między tymi miastami.
12. Gracz obserwuje wjazd autobusu na przystanek w mieście J oraz wysiadających z pojazdu pasażerów. Nad ikonami podróżnych, którzy zmierzali do miasta J, pojawiają się przesuwające się w górę kwoty wskazujące na zapłatę za przejazd uiszczoną przez pasażera (tak, jak pokazano na rysunku 5.4). Nad ikonami wysiadających pasażerów podróżujących do miasta I nie pojawia się kwota, ponieważ ci pasażerowie przesiadają się do autobusu linii 7. Ci podróżni pojawiają się na przystanku J, oczekując kolejnego autobusu. Pasażerowie, którzy osiągnęli swój przystanek docelowy (w tym wypadku w miejscowości J), znikają z mapy. Autobus linii 2 odjeżdża w stronę przystanku, z którego przyjechał, ponieważ osiągnął koniec swojej trasy i teraz przemierzy ją w przeciwnym kierunku.
13. Gracz obserwuje dalej przystanek J i oczekuje, aż przyjedzie autobus linii 7 (czerwonej). Do tego pojazdu wsiągają oczekujący na przesiadkę pasażerowie oraz pojawiający się na przystanku nowi podróżni chcący dojechać do miasta I. W razie potrzeby gracz może zwiększyć tempo upływu czasu.
14. Gracz obserwuje, czy pasażerowie, którzy przesiedli się w mieście J, dotarli do miasta I. Pasażerowie ci płacą za przejazd podczas wysiadania.

5.3 Testy wydajnościowe

W celu zbadania wydajności działania gry na różnych platformach systemowych i sprzętowych, przygotowano specjalny tryb działania aplikacji, którego zadaniem było zliczanie liczby wyświetlanych klatek oraz czasu działania aplikacji. Przy zamykaniu gry program drukował w konsoli zmierzone wartości oraz wyliczoną średnią liczbę klatek wyświetlanych w ciągu sekundy działania aplikacji (*framerate*).

Testy przeprowadzono na poniżej przedstawionych urządzeniach:

- Komputer klasy PC

Procesor Intel Core i3-3220 (3,3 GHz, 3 MB Smart Cache, 64-bit), 8 GB pamięci RAM DDR3, karta graficzna NVIDIA GeForce GT220 1 GB RAM, system operacyjny Windows 8.1, przeglądarka internetowa Google Chrome w wersji 40.0.2214, rozmiar okna gry 800 x 600 pikseli.

- Smartfon LG G2 (D802)

Procesor Qualcomm Snapdragon 800 2,26 GHz, GPU Adreno 330, 2 GB RAM, rozdzielcość ekranu 1920 x 1080 pikseli, system Android 4.4.2.

- Smartfon Samsung Galaxy Young (GT-S6310)

Procesor Qualcomm MSM7227A 1,00 GHz, GPU Adreno 200, 768 MB RAM, rozdzielcość ekranu 480 x 320 pikseli, system Android 4.2.

Na wszystkich wyżej wymienionych platformach przeprowadzono ten sam test, którego scenariusz przedstawał się następująco:

1. Uruchomienie aplikacji. Stworzenie nowej gry, używając ziarna **test123**.
2. Otworzenie okna posiadanych autobusów i sprzedanie trzech pierwszych pojazdów, licząc od początku listy.
3. Stworzenie trasy łączącej miasta F, G i B, w takiej kolejności. Przydzielenie do tej trasy dwóch autobusów, które graczowi pozostały.
4. Zmiana prędkości upływu czasu na x4.
5. Zakup 4 autobusów modelu *Mircodes X433*.
6. Stworzenie nowej linii o trasie G-H-I. Przydzielenie do niej dwóch dostępnych autobusów.
7. Stworzenie kolejnej linii relacji L-G-H. Przydzielenie do niej dwóch pozostałych autobusów.
8. Ustawienie prędkości upływu czasu na x16. Oddalenie kamery na maksymalną wartość.

9. Przeczekanie do daty 10 stycznia, obserwując sytuację w grze.
10. Gdy nastąpi data 10 stycznia, ustawienie prędkości upływu czasu na x1.
11. Usunięcie linii 2. Sprzedaż jednego z autobusów nieprzypisanych do żadnej linii.
12. Postawienie przystanku w mieście K.
13. Stworzenie nowej linii o trasie L-K-F i przydzielenie do niej dostępnego autobusu.
14. Wejście w okno statystyk. Wygenerowanie statystyk z całego okresu gry, dla wszystkich miast. Przejrzenie statystyk i zamknięcie okna po ok. 10 sekundach.
15. Zwiększenie prędkości upływu czasu do x32. Maksymalne oddalenie widoku kamery.
16. Pozostawienie symulacji aż łączny czas gry wyniesie 10 minut. Odczytywanie pojawiających się w tym czasie komunikatów o wydarzeniach. Tester mógł teraz swobodnie poruszać się po interfejsie gry, jednak nie mógł wykonywać zadań skutkujących wstrzymaniem upływu czasu (jak wejście do menu czy otworzenie panelu tworzenia trasy). Zabronione było też wstrzymywanie upływu czasu gry.

Test trwał ok. 10 minut. Przeprowadzono procedurę testową dla każdej przygotowywanej wersji aplikacji. Wersje „desktopową” oraz HTML5 (uruchomioną w przeglądarce Google Chrome) testowano na komputerze PC, natomiast wersje dla smartfonów z systemem Android przetestowano na dwóch urządzeniach, z których jedno (LG G2) jest stosunkowo nowym urządzeniem (w momencie przeprowadzania testów), natomiast drugie (Samsung Galaxy Young) jest kilkuletnim modelem. W trakcie przeprowadzania testów ograniczono do niezbędnego minimum liczbę aplikacji działających w tle na używanym urządzeniu.

Dzięki użyciu tego samego ziarna generatora świata gry na każdej testowanej platformie, test mógł być przeprowadzany w taki sam sposób. Rezultaty testów przedstawiono w tabeli 5.1.

Warto zaznaczyć, że używana biblioteka libGDX ogranicza maksymalną możliwą do uzyskania liczbę klatek na sekundę do 60. Ponadto cała aplikacja jest jednowątkowa, co było koniecznością wynikającą z użycia technologii GWT.

TABLICA 5.1: Wyniki testów wydajnościowych.

Urządzenie	Wersja gry	Czas trwania testu [s]	Liczba wyświetlonych klatek	Średnia liczba kl.\s.
PC	Desktop	601,64	35949	59,75
PC	HTML5	607,61	34516	56,86
LG G2	Android	608,34	35189	57,88
Galaxy Young	Android	606,32	16234	26,79

Z przedstawionych rezultatów wynika, że w każdej z opracowywanych wersji gry można z powodzeniem grać z niemal najwyższą możliwą do uzyskania wydajnością. Komputer PC, na którym przeprowadzano testy, jest kilkuletnim, używanym sprzętem i jego konfigurację można określić jako przeciętną. Mimo to gra w wersji uruchamianej z pliku JAR uzyskała najlepsze wyniki. Wersja HTML uruchomiona na tym samym komputerze uzyskała nieco gorszy wynik, co może wynikać z natury aplikacji online, którą to stanowi aplikacja JavaScript uzyskana z kodu języka Java (używając narzędzia GWT, o którym pisano w pierwszym rozdziale). Aplikacja ta działa po stronie klienta, więc nie należy tu się doszukiwać opóźnień związanych z działaniem sieci Internet.

Opracowana aplikacja mobilna uruchomiona na nowszym z używanych smartfonów testowych również uzyskała bardzo dobry wynik. Znacznie gorzej przedstawia się rezultat uzyskany na urządzeniu Samsung Galaxy Young. Smartfon ten posiada słabszy procesor, od tego użytego w urządzeniu LG. Mimo to, wg subiektywnej oceny osoby testującej, uzyskana wydajność działania aplikacji była wystarczająca do prowadzenia satysfakcjonującej rozgrywki.

5.3.1 Dodatkowy test funkcjonalny

Podczas przeprowadzania testów wydajnościowych, oprócz wyników tychże testów, odnotowano także informacje o stanie prowadzonej rozgrywki w ostatnim momencie przed zakończeniem każdego z testów. Zgromadzone dane przedstawiono w tabeli 5.2.

TABLICA 5.2: Stany gry w momencie zakończenia testów.

Urządzenie	Wersja gry	Data zakończenia gry	Stan konta
PC	Desktop	2 kwietnia 1 r. 2:00	2381\$
PC	HTML5	11 kwietnia 1 r. 5:00	-9323\$
LG G2	Android	3 kwietnia 1 r. 16:00	2448\$
Samsung Galaxy Young	Android	20 marca 1 r. 5:00	-5971\$

Pomimo przeprowadzenia dokładnie tego samego scenariusza testowego na różnych konfiguracjach testowych, uzyskano różne daty zakończenia gry. Gra zawsze rozpoczynała się 1 stycznia 1 r., na domyślnej prędkości upływu czasu (1x). Grę zakończono z najwcześniejszą datą na smartfonie Galaxy Young, co wynika z faktu, iż na tym urządzeniu aplikacja działała naj wolniej, zatem czynności zawarte w scenariuszu zajmowały adekwatnie więcej czasu. Najwięcej czasu gry upłynęło podczas testu gry w przeglądarce. Może to wynikać z faktu, iż test ten przeprowadzany był jako ostatni, a osoba testująca aplikację nabrała wprawy w przeprowadzaniu testu polegającego na wykonaniu tego samego scenariusza w każdej wersji gry. Zatem dobre poznanie interfejsu gry może prowadzić do szybszej rozgrywki.

Uzyskane stany kont diametralnie różnią się od siebie. Wynikać może to różnych zdarzeń losowych występujących podczas gry. Zdarzenia te skutkowały zyskami bądź stratami dla gracza. Ponadto, pomimo tworzenia takich samych tras dla autobusów, generowani podróżni mogli wybierać inne cele podróży, także takie, przez które nie kursowały autobusy wyznaczonych tras. Pomimo gry w tym samym środowisku (na tej samej mapie), rozgrywka toczy się nieprzewidywalnie, co było jednym z celów projektowych gry.

5.4 Analiza rozmiaru aplikacji

W celu zbadania zysku w obszarze rozmiaru aplikacji uzyskanego z użycia generowania proceduralnego do tworzenia części używanych w grze zasobów, przeanalizowano zawartość wykonywalnego pliku JAR, stanowiącego wersję aplikacji przygotowaną dla komputerów PC. Ponadto zestawiono rozmiary plików dystrybucyjnych aplikacji dla wszystkich przygotowanych wersji gry i zestawiono je w tabeli 5.3.

TABLICA 5.3: Rozmiary poszczególnych wersji aplikacji.

Wersja gry	Format wyjściowy	Rozmiar [MiB]
Desktop	Wykonywalny plik JAR	8,49
Android	Plik APK	2,47
HTML5	Folder z plikami strony WWW	10,00

Pliki dystrybucyjne dla platform PC oraz Android to pojedyncze pliki, które łatwo dystrybuować wśród potencjalnych graczy. W przypadku PC, jest to specjalny plik JAR, którego wykonanie powoduje uruchomienie gry (bez instalacji). W przypadku wersji gry dla systemu Android, konieczna jest instalacja gry z przygotowanego pliku APK. Po instalacji gra zajmuje w pamięci urządzenia 5,67 MiB.

Wygenerowana „przeglądarkowa” wersja gry to zestaw plików, które należy umieścić na serwerze w Internecie (możliwe jest uruchomienie lokalne używając serwera, np. *mongoose*¹). Pliki zasobów gry zajmują tyle samo miejsca, co w przypadku pozostałych wersji aplikacji, ponieważ są zapisane w niezmienionych formatach.

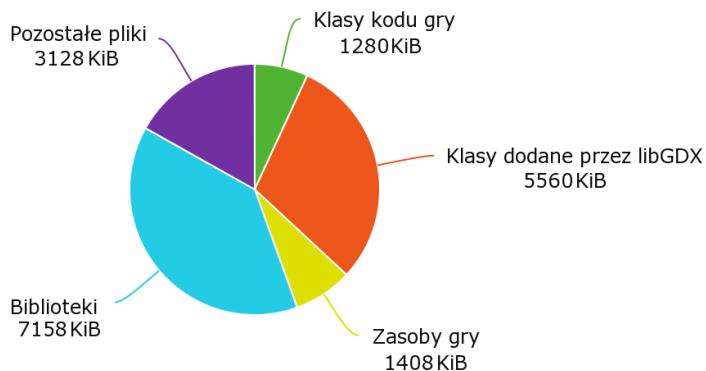
Analizę zawartości pliku JAR przedstawiono w formie wykresu na rysunku 5.5. Po rozpakowaniu pliku uzyskano katalog o rozmiarze 18,1 MiB. Jak można odczytać z wykresu, pliki klas aplikacji oraz pliki zasobów gry stanowią niewielką część całego pliku wykonywalnego aplikacji. Pliki zasobów gry, do których zaliczyć można tekstury, ikony, fonty, skórki i shadery, zajmują łącznie 1408 KiB, co stanowi 8% rozmiaru rozpakowanego, analizowanego pliku. Taki rozmiar dołączonych zasobów uznać można za dobry rezultat, ponieważ w przypadku większości gier to właśnie zasoby graficzne mają największy wpływ

¹<https://code.google.com/p/mongoose/>

na rozmiar aplikacji. Tekstury budynków, poszczególnych autobusów czy pliki zawierające struktury mapy gry nie są dołączone do zasobów, a są tworzone w trakcie działania aplikacji. Jednym z celów takiego podejścia była redukcja rozmiaru pliku wynikowego gry i można stwierdzić, że cel ten udało się zrealizować.

Spośród 1280 KiB zajmowanych przez klasy kodu gry, 166 KiB zajmowane jest przez klasy generatorów używanych w procesie przygotowania świata dla tworzonej rozgrywki. Jako że ziarno używane w procesie tworzenia mapy jest reprezentowane przez 32-bitową liczbę całkowitą ze znakiem, możliwe jest uzyskanie $2^{32} - 1$ map (dopuszczalne są ziarna o ujemnej wartości). Ręczne przygotowanie choćby kilkudziesięciu map zajęłoby autorowi pracy wiele godzin, a pliki opisujące struktury map zwiększyłyby rozmiar aplikacji. Największą niekorzyścią spowodowaną dołączeniem map do aplikacji, zamiast ich generowania, byłoby jednak znaczące ograniczenie ich liczby.

30% zawartości pliku stanowią klasy dodane przez libGDX, które stanowią szkielet aplikacji oraz odpowiadają za niskopoziomowe działanie (m.in. obsługę wejścia czy obsługę OpenGL). Pod nazwą *biblioteki* zawarto na wykresie łączny rozmiar dołączonych plików bibliotecznych odpowiedzialnych m.in. za działanie aplikacji na wielu systemach operacyjnych (Windows, Linux, Mac) oraz obsługę OpenGL i OpenAL. Biblioteki te są automatycznie dołączane przez libGDX, podobnie, jak pliki z kategorii *pozostałe*. Wśród nich zawarto dodatkowe zasoby, takie jak domyślny font libGDX czy przykładowe tła okien i przycisków. W opracowanej aplikacji zasoby te są zbędne, jednak nie znaleziono sposobu na wyłączenie automatycznego dołączania ich do pliku wynikowego.



RYSUNEK 5.5: Wykres przedstawiający wynik analizy zawartości rozpakowanego pliku JAR opracowanej aplikacji.

Rozdział 6

Podsumowanie

W efekcie prac nad projektem powstała gra będąca przykładem wykorzystania generowania proceduralnego do programowego tworzenia typowych elementów gry. Autorowi udało się zrealizować postawione przed nim cele. Dla każdej nowej rozgrywki przygotowywana jest mapa, która w widocznym stopniu różni się od otrzymywanych w pozostałych prowadzonych rozgrywkach, spełniając jednocześnie ustalone założenia dotyczące mapy świata gry. Gracz ma też możliwość ponownej gry na tej samej mapie, po podaniu znanego ziarna generatora. Ponowna rozgrywka na tej samej mapie może jednak wyglądać zupełnie inaczej, co zaobserwowano i odnotowano w testach. Udało się też spełnić założenia stawiane przed logiką gry. Pojawiający się na przystankach wraz z upływem czasu podróżni wybierają swoje cele podróży zgodnie z przyjętymi w założeniach gry kryteriami, wsiadając do autobusów mogących dowieźć ich do celu, być może z przesiadką. Gracz do pomocy w zarządzaniu dostał zestaw statystyk oraz pewne formy śledzenia sytuacji w grze, jak możliwość podglądu zapełnienia autobusów i przystanków w czasie rzeczywistym. Działania te przetestowano na drodze testów funkcjonalnych.

Opracowana aplikacja działa na wielu platformach sprzętowych i systemowych, a jej wydajność sprawdzono w toku testów. Dzięki użyciu biblioteki libGDX oraz technologii GWT, opracowanie gry wieloplatformowej miało ograniczyć się do napisania wspólnego kodu aplikacji w jednym języku oraz do uruchomienia dostarczonych mechanizmów komplikacji, które miały wykonać za programistę działania niezbędne do przygotowania poszczególnych wersji aplikacji dedykowanym obsługiwany systemom. Niestety, praktyka pokazała, że nie zawsze jest to takie proste. W trakcie pracy nad projektem konieczne okazało się napisanie dodatkowego kodu specjalnie dla wersji HTML5. Spowodowane było to faktem, iż część potrzebnych klas języka Java wciąż nie jest obsługiwana przez wykorzystywaną wersję narzędzia GWT i konieczne było zaimplementowanie pewnych funkcji, takich jak wypisywanie sformatowanego łańcucha znaków (jak statyczna metoda *Format* klasy

String w języku Java) czy własny format przechowywania daty i czasu oraz dokonywania operacji na nich. Poprawność działania wymienionych mechanizmów została sprawdzona testami jednostkowymi.

Opracowana gra posiada duże możliwości dalszego rozwoju. Wymienić można wiele elementów, o które można by wzbogacić aplikację, a które nie zawierały się w założeniach projektu. Są to elementy, które mogłyby znaleźć się w grze, gdyby miała być przedstawiona większemu gronu graczy, być może udostępniona w Internecie. Wymienić można tu:

- tryb gry wieloosobowej przez sieć (wielu graczy grających jednocześnie na tej samej mapie, rywalizujących o podróžnych i o pokrycie mapy swoimi liniami),
- lokalizacja językowa gry (gra została przygotowana tylko w języku polskim, jednak jako że wszystkie teksty wyświetlane graczowi zostały zawarte w jednym pliku, przygotowanie gry dla odbiorcy posługującego się innym językiem sprowadzałoby się do przetłumaczenia zawartości tego pliku),
- wyzwania dla gracza, w formie scenariuszy z określonymi warunkami początkowymi gry oraz warunkami zwycięstwa i porażki,
- oprawa dźwiękowa (opracowana gra nie wykorzystuje żadnych efektów dźwiękowych ani muzycznych),
- mechanizm zapisu i odczytu stanu gry,
- system osiągnięć (*achievements*), których zdobycie mogłoby przekładać się na określone korzyści dla gracza,
- efekty wizualne oraz ulepszenia grafiki, także efekty pogodowe, zależne od pory roku, czy efekty wizualne związane z cyklem dobowym, a także dodatkowe elementy takie jak poruszające się pojazdy po drogach, nie będące autobusami gracza, czy rozszerzenie zestawu tekstur terenu.

Inny element, który wzbogaciłby grę, to możliwość generowania map różnych rozmiarów. Gracz mógłby mieć możliwość wyboru rozmiaru mapy spośród kilku możliwości w trakcie rozpoczęcia nowej rozgrywki. Testowano różne rozmiary map. Zdecydowano się generować mapy stałego rozmiaru (20 regionów, w których mogą znajdować się miasta), ponieważ w toku prowadzonych rozgrywek testowych zauważono, iż jest to optymalny rozmiar mapy, a rozgrywka na mniejszych mapach, z mniejszą liczbą miast, byłaby mniej ciekawa i bardziej monotonna. Generowanie z kolei map większych rozmiarów mogłoby wpłynąć negatywnie na wydajność aplikacji oraz na orientację gracza w terenie gry zawierającej zbyt dużą liczbę miast.

Zbiór elementów możliwych do generowania w opracowanej grze nie został wyczerpany. Nazwy miast przedstawiane są jako pojedyncze, unikalne w skali mapy litery, a mogłyby być wyrazami (bądź ciągami wyrazów) powstałymi przy użyciu generatora opartego o słownik. Także tekstury terenu są elementem możliwym do tworzenia w trakcie działania programu. W pewnym zakresie jest to realizowane w projekcie, jednak przygotowanie rozwiązania bazuje na łączeniu obróconych o wielokrotności 90 stopni tekstur przygotowanych dla pojedynczych pól i tworzeniu z nich mozaiki przedstawianej graczowi jako mapa terenu. Programowo przygotowywane są także tekstury autobusów, jednak w tym wypadku generator również bazuje na przygotowanej tekstrze, która jest modyfikowana. W pełni proceduralnie tworzone są tekstury budynków miast, jednak te z kolei oparte są o kolorowe prymitywy geometryczne. Prezentację graficzną budynków można by wzbogacić o dodatkowe elementy (kominy, rynny, donice itd.) oraz wykorzystać tekstury (np. faktur ścian, wzorów ułożenia dachówek).

Użycie generowania proceduralnego do tworzenia części tekstur pozwoliło zredukować łączny rozmiar zasobów dołączonych do gry, co niewątpliwie jest dużą zaletą zastosowanego podejścia. Wadą jest większe zużycie pamięci operacyjnej w trakcie działania programu, potrzebnej na przechowywanie tekstur wygenerowanych w programie, które nie mogą pozostać wczytane z pliku w momencie, gdy są potrzebne.

Zastosowanie podejścia proceduralnego do przygotowania środowiska gry pozwoliło uzyskać różnorodne mapy, w liczbie nierealnej do przygotowania przez projektanta. Dodatkowym zyskiem jest minimalizacja rozmiaru aplikacji, ponieważ uniknięto dołączania zasobów w postaci map gry. Opracowanie zestawu generatorów, testowanie ich oraz znajdowanie i poprawianie błędów było jednak zadaniem trudnym, czasochłonnym i często żmudnym. Mimo to udało się uzyskać zadowalające efekty, spełniające postawione przed projektem założenia.

Rozdział 7

Instrukcja dla użytkownika

Poniżej przedstawiono skróconą pomoc dla użytkownika. Jej obszerniejszą wersję można znaleźć na płycie CD dołączonej do pracy, w formie pliku HTML. W wersji elektronicznej zawarto również zbiór porad dla gracza.

7.1 Instalacja aplikacji

Wersja gry przygotowana w postaci pliku wykonywalnego JAR nie wymaga instalacji. Do jej uruchomienia potrzebny jest komputer wyposażony w system operacyjny, dla którego dostępne jest środowisko uruchomieniowe języka Java w wersji 7 lub wyższej. Środowisko to musi zostać wcześniej zainstalowane (można je pobrać z serwisu firmy Oracle¹).

Aby uruchomić grę na urządzeniu pracującym pod kontrolą systemu Android, należy umieścić dostarczony plik o rozszerzeniu apk na karcie pamięci używanej w urządzeniu bądź wgrać ten plik bezpośrednio do pamięci urządzenia np. przez kabel USB. Następnie należy odnaleźć i uruchomić ten plik już na urządzeniu mobilnym, używając systemowego menadżera plików. Po zainstalowaniu aplikacji można ją uruchomić i rozpocząć grę. Przed zainstalowaniem aplikacji może być konieczne włączenie możliwości uruchamiania aplikacji pochodzących z obcych źródeł. Opcja taka jest dostępna w ustawieniach systemu Android.

Dostarczone pliki aplikacji dla wersji HTML5 są wystarczające do umieszczenia ich na serwerze WWW. Możliwa jest gra przez Internet bez dodatkowych czynności, ponieważ gra dostępna jest pod adresem www.g-ojdana.vxm.pl/busgame. Do uruchomienia gry niezbędna jest przeglądarka internetowa z obsługą WebGL (zaleca się Google Chrome).

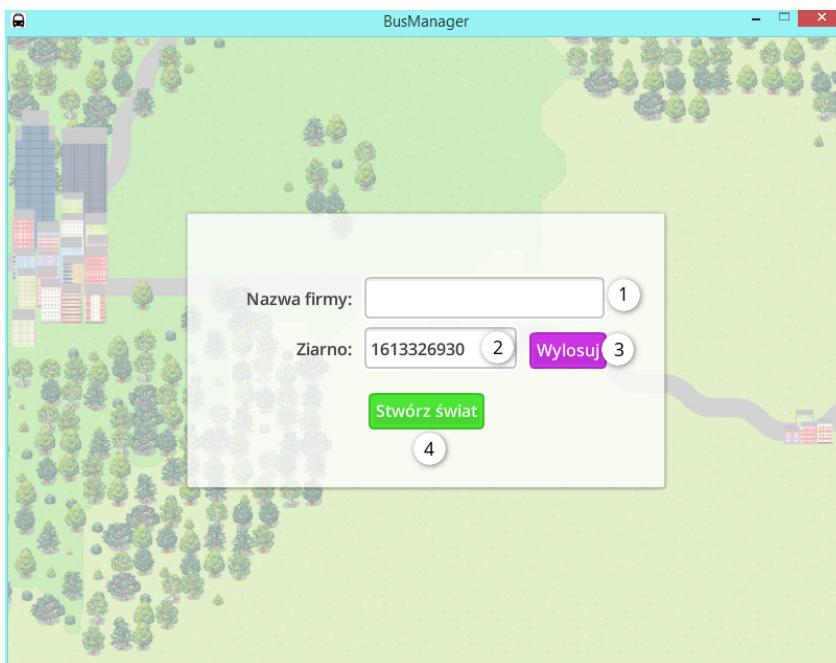
¹<http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html> (dostęp: 28.01.2015)

7.2 Główne ekrany gry

Po uruchomieniu aplikacji, graczowi przedstawiany jest ekran tworzenia nowej gry (ukazany na rysunku 7.1). Gracz podaje na nim informacje potrzebne do przygotowania rozgrywki. Wyróżnione elementy interfejsu to:

1. Pole tekstowe pozwalające podać nazwę przedsiębiorstwa. Może pozostać puste, jeśli gracz nie ma pomysłu na nazwę swojej firmy.
2. Ziarno dla generatora świata gry. Można wpisać tekst, który zostanie zamieniony na wartość liczbową (tekst jest łatwiejszy do zapamiętania w przypadku, gdyby gracz chciał ponownie zagrać na tej samej mapie). Jeśli pole będzie puste lub gracz poda wartość 0, wtedy wartość ziarna zostanie wylosowana.
3. Przycisk *Wylosuj*, którego naciśnięcie spowoduje wylosowanie wartości ziarna i wpisanie uzyskanej wartości w pole tekstowe przeznaczone na ziarno.
4. Przycisk *Stwórz świat*, którego naciśnięcie jest równoznaczne z potwierdzeniem podanych parametrów i skutkuje uruchomieniem procesu tworzenia świata gry.

W trakcie przygotowywania gry graczowi wyświetlany jest osobny ekran, ukazujący pasek postępu, wskazujący na faktyczny stan procesu przygotowywania rozgrywki.



RYSUNEK 7.1: Ekran tworzenia nowej gry.

Kiedy gra zostanie przygotowana, gracz zostanie przeniesiony na ekran rozgrywki (rysunek 7.2). Jest to ekran, na którym gracz spędzi większość czasu gry. Widoczna jest



RYSUNEK 7.2: Ekran rozgrywki.

mapa świata gry, po której gracz może się poruszać, oraz przyciski interfejsu użytkownika. Role zaznaczonych elementów interfejsu przedstawiają się następująco:

1. Przycisk menu gry. Po naciśnięciu tego przycisku upływ czasu w grze zostanie wstrzymany. Pokazane zostanie wyśrodkowane okno menu, z którego można powrócić do gry, opuścić bieżącą rozgrywkę lub pobrać ziarno generatora użyte do przygotowania świata gry.
2. Przycisk otwierający panel zarządzania, z którego gracz może otworzyć okno floty (posiadanych autobusów), okno zakupu autobusów oraz okno statystyk.
3. Przycisk otwierający oraz zamkujący panel linii, przedstawiający informacje o wyznaczonych liniach autobusowych.
4. Stan konta bankowego. Jeśli kwota będzie ujemna, wtedy zostanie wyświetlona czerwoną kolorem tekstu.
5. Bieżąca data i czas gry.
6. Przyciski umożliwiające sterowanie tempem upływu czasu (wstrzymywanie, spowolnienie oraz przyspieszenie). Bieżące ustawienie wyświetcone jest w formie mnożnika.
7. Przycisk otwierający panel tworzenia linii. Po jego otwarciu upływ czasu zostaje wstrzymany. Gracz może na nim przygotować nowe połączenie autobusowe. Jest też dostępna pomoc, pojawiająca po naciśnięciu odpowiedniego przycisku.

8. Przycisk otwierający okno powiadomień. Na przycisku tym wyświetlona jest liczba nieprzeczytanych wiadomości. Przycisk jest ukryty, jeśli gracz odczytał wszystkie wiadomości.

7.3 Sterowanie

Większość akcji użytkownik wykonuje za pomocą myszy. W przypadku wersji gry dla systemu Android, do sterowania w grze używane są gesty palców. Akcje zestawiono w tablicy 7.1. Niektóre z nich można wykonać na kilka sposobów, wymienionych w odpowiedniej rubryce tabeli, rozdzielonych przecinkami. Akronim LPM oznacza lewy przycisk myszy.

TABLICA 7.1: Sterowanie w grze.

Akcja	Metody (Desktop, HTML5)	Metody (Android)
Interakcja z elementami interfejsu	LPM	Dotyk
Interakcja z obiektami na mapie	LPM	Dotyk
Przybliżenie kamery (<i>zoom in</i>)	Ruch kółka myszy od siebie, klawisz plus	Gest rozsunięcia dwóch palców
Oddalenie kamery (<i>zoom out</i>)	Ruch kółka myszy do siebie, klawisz minus	Gest uszczypnięcia (dosunięcia dwóch palców do siebie)
Przesuwanie kamery	Klawisze WSAD, klawisze strzałek, przesuwanie myszy z wciśniętym kółkiem myszy	Przesuwanie palcem po ekranie

Bibliografia

- [1] Dechter R., Pearl J., *Generalized best-first search strategies and the optimality of A**, Journal of the ACM, 505–536, 1985, URL <http://dl.acm.org/citation.cfm?id=3830&coll=portal&dl=ACM>, dostęp 10.12.2014.
- [2] Ebert D., Musgrave F., Peachey D., Perlin K., Worley S., *Texturing and Modeling: A Procedural Approach*, Morgan Kaufmann, trzecie wyd., 2003.
- [3] Gustavson S., *Simplex noise demystified*, 2005, URL <http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf>, dostęp 30.10.2014.
- [4] Johnson L., Yannakakis G., Togelius J., *Cellular automata for real-time generation of infinite cave levels*, PCGames '10 Proceedings of the 2010 Workshop on Procedural Content Generation in Games, 2010, URL <http://dl.acm.org/citation.cfm?id=1814266>, dostęp 5.11.2014.
- [5] Olsen J., *Realtime procedural terrain generation*, 2004, URL <http://web.mit.edu/cesium/Public/terrain.pdf>, dostęp 29.10.2014.
- [6] Patel A., *Amit's A* pages. Heuristics for grid maps*, URL <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>, dostęp 13.12.2014.
- [7] Patel A., *Polygonal map generation for games*, URL <http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/>, dostęp 13.12.2014.
- [8] Perlin K., *Making noise*, URL <http://www.noisemachine.com/talk1/>, dostęp 30.10.2014.
- [9] Shaker N., Togelius J., Nelson M., *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, Springer, 2014, URL <http://pcgbook.com/>, dostęp 15.11.2014.

- [10] Twigg C., *Catmull-rom splines*, 2003, URL <http://www.cs.cmu.edu/~462/projects/assn2/assn2/catmullRom.pdf>, dostęp 16.11.2014.

Załącznik

Do pracy dołączono płytę CD, na której umieszczono:

- elektroniczną wersję tekstu pracy w formacie PDF,
- kod źródłowy aplikacji,
- pliki aplikacji służące do jej uruchomienia bądź instalacji, z podziałem na opracowane wersje gry,
- instrukcję użytkownika w formie pliku HTML.