

Uniwersytet Śląski w Katowicach
Wydział Informatyki i Nauki o Materiałach
Instytut Informatyki

Paweł Kaleta
numer albumu: 292684

**Implementacja gry strategicznej typu 4X polegającej na przejmowaniu
obszarów mapy**

PRACA DYPLOMOWA
INŻYNIERSKA

Promotor: dr hab. inż. Mariusz Boryczka

Sosnowiec 2017

Słowa kluczowe: Java, libGDX, gra strategiczna, gra 4X

Oświadczenie autora pracy

Ja, niżej podpisany/a:

imię (imiona) i nazwisko: **Paweł Kaleta**

autor pracy dyplomowej pt. **Implementacja gry strategicznej typu 4X
polegającej na przejmowaniu obszarów mapy**

Numer albumu: **292684**

Student **Wydziału Informatyki i Nauki o Materiałach Uniwersytetu
Śląskiego w Katowicach**

kierunku studiów: **Informatyka inżynierska - studia stacjonarne I stopnia**

Oświadczam, że ww. praca dyplomowa:

- została przygotowana przeze mnie samodzielnie¹,
- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity Dz. U. z 2006 r. Nr 90, poz. 631, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- nie zawiera danych i informacji, które uzyskałem/łam w sposób niedozwolony,
- nie była podstawą nadania dyplomu uczelni wyższej lub tytułu zawodowego ani mnie, ani innej osobie.

Oświadczam również, że treść pracy dyplomowej zamieszczonej przeze mnie w Archiwum Prac Dyplomowych jest identyczna z treścią zawartą w wydrukowanej wersji pracy.

Jestem świadoma/y odpowiedzialności karnej za złożenie fałszywego oświadczenia.

.....

Data

.....

Podpis autora pracy

¹ uwzględniając merytoryczny wkład promotora (w ramach prowadzonego seminarium dyplomowego)

Spis treści

Wstęp.....	6
1. Problematyka gier komputerowych.....	7
1.1 Gry komputerowe.....	7
1.2 Historia.....	8
1.3 Cechy gier komputerowych.....	8
1.4 Odbiór gier.....	9
1.5 Klasyfikacje gier.....	10
1.6 Gry strategiczne.....	11
1.7 Gry 4X.....	11
2. Podobne rozwiązania.....	13
2.1 Podobne gry.....	13
2.1.1 Sid Meier's Civilization V.....	13
2.1.2 Warlock – Master of the Arcane.....	15
2.1.3 Freeciv.....	17
2.1.4 Master of Orion.....	19
2.1.5 Heroes of Might and Magic III.....	20
2.1.6 Disciples.....	23
2.2 Pola.....	24
2.2.1 Pola kwadratowe.....	25
2.2.2 Pola sześciokątne.....	28
2.2.3 Pola predefiniowane.....	29
2.3 Toczenie bitew.....	30
2.3.1 Rozgrywanie bitew na osobnej mapie.....	31
2.3.2 Symulacja.....	31
2.3.3 Siła oraz siła i losowość.....	32
3. Założenia gry Land Terror.....	33
3.1 Opis gry.....	33
3.2 Wybrane rozwiązania.....	34
3.2.1 Rodzaj pól.....	34
3.2.2 Prowadzenie walk.....	35
3.3 Elementy gry.....	36
3.4 Założenia funkcjonalne aplikacji.....	39
4. Dokumentacja użytkowa.....	40
4.1 Wymagania sprzętowe i programowe aplikacji.....	40
4.2 Menu.....	40
4.2.1 Nowa gra.....	41
4.2.2 Kontynuacja.....	41
4.2.3 Opcje.....	42
4.3 Wybór rasy.....	42
4.3.1 Ludzie.....	42
4.3.2 Nieumarli.....	43
4.3.3 Demony.....	43
4.4 Mapa.....	44
4.5 Budynki.....	45
4.6 Armie.....	46
4.7 Pozyskiwane dobra.....	47
4.8 Technologie.....	47
4.9 Tury.....	48

4.10 Tura gracza komputerowego.....	48
4.11 Cele.....	49
4.12 Zapisywanie i wczytywanie.....	50
5. Dokumentacja programowa.....	51
5.1 Założenia do implementacji aplikacji.....	51
5.2 Środowisko.....	52
5.2 Kamera i widok.....	55
5.3 Elementy graficzne.....	56
5.3.1 Atlasy.....	57
5.3.2 Style.....	58
5.3.3 Tabele.....	58
5.3.4 Etykiety.....	59
5.3.5 Przyciski.....	59
5.3.6 Pola wyboru.....	60
5.3.7 Pasek postępu.....	60
5.3.8 Sprite.....	61
5.4 Preferencje.....	61
5.5 Ekrany.....	62
5.6 Muzyka.....	64
5.7 Mapa.....	64
5.7.1 Program Tiled.....	65
5.7.2 Ładowanie i renderowanie mapy.....	66
5.8 Pola sześciokątne i ich obsługa.....	67
5.9 Obsługa myszy.....	69
5.10 Zapisywanie i wczytywanie (pliki XML).....	70
5.11 Gracze.....	72
5.12 Budynki.....	73
5.13 Oddziały.....	74
5.14 AI.....	75
Zakończenie.....	82
Bibliografia.....	84
Spis rysunków.....	85
Spis tabel.....	86
Spis Listingów.....	87

Wstęp

Celem niniejszej pracy jest prezentacja procesu implementacji gry z gatunku 4X w języku Java przy użyciu biblioteki libGDX, jak i przedstawienie wybranych elementów tej biblioteki wykorzystanych przy pisaniu aplikacji. Praca ta pokazuje wybrane elementy używanych narzędzi, opisuje je oraz prezentuje sposoby ich wykorzystania.

Praca została podzielona na rozdziały. W pierwszym opisane zostały podstawowe pojęcia dotyczące gier komputerowych. W drugim rozdziale przedstawiono rozwiązania podobne oraz rodzaje pól i systemy prowadzenia walk, które są wykorzystywane w produkcjach gier 4X. W trzecim rozdziale opisano również założenia, które były brane pod uwagę w procesie implementacji gry. Czwarty rozdział zawiera instrukcję dla użytkowników aplikacji. W piątym rozdziale zamieszczono dokumentację programową, której celem jest przedstawienie wybranych elementów biblioteki libGDX oraz zastosowanych algorytmów.

Z uwagi na objętość elementów biblioteki zdecydowano się pokazać tylko wybrane jej mechanizmy, jak i niektóre z napisanych na cele gry – w całości lub też fragmentarycznie. Natomiast w kodzie źródłowym aplikacji zostały one opisane ze standardem umożliwiającym wykorzystanie narzędzia *Javadoc* w celu generowania dokumentacji.

1. Problematyka gier komputerowych

Opisane zostały tutaj definicje gier komputerowych, a w szczególności strategicznych i 4X, a także ich miejsce w historii oraz cechy, klasyfikacje, podział pracy nad nimi i ich odbiór wśród społeczeństwa. W celu opracowania informacji liczbowych opierano się między innymi na danych statystycznych dotyczących innych źródeł rozrywki oraz tych dotyczących graczy w Stanach Zjednoczonych oraz Polsce.

1.1 Gry komputerowe

Gry komputerowe są aplikacjami, które umożliwiają rozgrywanie określonych historii na platformach sprzętowych, dając osobie lub osobom, które z nich korzystają, swobodę działań, które aplikacja obejmuje, przy czym wprowadza również jeden lub więcej celów, do których gracz musi dojść, by ukończyć rozgrywaną historię lub też pewien jej fragment. Takie historie mogą być fikcyjne lub też realne – mogą się rozgrywać w miejscach i czasach, które istnieją lub też takich, które są wynikiem fikcji. Mogą dotyczyć rozgrywania bitew pomiędzy armiami, zarządzania różnego rodzaju biznesami, kierowania pewnym bohaterem gry.

Sama definicja wprowadzać może rozbieżność w klasyfikacji programów, bowiem wiele oprogramowań można by podpiąć pod grę komputerową, przykładowo programów do obliczania ze średniej kilku liczb, która stawia cel (obliczenie średniej) oraz daje osobie z niego korzystając swobodę (wpisywane liczby). Na szczęście producenci sami określają, czy ich aplikacja jest grą komputerową czy też nie.

Po gry komputerowe sięga się już częściej niż po książki, odnosząc się do badań przeprowadzonych przez *Statistic Brains* 70% ludzi nie kupiło ani jednej książki w ciągu pięciu lat, przy czym 42% studentów deklaruje się, że po ukończeniu studiów nie przeczyta żadnej książki [1]. W wypadku gier komputerowych w USA, odnosząc się do *The ESA* jest 42% aktywnych graczy, a na każdy dom przypada średnio dwóch graczy i aż w 51% domów jest konsola do gier [2]. W Polsce natomiast, odnosząc się do *Newzoo*, jest 32% aktywnych graczy [3].

Gry komputerowe skupiają w sobie wiele platform sprzętowych, czyli konsole, automaty, telefony komórkowe oraz komputery osobiste.

1.2 Historia

Najczęściej za pierwszą grę komputerową uważa się symulator pocisku rakietowego z 1947r, który używał lamp elektronowych. Tak naprawdę nie była to gra komputerowa, a jedynie jej prototyp. Pierwszą grą komputerową była natomiast komputerowa adaptacja gry kółka i krzyżyk. Początki gier były trudne, głównie ze względu na to, że nie cieszyły się wielką popularnością, a ich wydanie w taki sposób, by były dostępne dla każdego, było niezwykle trudne. Do roku 1972 były to gry stworzone dla potrzeb uniwersyteckich technologii. Dopiero po tych latach stworzono pierwszą konsolę do gier – *Magnavox Odyssey* i wydano pierwszą grę na automaty – *Pong*. W ciągu kolejnych dziesięciu lat spopularyzowane zostało automaty do gier, a na rynku pojawiło się wiele popularnych gier takich jak: *Breakout*, *Space Invaders*, *Asteroids*, *Pac-Man*, *Dragon's Lair*, *Donkey Kong* i *Mario Bros*. Pod koniec tych ostatnich lat wprowadzono też dużo konsol i komputerów uniwersalnych. Następne 15 lat owocowały popularyzacją kolejnych gier, konsol i komputerów, a gry takie jak *King's Quest*, *Tetris* czy *SimCity* spopularyzowały domową rozgrywkę. Od roku 1989 aż do teraz pojawiały się kolejne generacje konsol, silników graficznych, a co za tym idzie coraz to lepiej wyglądających gier. W większości gier zastępowano grafikę 2d, grafiką 3d, a światy gier stawały się coraz większe [4].

1.3 Cechy gier komputerowych

Gry komputerowe stawiają zawsze jakiś cel, które są dość typowe dla każdego z gatunków gier, a więc, przykładowo, w grach platformowych przejście wszystkich plansz, w grach strategicznych pokonanie armii przeciwnika (jest to też jedyny gatunek, który wprowadza wiele więcej celów przejścia gry – zebranie danej liczby dóbr, zebranie pewnej liczby jednostek danego typu, odkrycie pewnego obiektu), dla bijatyk celem jest oczywiście pokonanie jednego lub większej ilości przeciwników.

Gry powstały w celu zaspokojenia potrzeb rozrywkowych ludzi, co wcale

nie oznacza, że gry nie potrafią uczyć, a najlepiej o tym przekonują gry strategiczne, które zmuszają ludzi do logicznego myślenia, przewidywania działań przeciwników, zarządzania.

Popularność gier spowodowała stworzenie terminu „e-sport”, który oznacza rozgrywki sportowe przy użyciu gier komputerowych. Coraz więcej się też inwestuje w takie rozgrywki. W 2010r. pula nagród w taki rozgrywkach wyniosła 5 499 872\$, natomiast w 2015r. tylko do września wynosiła już 45 542 556\$. W ciągu tych lat liczba turniejów wzrosła niemal czterokrotnie.

Gry komputerowe są możliwe do uruchomienia na platformie lub platformach sprzętowych wskazanych przez producentów. W generacji pierwszych gier komputerowych, ich produkcja na wiele platform kierowała się do przepisania całego kodu w taki sposób, by możliwe było uruchomienie skompilowanej aplikacji na danej platformie. Takie działania zostały już niemal całkowicie wyparte przez rozwiązania umożliwiające pisanie aplikacji wieloplatformowych. Większość gier powstaje teraz na wiele platform sprzętowych, o ile nie są to gry przeznaczone tylko na jedną konsolę(wielu producentów należy do koncernów, które tworzą dany rodzaj konsol i wymaga się od nich stworzenia gier tylko na ich platformę). Tworzenie gier wieloplatformowych wiąże się jedynie z przepisaniem mniejszych fragmentów kodu.

1.4 Odbiór gier

Spopularyzowane jest stwierdzenie, że gry komputerowe, a w szczególności te brutalne, zwiększają ryzyko agresji wśród graczy. Amerykańskie badania mówią o tym, że gry wpływają bardziej na ludzi niż telewizja czy też książki. Wiąże się to przede wszystkim z faktem, że w grach komputerowych to właśnie gracze sterują postaciami i mają wpływ na decyzję. Zwiększenie agresji występuje głównie u graczy, którzy grają samotnie.

Gry komputerowe pomagają również w podejmowaniu dobrych wyborów w życiu realnym i są też nierzadko wykorzystywane w celu leczenia ludzi, czy to w przypadku zdolności manualnych czy też psychologicznych. Gry komputerowe poprawiają również reakcje ręka-oko i uczą wielozadaniowości, a dodatkowo mogą uczyć szybkiego czytania i liczenia.

Każde państwo stosuje swoje oznaczenia, które mówią o tym dla jakiej kategorii wiekowej jest dana gra, często jednak gry stworzone dla dorosłych graczy są grane przez młodych graczy. Rząd Australii stosuje kary pieniężne jak i więzienne dla sprzedawców, którzy nie stosują się do sprawdzania wieku klientów przed sprzedażą gier. W innych krajach takie decyzje zmniejszyłyby fundusze państw o przynajmniej kilka milionów.

1.5 Klasyfikacje gier

Gry komputerowe muszą być opisane oznaczeniami, które informują graczy i sprzedawców od jakiego wieku gry mogą być sprzedawane i grane.

W Stanach Zjednoczonych stosuje się ESRB (Entertainment Software Rating Board), w większości krajów europejskich PEGI (Pan European Game Information). Istnieje oczywiście więcej systemów klasyfikacji (np. W Japonii jest to CERO, w Niemczech USK), ale to właśnie ESRB i PEGI są najbardziej popularną klasyfikacją gier. Dla amerykańskiej klasyfikacji stosuje się system:

- EC (Early Childhood) – dla wszystkich graczy od 3 lat,
- E (Everyone) – od 6 lat,
- E10+ (Everyone 10+) – od 10 lat,
- T (Teen) – od 13 lat,
- M (Mature) – od 17 lat,
- A (Adults Only) – od 18 lat.

Dla klasyfikacji PEGI stosuje się oznaczenia liczbowe, które informują o tym, że gra jest dozwolona od wieku, który ta liczba określa. Ponadto stosuje się tak zwane opisy zawartości:

- przemoc,
- seks,
- dyskryminacja,
- narkotyki,
- strach,
- wulgarny język,

- hazard,
- gra online.

Opisy te informują o treściach występujących w danej grze.

1.6 Gry strategiczne

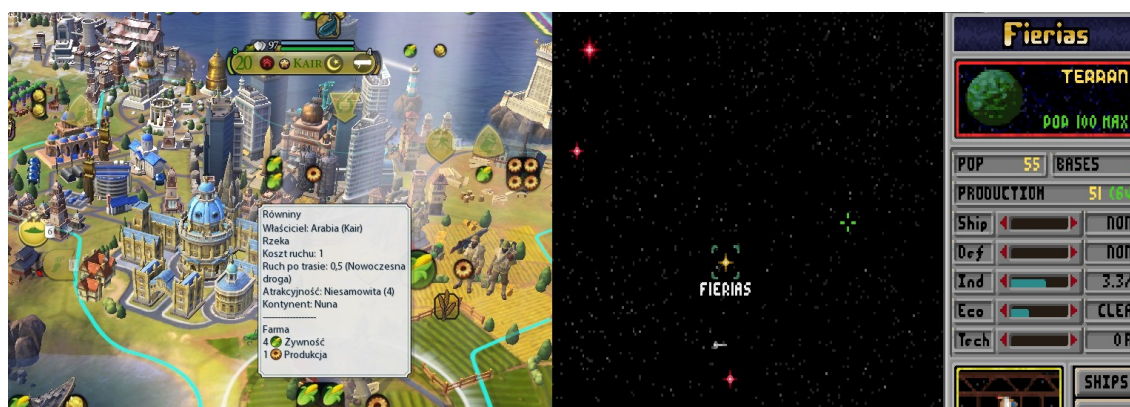
Gry strategiczne to gatunek gier komputerowych, które opierają się na rozgrywce polegającej na planowaniu i podejmowaniu ważnych decyzji taktycznych oraz strategicznych, które pozwalają zdobyć przewagę nad przeciwnikiem. Gatunek ten skupia się na logicznej rozgrywce, stawiając nacisk przede wszystkim na rozmyślanie nad kolejnymi podejmowanymi działaniami oraz przewidywaniu kolejnych ruchów, zarówno graczy jak i komputera. Mechanika tego gatunku, poza istnieniem taktyki i strategii, zakłada istnienie przynajmniej kilku elementów takich jak: jednostki, eksploracja, ekonomia, dyplomacja.

Strategia graczy jest zbiorem pewnych akcji, które podejmują, a jej wynikiem nie jest jedynie wynik działań danego gracza, lecz także innych graczy, przy czym strategia obejmuje działania na całą grę, nie tylko na jej dany etap, co oznacza, że każda wykonana akcja musi zakładać pewien wynik w przyszłości, przy czym założenie to niekoniecznie musi być spełnione.

1.7 Gry 4X

Gry 4X są jednym z gatunków gier strategicznych, które skupiają się na czterech założeniach: eksploracji, ekspansji, eksploatacji oraz eksterminacji [5]. Z angielskich odpowiedników też pochodzi nazwa gatunku (każde słowo zawiera w sobie literę „X”). Termin też wywołuje sporo kontrowersji w zaliczaniu gier do tego gatunku, bowiem nie każda gra, która opiera się na tych czterech założeniach, jest uznawana za grę 4X. Dzieje się to ze względu na wyznaczanie gatunku gry głównie przez deweloperów, którzy wolą określać swoje produkcje mianem gry strategicznej niż jej podgatunku. Oczywiście jest to dość względne, bowiem często różne platformy czy też portale stosują dodatkowe znaczniki (np. Steam). Najczęściej przy określaniu, czy grę można zaliczyć do gatunku 4X,

stosuje się dodatkowe kryteria jak np. więcej możliwości zwycięstw, złożoność technologii czy też dyplomacji. Najbardziej popularnym kryterium jest natomiast to, że grę można zaliczyć do gatunku 4X, jeżeli nie skupia się ona na eliminacji przeciwników za pomocą siły militarnej, lecz skupia się na kontrolowaniu rozwoju imperium pod względem technologicznym, gospodarczym i dyplomatycznym. Nie oznacza to, że gra w której nie ma dyplomacji, dodatkowych celów zwycięstwa nie jest grą 4X, bo to zależy przede wszystkim od dewelopera. Do najpopularniejszych gier 4X zalicza się serie *Master of Orion* oraz *Sid Meier's Civilization* (rys. 1).



Rys. 1: Przykładowe ekrany z Sid Meier's Civilization VI oraz Master of Orion

2. Podobne rozwiązania

W rozdziale tym przedstawione są podobne aplikacje oraz podobne elementy rozgrywki. Nie wszystkie wymienione w tym rozdziale gry są oficjalnie zaliczane do gatunku 4X, lecz wszystkie gry są zaliczane do tego gatunku przez deweloperów lub graczy. Elementy gier opisywane w tym rozdziale dotyczą sposobu reprezentowania określonych cech w przedstawionych poniżej grach oraz w Land Terror. Do wspomnianych elementów zaliczają się trzy podstawowe rodzaje pól oraz rekrutacja jednostek i sposób prowadzenia walk pomiędzy nimi.

2.1 Podobne gry

Opisane zostały tutaj aplikacje, które przyczyniły się w dużej mierze do określenia gatunku 4X oraz takie, które pomimo spełnienia warunków klasyfikacyjnych tego gatunku, nie zostały do niego oficjalnie zaliczone. Niektóre rozwiązania z Land Terror są jednak do nich w dużej mierze podobne. Opisano sześć różnych aplikacji oraz ich zalety i wady.

2.1.1 Sid Meier's Civilization V

Seria *Sid Meier's Civilization* była jedną z pierwszych z gatunku 4X. Do każdej kolejnej części wprowadzano wiele zmian, zmieniając tym samym jej mechanikę. Dopiero w piątej odsłonie serii wprowadzono pola sześciokątne (wcześniej były to czworokąty). Ta pojedyncza zmiana wprowadziła największą innowację w cyklu, bowiem zwiększyła możliwość ruchu o dwa pola i wyeliminowała problem z poruszaniem jednostek na pola sąsiadujące wierzchołkami, a także dała większą możliwość ruchu w wypadku pól, przez które jednostki nie mogły przechodzić – np. góry.

Gra oferuje rozwój cywilizacji od pewnego etapu rozwojowego lub też od samego początku, co oznacza, że gracz może zaczynać od zakładania pierwszego miasta dla wybranego na początku rozgrywki państwa lub też od pewnego okresu dziejowego – np. średniowiecza czy renesansu. Gra wraz z dodatkami oferuje 43 różne narody, które posiadają unikalnego przywódcę oraz zdolności. *Sid Meier's Civilization V* (rys. 2) jest jedną z najbardziej rozbudowanych gier

4X, dając graczom duże możliwości dostosowania rozgrywki. W trakcie gry gracz dostaje możliwość rozmów z innymi państwami, handlowania z nimi, zawierania sojuszy i różnego rodzaju paktów, wprowadzania ustaw światowych na kongresie lub ustrojów w prowadzonym przez siebie państwie, zakładania religii oraz jej rozwoju.

Sid Meier's Civilization V umożliwia budowanie wielu miast, których wielkość reprezentowana jest poprzez poziom odwzorowujący ilość ich mieszkańców. Zdobywane punkty kultury wpływają na granice miast, a więc liczbę pól, które należą do miasta, i tylko na tych polach w granicach miast można budować ulepszenia – farmy, tartaki, mennice oraz wiele innych. W grze występują surowce strategiczne i luksusowe, a aby je pozyskiwać należy pole ulepszyć do określonego rodzaju. Surowce strategiczne są niezbędne do rekrutowania danych jednostek, natomiast surowce luksusowe wpływają na zadowolenie mieszkańców.

Gra oferuje wiele jednostek, których rekrutacja ma określone wymagania, a każda z nich ma określone zadania. Osadnicy są jednostką, która jest wymagana do założenia nowego miasta. Robotnicy służą do ulepszania pól. Wielcy ludzie do określonej roli jak przyspieszenie odkrywania technologii lub również do ulepszania pól. Jednostki militarne służą do walk na lądzie, wodzie lub powietrzu.

Rozgrywka podzielona jest na tury, zaś okres czasowy na epoki. Każda z epok cechuje się określonymi jednostkami i budowlami, które odkrywa się za punkty nauki, co oznacza, że gracz nie ma możliwości budowy samolotów w epoce starożytnej, lecz dopiero po odkryciu odpowiedniej technologii w epoce współczesnej.

Gra posiada wiele możliwości zwycięstw. Rozgrywkę można ukończyć na pięć różnych sposobów – od dominacji, czyli pokonania pozostałych graczy na drodze militarnej, poprzez zwycięstwo naukowe, wysyłając ludzi w celu kolonizacji innej planety, przez stworzenie największej na świecie turystyki oraz poprzez zwycięstwo dyplomatyczne i punktowe. To ostatnie zakłada, że gdy w rozgrywce nastanie pewien rok (domyślnie 2500r.), to zwycięży ten gracz, który zgromadził najwięcej punktów. Tabela 1 przedstawia wybrane zalety i wady gry *Sid Meier's Civilization V*.

Tabela 1: Zalety i wady Sid Meier's Civilization V

Zalety:	Wady:
– wiele nacji do wyboru – rozbudowany system technologii – system dyplomacji – pola sześciokątne – wiele rodzajów jednostek i budynków – wiele ulepszeń pól – umiejętności specjalne – budowa stolicy w miejscu wybranym przez gracza	– źle zaprogramowane AI przeciwników – system początkowego usytuowania graczy – zwycięstwo przez dominację niemal nieosiągalne na większych mapach – brak GUI wyboru jednostek



Rys. 2: Widok z gry Sid Meier's Civilization V

2.1.2 Warlock - Master of the Arcane

Warlock – Master of the Arcane (rys. 3) posiada podobne założenia do *Sid Meier's Civilization V*, ale opiera się na uniwersum fantasy. Gracz kieruje

działaniami wybranej przez siebie postaci, a każda postać ma swoje zdolności i rasę, do której należy.

Na początku rozgrywki gracz otrzymuje już wybudowane miasto oraz jednostkę walczącą w zwarciu i jedną na dystans. W przeciwieństwie do *Sid Meier's Civilization V* rozbudowa terenów polega na ulepszaniu pól z poziomu miasta zamiast wykorzystywania w tym celu jednostek. Na jednym sześciokącie można usytuować tylko jeden budynek, co wymaga od gracza planowania rozbudowy miast. Inną cechą odróżniającą tę aplikację od poprzedniej jest fakt, że ulepszanie pól może być współbieżne z rekrutacją jednostek, co oznacza, że gracz może rozbudowywać swoje tereny i pozyskiwać jednostki w tym samym czasie.

Zamiast technologii wprowadzono do gry system zastępczy, który polega na opracowywaniu zaklęć. Mogą one służyć do osłabiania lub wzmacniania jednostek, modyfikacji terenów, przyzywania jednostek lub atakowaniu ich. Opracowanie zaklęcia wymaga jednak pewnego czasu.

Rozgrywka podzielona jest na tury podczas których gracz podejmuje decyzje dotyczące rozbudowy miast, szkolenia jednostek, działań lądowych oraz morskich, a także rzucania czarów i wykonywania działań dyplomatycznych.

Podobnie jak *Sid Meier's Civilization* gra dostarcza kilku możliwości odniesienia zwycięstwa. Pierwszą taką możliwością jest pokonanie pozostałych graczy. Drugą jest pokonanie awatara – jednostki neutralnej. Kolejną możliwością jest rzucenie czaru jedności, który można opracować po odkryciu wszystkich pozostałych. Ostatnimi możliwościami jest budowa miast na ponad połowie świętych miejsc (specjalnych pól) lub pokonanie jednego przeciwnika specjalnego. Tabela 2 przedstawia wybrane zalety i wady gry *Warlock – Master of the Arcane*.

Tabela 2: Zalety i wady Warlock - Master of the Arcane

Zalety:	Wady:
– dobrze zaprogramowane AI przeciwników – system opracowywania czarów – wiele rodzajów budynków – umiejętności bohaterów	– mały wybór ras – brak możliwości budowania stolicy w miejscu wybranym przez gracza – mała różnorodność jednostek – oddziały neutralne są losowe



Rys. 3: Widok z gry Warlock - Master of the Arcane

2.1.3 Freeciv

Gra powstała na podstawie *Sid Meier's Civilization*, jako jej darmowy odpowiednik. Udostępniona jest na licencji GNU GPL, co oznacza, że oprogramowanie jest udostępnione wraz z kodami źródłowymi, a każdy może je rozpowszechniać oraz wprowadzać w kodzie dowolne zmiany.

Freeciv (rys. 4) może być uruchomiany na wielu systemach. W wersji 2.5 liczba dostępnych nacji wynosi 555, wliczając w nie nacje niehistoryczne, takie jak wampiry, a liczba graczy, którzy mogą grać na jednym serwerze wynosi 126.

Umożliwia ustawienie wielu elementów rozgrywki – od rodzaju pól i ich częstości występowania, przez początkowe jednostki, wynalazki, punkty aż po system walki.

Z podstawowymi ustawieniami gracz rozpoczyna rozgrywkę z kilkoma jednostkami i dostaje dużą swobodę w usytuowaniu pierwszego miasta. Podobnie jak w *Sid Meier's Civilization*, dostaje możliwość budowy kolejnych miast za pomocą specjalnej jednostki. Drzewo technologii daje graczowi możliwość odblokowania kolejnych jednostek i budowli, a także ustrojów, które nastawiają miasta ku pewnej określonej drodze rozwoju. Miasta mogą się zajmować budową budynków, jednostek lub zmianą ustrojów, ale nie w tym samym czasie.

Gracz posiada trzy możliwości na odniesienie zwycięstwa, czyli pokonanie reszty graczy, odkrycie technologii lotów w kosmos i wysłanie rakiety. Jeżeli nie uda się osiągnąć żadnego z powyższych, to po określonej liczbie tur, jeżeli gracz zgromadził najwięcej punktów, to odnosi zwycięstwo. Tabela 3 przedstawia wybrane zalety i wady gry *Freeciv*.

Tabela 3: Zalety i wady Freeciv

Zalety:	Wady:
– system technologii – budowa stolicy w miejscu wybranym przez gracza – wiele nacji do wyboru – możliwość gry wielu graczy – ustawianie wielu jednostek na tym samym polu	– pola kwadratowe – mało ulepszeń pól – nieintuicyjne sterowanie – wszystkie surowce są widoczne już od początku rozgrywki – kiepski system dyplomatyczny



Rys. 4: Widok z gry Freeciv

2.1.4 Master of Orion

Rozgrywka w *Master of Orion* (rys. 5) prezentuje się nieco inaczej niż w poprzednich grach. Nie ma w grze żadnych pól, a jedynie punkty oznaczające planety i jednostki. Planety posiadają system rozwoju, polegający na określeniu nakładów na dany typ produkcji. Gracz ma możliwość wybrania jednej z dziesięciu dostępnych ras, a każda z nich posiada specjalne zdolności, które pomagają w pozyskiwaniu żywności, dyplomacji czy pozyskiwaniu ludności.

Statki kosmiczne służą jako jednostki. W grze występuje kilka typów jednostek – bojowe, zwiadowcze i kolonizacyjne. Statki można wyposażać w różne rodzaje broni, wzmacniając tym samym atak jednostek. Walka może rozgrywać się w przestrzeni kosmicznej oraz na powierzchni planet.

System badań opiera się na sześciu kategoriach – komputerach, polach siłowych, planetologii, napędzie, konstrukcji i broni. Badania pomagają w przeprowadzaniu zwiadu, życia na planecie oraz mobilności statków i sile ataku.

Istnieją dwie możliwości na odniesienie zwycięstwa w grze. Jedną z nich jest dominacja nad innymi rasami poprzez prowadzenie walk. Drugą możliwością jest wybranie przywódcy na władce galaktyki. Tabela 4 przedstawia wybrane zalety i wady gry *Master of Orion*.

Tabela 4: Zalety i wady Master of Orion

Zalety:	Wady:
– wiele ras do wyboru – umiejętności specjalne – dowolny rozkład produkcji – mała ilość możliwych dróg zwycięstwa	– system dyplomatyczny – system technologii – mała ilość jednostek – mapa jest nieintuicyjna, a jednostki na niej są słabo widoczne – kiepski system prowadzenia walk



Rys. 5: Widok z gry Master of Orion

2.1.5 Heroes of Might and Magic III

Seria *Heroes of Might and Magic* nie jest uznawana oficjalnie za grę 4X, ale spełnia wszystkie założenia tego gatunku, bo występuje w niej moduł eksploracji terenów, wydobywania materiałów, eksterminacji oraz poszerzania swoich terenów, chociaż ten ostatni jest kwestią bardziej umowną, bowiem

wszystkie obiekty, które służą do rekrutacji jednostek, wydobywania surowców oraz strzeżenia przejść są już wygenerowane na mapie, co oznacza, że tereny poszerzamy poprzez zdobywanie tych obiektów.

Dostosowanie rozgrywki w *Heroes of Might and Magic III* (rys. 6) umożliwia graczom na wybranie rodzaju imperium, początkowego bohatera oraz premii startowej w postaci słota, surowców lub artefaktów. Rozgrzywka opiera się na scenariuszach, które nakreślają cele, które należy wykonać, by odnieść zwycięstwo. Gra umożliwia również wygenerowanie losowej mapy. Każdy bohater ma swoją specjalność, która może dotyczyć premii do pewnej zdolności, do statystyk określonej jednostki, lub pozyskiwania pewnych dóbr.

Każde imperium posiada unikalny rodzaj miasta, te z kolei poza budynkami dostępnymi dla każdego imperium posiadają również budynki unikalne. Każde miasto umożliwia rekrutację siedmiu podstawowych jednostek dla danego imperium oraz ich ulepszeń. Jedno miasto posiada też unikalny budynek pozwalający rekrutować dodatkowy rodzaj jednostek.

Jednostki w grze mogą być rekrutowane w miastach i budynkach na mapie, które to umożliwiają. Istnieje również szansa, że jakieś jednostki przyłączą się do gracza przy próbie ich zaatakowania. Każda jednostka posiada statystyki, które określają atak, obronę, obrażenia, życie, ilość ataków dystansowych oraz prędkość jednostek, a sam system walki przenosi się na dodatkową mapę na której gracz może wydawać rozkazy swoim jednostką i korzystać z czarów.

W grze nie występuje system naukowy ani inny, który może być uznany za jego zamiennik, ale w zamian twórcy dali możliwość awansowania bohaterów na kolejne poziomy, dając graczom możliwość wybierania zdolności, które mogą między innymi zwiększyć ilość punktów ruchu, obrażenia jednostek czy ulepszyć efektywność czarów.

Gra podzielona jest na tury, które reprezentowane są w sposób kalendarzowy poprzez dni, tygodnie i miesiące. Najbardziej istotne są jednak wśród nich tygodnie, bo istnieje szansa na pewne efektywne lub nieefektywne wydarzenia na ich początku.

W zależności od wybranego scenariusza mogą istnieć maksymalnie dwie możliwości zwycięstwa, przy czym jedna z nich, to zawsze pokonanie wrogów. Tabela 5 przedstawia wybrane zalety gry *Heroes of Might and Magic III*.

Tabela 5: Zalety i wady Heroes of Might and Magic III

Zalety:	Wady:
– wiele ras do wyboru – system czarów – wiele rodzajów jednostek – wiele rodzajów budynków – walki prowadzone są na osobnej mapie, a gracz zyskuje kontrolę nad jednostkami.	– AI przeciwnika sprawdza się tylko na małych mapach. – brak możliwości przechodzenia przez innych bohaterów (częste blokowanie drogi) – niezbalansowane miasta i ich jednostki – losowe rozmieszczenie miasta – brak możliwości budowania budynków na mapie



Rys. 6: Widok z gry Heroes of Might and Magic III

2.1.6 Disciples

Disciples (rys. 7) jest serią, która podobnie jak *Heroes of Might and Magic* nie jest uznawana oficjalnie za grę 4X, lecz spełnia wszystkie jej założenia od eksploracji, przez wydobywanie i poszerzanie prowadzonego imperium, aż do eksterminacji. Gra umożliwia wybór imperium oraz jednej z trzech biegłości. Rozgrywka opiera się na rozgrywaniu scenariuszy, które mają określony cel i określone właściwości dotyczące tego, którą rasą można je rozgrywać.

Na początku rozgrywki gracze otrzymują bohatera i stolicę. W miastach można wznosić budynki odpowiedzialne za ulepszanie jednostek, ich wskrzeszanie, leczenie oraz za opracowywanie zaklęć. Zwykle miasta natomiast można rozbudowywać, zwiększając tym samym dzienną regenerację punktów zdrowia jednostek oraz liczbę miejsc służących do rekrutacji jednostek. Na mapie istnieją dodatkowe budynki odpowiedzialne za pozyskiwanie surowców oraz takie, które pełnią rolę sklepów.

Jednostki mogą być rekrutowane w miastach i można je umieścić w slotach jednostek na karcie bohaterów, o ile umożliwiają one przypisanie jednostek. Jednostki posiadają punkty doświadczenia podobnie jak bohaterowie. Po zebraniu odpowiedniej liczby tych punktów jednostka awansuje, o ile w stolicy istnieje budynek za to odpowiedzialny.

W grze nie występuje system naukowy odpowiedzialny za odkrywanie technologii, ale występuje w niej system odblokowywania czarów. By odblokować dany czar należy spełniać odpowiednie warunki określone poprzez ilość surowców, które w tym celu trzeba poświęcić.

Scenariusze określają cel, który należy zrealizować w celu odniesienia zwycięstwa. Może on dotyczyć zajęcia pewnego miasta, pokonanie wrogich bohaterów czy też posiadanie większych terenów. Tabela 6 przedstawia wybrane zalety i wady gry *Disciples*.

Tabela 6: Zalety i wady Disciples

Zalety:	Wady:
– mała ilość ras – umiejętności specjalne – wiele typów bohaterów – system zaklęć – wiele ulepszeń jednostek	– brak kontroli nad poszerzaniem granic miasta – po opracowaniu czarów dalej trzeba płacić w formie surowców, by ich użyć – czary są niezbalansowane



Rys. 7: Widok z gry Disciples

2.2 Pola

Z uwagi na to, że gry 4X realizowane są zazwyczaj, wykorzystując jeden z trzech rodzajów pól, zdecydowano się na opisanie ich w rozdziale o podobnych rozwiązaniach, bowiem ich rodzaj ma kluczowy wpływ na przebieg całej rozgrywki. W sporej części implementację rozwiązań w Land Terror opierano właśnie na polach.

Pola można podzielić na 3 kategorie: kwadratowe, sześciokątne oraz predefiniowane. Istnieją też mniej popularne rozwiązania, bo pola można też

tworzyć za pomocą dowolnych wielokątów, ale są to rozwiązania z całą pewnością mniej popularne w dzisiejszych grach komputerowych, choć każde z nich ma swoje zalety jak i wady.

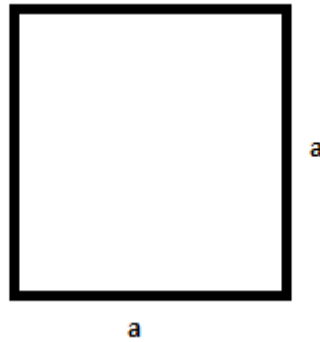
Główną wadą takich wielokątów jest to, że im więcej kątów, tym bardziej przypomina to ruch po okręgu, a też wielkość map musi być zwiększana, by zachować sporo możliwości strategicznych w kontroli terenów. Pola nie muszą być budowane z wielokątów foremnych, ale takie rozwiązanie jest zdecydowanie najłatwiejsze i najbardziej odpowiednie dla graczy, bowiem w wypadku figur, które nie są foremne, należy wypełniać luki figurami mniejszymi oraz większymi.

W wypadku pól predefiniowanych istnieje spora dowolność kształtów oraz pól sąsiednich, ale nie istnieje sens, by tworzyć je w jednym rozmiarze. W przypadku tych pól sąsiednie pola trzeba ręcznie ustawiać, często robi się to za pomocą macierzy, czyli im więcej jest pól, tym większa jest macierz. Takie rozwiązanie jest więc dobre w wypadku map, które składa się z małej ilości pól. Dodatkowo trudniejsze są przejścia rodzajów terenów, bowiem w wypadku pól przykładowo piaszczystych nie można sobie pozwolić na przejście w tereny, które są złożone przykładowo z lasów. Można oczywiście stosować przejścia mieszane, ale zmusza to nad przemyśleniem, czy w takim wypadku dany teren powinien być na przykład piaszczysty czy składać się z wielu rodzajów terenów.

2.2.1 Pola kwadratowe

Pole kwadratowe (*rys. 8*) składa się z czterech boków, więc każdy z boków stanowi pojedynczą możliwość przemieszczania jednostek z pola lub na pole. Największymi zaletami takich pól jest łatwość ich zaprogramowania, w tym również algorytmów ruchu po takich polach.

Jak łatwo można zauważyć, istnieje możliwość ruchu tylko na północ, południe, wschód oraz zachód. Stanowi to ogromny problem, jeżeli istnieje chęć, by przenieść wojsko na pole, które wierzchołkiem styka się z tym na którym ulokowane są wojska, bowiem potrzeba w takim wypadku dwóch ruchów. Problemem jest więc to, że mimo tego, iż taki teren jest blisko, to trzeba przejść przez jedno pole dodatkowe.



Rys. 8: Przykładowe pole kwadratowe

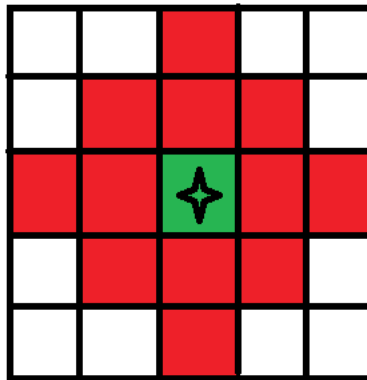
Jednym z rozwiązań takiego problemu jest danie jednostkom większej liczby punktów ruchu lub ustawienie możliwości ruchu na pola, które stykają się wierzchołkami. Opis dwóch tych rozwiązań znajduje się poniżej.

Rozwiązanie 1

W tym przypadku można dać jednostkom więcej punktów ruchu, czyli przykładowo dwa. Oznacza to, że można ruszyć jednostkami na każde pole, które przylega bokami oraz wierzchołkami do tego na którym aktualnie ustawione są jednostki (rys. 9). Tworzy to jednak kolejny problem – można się dostać na pole, które styka się wierzchołkiem z aktualnym lub o dwa pola na północ, południe, wschód lub zachód, czyli w praktyce istnieją cztery możliwości ruchu po dwa pola lub cztery możliwości ruchu o jedno pole. Daje to w wypadku dwóch punktów ruchu aż dwanaście różnych możliwości na przeniesienie wojsk z aktualnego pola.

Takie rozwiązanie skutecznie wyklucza problem, ale stwarza kolejny – nie można przenieść jednostek wojskowych o dwa pola pod kątem, czyli dostarcza to skuteczną możliwość obrony terenu na którym istnieją wojska, ale dalej przebywany dystans jest za mały.

Można dać więcej punktów ruchu, dając możliwość pozycjonowania i obrony jeszcze większego terenu, ale też istnieje możliwość atakowania pól oddalonych o sporą ilość pól, czyli zakładając, że pole ma długość 50km, możemy przebywać teren $50 \cdot n$ kilometrów, gdzie n jest ilością pól o które możemy się przemieścić. Należy jednak zauważyć, że więcej punktów ruchu oznacza też, że możliwość ruchu coraz bardziej przypomina ruch po okręgu

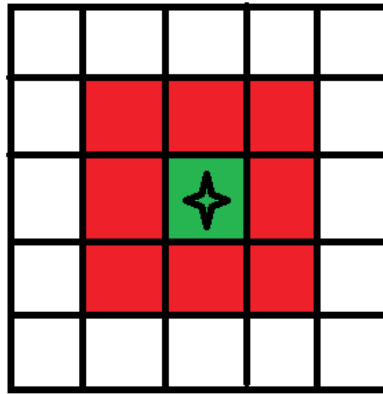


Rys. 9: Możliwe pola do przeniesienia oddziału w wypadku dodatkowego punktu ruchu

Rozwiązanie 2

To rozwiązanie wydaje się być bardziej uczciwe, bo umożliwia przemieszczanie wojsk o pewną stałą ilość pól (również pod kątem), czyli gdy istnieje możliwość przesunięcia jednostek o jedno pole, to istnieje osiem możliwości (*rys. 10*).

Problemem w takim rozwiązaniu jest rzutowanie na dystans, czyli oddanie rzeczywistej odległości, którą wojska przebywają. Zakładając, że jednostki są ustawione w centrum danego pola i przemieszczają się na centrum innego pola, to mają możliwość przebycia dystansu a w czterech kierunkach, oraz odległości $a\sqrt{2}$ również w czterech kierunkach. Przebywany dystans na północ, południe, wschód oraz zachód jest coraz mniejszy wraz z liczbą pól, które można przebyć w tych kierunkach, od tego, który przebyć można na ukos, czyli po przekątnej pola kwadratowego.



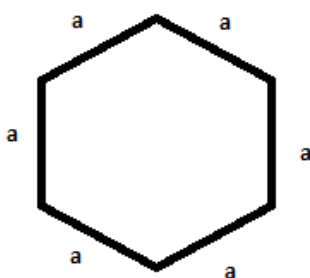
Rys. 10: Możliwe pola do przeniesienia oddziału w wypadku ruchu pod kątem

2.2.2 Pola sześciokątne

Ten rodzaj pól jest aktualnie najczęściej wykorzystywany w grach strategicznych, pomimo tego, że jest bardziej skomplikowany pod względem programistycznym. Jednak graczowi zapewnia sporą gamę możliwości ruchu, bo już przy możliwości przebycia tylko jednego pola gracz ma sześć możliwości wykonania ruchu (*rys. 11*).

Pole takie umożliwia ruch na ukos, co całkowicie eliminuje problem występujący dla pól kwadratowych. Mapy składające się z takich pól wyglądają na większe, a gracz dostaje więcej możliwości ruchu oraz pokrywania terenu zarówno w ofensywie jak i defensywie. Pola te dają również dodatkowe możliwości premii np. atakowanie jednej jednostki lub miasta z sześciu różnych stron lub przy flankowaniu (istnieją 3 możliwości flankowania w wypadku takich pól, w wypadku kwadratowych tylko dwie).

Ten rodzaj pól niemal całkowicie wyparł inne rozwiązania, ale też posiada pewne wady. Do największych można zaliczyć gry oparte na budowaniu miast, które składają się z budynków, które najczęściej mają budowę bardziej zbliżoną do kwadratu niż sześciokąta



Rys. 11: Przykładowe pole sześciokątne

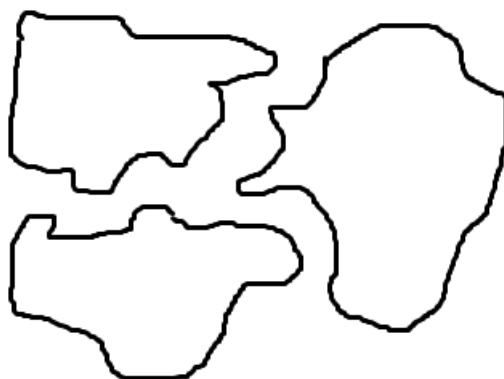
2.2.3 Pola predefiniowane

Pola tego typu często są stosowane w grach planszowych i ich odpowiednikach komputerowych. Kształty pól najczęściej są różne od siebie (rys. 12). Do największych zalet takiego rozwiązania zalicza się przede wszystkim dowolność kreowania map. Oznacza to, że osoba odpowiedzialna za budowę mapy może dowolnie nadawać kształt takim polom, nadawać im niemal dowolny wygląd, a także ustawiać niemal dowolnie liczbę pól sąsiednich.

Budowa takich map może mieć różny poziom trudności ich zaprogramowania. W wypadku małych map takie pola nie stanowią aż takiego problemu jak duże mapy. W wypadku pól sześciokątnych i kwadratowych istnieje możliwość dowolnego ustawienia wielkości mapy, zaś w wypadku pól predefiniowanych wielkość ogranicza się do mapy już zaprogramowanej.

Dodatkowym problemem, który przemawia za innymi rozwiązaniami, jest ręczne ustawianie sąsiednich pól, najczęściej za pomocą macierzy. W takim przypadku w macierz ta jest rozmiaru równego n pól na m , gdzie n jest ilością wszystkich pól, a m maksymalną ilością terenów sąsiednich, w których zapisywane są kody pól do których można przejść z aktualnego.

Mimo wszystko kwestie różnorodności pól, możliwości ręcznego dopracowania każdego z nich oraz różne ilości pól sąsiednich sprawiają, że to rozwiązanie jest wciąż spotykane.



Rys. 12: Przykładowe pola predefiniowane

2.3 Toczenie bitew

Z uwagi na to, że bitwy pełnią istotną funkcję w grach 4X, zdecydowano się umieścić tutaj opis najpopularniejszych rozwiązań tego aspektu rozgrywki. Walki w grach z tego gatunku najczęściej posiadają trzy możliwości ich zaprojektowania.

Pierwsza taka możliwość sprawia, że podczas ataku mapa zmienia się na inną, która odpowiedzialna jest za wyświetlanie jednostek na polu bitwy, a gracz dostaje możliwość, by dowolnie nimi sterować.

Drugim rozwiązaniem jest symulacja ataku, w wypadku której bierze się pod uwagę różne czynniki takie jak skuteczność ataku i obrony danych jednostek przeciw atakowi i obronie jednostek przeciwnika, premie do statystyk, premie terenu, szybkość jednostek, dowolność czynników ogranicza się właściwie do osób odpowiedzialnych za zaprogramowanie walk.

Ostatnią z najczęściej spotykanych możliwości są walki oparte na statystykach bez losowości lub wraz z losowością. W drugim przypadku najczęściej bierze się wartość siły jednostek, do której dolicza się wylosowaną w pewien sposób liczbę z danego zakresu, a wygrywa osoba, które miała większy wynik.

2.3.1 Rozgrywanie bitew na osobnej mapie

W tym rozwiązaniu, gdy gracz zaatakuje innego gracza, walka przenosi się na osobną mapę do tego przeznaczoną, najczęściej jest to mapa ze stałym rozmiarem. Jednostki graczy są ustawiane na polach tej mapy.

Takie same jednostki są ustawiane w jeden oddział na polu bitwy o sile równej ich liczbie. Gracz przejmuje kontrolę nad takimi jednostkami i może je przenosić na inne pola, wyprowadzać ataki skierowane w jednostki przeciwnika. To rozwiązanie przede wszystkim daje graczom kolejne opcje prowadzenia strategicznych działań i dodaje spory element rozgrywki do gry komputerowej. Dodatkowo gracz może oszacować swoje szanse na wygraną jeszcze przed walką. Rozwiązanie to jest znane graczom przede wszystkim z serii *Heroes of Might and Magic*.

Zalety są w takim wypadku też wadami, bo nie każdy gracz lubi prowadzić tego typu walki. Zajmują one sporo czasu, więc gdy gracz woli raczej zarządzać terenami, a walki rozgrywać szybko, to takie rozwiązanie jest sporym minusem, lecz nie każdy gracz jest taki sam – sporej liczbie osób takie rozwiązanie przypadło do gustu.

2.3.2 Symulacja

W wypadku symulacji o losach bitwy świadczy wiele czynników, zaczynając od statystyk danego rodzaju jednostek w armii, przez warunki panujące na danym terenie, morale, zdolności bohaterów kierujących oddziałami, nastawienie do przeciwnika oraz wiele innych. Na ich podstawie są prowadzone kalkulacje, które doprowadzają jedną ze stron do zwycięstwa lub obu do impasu.

Takie rozwiązanie jest zdecydowanie szybsze niż rozgrywanie walk na osobnej mapie, zmusza też gracza do dokładnego przemyślenia sytuacji przed atakiem, a znajomość terenu, stosunków dyplomatycznych czy nastawienia ludności swojej oraz przeciwnika do ich państwa jest istotne. Oczywiście nie wszystkie z tych czynników muszą być brane pod uwagę. Wszystko zależy od gry. Ogólnie rzecz biorąc – im więcej czynników jest wliczane w kalkulacje, tym bardziej walki są skomplikowane.

Duża liczba czynników bardziej komplikuje rozgrywkę niż rozgrywanie bitew na osobnej mapie, zaś ich mała liczba zdecydowanie ją ułatwia. Czynniki te świadczą o zaawansowaniu strategicznym danej gry.

2.3.3 Siła oraz siła i losowość

W wypadku tego rozwiązania brana jest pod uwagę siła jednostek, do której niekiedy dolicza się pewną wylosowaną liczbę (najczęściej wynik rzutu kością). W jednym i drugim przypadku zlicza się siłę jednostek, przykładowo poprzez sumę sił ataku każdego typu jednostek pomnożoną przez ich liczbę. W wypadku, gdy pod uwagę bierze się jedynie siłę, to jej wynik świadczy o zwycięstwie. Gracz, który posiada większą wartość ataku wychodzi zwycięsko z potyczki. Oznacza to tyle, że same walki odbywają się bez strategicznego zaangażowania gracza, czyli jedyna strategia, która istnieje dotyczy innych części gry.

W przypadku siły wraz z losowością, pod uwagę bierze się siłę jednostek, która zostaje zwiększona o losową liczbę. To rozwiązanie zapewnia brak strategicznego zaangażowania, a procentowe szanse na zwycięstwo można określić z góry. Losowość ma zastępować czynniki, które występują w symulacyjnym rozwiązywaniu kwestii walk, nie dając pewności zwycięstwa, o ile siła jednostek, do której doda się najmniejszą możliwą wylosowaną liczbę nie przekroczy siły wrogich jednostek, do których dodana jest zostanie największa możliwa do wylosowania liczba. Jest to proste rozwiązanie, które daje szansę słabszym armii, o ile nie spełniony został warunek co do pewności zwycięstwa u przeciwnika. Przykładem takiego rozwiązania jest gra *Small World 2*.

Najczęściej po przeprowadzeniu takich ataków maleje liczba jednostek jaką posiadają gracze – w wypadku zwycięzcy traci on pewną część jednostek, zaś gracz przegrany traci wszystkie jednostki, które zostały zaatakowane.

3. Założenia gry Land Terror

Opisane zostały tutaj założenia, które towarzyszyły w trakcie implementacji gry *Land Terror*, i dotyczą one sposobu w jaki był pisany kod oraz rozwiązań, które zostały w nim użyte. Znajduje się tutaj opis gry, jak i również założeń funkcjonalnych oraz niefunkcjonalnych, a także elementów występujących w aplikacji oraz wybranych rozwiązań dotyczących rodzaju pól i systemu prowadzenia walk.

3.1 Opis gry

Gra należy do gatunku 4X. Użytkownik ma możliwość podjęcia czterech akcji startowych – rozpoczęcia nowej gry, kontynuacji już rozpoczętej, zmiany ustawień oraz wyjścia z aplikacji. W wypadku rozpoczęcia nowej rozgrywki, gracz wybiera jedną z dostępnych ras, tym samym ustawia czynnik dodatkowy – zdolność rasową, która wpływa na premie startowe lub na elementy podczas rozgrywki.

Po rozpoczęciu gry gracz zostaje przeniesione na mapę podzieloną na sześciokąty. Posiada na początku swoją stolicę oraz armię. W toku rozgrywki stara się doprowadzić do zwycięstwa, ulepszając pola i rekrutując wojsko.

Gracz posiada możliwość wybudowania ulepszeń na każdym pojedynczym polu, które usprawniają jego gospodarkę oraz ma możliwość przejmowania tych ulepszeń, które należą do innych graczy. Dodatkowo posiada możliwość rekrutacji jednostek za złoto i jedzenie. Liczba oddziałów armii zostanie ograniczona do dwóch (aczkolwiek rasa nieumarłych będzie posiadać jeden oddział więcej z uwagi na zdolność rasową). Takie rozwiązanie ogranicza graczy do liczby pól, które może przejmować na początku poprzez uniknięcie zbyt szybkiej ekspansji, czyli zyskania zbyt dużej przewagi już na samym początku rozgrywki jak i uniknięcie zbyt dużej liczby wojsk dla stron, które nie były zaangażowane w konflikt militarny.

Gracz ma możliwość dostosowania jednostek w wojsku poprzez rekrutację jednostek danego typu. Każdy typ posiadać będzie posiadać różne cele, które należy spełnić do ich rekrutacji (określona liczba jedzenia i złota). Jednostki specjalne posiadać będą bonus do ataku na jednostki oraz pola z ulepszeniami,

który będzie odpowiednio trzykrotnie i czterokrotnie większy niż w wypadku innych jednostek.

Ruch jednostek zostanie ograniczony do przesunięcia jednostki o jedno pole na turę lub budowy pewnego ulepszenia danego pola. Po zakończeniu każdej z tur gracz będzie dostawał pewną ilość dóbr z każdego z ulepszonych pól, a gracze komputerowi będą wykonywali ruch w zależności od ich sytuacji oraz sytuacji reszty graczy.

3.2 Wybrane rozwiązania

Z opisywanych w poprzednim rozdziale rozwiązań dotyczących rodzaju pól oraz systemu prowadzenia walk, wybrano po jednym, jako najlepsze rozwiązania do zastosowania w *Land Terror*. Ich mechanika oraz wytłumaczenie wyboru takich rozwiązań znajdują się poniżej.

3.2.1 Rodzaj pól

Mapa w grze powinna składać się z sześciokątów z uwagi na liczbę możliwych ruchów, które te pola zapewniają. Dzięki takiemu rozwiązaniu gracz będzie mógł posiadać sześć możliwości wykonania ruchu o jedno pole. W przeciwieństwie jednak do rozwiązań przedstawionych w rozdziale 2. mapa składać się będzie z pól sześciokątnych w orientacji flat top zamiast pointly top, co oznacza, że pole posiadać będzie po dwa wierzchołki u góry i u dołu, i po jednym po bokach.

W przeciwieństwie do pól predefiniowanych pole to posiada stałą liczbę maksymalnych pól sąsiednich, ale zachowuje ono stały wygląd pola. Z tego powodu pole to zastosowano w aplikacji zamiast pól predefiniowanych. Pola kwadratowe zapewniają 4 lub 8 możliwości, ale trudno jest określić, które z dwóch zaprezentowanych w 2.2.1 rozwiązań jest odpowiednie, bowiem przekładają się na różne przebyte odległości. Z tego powodu zostaną zastosowane pola sześciokątne zamiast kwadratowych.

Ze względu na to, że gra umożliwia ulepszanie pól, przejmując tym samym pewien obszar mapy, zastosowana będzie, widoczna dla gracza, siatka. W grach z serii *Heroes of Might and Magic* oraz *Disciples* siatka nie została zastosowana ze względu na to, że mapa zostaje wygenerowana lub wczytana oraz

gry te nie umożliwiają ulepszania pojedynczych pól, a jedynie przejmowanie znajdujących się już na mapie obiektów.

3.2.2 Prowadzenie walk

W grze zastosowane będzie rozwiązanie opierające się na sile jednostek oraz wprowadzone zostaną czynniki symulacyjne, które są opisane przez procentowe wyrażanie przewagi gracza atakującego lub jej braku. Mnożnik ataku będzie określony jako średnia ilości pozyskiwanych dóbr gracza atakującego w stosunku do gracza broniącego oraz ilości posiadanych dóbr gracza atakującego w stosunku do gracza broniącego. Maksymalna wartość premii do ataku dla graczy wyniesie 120%, a w przypadku braku przewagi minimalna wartość mnożnika wynosić będzie 80%. Oznacza to, że w wypadku gdy jeden z graczy posiada maksymalną wartość premii, to siła jego jednostek zostanie zwiększona o 20%. Gdy natomiast mnożnik dla gracza wynosi 80%, to siła jego jednostek będzie zmniejszona o 20%. Takie rozwiązanie nie zmienia rodzaju prowadzenia walk na rozwiązanie symulacyjne z powodu zbyt małej ilości czynników, ale ulepsza rozwiązanie siłowe o morale oddziałów oparte na ekonomii gracza.

W grze nie zostanie zastosowane rozwiązania symulacyjne z powodu zbyt małej ilości racjonalnych czynników, które można by użyć do opisania rzeczywistej siły oddziałów. Mapa gry zostanie podzielona na pola, ale pola te będą jednego rodzaju, więc niemożliwe będzie zmniejszenie obrażeń ze względu na atak dystansowy na jednostki w lesie lub zmniejszenie obrażeń ze względu na to, że oddziały znajdują się na pustyni. By zastosować rozwiązanie symulacyjne należałoby wprowadzić więcej elementów do rozgrywki.

Zrezygnowano też z prowadzenia walk na osobnej mapie, ze względu na to, że w oddziale mogą istnieć jedynie trzy rodzaje jednostek oraz z powodu tego, że gra w przeciwieństwie do aplikacji opisanych w rozdziale 2. pozwala na krótkie rozgrywki, a przenoszenie walk na osobną mapę jedynie wydłużyłoby ich czas.

Rozwiązanie siłowe wraz z losowością opiera się na losowej liczbie dodawanej do wartości ataku oddziałów, co wprowadza czynnik losowy, który nie pasuje do gry, w której gracz powinien decydować o ataku na podstawie

sytuacji jego królestwa. Był to główny powód rezygnacji z tego rozwiązania, ale biorąc pod uwagę, że liczba jednostek w oddziałach nie jest ograniczona, to wraz z coraz większą ilością jednostek w oddziale czynnik losowy byłby coraz mniej istotny.

3.3 Elementy gry

Gra będzie podzielona na elementy, których pojęcia wraz z ich przedstawieniem będą przedstawione poniżej i na rys. 13, oraz będą stosowane w dalszej części pracy. Elementy te będą częściami składowymi pracy, co oznacza, że nie muszą być stosowane w wypadku każdej gry komputerowej.

- Menu główne – przedstawia akcje, które będzie można wykonać po uruchomieniu aplikacji. Menu będzie podzielone na cztery elementy:
 - Wyjście – akcja ta ma za zadanie wyjście z gry, zwalniając tym samym wszelkie zasoby komputera, które aplikacja wykorzystywała.
 - Nowa gra – akcja ta ma zadanie rozpoczęcie nowej gry, dając możliwość ustawienia rasy gracza.
 - Kontynuacja – akcja będzie miała możliwość wczytania stanu gry, która została zapisana. Stan gry nie powinien różnić się od tego, który był w trakcie wykonywania zapisu rozgrywki
 - Opcje – akcja ta będzie odpowiedzialna za zmianę ustawień dźwięków, wyświetlania aplikacji
- Mapa – będzie przedstawiać krainę, w której toczona będzie rozgrywka. Będzie podzielona na pola połączone krawędziami z innymi polami, za wyjątkiem krawędzi, które stanowią będą granice mapy.

- Pola – część mapy. Na każdym polu będzie możliwość stawiania ulepszenia pola (umożliwiający ulepszony rozwój żywnościowy lub finansowy) oraz przesuwania na nie jednostek.
- Ulepszone pole – pole, na którym umiejscowiony jest budynek, który gwarantuje zwiększony przychód zasobów dla królestwa.
- HUD – dodatkowy element gry, którego zadaniem jest wyświetlanie informacji o obiektach w sposób przejrzysty dla gracza. Niekiedy umożliwia również zarządzanie obiektami poprzez zmiany ich elementów.
- Statystyki królestwa (HUD statystyk) – przedstawiać będą aktualny stan ekonomiczny królestwa.
- HUD technologii– przedstawia technologie, które można odkryć oraz umożliwia wykonanie zapisu stanu gry.
- Rasa – jest to określenie stosowane do frakcji w grze. Każda rasa posiada swoje cechy unikalne oraz jest w stanie wojny z innymi.
- Umiejętność specjalna – dla każdej rasy istnieje unikalna umiejętność specjalna, taka jak zwiększone zasoby na początku gry, więcej oddziałów, więcej jednostek.
- Złoto – jednostka płatnicza w grze, która stosowana jest do rekrutowania wojsk oraz rozbudowy swoich terenów.
- Jedzenie – tak jak złoto jest jednostką płatniczą w grze, która służy do rekrutowania jednostek oraz rozbudowy terenów.
- Armia – liczba jednostek militarnych. Podzielona jest na daną liczbę oddziałów.

- Oddział – zbiór jednostek bliskiego i dalekiego zasięgu oraz jednostek specjalnych.
- Stolica – miasto stanowiące główną siedzibę dowodzenia królestwa. Po jego przejściu przez wroga, frakcja, do której należała stolica, przegrywa automatycznie.
- Cel zwycięstwa – zadanie, którego ukończenie będzie oznaczało wygraną gry. Będzie to mogło być pokonanie wszystkich wrogów, zebranie określonej liczby wojska jako pierwszemu, czy też np. zagarnięcie pewnej liczby pól.
- Mgła wojny – w grach komputerowych taką nazwę stosuje się w stosunku do terenów, do których gracze nie mają wglądu, więc takich o których nie posiadają żadnych informacji, które byłyby aktualizowane na bieżąco (np. w ciągu jednej tury). Oznacza to, że gracz nie posiada wglądu w sytuację militarną i gospodarczą terenów, które do niego nie należą lub są w jakiejś odległości od jego pól.



Rys. 13: Widok z gry Land Terror z opisem elementów

3.4 Założenia funkcjonalne aplikacji

Poniższe założenia będą definiować cele, które aplikacja powinna spełniać po procesie jej implementacji.

- Możliwość wczytania stanu gry – użytkownik będzie posiadał możliwość wczytania stanu poprzedniej rozgrywki, o ile taki zapisany stan istnieje. Użytkownik będzie mógł tego dokonać poprzez menu główne jak i w trakcie aktualnie rozgrywanej gry.
- Możliwość zapisania aktualnego stanu gry – użytkownik będzie posiadał możliwość zapisania aktualnego stanu gry, który mógłby służyć późniejszemu jej przywróceniu do stanu, w którym gra została zapisana, nie tracąc tym samym czasu, który poświęcił na jej rozgrywanie.
- Możliwość dostosowania ustawień pojedynczej rozgrywki – użytkownik będzie posiadał możliwość, by dostosować nową grę do swoich preferencji. Będzie miał możliwość wyboru rasy.
- Możliwość wykonania akcji oddziałami wojskowymi – użytkownik będzie posiadał możliwość zarządzania oddziałami wojskowymi takimi jak: przemieszczanie oddziałów, rekrutacja jednostek, atakowanie wskazanych terenów.
- Możliwość ulepszania pól – użytkownik będzie posiadał możliwość, by na wskazanym polu wybudować budynek, który będzie dostarczał określonej premii na turę.
- Możliwość ustawienia głośności aplikacji – użytkownik będzie posiadał możliwość dostosowania dźwięków aplikacji z ekranu opcji w menu głównym aplikacji. Dźwięki będą wyrażane procentowo (od 0% do 100%).

4. Dokumentacja użytkowa

W rozdziale tym opisane są elementy gry w taki sposób, by gracz wiedział w jaki sposób toczy się rozgrywka w Land Terror. Opisane są tutaj elementy menu głównego, a także ich rola. Dostępne w grze rasy, mapa i jej podział na pola, a także ulepszanie pól, armie, środki płatnicze w grze oraz dostępne do odkrycia technologie oraz tury i zapisywanie oraz wczytywanie gry. Opisane są tu również metody postępowania gracza komputerowego.

4.1 Wymagania sprzętowe i programowe aplikacji

By możliwe było uruchomienie aplikacji, wymagany jest odpowiedni sprzęt oraz oprogramowanie, które umożliwi uruchomienie gry oraz swobodne z niej korzystanie, co rozumiane jest jako brak spadku liczby klatek wyświetlanych na sekundę poniżej 30.

Wymagania sprzętowe:

Procesor: Pentium 4 1.5 GHz lub Athlon XP 1500+ lub lepszy

Pamięć RAM: 256 MB lub więcej

Karta graficzna: NVIDIA GeForce FX 5200 lub ATI Radeon 9600 lub lepsza

Dysk: 250 MB

Wymagania programowe:

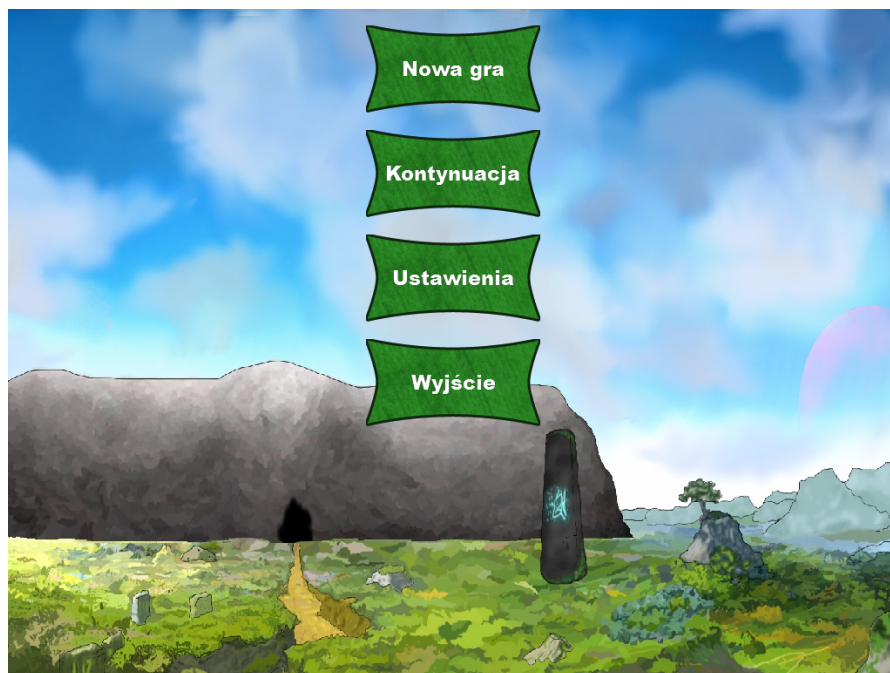
System: Windows Vista lub nowszy, Mac OS X 10.8.3 lub nowszy, Linux 2.6 lub nowszy

Środowisko: Java SE Runtime Environment 7 lub nowsze

4.2 Menu

Menu (*rys. 14*) odpowiada za wyświetlanie początkowych wyborów w aplikacji oraz implementacje dla rozwiązań potrzebnych do ich obsługi.

Umożliwia rozpoczęcie nowej gry, kontynuację zapisanych rozgrywek, dostosowanie opcji muzyki oraz wyjście z aplikacji. Menu to odpowiada za przełączanie pomiędzy trzema ekranami odpowiedzialnymi za wybraną przez gracza akcję. Gdy ekran aplikacji zostanie przełączony na menu, to zaczyna być odtwarzana muzyka, o ile nie została ona wyciszona.



Rys. 14: Menu główne gry Land Terror

4.2.1 Nowa gra

Opcja umożliwia rozpoczęcie nowej rozgrywki. Wybranie tej pozycji spośród wszystkich dostępnych w menu przełącza ekran na ten odpowiedzialny za dostosowanie opcji rozgrywki, a więc wybór rasy, a także za rozpoczęcie rozgrywki z wybranymi przez gracza opcjami.

4.2.2 Kontynuacja

Opcja ta umożliwia rozpoczęcie zapisanej wcześniej rozgrywki od momentu, w którym została ona zapisana. Opcja ta sprawia, że zostaje wczytany

plik *save* z katalogu zasobów aplikacji, a informacje tam zawarte są wykorzystane do ustawienia zasobów graczy, jednostek w oddziałach oraz umiejscowienia danych budynków na mapie świata.

4.2.3 Opcje

Opcja ta umożliwia dostosowanie ustawień głośności muzyki w skali procentowej oraz włączenie lub wyłączenie odtwarzania muzyki. Wybranie tej pozycji z menu głównego gry zmienia ekran aplikacji na ekran z ustawieniami gry.

4.3 Wybór rasy

W grze występują trzy rasy, a każda z nich wpływa na sytuację gracza na samym początku rozgrywki. Każdą z ras jest łatwiej dążyć do określonego rodzaju zwycięstwa. Wybór rasy może poprawiać sytuację ekonomiczną królestwa, może również zwiększyć pole możliwej ekspansji, a także poprawić sytuację militarną. Dla każdej z ras proponowana jest pewna strategia działań, choć nie zmusza gracza do podejmowania działań określonych przez strategię działań.

4.3.1 Ludzie

Ludzie są rasą, która dąży do szybkiej ekspansji, a tym samym polepszenia sytuacji ekonomicznej królestwa. Wybranie rasy ludzkiej powoduje zwiększenie dostępnych na początku rozgrywki dóbr. Ludzie są jedyną rasą, która na początku drugiej tury (a więc pierwszej która umożliwia ekspansję terenów) jest w stanie zakupić dwa ulepszenia pól, co skutkuje możliwością polepszenia sytuacji ekonomicznej dużo bardziej niż w wypadku innych dostępnych w aplikacji ras, które gracz może wybrać przed rozpoczęciem rozgrywki. Największą zaletą tej rasy jest szybka rozbudowa terenów i armii.

Zalecaną strategią dla ludzi jest szybka rozbudowa terenów w pierwszych etapach rozgrywki, a następnie przy dużym wydobyciu dóbr, rozbudowa armii.

4.3.2 Nieumarli

Nieumarli to rasa, która podobnie jak ludzie dąży do ekspansji. Wybranie rasy nieumarłych sprawia, że zamiast dwóch początkowych oddziałów, które gracz dostałby w wyniku wyboru ludzi lub demonów, otrzymuje jeden dodatkowy oddział, ale wszystkie składają się z mniejszej liczby jednostek. W przeciwieństwie do ludzi nie są w stanie zakupić dwóch ulepszeń pól w drugiej turze, ale liczba oddziałów umożliwia jeszcze szybszą ekspansję w późniejszych etapach rozgrywki niż w wypadku innych ras. Rasa ta ma małe szanse na zwycięstwo w bitwach w pierwszych turach, ale umożliwia podchodzenie do podejmowanych działań ze sprytem, wykorzystując możliwości pozycjonowania oddziałów.

Zalecaną strategią dla nieumarłych jest ekspansja terenów w początkowych fazach rozgrywki poprzez ulepszanie pól i przejmowanie już tych ulepszonych, o ile nie prowadzi to do walk. Takie rozwiązanie utrzymuje wrogiego gracza w granicach zagrożonych pól, uniemożliwiając tym samym szybką ekspansję. W późniejszych fazach zaleca się obronę własnych terenów połączoną z atakiem na królestwo wroga lub zmasowany atak na stolicę wrogiego gracza.

4.3.3 Demony

Demony to rasa, która nie dąży do szybkiej ekspansji, ale do szybkiego pokonanie wrogich oddziałów lub osłabienia ich na tyle, aby nie dały radę odbudować swoich sił wystarczająco, by pokonać armię gracza. Wybranie rasy demonów sprawia, że każdy z dwóch początkowych oddziałów otrzymuje dodatkową jednostkę bliskiego zasięgu, dalekiego zasięgu oraz jednostkę specjalną. W przeciwieństwie do ludzi nie są w stanie zakupić dwóch ulepszeń pól w drugiej turze rozgrywki. Największą siłą tej rasy są potężne oddziały na początku rozgrywki, które umożliwiają zlikwidowanie oddziałów przeciwnika już w pierwszych turach, o ile oponent nie zrezygnuje z ekspansji na rzecz umocnienia swoich sił militarnych. W taki sposób odcina to gracza przeciwnego od dużego wydobycia dóbr i zyskania przewagi nad graczem w późniejszych etapach rozgrywki.

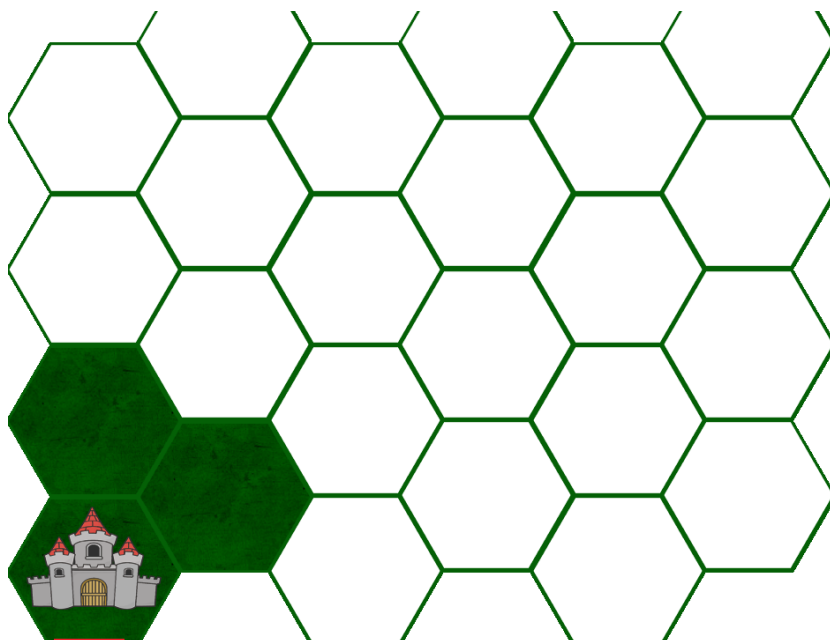
Zalecaną strategią dla tej rasy jest szybkie przemieszczenie armii na tereny wroga już w pierwszych turach rozgrywki, ulepszając przy tym małą liczbę pól, ale wystarczającą by nie przeważać szali na stronę oponenta pod względem rozbudowy armii. Powstrzymuje to wroga od ekspansji terenów usytuowanych dalej od jego stolicy, hamując tym samym jego rozwój ekonomiczny. W późniejszych fazach demony tracą na sile, o ile nie zastopowano wrogiego gracza, więc zachowanie w tej fazie zależy od kontrolującego tę rasę gracza.

4.4 Mapa

Świat gry przedstawiony jest na planszy, która ma zadanie odzwierciedlać tereny na które gracze mogą przemieszczać swoje oddziały, oddzielając tym samym te ziemie od tych niedostępnych dla graczy.

Mapa jest podzielona na kolumny i wiersze, a każda para kolumna-wiersz jest reprezentowana przez sześciokątne pole, na którym mogą znaleźć się budynki biorących udział w rozgrywce ras oraz ich jednostki. Pola w kolumnach parzystych i dostępnych dla nich wierszach są umiejscowiona o połowę wysokości pola niżej te w kolumnach nieparzystych i dostępnych dla nich wierszach, co skutkuje tym, że istnieje możliwość przeniesienia jednostek z pola, na którym się znajdują, na pola, które stykają się bokami z tym na którym znajdują się jednostki.

Mapa może być przysłonięta przez mgłę wojny, a więc element, który uniemożliwia graczowi wgląd w określone pola oraz obiekty które się na nich znajdują. Mgła wojny umiejscowiona jest na polach oddalonych o więcej niż jedno pole od ulepszeń gracza oraz jego oddziałów (*rys. 15*).



Rys. 15: Widok mapy z gry Land Terror

4.5 Budynki

Budynki to zbiór obiektów, którego elementami są stolice oraz ulepszenia pól. Każda rasa posiada po trzy rodzaje budynków, którymi są stolice, budynki skupiające się na wydobywaniu złota oraz budynki skupiające się na wydobywaniu jedzenia.

Każdy obiekt tego zbioru jest określony przez jego identyfikator, a więc numer obiektu, współrzędne kubiczne, które opisują umiejscowienie budynku na konkretnym polu, aktualne i maksymalne punkty wytrzymałości, nazwę gracza, do którego budynek należy i jego rasę, rodzaj budynku, złoto oraz jedzenie i punkty nauki, które obiekt zapewnia graczowi, do którego należy.

Na tabeli (Tabela 7) opisane są maksymalne punkty wytrzymałości budynków, wydobywane dobra oraz punkty nauki i koszt ulepszenia pola.

Tabela 7: Statystyki i koszt ulepszeń pól

Rodzaj budynku	Punkty wytrzymałości	Złoto na turę	Jedzenie na turę	Punkty nauki na turę	Koszt(złoto, jedzenie)
Stolica	100	8	3	0.1	Budynek darmowy
Farma	40	5	3	0	25, 10
Dzielnica rzemieślnicza	40	10	2	0	15, 15

4.6 Armie

Armia to zbiór wszystkich oddziałów danego gracza wraz z ich jednostkami. W grze stanowi element militarny, co oznacza, że za jej pomocą gracz przeprowadza wszystkie akcje służące do eksterminacji, a więc pokonywania wrogich oddziałów lub też zmniejszania liczby ich jednostek. Armia w aplikacji służy również do przejmowania ulepszeń wrogiego gracza, a także występuje w charakterze eksploracyjnym i budowniczym, co oznacza, że ma możliwość zwiedzania terenów mapy oraz poszerzania królestwa poprzez ulepszanie pól. Oddziały to zbiór jednostek, którymi steruje gracz. Można za ich pomocą ulepszać pola, budując na nich farmy lub dzielnice rzemieślnicze, a także można rekrutować kolejne jednostki. Oddział może atakować oddziały należące do wrogiego gracza, a także przejmować wrogie ulepszenia pól.

Jednostki są podzielone na trzy rodzaje: jednostki bliskiego zasięgu, dalekiego zasięgu oraz jednostki specjalne. Każda jednostka bliskiego i dalekiego zasięgu posiada wartość ataku równą jeden, ale posiadają inny koszt ich rekrutacji. Jednostki specjalne posiadają trzykrotnie większy atak na wrogie oddziały niż jednostki bliskiego lub dalekiego zasięgu, natomiast w wypadku ataków na wrogie ulepszenia siła ta jest już czterokrotnie większa, co sprawia, że jednostki te opłaca się najbardziej kupować w celu przejmowania wrogich budynków lub po zmniejszeniu cen rekrutacji poprzez odblokowanie technologii Lepsza rekrutacja. Ceny rekrutacji przedstawione zostały w tabeli (Tabela 8).

Tabela 8: Koszt rekrutacji jednostek

Rodzaj jednostki	Kosz w złocie	Koszt w jedzeniu
Bliskiego zasięgu	10	3
Dalekiego zasięgu	15	2
Specjalna	30	8

4.7 Pozyskiwane dobra

W grze występują dwa rodzaje dóbr: złoto oraz jedzenie. Służą one w celu budowania ulepszeń pól oraz pozyskiwania nowych jednostek dla armii gracza. Po zakończeniu tury gracze otrzymują pewną ilość tych dóbr zdefiniowaną przez sumę pozyskiwanego na turę złota oraz jedzenia ze wszystkich budynków należących do gracza, a więc każde z ulepszeń dostarcza w ciągu tury określoną ilość złota i jedzenia dla gracza, który jest właścicielem pola z tym ulepszeniem

4.8 Technologie

W grze występuje możliwość odkrywania technologii. Stolica dostarcza graczowi w ciągu tury 0.1 punktu nauki. Gracz posiada możliwość odkrycia technologii, o ile spełnia określone warunki – posiadanie pewnej ilości punktów nauki. Technologia może wpływać na zdobywanie dóbr, rekrutację oraz budowę, a dokładne skutki ich odkrycia opisane są w tabeli (Tabela 9).

Tabela 9: Nazwa, koszt i opis technologii w Land Terror

Ikona	Nazwa	Koszt	Opis
	Złoty wiek	1	Podwaja liczbę pozyskiwanych dóbr ze stolicy
	Lepsza budowa	2	Zmniejsza koszt budowy ulepszeń o 10%
	Lepsza rekrutacja	2	Zmniejsza koszt rekrutacji jednostek o 10%

4.9 Tury

Rozgrywka w aplikacji jest podzielona na tury, czyli nieokreślone etapy czasowe pewnej fazy podczas której gracz może przemieszczać swoje oddziały, budować ulepszenia pól, rekrutować jednostki, atakować wrogie jednostki, atakować wrogie ulepszenia pól oraz odblokowywać technologie. Każdy z graczy musi znajdować się w pewnej określonej turze, a ich działania są wykonywane po kolei, co oznacza, że każdy z graczy, który wykonał już swoje działania i zakończył turę, przechodzi do kolejnej. W pierwszej kolejności turę wykonuje człowiek, a po nim gracze komputerowi.

Po zakończeniu tury danego gracza resetowane są punkty ruchu jego oddziałów, uszkodzone budynki są nieznacznie odnawiane, a ulepszenia pól tego gracza dostarczają mu złoto, jedzenie oraz punkty nauki.

4.10 Tura gracza komputerowego

Dla graczy komputerowych w ich turze są podejmowania działania, które mają określić, które z możliwych do wykonania działań okażą się dla tego gracza najbardziej korzystnymi. W tych turach dla decyzji obliczane są współczynniki określające ich wagi, a te z największymi współczynnikami decyzyjnymi są uznawane za najlepsze. Działania tych graczy są dwuetapowe.

Pierwszym etapem jest zdecydowanie na co przeznaczyć dobra posiadane przez tego gracza. W wypadku gorszej sytuacji ekonomicznej gracze

komputerowi będą starać się budować ulepszenia pól lub dążyć do przejmowania ulepszeń przeciwnika. Jeżeli będą posiadać gorszą sytuację militarną niż przeciwnik, to będą starać się przeznaczać te dobra na rekrutację kolejnych jednostek. Działania te są uwarunkowane przez sytuację na sąsiednich polach – jeżeli na tych polach będzie istnieć więcej jednostek w oddziałach wroga niż w oddziale gracza komputerowego, to będzie się on starać unikać walk.

Drugim etapem jest zdecydowanie co zrobić z punktem ruchu oddziałów, o ile te punkty nie zostały wykorzystane. Gracz komputerowy będzie podejmował decyzje, czy atakować jednostkę wroga czy wroga ulepszenie pola. Jest to uzależnione przede wszystkim od siły wrogich oddziałów oraz liczby tur potrzebnych do przejścia określonego budynku. Jeżeli żadne z takich działań nie będzie dla niego korzystne, to w zależności od sytuacji tego gracza będzie on poszukiwał najszybszej drogi do polepszenia takiej sytuacji poprzez przejście ulepszenia wroga, budowę własnego lub też w wypadku braku zagrożenia będzie starał się on przejść stolicę wroga.

4.11 Cele

Dla każdego z graczy, niezależnie od wybranej rasy, istnieją dwie możliwości na odniesienie zwycięstwa. Pierwszą taką możliwością jest zwycięstwo przez dominację. W wypadku gry nie wymaga to pokonania oddziałów wroga lub przejścia pewnej liczby pól z farmą lub dzielnicą rzemieślniczą, a jedynie zdobycie stolicy wroga, czyli przejście tylko jednego pola z ulepszeniem. Drugą możliwością na wygraną jest zebranie w armii siedemdziesięciu jednostek, przy czym każda jednostka bliskiego zasięgu, dalekiego zasięgu oraz specjalna jest brana tylko za jedną jednostkę, co oznacza, że pomimo tego, iż jednostki specjalne posiadają siłę trzykrotnie lub czterokrotnie większą niż inne jednostki, to nie jest ona uznawana za większą liczbę jednostek.

4.12 Zapisywanie i wczytywanie

Gra oferuje możliwość zapisywania i wczytywania stanów gry. Pierwszą czynność wykonuje się klikając przycisk zapisu po prawej stronie okna aplikacji (ikona dyskietki na zielonym tle). Zapisywane są wszystkie ulepszenia pól, oddziały, odkryte technologie oraz stan dóbr graczy i punktów nauki. Zapisywanie ma miejsce zawsze do jednego pliku.

Wczytywanie odbywa się natomiast z menu głównego gry. Wszystkie zapisane obiekty są z powrotem przenoszone na mapę, a stan dóbr i nauki jest ustawiany na ten z pliku zapisu. Gracz może kontynuować rozgrywkę od momenty w którym została ona zapisana.

5. Dokumentacja programowa

W rozdziale tym opisane będą najistotniejsze algorytmy aplikacji i najważniejsze elementy biblioteki libGDX, które umożliwiły implementację Land Terror. Rzeczy te pełnią ważne role w grze komputerowej. Pokazane są tutaj między innymi metody przełączania ekranów, czyli zmianę widoku z jednego na inny, ustawiania kamer i widoków, obsługi myszy, odtwarzania muzyki, jak i programowania interfejsu użytkownika oraz AI gracza komputerowego. W procesie pisania aplikacji korzystano z książki opisującej tworzenie gier w bibliotece [6] oraz z API libGDX [7], czyli interfejsu programistycznego aplikacji oraz w największym stopniu z dokumentacji biblioteki [8].

5.1 Założenia do implementacji aplikacji

Przy realizacji gry kierowano się zasadami, które ułatwiałyby pisanie kodu oraz jego modyfikacje – zarówno podczas tworzenia pierwotnej wersji aplikacji, jak i jej rozwijania, także dotyczy to organizacji plików binarnych, tekstowych, graficznych oraz dźwiękowych, jak i organizacji samego kodu źródłowego oraz komentarzy. Dokładną specyfikację wymagań przedstawia poniższa lista.

- Podział klas – klasy będą się dzielić na takie, które będą odpowiadać za dane aspekty rozgrywki, czyli np. graczy, ekrany, HUD (ang. heads-up display) – czyli wyświetlany graczowi element opisujący pewne obiekty.
- System komentarzy – stosowanie komentarzy do objaśnień zadań funkcji oraz zmiennych i stałych w taki sposób, by wracając do projektu po długiej przerwie można było w sposób przejrzysty znaleźć fragment kodu odpowiedzialny za dany element.

- Przejrzysty podział plików gry – podział plików na taki, by wszystkie pliki były umieszczone w odpowiednich gałęziach gry, a więc pliki graficzne w folderze przeznaczonym dla nich, muzyka w katalogu przeznaczonym do takich plików.
- Zgodność plików konfiguracyjnych – pliki te powinny być zgodne ze standardami przechowywania ustawień oraz treści (np. .cfg, .xml) i w wypadku wielu plików odpowiedzialnych za takie elementy, przechowywanie ich w plikach jednego typu. Oznacza to, że np. zapisy rozgrywki nie mogą znajdować się równocześnie w wielu typach plików (np. .json, .xml).
- Unikanie zależności od plików zewnętrznych – możliwość zmian plików zewnętrznych bez potrzeby zmian w kodzie źródłowym gry komputerowej (z wyjątkami, takimi jak: zmiana nazw plików, katalogów).
- Podział elementów kodu – stałe oraz zmienne nie powinny być mieszane z funkcjami oraz ich deklaracjami, o ile nie ma takiej potrzeby, co umożliwia swobodny dostęp do nich bez konieczności szukania ich w całym kodzie aplikacji.
- Przejrzyste nazewnictwo – nazwy stałych, zmiennych oraz funkcji będą zapisywane w języku angielskim i będą posiadać unikalne nazwy.
- Zarządzanie zasobami – pamięć oraz procesor nie mogą wykorzystywać swoich zasobów na niepotrzebne akcje, takie jak wczytywanie jednego pliku zewnętrznego kolejny raz do pamięci, o ile poprzedni zapis nie został usunięty.

5.2 Środowisko

Gra została stworzona w języku Java przy użyciu biblioteki libGDX, która umożliwia pisanie graficznych aplikacji wieloplatformowych. Powstała ona w

celu pisania aplikacji w sposób wygodny i szybki. Biblioteka ta korzysta z jeszcze innej – LWJGL (*Lightweight Java Game Library*). LWJGL służy do pisania gier komputerowych i wykorzystuje API OpenGL. Dzięki takim rozwiązaniom libGDX jest przeznaczone zarówno dla programistów posiadających podstawową wiedzę w zakresie pisania gier komputerowych, jak i profesjonalnych twórców.

Narzędzie jest na licencji Apache 2.0, czyli jest darmowe i istnieje możliwość przejrzenia kodu źródłowego biblioteki jak i jego modyfikację. Posiada obszerną i darmową dokumentację, która bardzo dobrze wyjaśnia działanie funkcji narzędzia. Właśnie ze względów takich jak pisanie aplikacji na wiele platform, brak potrzeby pisania własnego silnika oraz ułatwienia programistyczne, jest to doskonałe narzędzie. Sama biblioteka posiada również możliwość dostępu do plików systemowych, urządzeń wejściowych i wyjściowych.

Pisanie aplikacji przy wykorzystaniu tego narzędzia na wiele platform polega jedynie na modyfikacji klas startowych, które polegają przede wszystkim na ustawieniu rozmiaru okna aplikacji oraz dodatkowych opcji, takich jak automatyczne dostosowanie rozmiaru okien do rozdzielczości. Klasy startowe służą jedynie do uruchomienia głównej części aplikacji (ang. *core*) na danej platformie.

W skład libGDX wchodzi biblioteki dodatkowe, które współpracują z tym narzędziem:

- OpenGL – API, które służy do budowy zaawansowanych gier 3d, a nawet 2d. Obliczenia są wykonywane szybko na GPU. Do największych zalet OpenGL należy przede wszystkim multiplatformowość oraz zaawansowanie działań, które można dzięki niej wykonać – od budowy scen przez figury geometryczne aż po fizykę tych obiektów. Biblioteka ta pracuje na palecie barw RPG, co umożliwia odwzorować idealne barwy.
- FreeType – biblioteka, która służy do uzyskania dostępu do treści czcionek. Obsługuje bitmapy i formaty skalowalne, czyli można dzięki niej uzyskać dostęp do takich formatów jak TrueType, OpenType,

CID, CFF oraz wielu innych. Potrafi obsługiwać czcionki w 256 odcieniach szarości. Umożliwia wyświetlanie tekstu, jego renderowania oraz ustawienia wielkości tekstu.

- Mpg123 – darmowy odtwarzacz dźwięków z otwartymi źródłami. Działa w czasie rzeczywistym i pozwala na odtwarzanie dźwięków zakodowanych w MPG.
- xiph.org – narzędzie z otwartymi źródłami, które pozwala na kodowanie oraz dekodowanie dźwięków kompresowanych kodekiem Vorbis, czyli odpowiednik formatu MP3. Najczęściej jest stosowany w systemach GNU/Linux.
- Soundtouch – biblioteka z otwartymi źródłami. Służy do odtwarzania dźwięków za pomocą strumieni oraz plików audio. Największą zaletą biblioteki jest jej wieloplatformowość, łatwość w użytkowaniu, 16 i 32-bitowe wskaźniki na dźwięki monofoniczne, stereo i wielokanałowe. Biblioteka odtwarza dźwięki w czasie rzeczywistym.
- Box2d – biblioteka umożliwiająca pracę z fizyką w grach. Początkowo została napisana w C++, ale została też przeniesiona na inne języki (w tym Javę). Pozwala na ustawienia właściwości obiektów takie jak gęstość czy tarcie.
- LWJGL – biblioteka, która służy do pracy nad stroną audiowizualną aplikacji. Jest biblioteka wieloplatformową. Opiera się na innych bibliotekach przez co umożliwia zarządzanie grafiką, dźwiękami, urządzeniami wejściowymi (również gamepadami).
- OpenAL – biblioteka z otwartymi źródłami. Służy do generowania wielokanałowych dźwięków. Jest wieloplatformowa.
- KissFFT – biblioteka, która umożliwia szybką transformację Fouriera,

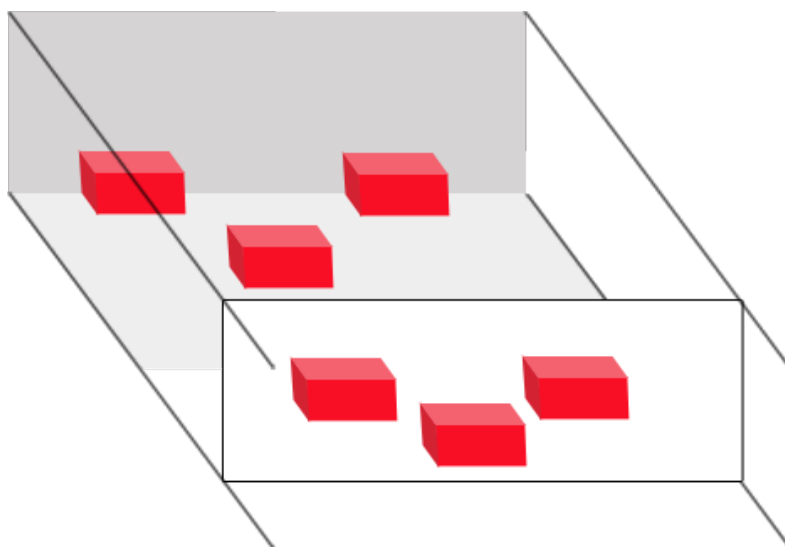
czyli usuwanie szumów z funkcji aproksymowanych sinusami. Umożliwia to między innymi usuwanie szumów z dźwięków jak i różnych algorytmów (również graficznych). Umożliwia stosowanie wskaźników typu fixed i float do typów danych.

Wersją biblioteki która została wykorzystana w celu implementacji aplikacji była wersja 1.6.1 wykorzystana w zintegrowanym środowisku programistycznym *NetBeans* w wersji 8.2. Dodatki wykorzystane dla tego środowiska to: *Gradle Support* w wersji 1.3.8.1, *Android* w wersji 1.280 i *Android Gradle Supports* w wersji 1.271.

5.2 Kamera i widok

Kamera w libGDX pełni rolę oczu, a więc definiuje w jaki sposób postrzegane jest to, co znajduje się na scenie, i czy w ogóle jest to postrzegane. Biblioteka zapewnia dwa rodzaje kamer – kamerę izometryczną oraz kamerę perspektywiczną. Pierwsza z nich postrzega obiekty jako elementy 2D, natomiast druga bardziej naśladuje ludzkie oko i postrzega dodatkowo głębie obiektów. W aplikacji zastosowano kamerę izometryczną, bowiem mapa, jednostki i budynki są dwuwymiarowe, więc nie ma potrzeby, by podejmować starania na ukazanie głębi obiektów oraz odległości obiektów od siebie, co oznacza, że obiekty postrzegane są w taki sposób jak wyglądają ich tekstury, zaś odległość obiektów od siebie nie ma znaczenia (*rys. 16*). Dlatego też wykorzystano do tego rzut równoległy, który pokazuje identyczne obiekty w takiej samej skali, nawet, gdy obiekty znajdują się na końcu planu. Kamerę izometryczną tworzy się przez stworzenie obiektu klasy *OrthographicCamera*, a następnie ustawienie kadru.

LibGDX umożliwia określenie sposobu w jaki dane sceny są pokazywane użytkownikom aplikacji. Możliwe jest, by na daną rozdzielczość rozciągnąć całą scenę, dopasować scenę przez jak najmniejsze skalowanie, by dopasować szerokość lub wysokość do określonego rozmiaru lub jak najmniejsze skalowanie, by szerokość i wysokość dopasować do podanej rozdzielczości lub rozmiaru okna, możliwe jest też rozszerzanie widoku tylko w jednym kierunku.



Rys. 16: Postrzeganie sceny przez kamerę izometryczną

```
private void createCamera() {
    //stworzenie obiektu kamery izometrycznej
    camera = new OrthographicCamera();
    //ustawienie kamery na rozdzielczość aplikacji i z
    //osią y //rosnącą w górę.
    camera.setToOrtho(false, LandTerrorGame.WIDTH,
        LandTerrorGame.HEIGHT);
}
```

Listing 1: Tworzenie kamery i jej ustawianie

W Land Terror wykorzystano klasy *FitViewport*, *StretchViewport*. Pierwszy widok by rozszerzać rozmiary scen w wypadku większej szerokości niż 1920 pikseli oraz wysokości przy większych rozmiarach niż 1080 pikseli dla wszystkich ekranów oprócz ekranu gry. Drugi natomiast, by zachować poprawny rozmiar obiektów HUD or mapy. Obiekty rzutowania tworzone są poprzez podanie jako argumentów: szerokości, wysokości oraz obiektu kamery.

5.3 Elementy graficzne

LibGDX posiada wiele elementów interfejsu użytkownika, które umożliwiają nadanie aplikacji przyjaznego dla użytkownika wyglądu. Elementy te znajdują się w pakiecie *ui* biblioteki. I te najbardziej istotne z nich oraz inne

elementy odpowiedzialne za wyświetlaną grafikę, które zostały wykorzystane w grze, będą opisane w poniższych punktach.

5.3.1 Atlasy

Biblioteka umożliwia obsługę atlasów, a więc plików tekstowych, które opisują pliki graficzne, które składać się mogą z wielu tekstur (*rys. 17*). Do tworzenia takich plików został stworzony program TexturePacker, ale można je utworzyć również ręcznie. Plik tekstowy musi zawierać opis atlasu, a więc nazwę pliku, który opisuje, format graficzny oraz filtr wyszukiwania tekstur. Poza opisem atlasu musi zawierać jeszcze opis elementów, który opisuje nazwy, ułożenie, położenie i rozmiar tekstury. Poniżej znajduje się przykładowa treść opisu atlasu i jednego jego elementu oraz przykładowy wygenerowany obraz z kilku tekstur.

```
units.png
format: RGBA8888
filter: Nearest, Nearest
repeat: none
unitButton
  rotate: false
  xy: 1, 103
  size: 100, 100
  orig: 100, 100
  offset: 0, 0
  index: -1
```

Listing 2: Fragment atlasu opisującego HUD oddziałów



Rys. 17: Przykładowa grafika opisywana przez atlas

5.3.2 Style

Wygląd elementów interfejsu użytkownika jest określony przez plik zawierający opis stylów tych obiektów oraz zbiór obiektów, które służą do nadania elementom określonej prezentacji, a więc między innymi czcionek, tekstur, kolorów. Style takie tworzy się przy pomocy obiektu klasy *Skin*, który zawiera opis stylów oraz atlas.

W aplikacji do tworzenia stylów użyto plików tekstowych w formacie JSON, które opisują style obiektów konkretnych klas. Muszą one być opisane przez jedną lub dwie klasy, przy czym w wypadku drugiej klasy musi on być strukturą stylów dla pierwszej klasy. Opisy klas natomiast muszą zawierać nazwę dla stylu oraz uzupełniać wszystkie pola stylów tych obiektów, co oznacza, że struktura takiego pliku musi wyglądać jak poniżej.

```
{
    Klasa$Klasa_struktury_stylów: {
        nazwa_stylu: { pole: wartość_dla_pola }
    }
}
```

Listing 3: Syntaktyka tworzenia styli

5.3.3 Tabele

Tabele w bibliotece mają za zadanie usprawniać pozycjonowanie i ustawianie rozmiarów elementów w komórkach. W wierszach tabel może znajdować się wiele komórek i wiersze te nie muszą zawierać tyle samo komórek. Tabele umożliwiają kontrolowanie wielkości komórek, wyrównywania elementów, ustawiania dla nich marginesów lub ustawiania odstępów od innych, jak i pozycjonowanie tabeli do układu rodzica elementu. W grze użyto ich do pozycjonowania elementów interfejsu użytkownika przy pomocy pliku stylów.

```

Table mainTable = new Table();
//ustawienie rodzica tabeli na scenę i dopasowanie tabeli
//do rozmiaru sceny
mainTable.setFillParent(true);
//ustawienie położenie tabeli na górny środek sceny
mainTable.top();
//dodanie komórki z podanym elementem
mainTable.add(playButton);
//dodanie wiersza i ustawienie odstępu od góry
mainTable.row().padTop(10);
mainTable.add(selectBox);
mainTable.row().padTop(10);
mainTable.add(backButton);

```

Listing 4: Tworzenie tabeli dla elementów ekranu ustawiania opcji rozgrywki

5.3.4 Etykiety

Etykiety są prostymi elementami klasy `Label`, które wyświetlają podany tekst przy pomocy koloru opisanego przez obiekt klasy `Color` oraz czcionkę bitmapową. Pełnią one rolę informacyjno-opisową dla graczy o elementach aplikacji. Tekst formatowany jest dzięki statycznej metodzie *String.format*, która przyjmuje jako argumenty zasadę formatowania oraz tekst.

5.3.5 Przyciski

Przyciski stanowią elementy, które posiadają reprezentację graficzną i wyzwalającą (rys. 18), a więc posiadają pewien wygląd i obsługę wydarzeń takich jak kliknięcie czy skierowanie na nie kursora myszki. Wyzwalacze mogą dodatkowo zawierać pewne wydarzenia opisane przez obiekty klasy *InputEvent*, które dają możliwość wykonania określonego fragmentu kodu w wypadku uruchomienia wyzwalacza.

Biblioteka oferuje programistom cztery rodzaje przycisków. Mogą to być puste przyciski, przyciski z obrazem i tekstem, przyciski z tekstem oraz przyciski z obrazem. Dwa ostatnie wykorzystano w aplikacji, i rozszerzają one klasę `Button` odpowiedzialną za puste przyciski, dodatkowo o tekst i grafikę.

TextButton tworzy pusty przycisk a następnie dodaje do niego etykietę z

podanym tekstem, natomiast *ImageButton* również tworzy pusty przycisk, lecz dodaje do niego obiekt klasy *Image*, czyli obraz, który dzięki metodzie *draw* jest rysowany na przycisku. W aplikacji, dzięki dostosowaniu rozmiaru przycisków do obiektu klasy *Image*, przyciski wyglądają tak jakby same obrazy były przyciskami. Poniżej znajdują się metody tworzenia przycisków z tekstem oraz grafiką.

```
TextButton przycisk = new TextButton("Tekst na przycisku",  
style, "nazwa stylu");
```

Listing 5: Przykład tworzenia przycisku tekstowego



Rys. 18: Przykłady stworzonych przycisków(dwa tekstowe i jeden z obrazem)

5.3.6 Pola wyboru

Pola wyboru to obiekty klasy *CheckBox*, które rozszerzają klasę *TextButton*, a więc posiadają metodę *isChecked* sprawdzającą, czy zostały one zaznaczone. Pola wyboru posiadają dwie grafiki opisywane przez style – pierwsza dla niezaznaczonego pola, a druga do już zaznaczonego. Obraz pola wyboru jest wyświetlany po lewej stronie obiektu *Label*, którą zawierają te obiekty. W grze nie istniała potrzeba opisywania tych pól przez etykiety, więc obiekt *Label* zawiera pusty tekst.

5.3.7 Pasek postępu

Paski postępu są obiektami klasy *ProgressBar* i służą do pokazania pewnego rodzaju postępu do określonego warunku, który opisany jest przez pewną wartość. Konstruktor przyjmuje cztery warunki, a są to: wartość minimalna, wartość maksymalna, minimalny procentowy postęp, układ horyzontalny lub wertykalny, a także element stylów, który opisuje wygląd obiektu. Używane są w aplikacji do pokazania procesu przejmowania ulepszeń

pól, reprezentując tym samym aktualne punkty wytrzymałości budynków na punkty maksymalne.

5.3.8 Sprite

Sprite to inaczej jakikolwiek dwuwymiarowy obrazek, który jest częścią większej sceny. Są one tworzone przez klasę *Sprite* na podstawie załadowanej do pamięci tekstury. Jest to wygodny sposób na tworzenie statycznej grafiki oraz prostych animacji. W *Land Terror* służą do wyświetlania obiektów oddziałów oraz obiektów budynków na scenie aplikacji. Sposób tworzenia takich obiektów polega jedynie na stworzeniu obiektu *Sprite* z argumentem, który stanowi tekstura. W wypadku wielu identycznych obiektów do pamięci ładowana jest tylko jedna tekstura, co sprawia, że stanowią one idealny sposób na zmniejszenie zapotrzebowania na pamięć komputera.

5.4 Preferencje

Preferencje w bibliotece *libGDX* są obiektami klasy *Preferences* i stanowią kontenery danych, które są zapisywane jako pliki w folderze *.prefs* w katalogach domowych użytkowników. Nazwy plików przybierają nazwy obiektów *Preferences*, których użyto do zapisu danych. Pliki te są plikami XML, których głównym korzeniem jest *properties*, a atrybuty są zapisywane jako potomkowie oznaczeni jako *entry key* z określonymi wartościami kluczy. Tworzenie takich plików jest możliwe dzięki metodom zapewnionym przez interfejs *Preferences*, które umożliwiają zapis podstawowych typów danych, zaś wczytywanie wartości dzięki metodom odpowiedzialnym za wczytywanie podstawowych typów danych.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM
"http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="switch">false</entry>
<entry key="volume">0.1</entry>
</properties>
```

Listing 6: Zapisane ustawienia gry Land Terror

```
p = Gdx.app.getPreferences("settings");  
float volume = p.getFloat("volume");  
boolean switchOff = p.getBoolean("switch");
```

Listing 7: Wczytanie ustawień głośności i odtwarzania dźwięku

5.5 Ekran

Zarządzanie ekranami odbywa się za pomocą klasy *Game* oraz podstawowej klasy abstrakcyjnej i kilku, które rozszerzają jej funkcjonalność o kolejne elementy i metody. Klasami rozszerzającymi strukturę *AbstractScreen* są: *SplashScreen*, *MenuScreen*, *SettingsScreen*, *MenuSettingsScreen*, *GameScreen* oraz *EndScreen* (rys. 19).

Klasa *Game* pozwala na zmianę rozdzielczości ekranu, zatrzymywanie i wznowianie działania ekranów oraz pozwala na ich renderowanie w pętli głównej i ukrywanie, gdy nie będzie potrzeby ich użycia. Jednak poza renderowaniem pozwala również na przełączanie ekranów, co pozwala na używanie wielu ekranów w aplikacji, umożliwiając tym samym zmiany scen, które są pokazywane graczowi.

Zadaniem klasy abstrakcyjnej *AbstractScreen* jest stworzenie struktur dla sceny i kamery która ją pokazuje oraz stworzenie procesora, który zajmuje się przechwytywaniem działania myszy i klawiatury. Także tworzy interfejs dla klas rozszerzających, który umożliwia ich obsługę przez klasę *Game*.

SplashScreen odpowiada za wyświetlanie ekranu początkowego aplikacji. Renderuje on na scenie nałożoną na prostokąt teksturę, sprawiając, że gracz widzi podany jako plik obraz. Po upływie sekundy ekran jest przełączany.

MenuScreen odpowiada za wyświetlanie menu aplikacji. Podobnie jak *SplashScreen* rozszerza on metodę renderowania klasy *Game* o renderowanie tła. Przy udziale atlasu zawierającego informacje o elementach graficznych tekstury oraz informacji o stylach wyświetla on przyciski odpowiadające za rozpoczęcie nowej rozgrywki, kontynuowania zapisanej, wejście do opcji gry oraz wyjście z aplikacji. Przyciski te umieszczono w tabeli, by zapewnić poprawne ich ułożenie w jednej linii. Do każdego przycisku dodano zdarzenie, które zapewnia przełączanie ekranów lub wyjście z aplikacji.

SettingsScreen odpowiada za wyświetlanie ustawień aplikacji. Tworzy on etykiety do opisywania elementów ustawień oraz suwak do ustawiania głośności i pole wyboru. Jeżeli istnieje plik zapisanych opcji, to za pomocą tych danych ustawiane są wartości tych ustawień. Elementy interfejsu użytkownika zamieszczono w tabeli. Ekran ten umożliwia zmianę ekranu na *MenuScreen* oraz zapisanie ustawień.

MenuSettingsScreen odpowiada za ustawianie opcji rozgrywki. Za pomocą atlasu oraz informacji o stylach umożliwia on stworzenie listy wyboru oraz przycisków do zmiany ekranu na *MenuScreen* i *GameScreen*. Lista wyboru posiada wartości odpowiadające możliwym do wyboru rasom. Elementy interfejsu użytkownika zamieszczono w tabeli, by dostosować położenie elementów.

GameScreen odpowiada za rozpoczęcie rozgrywki lub wczytanie już istniejącej, a także za wyświetlanie sceny rozgrywki graczowi. Klasa ta tworzy lub wczytuje obiekty: graczy, budynków, jednostek, mapy oraz HUD technologii oraz statystyk. Tworzy on także scenę dla rozgrywki, renderuje mgłę wojny. Klasa ta sprawdza również, czy osiągnięto zwycięstwo lub porażkę gracza.

EndScreen odpowiada za wyświetlanie informacji, czy dla gracza nastąpiła porażka czy wygrana przez cztery sekundy.



Rys. 19: Ekran *SplashScreen*, *MenuScreen* i *GameScreen*

5.6 Muzyka

Muzyka w LibGDX działa poprzez strumieniowanie, a więc wysyłanie cyfrowych danych do urządzeń do odtwarzania, czyli przykładowo słuchawek. Oznacza to, że pliki z muzyką nie są wczytywane do pamięci RAM, lecz odtwarzane bezpośrednio z dysku komputera. Odpowiada za to interfejs *Music*, który umożliwia włączanie, wyłączanie i pauzowanie muzyki, a także ustawienie głośności.

```
Music = Gdx.audio.newMusic( Gdx.files.internal
("nazwa_pliku"));
//ustawienie głośności na wartość zmiennej musicVolume
setVolume(musicVolume);
//wyłączanie muzyki. True dla wyłączania; false dla
//włączania
turnOffMusic(musicOff);
//włączenie zapętlenia muzyki
music.setLooping(true);
//włączenie odtwarzania muzyki
music.play();

public void setVolume(float vol) {
    music.setVolume(vol);
}
public void turnOffMusic(boolean switchMusic) {
    if(switchMusic) music.stop(); else music.play();
}
```

Listing 8: Odtwarzanie i zatrzymywanie muzyki oraz ustawianie jej głośności

5.7 Mapa

LibGDX pozwala na operowanie na mapach kafelkowych, a więc mapach, które są podzielone na pojedyncze pola. Mapy takie składają z pewnej ilości pól w kolumnach i pewnej liczby pól w wierszach. Kafelki mogą posiadać dowolny wygląd – od pojedynczych pikseli po obrazy w dużej rozdzielczości, mogą reprezentować małe obiekty, jak na przykład pniak drzewa, ale również duże, jak na przykład budynki.

Dodatkowym atutem takich map jest posiadanie wielu warstw, dzięki czemu można stworzyć kafelki, które znajdują się na innych, by oddzielić reprezentacje położenia obiektów względem ziemi lub też oddzielić określone typy obiektów. Jako, że Land Terror jest grą, która pokazuje określone wyobrażenia regionów w małej skali, to nie istniała żadna potrzeba wykorzystania większej ilości warstw. Zastosowanie większej ich ilości sprawdza się przeważnie jedynie w wypadku gier platformowych lub gier, które są pokazane ze względnie bliskiej odległości, czyli na przykład takiej, w której kafelki reprezentujące drzewa czy budynki byłyby widoczne dla gracza.

By tworzyć mapy kafelkowe potrzebne są kafelki, a więc jeden lub więcej plików z teksturami. Następnie należy stworzyć plik XML, który reprezentuje taką mapę, a więc między innymi liczbę kafelków w rzędach, ich ułożenie, orientacje oraz wiele innych atrybutów. Z powodu skomplikowania takiego procesu, wykorzystano program Tiled, który ułatwia ich tworzenie.

Ładowanie map następuje dzięki wykorzystaniu klasy *TmxMapLoader*, natomiast renderowana jest za pomocą renderera klasy *HexagonalTiledMapRenderer*.

5.7.1 Program Tiled

Program Tiled jest aplikacją, która usprawnia tworzenie map kafelkowych. Aby być w stanie taką mapę zaprojektować, należy posiadać zbiór obrazków z teksturami w równych odległościach od siebie.

Opcja *Nowy* w menu kontekstowym *Plik* tworzy mapę o określonych parametrach takich jak rozmiar mapy, rozmiar kafelków, orientacja, format warstwy kafelków oraz kolejność renderowania (*rys. 20*). Właściwości mapy można po utworzeniu podstawowej jeszcze bardziej dostosować. Opcja *Nowy zestaw kafelków* z obszaru *Zestawy kafelków* pozwoliła na utworzenie pojedynczych tekstur na podstawie pliku graficznego, które posiadają swoje identyfikatory, rozmiar i prawdopodobieństwo (*rys. 21*). Następnie zostały naniesione na utworzoną mapę. Została ona następnie zapisana do formatu *TMX* (ang. *Tile Map XML*).

Właściwość	Wartość
▼ Mapa	
Orientacja	Heksagonalna
Szerokość	6
Wysokość	4
Szerokość Kafelka	198
Wysokość Kafelka	172
Długość Ścianki Bocznej Kafelka (Hex)	99
Osie Rozłożenia	X
Indeks Rozłożenia	Nieparzysty
Format Warstwy Kafelków	Base64 (niekor
Kolejność Renderowania Kafelków	Lewy Góra
► Kolor Tła	Not set

Rys. 20: Właściwości mapy

Właściwość	Wartość
▼ Kafelk	
ID	0
Szerokość	198
Wysokość	172
Prawdopodobieństwo	1,000

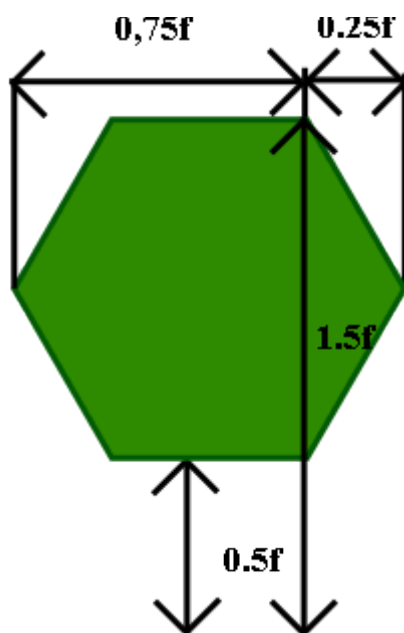
Rys. 21: Właściwości kafelka

5.7.2 Ładowanie i renderowanie mapy

Ładowanie mapy odbywa się dzięki stworzeniu obiektu klasy *TiledMap* i załadowanie do niego mapy dzięki metodzie *load* klasy *TmxMapLoader*. Odbywa się to poprzez stworzenie obiektu mapy na podstawie pliku oraz struktury klasy *Parameters*. Metoda *load* ma za zadanie załadowanie wszystkich tekstur wskazanej mapy i generowanie pól na ich podstawie oraz właściwości mapy przez metodę *loadTileSet*.

Renderowanie takich map odbywa się przez renderer *HexagonalTiledMapRenderer* przy użyciu metody *renderTileLayer*. Na podstawie szerokości i wysokości pól tworzone są stałe, które odpowiadają położeniu kluczowych punktów pól sześciokątnych, a więc dla szerokości jest to 0.25 i 0.75 szerokości pól, natomiast dla wysokości jest to 0.50 i 1.50 ich wysokości. Pola są rysowane w punktach szerokości równej iloczynowi 0.75 i

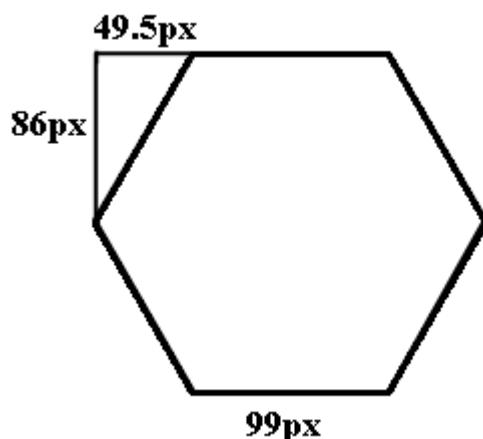
aktualnej kolumny oraz iloczynowi wysokości pola i aktualnego wiersza, którego wartość jest zwiększana o połowę wysokości pola w wypadku wiersza nieparzystego (rys. 22). Następnie kafelki są rysowane na ekranie jako pola kwadratowe, ale z uwagi na ich umieszczenie oraz przezroczyste obszary, mapa wygląda jakby była rysowana z sześciokątów.



Rys. 22: Poszczególne długości i odległości pól sześciokątnych

5.8 Pola sześciokątne i ich obsługa

W Land Terror rozmiar pól wynosi 198x172 piksele i są one układane w orientacji flat-top, czyli na górze i dole tych pól są boki, które mają długość połowy szerokości, czyli 99 pikseli. Sześciokąty te są podzielone na sześć trójkątów równobocznych o długości boków wynoszącej 99 pikseli. Wykorzystano dla nich wertykalne ułożenie even-q, czyli położenie parzystych kolumn jest od samego dołu mapy, gdy kolumny nieparzyste położone są o połowę wysokości tych pól wyżej, czyli o 86 pikseli (rys. 23).



Rys. 23: Podstawowe rozmiary wykorzystywane przy obsłudze pól

Tworzenie takich pól i ich obsługa jest realizowana poprzez stworzony na potrzeby aplikacji pakiet *com.pk.ltgame.hex*. Składa się on z sześciu klas, których zadaniem jest trzymanie informacji o tych polach w różnych postaciach, przechodzenie pomiędzy nimi przy użyciu metod tego pakietu oraz operacje na nich takie jak znajdowanie sąsiadów, odległości pól od siebie i wielu innych.

Klasa *Point* zajmuje się przechowywaniem koordynatów osiowych, czyli trapezoidalnych oraz podstawowych rozmiarów punktów wykorzystywanych w ułożeniu mapy, Klasa ta realizuje system koordynatów i trzyma podstawowe informacje o polach, mimo że jego struktura składa się zaledwie z dwóch wartości typu *double* i posiada tylko jeden konstruktor.

Klasa *Orientation* jest strukturą, która przechowuje kąt początkowy dla orientacji flat-top (rysowanie pól w tej orientacji zaczyna się od kąta 0°) oraz wartości umożliwiające układanie pól i rysowanie kątów dzięki ich odwracaniu, czyli odwracaniu macierzy z tymi informacjami.

Klasa *Layout* odpowiada za układ pól sześciokątnych przy pomocy struktury klasy *Orientation* oraz podanych rozmiarów pól. Wymagane rozmiary, które są w tym celu potrzebne, to takie zapisane jako obiekty klasy *Point*, których wartości oznaczają kolejno szerokość i wysokość. Są to obiekty oznaczające połowę większej wartości rozdzielczości wykorzystywanych pól. Pola te jednak można jednak rozciągać na szerokość i wysokość, podając dwie różne wartości. Drugi obiekt natomiast posiada wartości połowy szerokości i pełną wysokość tych pól, które definiują boczne wierzchołki do których

przylegają po dwa wierzchołki pól sąsiednich.

Klasa *OffsetCoord* zajmuje się przechowywaniem koordynatów w postaci offsetowej, a więc wektorów przesunięć od pola początkowego. Posiada metody umożliwiające zamienianiu koordynatów na offsetowe z kubicznych, jak i na odwrót. Jest wykorzystywana również w celu znajdowania wszystkich pól widocznych i niewidocznych dla gracza oraz definiuje pola, które zostały kliknięte kursorem myszy przez gracza i jest wykorzystywane do renderowania obiektów na polach mapy.

Klasa *Hex* zajmuje się przechowywaniem koordynatów kubicznych pól, a więc ich położenia względem trzech osi, jak i również zawiera metody, które służą do dodawania i odejmowania pól oraz wybierania określonego kierunku, znajdowania sąsiednich pól, obliczania odległości pól od pola początkowego oraz ilości pól potrzebnych do przebycia z jednego pola na drugie. Jest głównym źródłem określania sposobu poruszania się oddziałów wrogiego gracza oraz pól na które gracze mogą wykonywać ruch.

Klasa *FractionalHex* odpowiada za trzymanie koordynatów pól frakcyjnych, a więc takich, które w wypadku pól sześciokątnych są rozpatrywane jako sześć trójkątów składających się w pole sześciokątne. Klasa ta posiada możliwość zaokrąglenia koordynatów frakcyjnych do kubicznych. Używana jest w wypadku zaokrąglania koordynatów kliknięć kursora myszki do koordynatów używanych w grze.

5.9 Obsługa myszy

Obsługa myszy odbywa się poprzez zgłaszanie klikniętego punktu oraz ich zamianę na koordynaty używane w aplikacji, licząc punkt początkowy od górnego lewego rogu. Aplikacja jest informowana, który z klawiszy został wciśnięty.

W grze realizowane jest to za pomocą klasy *GameInput*, która implementuje interfejs *InputProcessor* i rozszerza jej metodę o nazwie *touchDown* o obsługę kliknięcia lewym przyciskiem myszy, następnie dane te są przekazywane poprzez klasę *InputStream* do sceny rozgrywki, która wykonuje odpowiednie zadania zależne od zdefiniowanych tam czynników.

```

@Override
public boolean touchDown (int x, int y, int pointer, int
button) {
    //Sprawdzenie kliknięcia dla lewego przycisku myszy
    //zdefiniowanego w interfejsie Input
    if (button == Input.Buttons.LEFT) {
        // wysyłanie danych o klikniętym punkcie.
        postMC.postMouseClicked(x, y);
        return true;
    }
    return false;
}

```

Listing 9: Obsługa kliknięcia lewego przycisku myszy

5.10 Zapisywanie i wczytywanie (pliki XML)

Zapisywanie i odczytywanie plików XML odbywa się przy udziale klas *XmlWriter* i *XmlReader*. Użyto tych plików do zapisywania i wczytywania stanu gry oraz pośrednio do zapisywania i wczytywania zapisanych ustawień poprzez obiekt *Preferences*.

Zapisywanie polega na zapisaniu obiektów graczy, budynków i jednostek do pliku o nazwie *save*. Wykonuje się to za pomocą obiektu *XmlWriter*. Jego działanie polega na tworzeniu elementów i atrybutów o podanych wartościach, a następnie zapisanie ich wszystkich za pomocą metody *writeString* klasy *FileHandle* do podanego w obiekcie tej klasy pliku. Elementy tworzy się poprzez wywołanie metody *element* z nazwą tagu jako argumentem, które następnie zamyka się poprzez metodę *pop*. Dodawanie atrybutów odbywa się przy pomocy metody *attribute* z nazwą atrybutu i jej wartością jako argumentami.

Wczytywanie polega natomiast na wczytaniu zapisanych obiektów poprzez działania na obiekcie klasy *XmlReader* oraz obiektów klasy *Element*. Klasa ta przechowuje elementy o podanych nazwach i umożliwia pobieranie wartości podanych atrybutów przez metodę *getAttribute*. Na podstawie wczytanych elementów tworzone są nowe, które są identyczne z tymi, które istniały w momencie zapisu.

```

<root>
<GameHUD playerColor="RED" gold="130" food="48" techPoints="1"
    turn="11"/>
<HumanPlayer color="RED" gold="130" food="48" techpoints="1" turn="1"
    race="Ludzie" tech1="true" tech2="false" tech3="false"/>
<AIPlayer color="BLUE" gold="275" food="43" techpoints="0" turn="1"
    race="Ludzie"/>
<TileBuildings id="0" q="0" r="3" s="-3" maxHP="100" HP="100.0"
    dayGold="8" dayFood="3" dayTechPoints="0.1" playerColor="RED"
    textureName="castle" race="Ludzie"/>
<TileBuildings id="1" q="3" r="-2" s="-1" maxHP="100" HP="100.0"
    dayGold="8" dayFood="3" dayTechPoints="0.1" playerColor="BLUE"
    textureName="castle" race="Ludzie"/>
<Units id="0" q="1" r="2" s="-3" meleeUnits="4" rangeUnits="2"
    specialUnits="0" move="1.0" playerColor="RED" race="Ludzie"/>
<Units id="3" q="4" r="-1" s="-3" meleeUnits="4" rangeUnits="2"
    specialUnits="0" move="1.0" playerColor="BLUE" race="Ludzie"/>
</root>

```

Listing 10: Przykładowy plik zapisu gry

```

Array<Element> o = root.getChildrenByName("GameHUD");
for (Element e: o)
{
    String playerColor = e.getAttribute("playerColor");
    int gold = Integer.parseInt(e.getAttribute("gold"));
    int food = Integer.parseInt(e.getAttribute("food"));
    float techPoints =
Float.parseFloat(e.getAttribute("techPoints"));
    int turn = Integer.parseInt(e.getAttribute("turn"));
    gameHUD = new GameHUD(playerColor, gold, food,
    techPoints, turn);
    this.gamehud = gameHUD;
}

```

Listing 11: Wczytywanie pliku XML

5.11 Gracze

Graczy w aplikacji można podzielić na graczy ludzkich i komputerowych i reprezentowani są przez klasy *HumanPlayer* i *AIPlayer*. Oboje posiadają strukturę, które pozwala na określenie koloru gracza, który stanowi ich unikalny identyfikator, a ponadto posiadane złoto, jedzenie, punkty nauki, aktualną turę i rasę gracza. Gracz ludzki posiada również listę z dostępnymi w grze technologiami i metodę *payByTechPoints*, służącą do odejmowania wskazanej ilości punktów nauki w strukturze obiektu. Obie klasy posiadają metodę *payFor*, która jako argumenty przyjmuje ilość złota oraz jedzenie, które należy odjąć od tych wartości, które posiada obiekt. Klasa gracza komputerowego jest opisywana dodatkowo przez zbiór metod, które służą jako ich AI i będą opisane w 5.14.

```
//Konstruktor klasy gracza.
public HumanPlayer(String color, int gold, int food, float
techpoints, int turn, String race, ArrayList<Boolean> tech) {
    this.technology = tech;
    this.color = color;
    this.gold = gold;
    this.food = food;
    this.techpoints = techpoints;
    this.turn = turn;
    this.race = race;
}
//Odejmnowanie wskazanej ilości złota i gracza dla tego
obiektu
public void payFor(int gold, int food)
{
    this.gold = this.gold - gold;
    this.food = this.food - food;
}
//Odejmnowanie wskazanej ilości nauki dla tego obiektu.
public void payByTechPoints(float techpoints) {
    this.techpoints -= techpoints;
}
```

Listing 12: Konstruktor obiektu gracza oraz metody do płacenia złotem, jedzeniem i punktami nauki.

5.12 Budynki

Obiekty budynków zostały napisane jako graficzna reprezentacja określonych ulepszeń pól, które posiadają zbiór atrybutów je opisujących oraz metod w celu wykonywania na nich operacji. Tworzenie budynków odbywa się poprzez konstruktor klasy *TileBuildings*, który jako argumenty przyjmuje: identyfikator budynku, współrzędne q oraz r i s, maksymalne oraz aktualne punkty wytrzymałości ulepszenia, dzienny przyrost złota oraz jedzenia i punktów nauki, kolor gracza, teksturę do reprezentacji graficznej oraz rasę. Konstruktor wywołuje metodę *create* z teksturą jako jej argumentem. Metoda ta tworzy *Sprite* oraz *ProgressBar* będący reprezentacją punktów wytrzymałości.

Jako obiekt, który posiada graficzną reprezentację posiada metody do ich renderowania oraz niszczenia zasobów tego obiektu. Klasa posiada również metodę *updateFillBar*, która służy do aktualizacji paska postępu do aktualnej wytrzymałości i jest wywoływana przy zmianie aktualnych punktów wytrzymałości budynków. Kolejną metodą jest *updateTexture*, która jest wywoływana w celu zmiany tekstury według opisu obiektu. Korzysta się z niej, gdy punkty wytrzymałości spadną do zera, czyli następuje tym samym przejęcie tego obszaru przez innego gracza.

```
//Aktualizacja tekstury obiektu.  
public void updateTexture() {  
    this.img = new  
        Texture(Gdx.files.internal(this.textureName +  
            playerColor + race + ".png"));  
    this.tileBuilding.setRegion(img);  
}  
//Aktualizacja wyświetlanych na pasku punktów  
wytrzymałości  
public void updateFillBar(float progress) {  
    this.bar.setValue(progress);  
}
```

Listing 13: Metody do aktualizacji tekstury i paska postępu ulepszeń pól

5.13 Oddziały

Oddziały są obiektami, które posiadają reprezentację graficzną i informacyjną oraz zbiór atrybutów je opisujący i metody, które służą do działań na obiektach. Tworzenie oddziałów odbywa się poprzez konstruktor klasy *Units*, który jako argumenty przyjmuje: identyfikator obiektu, koordynaty *q* oraz *r* i *s*, liczbę jednostek bliskiego zasięgu oraz dalekiego zasięgu i specjalnych, liczbę punktów ruchu, kolor gracza oraz rasę. Konstruktor ten wywołuje metodę *create*, która tworzy reprezentację graficzną i informacyjną obiektów. Pierwsza reprezentacja jest realizowana dzięki obiektom *Sprite*, natomiast druga dzięki obiektom *Label*, które wyświetlają liczbę jednostek w oddziale.

Klasa posiada reprezentację graficzną, więc wymagane było stworzenie metody *render* oraz *dispose*, które odpowiadają za rysowanie tekstury na obiekcie *Sprite* oraz jej niszczenie. Aktualizacja informacji o jednostkach w oddziale jest realizowana poprzez metodę *updateArmy*, która zmienia tekst obiektu *Label*, gdy rekrutowane są jednostki lub dochodzi do ataku na oddział.

```
//Tworzenie reprezentacji graficzno-informacyjnej obiektu
public void create () {
    this.batch = new SpriteBatch();
    this.stage = new Stage();
    this.img = new Texture(Gdx.files.internal("knight" +
        playerColor + ".png"));
    this.unit = new Sprite(img);
    this.unitsCount = new Label(String.format("%2d/%2d/%2d",
        meleeUnits, rangeUnits, specialUnits), new
        Label.LabelStyle(new BitmapFont(), Color.WHITE));
    Pixmap labelColor = new Pixmap(50, 20,
        Pixmap.Format.RGB888);
    labelColor.setColor(Color.BLACK);
    labelColor.fill();
    unitsCount.getStyle().background = new Image(new
        Texture(labelColor)).getDrawable();
    stage.addActor(unitsCount);
}
```

Listing 14: Tworzenie obiektu oddziału

5.14 AI

Działania gracza komputerowego realizowane są przez zbiór siedemnastu metod w klasie *AIPlayer*, odpowiadające za wykonywanie czynności, które zmieniają atrybuty obiektu oraz takie, które odpowiadają za kalkulacje wag możliwych czynności i podejmowanie na ich podstawie najlepszych decyzji. Działania graczy komputerowych mają miejsce po zakończeniu tury użytkownika aplikacji.

Główną metodą działania AI jest *makeADecision*, która wywołuje kolejne metody odpowiedzialne za obliczanie wag decyzyjnych, czyli *buyBuildingCalculations*, *buyUnitCalculations*, *attackEnemyCalculations* oraz *attackTileCalculations*. Następnie są wykonywane dwie fazy akcji. Pierwszą z nich jest decyzja co zrobić z posiadanymi przez gracza komputerowego złotem i jedzeniem. Możliwymi działaniami jest kupno ulepszenia pola oraz rekrutacja jednostek. Wykonane mogą być dwa powyższe działania, jedno lub też żadne.

```
//Kupno ulepszenia, gdy true
if (buyBuildingCalculations.get(oF) >= buyUnitCalculations.get(oF)) {
    buyTileBuilding(unitsList.get(i), buildingsList);
    //Obliczanie nowej wagi dla zakupu jednostek
    buyUnitCalculations.set(oF, buyUnits(unitsList.get(i), mapX,
    mapY));
    // Zakup jednostek dla wagi równej lub większej niż 1
    if (buyUnitCalculations.get(oF) >= 1) {
        buyUnit(unitsList.get(i), mapX, mapY);
        attackEnemyCalculations.set(oF, attackEnemy(unitsList.get(i),
        mapX, mapY));
    }
    //Rekrutacja, gdy false
} else {
    buyUnit(unitsList.get(i), mapX, mapY);
    //Obliczanie nowej wagi dla zakupu ulepszenia pola
    buyBuildingCalculations.set(oF, buyBuilding(unitsList.get(i),
    buildingsList));
    attackEnemyCalculations.set(oF,
    attackEnemy(unitsList.get(i), mapX, mapY));
    //Zakup ulepszenia dla wagi równej lub większej niż 1
    if (buyBuildingCalculations.get(oF) >= 1) {
        buyTileBuilding(unitsList.get(i), buildingsList);
    }
}
```

Listing 15: Dokonywanie decyzji nad zakupem ulepszenia lub jednostek

Drugą fazą jest wykonywanie ruchu. By ta faza nastąpiła, to oddział musi posiadać przynajmniej jeden punkt ruchu. W wypadku możliwego ruchu są podejmowane decyzje dotyczące atakowania pola z ulepszeniem, pola z wrogiem oddziałem lub wyszukiwanie i ruch w kierunku pola, które wydaje się najbardziej korzystne pod względem ekonomicznym lub militarnym, a bardziej uprecyzowane będzie to w opisie metody *makeMove*.

```
//Gdy punkty ruchu wynoszą przynajmniej 1
if(unitsList.get(i).move>=1){
//Jeżeli istnieje pole, którego waga ataku na ulepszenie
wynosi
//więcej niż waga ataku na wrogie oddziały
if(attackTileCalculations.get(oF).calculation>=
attackEnemyCalculations.get(oF).calculation &&
attackTileCalculations.get(oF).hex != null){
//Atakowanie ulepszenia pola
tMS.getNeighbors(selectedID);
tMS.moveUnit(selectedID,
attackTileCalculations.get(oF).hex, this.color);
//Jeżeli istnieje pole, którego waga ataku na wrogie
oddziały //była większa niż waga ataku na ulepszenie
} else if(attackEnemyCalculations.get(oF).hex != null){
//Gdy siła oddziałów jest większa od siły oddziałów
//wroga, to atakuj
if(attackEnemyCalculations.get(oF).calculation>=0.9){
tMS.getNeighbors(selectedID);
tMS.moveUnit(selectedID,
attackEnemyCalculations.get(oF).hex, this.color);
//Gdy wrogie oddziały były silniejsze, to szukaj
//najlepszego rozwiązania
} else {
makeMove(unitsList.get(i), buildingsList, unitsList,
mapX, mapY);
}
}
//Jeżeli nie ma możliwości ataku ani na ulepszenie, ani
na wroga //oddziały, to szukaj najlepszego rozwiązania
} else {
makeMove(unitsList.get(i), buildingsList, unitsList,
mapX, mapY);
}
}
```

Listing 16: Dokonywanie decyzji dotyczącej ruchu

Metody *getSummaryGold* i *getSummaryFood* służą w celu obliczenia dziennego przyrostu jednostek złota i jednostek jedzenia. Są używane w celu obliczenia wag skierowanych na poprawę ekonomicznej sytuacji królestwa na podstawie listy z obiektami typu *TileBuildings* i koloru gracza komputerowego.

```
public int getSummaryFood(ArrayList<TileBuildings>
buildingsList, String color) {
    int foodSummary = 0;
    //Przechodzenie po obiektach listy i dodawanie do
    //foodSummary wartości dayFood, gdy kolor gracza
    //będzie identyczny jak ten podany
    for(int i=0;i<buildingsList.size();i++){
        if(buildingsList.get(i).playerColor.equals(color))
        {
            foodSummary += buildingsList.get(i).dayFood;
        }
    }
    //Zwracanie obliczeń
    return foodSummary;
}
```

Listing 17: Obliczanie dziennego przyrostu jedzenia

Poza przeliczaniem dziennego przyrostu dóbr, klasa *AIPlayer* oferuje jeszcze inne metody, które są realizowane pośrednio z metody *makeADecision* i odpowiadają za maksymalną liczbę jednostek możliwych do zakupu oraz korektę siły oddziałów. Pierwsza czynność jest realizowana przez metodę *maxAbleToBuy* i *getMax*, które definiują maksymalną liczbę jednostek, opierając się na podanych w argumentach jednostek złota i jedzenia, które opisują cenę rekrutacji danej jednostki. Druga czynność natomiast jest realizowana przez metodę *recalcPower* i zwraca *true* lub *false* w zależności, czy siła oddziału po rekrutacji jednostek będzie większa niż siła oddziałów wroga.

```
//Pobieranie maksymalnej liczby jednostek możliwych do
zakupu
private int getMax(int goldCost, int foodCost) {
    int maxGold = (int) (this.gold/goldCost);
    int maxFood = (int) (this.food/foodCost);
    //zwracanie mniejszej wartości
    if (maxGold<=maxFood) return maxGold; else return
    maxFood;
}
```

Listing 18: Obliczanie maksymalnej liczby jednostek, którą można zakupić

Metoda *buyBuilding* zajmuje się obliczaniem wagi dla zakupu ulepszenia pola. Wykonywane są działania przy pomocy metod *getSummaryGold* i *getSummaryFood* dla obu graczy w celu obliczenia różnicy w ich dziennym przyroście dóbr. Wartość decyzyjna jest ustawiona początkowo na 0.5, by w wypadku małej różnicy nie rezygnowano całkowicie z ulepszeń pól oraz jest zerowana, gdy ulepszenie już istnieje. Jeżeli w *makeADecision* podjęto decyzję dotyczącą zakupu ulepszenia, to wywoływana jest metoda *buyTileBuilding*, która odpowiada za zakup ulepszenia na aktualnym polu.

```
//Fragment obejmuje wypadek, gdy ulepszenie na aktualnym
polu nie istnieje. Zmienne goldSummary i foodSummary
definiują przyrost dóbr gracza komputerowego, gdy
goldHumanSummary i foodHumanSummary definiują przyrost
gracza ludzkiego. Zmienna calculation natomiast stanowi
wagę kalkulacji dla ulepszenia pola.
if (goldSummary<=goldHumanSummary) {
    calculation += (goldHumanSummary-goldSummary)*0.1;
}
if (foodSummary<=foodHumanSummary) {
    calculation += (foodHumanSummary-foodSummary)*0.2;
}
```

Listing 19: Obliczanie wag do zakupu ulepszenia pola

Metoda *buyUnits* zajmuje się obliczaniem wagi dla zakupu jednostek. Bierze ona pod uwagę sytuację militarną na sąsiednich do jednostki polach oraz przelicza siłę wrogich oddziałów, jeżeli się tam znajdują. Jeżeli siła jednostek w oddziale jest mniejsza od 90% siły wrogich oddziałów, to następuje przeliczanie maksymalnej liczby jednostek, które można rekrutować oraz waga jest regulowana w zależności od korekty siły dzięki metodzie *recalcPower*. Gdy w *makeADecision* podjęto działania prowadzące do zakupu jednostek, to wywoływana jest metoda *recruitUnit*, która odpowiada za rekrutację jednostek do oddziału do czasu, gdy siła oddziału nie będzie większa o 25% od siły oddziałów wroga lub sytuacja w złocie i jedzeniu nie będzie korzystna dla dalszej ich rekrutacji.

```
if((thisUnitPower<enemyUnitsPower*0.9)) {
    //Obliczanie maksymalnej liczby jednostek do rekrutacji
    maxMelee = maxAbleToBuy(10,3);
    maxRange = maxAbleToBuy(15,2);
    maxSpecial = maxAbleToBuy(30,8);
    if(maxMelee >=maxRange && maxMelee >=maxSpecial*3.5) {
        if(recalcPower(enemyUnitsPower, thisUnitPower,
            maxMelee, 0, 0))
        {
            calculation += 1;
        } else if(maxSpecial*3.5>=maxRange &&
            maxSpecial*3.5>=maxMelee )
        {
            if(recalcPower(enemyUnitsPower, thisUnitPower, 0, 0,
                maxSpecial))
            {
                calculation += 1;
            }
        } else {
            if(recalcPower(enemyUnitsPower, thisUnitPower, 0,
                maxRange, 0))
            {
                calculation += 1;
            }
        }
    }
    } else calculation -= 0.3;
```

Listing 20: Obliczanie decyzji zakupu jednostek

Metoda *attackEnemy* zajmuje się obliczaniem wag dla decyzji dotyczącej ataku na wrogie oddziały. Bierze ona pod uwagę sytuację militarną na sąsiednich polach i w zależności od siły militarnej bieżącego oddziału i sąsiednich obliczana jest waga, która przyjmuje wartość stosunku siły oddziału gracza komputerowego do sumy siły oddziałów wroga, które znajdują się o jeden punkt ruchu od tego oddziału.

Metoda *attackTile* zajmuje się obliczaniem wag decyzyjnych dotyczącej przejmowania wrogich ulepszeń pól. Początkowa waga decyzyjna jest równa 3. Waga ta jest zmniejszana stosunkowo do ilości potrzebnych do przejścia ulepszenia tur. Zastosowano takie rozwiązanie ze względu na sposób podejmowania decyzji o ulepszaniu pola – gdy sytuacja ekonomiczna byłaby znacząco gorsza, to waga decyzji dotyczącej ulepszenia pola mogłaby znacząco przeważać nad przejmowaniem ulepszenia.

```
for (int i=0; i<neighborBuildings.size(); i++)
{
    calcHex.calculation = calculation -
    (neighborBuildings.get(i).HP/conquerPower);
    return calcHex;
}
```

Listing 21: Obliczanie decyzji nad atakiem pola

Metoda *makeMove* ma za zadanie podejmować decyzję dotyczące ruchu, gdy inne rozwiązania w *makeADecision* nie zostaną uznane za wartościowe lub nie będzie możliwości ich wykonania. Metoda ta bierze pod uwagę sytuacje aktualnie posiadanych dóbr, jak i tych zdobywanych w ciągu tury, a także sytuację militarną. Na ich podstawie podejmowana jest decyzja do czego oddziały gracza komputerowego powinny dążyć. Następnie sprawdzane są pola na które oddziały powinny się przemieścić według zebranych informacji i wybierane jest to, które znajduje się w najmniejszej odległości od oddziału. Jest to możliwe dzięki metodom *getDistance* oraz *getHexToGetTo*. Pierwsza z nich oblicza dystans do wskazanego pola z pola aktualnego. Druga natomiast oblicza koordynaty pola na które oddziały muszą wykonać ruch, by przechodzić w kierunku pola wskazanego.

```
//Sprawdzanie dystansu potrzebnego do przebycia z pola na
którym znajduje się oddział do pola wskazanego
private int getDistance (Units thisUnit, Hex hex) {
    return Hex.dis(new Hex(thisUnit.q, thisUnit.r,
thisUnit.s), hex);
}
```

Listing 22: Sprawdzanie dystansu pomiędzy polami

```
//Fragment metody getHexToGetTo pokazujący działania w
wypadku, gdy koordynat q aktualnego pola i wskazanego są
takie same
if (start.q == end.q)
{
    checkpointQ = start.q;
    if (start.r > end.r)
    {
        checkpointR = start.r-1;
        checkpointS = start.s+1;
    } else {
        checkpointR = start.r+1;
        checkpointS = start.s-1;
    }
} else if (start.q != end.q)
```

Listing 23: Obliczanie współrzędnych pola na które oddział ma przejść

Zakończenie

Celem pracy było zaimplementowanie gry 4X, polegającej na przejmowaniu obszarów oraz spełnienie postawionych wymagań opisanych w założeniach. Spełniono wszystkie założenia funkcjonalne. Opis gry został przełożony na aplikację w sposób zgodny z założeniami.

Gracz dostał możliwość wyboru rasy spośród trzech dostępnych, co umożliwia mu dostosowanie premii startowych. Każda rasa posiada inaczej wyglądające oddziały i budynki. Gracz może rekrutować jednostki, przemieszczać je po mapie oraz atakować ulepszenia i oddziały przeciwników. Pola można ulepszać, co sprawia, że gracz otrzymywał będzie pewną liczbę dóbr za każde ulepszenie. W aplikacji zaimplementowano ustawienia gry, które umożliwiają dostosowanie poziomu głośności muzyki i jej wyłączenie. W przeciwieństwie do innych gier z tego gatunku, gracz posiada określoną liczbę oddziałów, które otrzymuje na początku rozgrywki. Sprawia to, że na gracza jest nakładana większa presja w związku z nieprzemyślanymi decyzjami. Oddziały te składają się też z trzech rodzajów jednostek, co sprawia, że oddziały można bardziej dostosować niż w wypadku gier, które nie posiadają oddziałów, lecz pojedyncze jednostki. Pola mapy są widoczne jedynie, gdy na sąsiednich polach znajdują się ulepszenia pól lub oddziały, które należą do gracza. Zastosowano do tego obraz pustego pola, co w porównaniu z efektami cząsteczkowymi z podobnych aplikacji, sprawia, że aplikacja nie wymaga znacząco większego zapotrzebowania na moc obliczeniową procesora. Szereg algorytmów w AI gracza komputerowego sprawia, że działania przeciwników nie opierają się na losowości, lecz brane jest pod uwagę wiele czynników, które prowadzą do decyzji, dotyczących podjęcia określonych działań. Gra posiada możliwość zapisu stanu gry i wczytywania ustawień oraz rozgrywek.

Aplikacja została podzielona na klasy, zastosowano podział plików gry i system komentarzy, a sam kod gry używał przejrzystego nazewnictwa. Pliki gry mogą być zmieniane bez wprowadzania zmian w kodzie. Dodatkowo załadowane do zasobów pliki są usuwane, gdy nie zachodzi już potrzeba ich używania. Aplikacja jest warstwą podstawową gier 4X w porównaniu z najbardziej znanymi grami tego gatunku, dlatego też można ją rozwijać na wiele sposobów –

dodać moduły odpowiedzialne za dyplomację lub handel, wprowadzić podział na ery, oferujący nowe jednostki, rozbudować system technologii, wprowadzić wiele rodzajów terenów i surowce luksusowe lub strategiczne. Kod aplikacji zawiera metody odpowiedzialne za sprawdzanie, czy dane obiekty znajdują się na wskazanym polu, metody do rekrutacji, podstawowy system technologii oraz wiele innych mechanizmów, co w znaczący sposób usprawnia rozwijanie aplikacji.

Bibliografia

- [1] Statistic Brain, Reading Statistics, <http://www.statisticbrain.com/reading-statistics>, 2016, czas dostępu: 12.01.2017r.
- [2] Entertainment Software Association, Essential facts about the computer and video game industry, <http://www.theesa.com/wp-content/uploads/2015/04/ESA-Essential-Facts-2015.pdf>, 2015, czas dostępu: 16.01.2017 r.
- [3] Newzoo, NEWZOO SUMMER SERIES #11: POLISH GAMES MARKET, <https://newzoo.com/insights/infographics/newzoo-summer-series-11-polish-games-market>, 2015, czas dostępu: 16.01.2017 r.
- [4] Kluska B., Dawno temu w grach. Czas pionierów: szkice z historii gier komputerowych, 2008
- [5] Endrich A. „MicroProse' Strategic Space Opera Is Rated XXXX!”, Computer Gaming World, 10/1993
- [6] Hoey P., Mastering LibGDX Game Development, 2015
- [7] Badlogic Games, libgdx API, <http://libgdx.badlogicgames.com/nightlies/docs/api>, , czas dostępu: 12.03.2017 r.
- [8] Badlogic Games, libGDX Wiki, <https://github.com/libgdx/libgdx/wiki>, , czas dostępu: 12.03.2017 r.

Spis rysunków

Rys. 1: Przykładowe ekrany z Sid Meier's Civilization VI oraz Master of Orion.....	12
Rys. 2: Widok z gry Sid Meier's Civilization V.....	15
Rys. 3: Widok z gry Warlock - Master of the Arcane.....	17
Rys. 4: Widok z gry Freeciv.....	19
Rys. 5: Widok z gry Master of Orion.....	20
Rys. 6: Widok z gry Heroes of Might and Magic III.....	22
Rys. 7: Widok z gry Disciples.....	24
Rys. 8: Przykładowe pole kwadratowe.....	26
Rys. 9: Możliwe pola do przeniesienia oddziału w wypadku dodatkowego punktu ruchu	27
Rys. 10: Możliwe pola do przeniesienia oddziału w wypadku ruchu pod kątem.....	28
Rys. 11: Przykładowe pole sześciokątne.....	29
Rys. 12: Przykładowe pola predefiniowane.....	30
Rys. 13: Widok z gry Land Terror z opisem elementów.....	38
Rys. 14: Menu główne gry Land Terror.....	41
Rys. 15: Widok mapy z gry Land Terror.....	45
Rys. 16: Postrzeganie sceny przez kamerę izometryczną.....	56
Rys. 17: Przykładowa grafika opisywana przez atlas.....	57
Rys. 18: Przykłady stworzonych przycisków(dwa tekstowe i jeden z obrazem).....	60
Rys. 19: Ekrany SplashScreen, MenuScreen i GameScreen.....	63
Rys. 20: Właściwości mapy.....	66
Rys. 21: Właściwości kafelka.....	66
Rys. 22: Poszczególne długości i odległości pól sześciokątnych.....	67
Rys. 23: Podstawowe rozmiary wykorzystywane przy obsłudze pól.....	68

Spis tabel

Tabela 1: Zalety i wady Sid Meier's Civilization V.....	15
Tabela 2: Zalety i wady Warlock - Master of the Arcane.....	17
Tabela 3: Zalety i wady Freeciv.....	18
Tabela 4: Zalety i wady Master of Orion.....	20
Tabela 5: Zalety i wady Heroes of Might and Magic III.....	22
Tabela 6: Zalety i wady Disciples.....	24
Tabela 7: Statystyki i koszt ulepszeń pól.....	46
Tabela 8: Koszt rekrutacji jednostek.....	47
Tabela 9: Nazwa, koszt i opis technologii w Land Terror.....	48

Spis Listingów

Listing 1: Tworzenie kamery i jej ustawianie.....	56
Listing 2: Fragment atlasu opisującego HUD oddziałów.....	57
Listing 3: Syntaktyka tworzenia stylu.....	58
Listing 4: Tworzenie tabeli dla elementów ekranu ustawiania opcji rozgrywki.....	59
Listing 5: Przykład tworzenia przycisku tekstowego.....	60
Listing 6: Zapisane ustawienia gry Land Terror.....	61
Listing 7: Wczytanie ustawień głośności i odtwarzania dźwięku.....	62
Listing 8: Odtwarzanie i zatrzymywanie muzyki oraz ustawianie jej głośności.....	64
Listing 9: Obsługa kliknięcia lewego przycisku myszy.....	70
Listing 10: Przykładowy plik zapisu gry.....	71
Listing 11: Wczytywanie pliku XML.....	71
Listing 12: Konstruktor obiektu gracza oraz metody do płacenia złotem, jedzeniem i punktami nauki.....	72
Listing 13: Metody do aktualizacji tekstury i paska postępu ulepszeń pól.....	73
Listing 14: Tworzenie obiektu oddziału.....	74
Listing 15: Dokonywanie decyzji nad zakupem ulepszenia lub jednostek.....	75
Listing 16: Dokonywanie decyzji dotyczącej ruchu.....	76
Listing 17: Obliczanie dziennego przyrostu jedzenia.....	77
Listing 18: Obliczanie maksymalnej liczby jednostek, którą można zakupić.....	78
Listing 19: Obliczanie wag do zakupu ulepszenia pola.....	78
Listing 20: Obliczanie decyzji zakupu jednostek.....	79
Listing 21: Obliczanie decyzji nad atakiem pola.....	80
Listing 22: Sprawdzanie dystansu pomiędzy polami.....	81
Listing 23: Obliczanie współrzędnych pola na które oddział ma przejść.....	81