

UNIWERSYTET PEDAGOGICZNY

im. Komisji Edukacji Narodowej w Krakowie



INSTYTUT INFORMATYKI

Kierunek: INFORMATYKA

specjalność: Administracja Systemami Informatycznymi

Multimedia i technologie internetowe

Mateusz Bugaj

Nr albumu: 138093

Dominik Froński

Nr albumu: 138111

Gra platformowa z elementami algorytmiki

Praca inżynierska
napisana pod kierunkiem
dr. Romana Czapli

Kraków 2021

Spis treści

Wstęp	4
1 Tematyka projektu	6
1.1 Gra komputerowa	6
1.2 Gra platformowa	7
1.3 Tworzenie gier komputerowych	9
1.3.1 Zespoły tworzące gry komputerowe	9
1.3.2 Etapy tworzenia gier komputerowych	12
1.3.2.1 Faza Preprodukcyjna	12
1.3.2.2 Faza Produkcyjna	13
1.3.2.3 Faza Postprodukcyjna	14
2 Fabuła gry "Tree of life"	15
2.1 Pomysły zrealizowane	15
2.1.1 Przeszłość	15
2.1.2 Teraźniejszość	15
2.1.2.1 Poszukiwanie lekarstwa	15
2.1.2.2 Mroczna Kraina	16
2.2 Pomysły niezrealizowane	24
2.3 Scenariusze rozwoju gry <i>Tree Of Life</i>	25
3 Grafika	27
3.1 Rodzaje grafiki	27
3.1.1 Grafika 2D	27
3.1.2 Grafika 3D	30
3.2 Grafika w grach komputerowych	31
3.2.1 Graficzne aspekty gier	31
3.2.2 Animacje	34
3.3 Grafika w grze "Tree of Life"	35

4	Techniczne aspekty tworzenia gier komputerowych	38
4.1	Silniki gier komputerowych	38
4.2	Biblioteki do tworzenia gier komputerowych	39
4.2.1	The Lightweight Java Game Library	39
4.2.2	Biblioteka Pygame	40
4.3	Algorytmy i struktury użyte w grze "Tree Of Life"	40
4.3.1	Przeliczanie Binarne	41
4.3.2	Sortowanie bąbelkowe	41
4.3.3	Szyfr PlayFair	41
4.3.4	Drzewa Binarne	42
4.4	Struktura projektu	43
4.5	Testowanie Gry	48
	Zakończenie	49
	Spis rysunków	53

Wstęp

Celem pracy jest stworzenie gry platformowej inspirowanej klasyczną grą *Super Mario Bros.* Przeszukując różne źródła informacji autorzy nie spotkali się z połączeniem gry platformowej z nauką algorytmów i struktur danych. Twórcy chcąc przybliżyć osobom niezwiązanym z tą dziedziną nauki postanowili stworzyć grę edukacyjną, która połączy rozrywkę z nauką algorytmów powszechnie stosowanych w informatyce. Praca pisana była w Python 3.7 oraz bibliotece Pygame 2.0. Gra stworzona była przez dwie osoby z jasnym podziałem obowiązków:

- Pierwsza osoba (Mateusz Bugaj) odpowiedzialna była za część algorytmiczną całej rozgrywki. Jej zadania pokrywały się ze stanowiskami zaliczanymi do ról programistycznych w profesjonalnej produkcji gier. Ten autor stworzył również rozdziały poświęcone tematyce projektu oraz technicznym aspektom gier komputerowych.
- Druga osoba (Dominik Froński) odpowiedzialna była za część fabularną oraz graficzną całej rozgrywki. Jej zadania pokrywały się ze stanowiskami zaliczanymi do ról kreatywnych w profesjonalnej produkcji gier. Ten autor stworzył również rozdziały poświęcone grafice oraz fabule rozgrywki.
- Osoby tworzące grę, musiały wykazać również umiejętności pracy w grupie: wspólnie wybierały one algorytmy przedstawione w grze.

Pierwszy rozdział porusza kwestie teoretyczne związane z grami komputerowymi: ukazana została historia gier oraz etapy ich tworzenia. Drugi rozdział przedstawia fabułę, w jakiej umieszczona została rozgrywka, oraz opisane zostały pomysły niezrealizowane podczas jej produkcji. Trzeci rozdział opisuje zagadnienia związane z częścią artystyczną tej pracy - grafiką: ukazane zostały rodzaje grafik oraz sposoby ich użycia w grach komputerowych. Czwarty rozdział przedstawia część programistyczną podczas produkcji tychże projektów: opisane zostały aktualne metody ich tworzenia oraz przybliżone zostały etapy stworzenia niniejszego projektu. W tym

rozdziale zawarte zostały także opisy algorytmów oraz wykorzystane sposoby testowania oprogramowania.

Rozdział 1

Tematyka projektu

W tym rozdziale przedstawiona zostanie historia gier komputerowych, w szczególności opisane zostaną zagadnienia związane z grami platformowymi. Poruszone również będą kwestie odnoszące się do ich produkcji: scharakteryzowane zostaną zespoły tworzące tytuły oraz wymagane dokumenty powstające w poszczególnych etapach ich powstawania.

1.1 Gra komputerowa

Według słownika języka polskiego grą komputerową [1] jest "gra rozgrywana na ekranie monitora; też: program komputerowy umożliwiający tę grę". Definicja ta pozwala nam zrozumieć czym jest gra komputerowa, ale nic nie mówi o rodzajach gier - a jest ich naprawdę dużo. Możemy je dzielić ze względu na platformę, na której jest użytkowana, jak również dzielić z perspektywy programowania, czyli sposobu w jakim ta gra została napisana; w jakim języku, czy wykorzystane zostały silniki gier i tym podobne. Biorąc pod uwagę sposoby "używania" gier, czyli na urządzenia, na których są uruchamiane powinno się używać określenia *gry video*.

Powstaje pytanie, które z pewnością nurtuje niejednego użytkownika - "gracza": *Czym tak naprawdę jest gra komputerowa?* Przeszukując dostępne źródła literatury przedmiotu, główną cechą gry komputerowej jest dostarczanie rozrywki swoim użytkownikom. Jednak istnieją na rynku gry, których głównym przeznaczeniem jest edukacja użytkowników, wymagają one od graczy rozwiązywania łamigłówek logicznych jak również zręcznościowych.

Gry komputerowe wywodzą się od gier planszowych. Pierwszy prototyp gry elektronicznej pojawił się w 1947 roku. Do 1972 roku gry komputerowe pozostawały tylko w obrębie akademickim, kiedy to swoją premierę miała pierwsza gra komputerowa

Pong [2]. Początkowo gry były wydawane na automaty [3], czyli urządzenia umieszczone w miejscach publicznych, na których po wrzuceniu monety użytkownik może zagrać. Wraz z popularyzacją komputerów osobistych w latach osiemdziesiątych gry zaczęto produkować również na te platformy. Nie cieszyły się one jednak popularnością, ponieważ w tamtych czasach komputery były bardzo drogie, duże oraz przede wszystkim trudno dostępne. Pierwszymi grami, które trafiły na komputery osobiste były tak zwane *gry tekstowe*, gdzie gracz grał w nie poprzez linię komend. Prawdziwy przełom nastąpił w roku 1982, kiedy to odbyła się premiera komputera *ZX Spectrum*, który posiadał kolorowy wyświetlacz oraz umożliwiał odtwarzanie dźwięku. Wydarzenie to zapoczątkowało historię gier platformowych, które do obecnych czasów cieszą się ogromną popularnością. Bardzo duży wpływ miała na to premiera gry *Super Mario Bros*. Kolejne lata były bardzo intensywne pod względem liczby gier wydawanych na rynku: pojawiało się coraz więcej różnych gatunków gier dzięki wykorzystywaniu coraz to nowszych technologii. Obecnie na świecie istnieje wiele platform na które tworzone są gry: telefony, konsole oraz wspomniane już komputery osobiste.

1.2 Gra platformowa

Gry platformowe to jedne z pierwszych gier, które zostały wydane na komputery osobiste z kolorowymi wyświetlaczami. Idealnym przykładem takiej gry jest *Super Mario Bros* (zob. rysunek 1.1), które "wychowało pokolenia". Tego typu gry charakteryzują się przede wszystkim tytułowymi "platformami". Gracz porusza się po kolejnych poziomach, na których czekają go zadania o zróżnicowanym stopniu trudności takie jak: unikanie pułapek, pokonywanie przeciwników, zbieranie nagród takich jak: monety, tzw. potiony, czyli eliksiry regenerujące życie, dodatkowe życia itp. Takie gry wciąż powstają i cieszą się dużą popularnością. Jednym z nowszych tytułów jest gra *Hollow Knight* (zob. rysunek 1.2), której premiera odbyła się w 2017 roku.



Rys. 1.1: Gra "Super Mario Bros" z 1986. Źródło: [4]



Rys. 1.2: Gra "Hollow Knight" z 2017 Źródło: [5]

Obecnie na rynku gier istnieje bardzo dużo takich tytułów. Są one wydawane na każdą platformę. Zazwyczaj nie wymagają one wysokiej jakości sprzętu, ponieważ najczęściej są to gry typu 2D. Idealnie nadają się one do przedstawienia różnego typu zagadnień logicznych.

1.3 Tworzenie gier komputerowych

1.3.1 Zespoły tworzące gry komputerowe

Obecnie na świecie istnieje wiele firm, które produkują gry komputerowe. W zależności od wielkości studia skład takiego zespołu może się różnić: jedne zespoły mogą składać się z kilku osób, inne natomiast z kilkuset. W procesie tworzenia gry komputerowej można wyszczególnić kilka głównych grup. Podążając za Rafałem Nowocieniem [20] wyróżnia się kilka ролей niezbędnych do prawidłowego zrealizowania wszystkich etapów produkcji gry: zarządcze, kreatywne, artystyczne, programistyczne oraz wszystkie inne. Przyjętym na świecie standardem jest korzystanie z angielskiego nazewnictwa stanowisk, więc tak też będą one tutaj przedstawione. Do ról zarządczych możemy zaliczyć:

- *Investor* - jest to indywidualna osoba lub firma, która podjęła decyzję o finansowaniu produkcji gry. Zazwyczaj w zamian otrzymuje ona prawa do dystrybucji gry na rynku, dzięki czemu czerpie większość zysków ze sprzedaży. W małych zespołach czasami istnieje zjawisko polegające na tym, że wszyscy pracownicy są inwestorami - pracują oni za darmo, dzieląc się swoimi umiejętnościami podczas produkcji gry. Jeżeli ten zespół jest *startupem*, osoby dzielą się umiejętnościami w ramach *opensource*.
- *Producer* - osoba pełniąca rolę *Project managera*, która jest "łącznikiem" między zespołem produkującym grę, a inwestorem. Jej głównymi zadaniami są: planowanie budżetu, rozwiązywanie problemów, pojawiających się na etapie tworzenia produkcji oraz załatwianie komunikacji między poszczególnymi członkami zespołu.
- *Lead programmer* - zawsze jest to osoba, która ma największe doświadczenie na stanowisku programisty. Jego zadaniem jest zarządzanie zespołem programistów oraz tworzenie dokumentacji technicznej gry.
- *Lead designer* - jest to osoba, która odpowiada za przebieg rozgrywki, przygotowanie scenariusza gry oraz wszystkich niezbędnych w niej mechanizmów

oraz nadzorowanie pracy działu kreatywnego.

- *Art director* - osoba, która nadzoruje tworzenie warstwy wizualnej gry.

Kolejną grupą funkcji są role kreatywne. Osoby pracujące w tych zespołach odpowiadają za fabułę, wygląd poszczególnych etapów gry itp. Można tutaj zaliczyć takie stanowiska jak:

- *Game designer* - osoba będąca na tym stanowisku jest odpowiedzialna za projektowanie i testowanie mechanizmów rozgrywki. W mniejszych zespołach odpowiada ona za wszystkie poniższe pozycje w rolach kreatywnych.
- *Narrative designer/writer* - jest to osoba, która tworzy wszystkie elementy fabularne takie jak: opisy lokacji, opowieści, notatki czytane przez gracza oraz dialogi.
- *Level designer* - jest to projektant poziomów gry. Jego zadaniem jest opracowanie wyglądu danej lokacji oraz postaci spotykanych przez gracza jak również poziomy trudności gry.
- *Combat designer* - jest to osoba, która przygotowuje sekwencje walk: określa liczbę żywych przeciwników, poziomy trudności walk itp.
- *Economy/monetisation designer* - jego zadaniem jest ustalenie ekonomicznej wartości przedmiotów w grze. Ustala ceny towarów, tak aby w poszczególnych poziomach gracz miał odpowiednią ilość pieniędzy.

Osoby, które można przypisać do ról artystycznych są odpowiedzialne za wszystkie grafiki oraz animacje w produkcjach. Praca ta jest jedną z najważniejszych w całym procesie tworzenia gry, ponieważ to ich grafiki będą widoczne dla końcowych użytkowników. W ich skład wchodzą takie stanowiska jak:

- *Graphic designer* - jest to osoba przygotowująca poszczególne elementy graficzne do gry. Można tę pozycję podzielić na 2 typy: grafiki 2D (rysunki przedstawione są w 2 wymiarach, tzw. "płaski rysunek") oraz 3D (takie jak bryły trójwymiarowe, pojazdy i budynki).
- *GUI designer* - jego zadaniem jest przygotowanie interfejsu dla użytkownika, czyli: wyglądu ekranu gry, przycisków oraz ramki, dobór odpowiednich czcionek do tekstów wyświetlanych na ekranie itp.

- *Motion designer/animator* - jego zadaniem jest zaprojektowanie wszystkich animacji w grze, np.: realistyczne animacje chodzenia, walki lub efektów świetlnych.
- *Cinematic designer* - do jego zadań należy projektowanie tzw. *cutscen*^[6] - przerywników w grze, umieszczanych w najważniejszych momentach rozgrywki.
- *Concept artist* - jego praca traktowana jest jako wzór dla wszystkich osób z działu artystycznego. Jego głównym zadaniem jest przygotowanie odręcznych grafik, na podstawie których wykonywane są widzialne dla użytkownika animacje i rysunki.

Role programistyczne polegają na tworzeniu kodu, który wykorzystuje wcześniej przygotowane grafiki, aby gra stała się bardziej przystępna dla użytkownika. Możemy tutaj wymienić:

- *Game developer* - jest to osoba, która przygotowuje kod, w taki sposób, aby współgrał on z grafikami i stał się odpowiedni w odbiorze dla gracza.
- *Backend developer* - jest to specjalistyczna rola programistyczna. Osoba pracująca jako *"backendowiec"* odpowiada za warstwę logiczną gry, gdzie dokonywane są wszystkie obliczenia.
- *Engine programmer* - jest to osoba, która pisze bądź rozszerza zakres możliwości tzw. *silników gry*, które są narzędziami usprawniającymi pisanie kodu oraz ograniczającymi liczbę linii kodu, który musiałby być tworzony dla każdej gry.

Istnieją role, których nie można jednoznacznie zakwalifikować do żadnych z wyżej wymienionych kategorii. Osoby te często wykonują zadania z wyżej wymienionych stanowisk oraz do ich obowiązków należy popularyzacja gry. Możemy tutaj zauważyć takie stanowiska jak:

- *Sound/music designer* - osoba zatrudniona na tym stanowisku odpowiedzialna jest za dźwięki, które możemy usłyszeć podczas rozgrywki.
- *Tester* - jedno z cięższych stanowisk podczas produkcji gier. Osoba ta szuka błędów w grze i zgłasza je osobom odpowiedzialnym za poszczególne komponenty produkcji.
- *Marketing manager* - jego zadaniem jest promowanie gry na rynku oraz współpraca z mediami.

- *Community manager* - stosunkowo nowe stanowisko podczas produkcji gier. Osoba zatrudniona jako community manager odpowiedzialna jest za utrzymywanie zainteresowania użytkowników daną produkcją, udziela odpowiedzi na wszystkie pytania graczy związane z nadchodzącym tytułem.
- *Customer service* - jego praca rozpoczyna się po premierze gry. Jest on odpowiedzialny za pomoc użytkownikom w problemach technicznych związanych z daną produkcją.
- *ASO manager* - osoba, której zadaniem jest pozycjonowanie gry w najpopularniejszych serwisach sprzedażowych.
- *Translator* - osoba zatrudniona jako translator odpowiedzialna jest za tłumaczenie tekstów - zarówno pisanych jak i mówionych - aby gra była dostępna dla szerzej grupy potencjalnych konsumentów.
- *Porting team* - jest to jedno z najtrudniejszych zadań podczas całego etapu tworzenia gier. Osoby pracujące w tym zespole zajmują się przenoszeniem gry na inne dostępne na rynku platformy, co wymaga poważnych zmian w kodzie i grafice. Zazwyczaj w skład takiego zespołu wchodzą najbardziej doświadczeni programiści i graficy.

1.3.2 Etapy tworzenia gier komputerowych

Proces powstawania gry komputerowej jest bardzo czasochłonny. Aby dana produkcja odniósła sukces, osoby odpowiedzialne za projekt muszą się ciągle rozwijać, tak aby podążać za najnowszymi trendami na rynku. W obecnych czasach na świecie jest bardzo dużo osób, które grają w takie produkcje. Większość z nich nie jest świadoma tego ile czasu zajmuje powstanie gry. Cały proces tworzenia produkcji bardzo trafnie opisano w [20], został on podzielony na trzy fazy:

1. Faza preprodukcyjna,
2. Faza produkcyjna,
3. Faza postprodukcyjna.

1.3.2.1 Faza Preprodukcyjna

Jest to czysto teoretyczna faza w całym etapie tworzenia gier komputerowych. Podczas jej trwania przygotowywane zostają wszystkie informacje oraz dokumenty, które będą przydatne do otrzymania efektu końcowego - publikacji gry.

Pierwszym elementem tej fazy jest opracowanie *Koncepcji gry*, czyli tego czym produkcja ma być. Określony zostaje tutaj typ gry np.: gra platformowa, rodzaj (2D bądź 3D) oraz założenia projektowe, którymi są informacje o graczu, jego podstawowe zadania itp.

Kolejnym etapem jest utworzenie *Głównego Dokumentu Koncepcyjnego*, czyli *GDD*. W tym dokumencie zawarte są szczegółowe informacje dotyczące danej gry: tematyka, grupa docelowa graczy (określany jest typ graczy, którzy mogą być zainteresowani produkcją), przedmioty dostępne w grze, szczegóły opisujące przeciwników. Bardzo ważnym elementem tego dokumentu są grafiki koncepcyjne, które przedstawiają propozycje wyglądu poszczególnych obiektów ukazanych w produkcji. Projektant gry opracowując *GDD* konsultuje go z inwestorem, głównym programistą oraz dyrektorem artystycznym w celu przedstawienia i uzgodnienia wizji danej produkcji.

Na podstawie wyżej opisanego *GDD* główny programista tworzy *Dokument Techniczny Gry*, czyli *GTD*. Zawarte są w nim wymagania sprzętowe, informacje odnośnie wykorzystywanego silnika gry oraz inne przydatne informacje techniczne. Dokument ten zawiera również harmonogram produkcji poszczególnych elementów gry. Równocześnie z *GDD* tworzony jest *Dokument Poziomów Gry*, czyli *LD*. Zawiera on w sobie informacje o poszczególnych poziomach, opisy kampanii oraz zadania dla gracza. Obydwa te dokumenty podlegają ciągłym zmianom podczas kolejnych etapów tworzenia gry.

Podczas Fazy Preprodukcyjnej tworzone są również pomocnicze dokumenty, inne niż te wyżej wymienione, są to m.in.: plan produkcji oraz budżetowy, lista dźwięków, dokumenty fabularne, plany marketingowe i wiele innych.

1.3.2.2 Faza Produkcyjna

Podczas trwania *fazy produkcyjnej* wykorzystywane są materiały opracowane podczas poprzedniego etapu. Na początku zostaje przygotowany prototyp, który jest pierwszą, bardzo okrojoną wersją gry, mającą na celu przedstawienie działania i wyglądu końcowej produkcji. Sprawdzane są tutaj również założenia początkowe z fazy preprodukcyjnej, czy zostały one zrealizowane w tworzonej grze. Kolejnym prototypem jaki zostaje przygotowany jest *Vertical Slice*. W przeciwnieństwie do MVP, który służy zastosowaniom zespołu produkcyjnego, *VS* ma za zadanie pokazać odbiorcom jak będzie wyglądał efekt końcowy, ma ich zaciekać, aby wyczekiwali oni publikacji gry.

Ostatnim etapem tej fazy jest przygotowanie tzw. *beta buildu*. Jest to dużo bardziej rozbudowana wersja gry, z którą odbiorcy mieli styczność podczas prezentowa-

nia MVP i VS. Zawiera on w sobie wszystkie zaplanowane elementy wersji końcowej, jednakże jeszcze nie do końca działające poprawnie. Służy on przede wszystkim do testowania funkcjonalności oraz do znajdywania różnych błędów, tak aby użytkownik mógł cieszyć się w pełni działającą grą. Bardzo często zauważa się, że firmy działają na zasadzie *Open Beta Testów*. Oznacza to, że udostępniają oni swój produkt graczom, a oni grając sprawdzają poprawność działania.

1.3.2.3 Faza Postprodukcyjna

Ostatnia faza produkcji gry polega na opublikowaniu gry na rynku, oraz poprawianiu niezauważonych wcześniej błędów. Na tym etapie udostępniane są również dodatkowe materiały związane z tytułem, np.: ubrania z logiem gry, soundtracki, czyli ścieżki dźwiękowe użyte w rozgrywce i wiele innych.

Pierwszym krokiem jest wybranie odpowiedniego kandydata wersji gry do publikacji, tzw. *Release Candidate*. Decyzja o wyborze buildu końcowego podejmowana jest zazwyczaj przez *Project managera* oraz *Investora* bądź *Wydawcę*. Aby gra mogła przejść z fazy *Beta testów* do *RC*, musi ona przejść przez *Testy akceptacyjne*[7], które pomagają określić czy dana produkcja jest gotowa do publikacji. Sprawdza się tutaj między innymi zachowanie gry z określona wcześniej specyfikacją itp.

Gdy *RC* przejdzie wszystkie niezbędne procedury, podejmowana jest decyzja o publikacji. Od wcześniej zbudowanego tła przez osoby zajmujące się marketingiem i promowaniem produkcji zależy powodzenie sprzedaży produktu. Producenci bardzo często korzystają w celu zwiększenia sprzedaży swojego tytułu z różnych średników, takich jak na przykład Steam[8], który jest największym dystrybutorem gier na rynku. Przedsiębiorstwa programistyczne bardzo często decydują się na samodzielne wydanie gry. Bardzo ważną rolę odgrywają tutaj serwisy społecznościowe, na których prowadzona jest kampania marketingowa.

Bardzo często zdarza się, że po wydaniu danej gry firmy oferują wsparcie dla klienta. Dzieje się to, ponieważ podczas testowania nie da się znaleźć wszystkich błędów działania danego tytułu. Dzięki temu, użytkownik który zauważy problem może zgłosić się do osoby wykwalifikowanej która rozwiąże problem tak, aby z czasem gra stała się pozbawiona błędów i zapewniała rozrywkę dla osób, które zdecydują się ją zakupić.

Rozdział 2

Fabuła gry "Tree of life"

Rozdział skupia się na przedstawieniu fabuły gry - ukazana zostanie historia głównego bohatera. Opisane również zostaną pomysły, których nie udało się zrealizować podczas tworzenia tej produkcji oraz potencjalne możliwe scenariusze rozwoju tego tytułu.

2.1 Pomysły zrealizowane

2.1.1 Przeszłość

W grze *Tree of Life* gracz wciela się w postać legendarnego wikinga o imieniu *Björn Żelaznoboki*, który jest synem *Ragnara Lothbroka* pochodzącego od nordyckiego boga *Odyna* oraz wojowniczej królowej *Lagerthy*. Swój przydomek zdobył podczas jednej z wypraw, gdzie wyróżnił się wielką odwagą i determinacją w pokonaniu podbijanego przez wikingów ludu. Tak jak ojciec jest on inteligentny i zdeterminowany: pragnął przygód oraz poznania innych kultur i narodów. Po tragicznej śmierci *Ragnar*a został on królem miasta - państwa *Kattegat*. Z jednego ze swoich nieudanych małżeństw miał córkę *Siggy*.

2.1.2 Teraźniejszość

2.1.2.1 Poszukiwanie lekarstwa

Właściwa historia przedstawiona w grze rozpoczyna się w momencie odkrycia, że córka Björna jest ciężko chora i nie ma szans na przeżycie. Bohater nie załamuje się i postanawia wyruszyć na wyprawę do odległych krain, aby znaleźć dla niej lekarstwo. Podczas poszukiwań w pewnym kraju jeden z wikingów trafił na wędrowca. Włóczęga opowiedział mu o bardzo niebezpiecznym terytorium, w którym rośnie

drzewo łączące ze sobą wszystkie światy - *Yggdrasil*. Według dawnych podań osoba, która dostanie się na szczyt tego starodrzewa otrzyma napój leczący wszystkie choroby.

Wojownik ucieszył się z tej informacji i zabrał wędrowca ze sobą na spotkanie z królem. Podróżnik opowiedział ze szczegółami całą historię tej krainy i przestrzegał, żeby Björn nie udawał się tam, ponieważ czeka go niechybna śmierć. Miłość bohatera do córki jest jednak tak ogromna, że zamierza on poświęcić własne życie dla niej: nie przejmuje się tym, że cała jego kompania może nie wrócić do domów. Znając ryzyko dał możliwość podjęcia decyzji swoim wikingom zdecydować czy chcą pływać z nim czy wolą wrócić do domów. Większość z nich wybrała towarzyszenie swojemu *jarlowi* w wyprawie, ponieważ kochali go. Dzięki niemu ich wioska stała się najpotężniejszą osadą w całej Norwegii, to on zapewnił im dobrobyt i przyszłość.

Wraz z wojownikami, którzy zdecydowali się wyruszyć z królem, Björn rozpoczął przygotowania do wyprawy. Nie wiedząc, jak daleka podróż ich czeka, wyładowali oni drakkary do granic możliwości prowiantem, wodą oraz różnego rodzaju bronią, które zakupili w pobliskiej wiosce. Na morzu kompania spędziła dużo tygodni. Ich zapasy były na wykończeniu. Pewnego dnia, gdy byli na otwartym morzu, zastał ich sztorm: drakkary zostały oddzielone od siebie, część została zniszczona przez żywioł. Jednym ze zniszczonych statków była łódź, którą płynął Björn. Krew nordyckiego boga *Odyna* płynąca w żyłach nordyckiego króla sprawiła, że tylko on przetrwał ten kataklizm. Po wielu dniach dryfowania na szczątkach drakkaru na morzu trafił on na dziwnie wyglądającą wyspę.

2.1.2.2 Mroczna Kraina

Plaża

Ranny i wycieńczony wiking widząc, że miejsce w którym się znajduje nie sprzyja ludziom, zaczął szukać schronienia. Znalazł zacisną jaskinię, w której postanowił odpocząć. Po kilku dniach spędzonych w grocie Björn postanowił poszukać jedzenia i jakiekolwiek osady. Podczas wędrówki został on jednak złapany przez dziwnie wyglądające stwory - *Draugry*[9]. Są to najbardziej mroczne stworzenia pochodzące z pierwotnych skandynawskich legend. Potwory te przypominały z wyglądu ludzi - wikingów, ale w przeciwieństwie do króla Kattegat postacie te nie żyły. Demony te są zmarłymi wojownikami, które za sprawą czarnej magii zostały wskrzeszone. Uosabiają one lęk wikingów przed zmarłymi. W postaci tych potworów зло ukazywało się przez nienawiść do wszystkiego co żyje. Żelaznoboki został zabrany do dziwnie

wyglądającej cytadeli - siedziby demonów, tam został zamknięty w celi.

CONDEMNED CITADEL

Po kilku dniach przebywania w zamknięciu Björn został wezwany przed oblicze władcę Draugrów - *Jormungand* - który był bogiem, przeciwnikiem Thora, boga piorunów, w postaci gigantycznego węża. Król wyspy chce, aby wiking stał się jego sługa, na co ten nie chce się zgodzić. Z tego powodu zostaje zamknięty w więzieniu i torturowany. Pewnego dnia po męczarniach znajduje w celi mapę i list mówiący o *Drzewie Życia*, o którym opowiadał mu wcześniej napotkany podróżnik. Żelaznoboki uradował się, ponieważ dotarł do miejsca, gdzie chciał się znaleźć, aby uratować swoją córkę.

Kolejnego dnia, gdy znowu został zabrany na tortury, ukradł narzędzie, które będzie mu służyć jako wytrych do otwarcia drzwi celi. Bohater odkrył w sobie nowe pokłady siły i teraz, każdego dnia po męczarniach żłobił ukradzione narzędzie, tak aby mógł on otworzyć zamek i wydostać się na wolność. Znanemu ze swojej determinacji Björnowi zajęło to kilka wieczorów. Podczas jednej z pracowitych nocy, znalazł w pobliżu drzwi na ścianie dziwną liczbę, jednak zignorował ją - twierdząc, że jest to pozostałość po poprzednim więzniu.

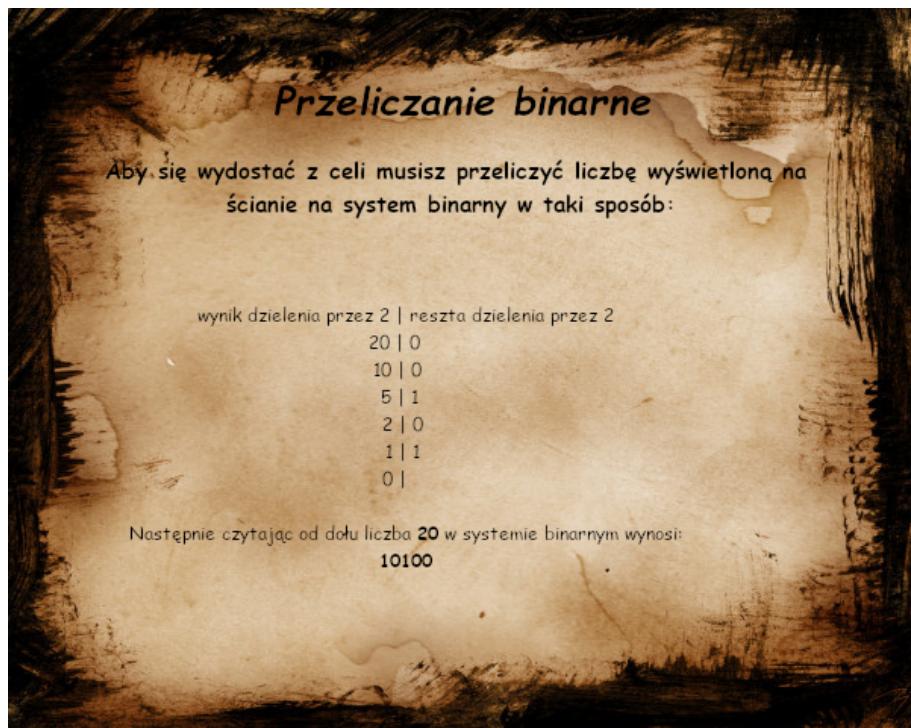
Pewnego wieczoru po dniu pełnym tortur postanowił wcielić swój plan w życie. Przed sobą miał mechanizm (zob. rysunek 2.1) - zamek do drzwi, który można było odblokować ustawiając w odpowiedniej kolejności piny.



Rys. 2.1: Wygląd mechanizmu zamka

Zamek składa się z 7 pinów. Król wikingów próbował różnych sposobów, aby otworzyć drzwi jednak mu się to nie udało. Przypomniał sobie o liczbie wyżłobionej

na ścianie w okolicy drzwi. Postanowił ją wykorzystać. Okazało się, że liczba ta niewiele pomaga w próbie otwarcia mechanizmu zamka. Zmęczony i poirytowany Björn zaczął czytać jeszcze raz znaleziony list, w którym odnalazł wskazówkę (zob. rysunek 2.2) jak przekształcić znalezioną liczbę w ciąg cyfr, służący do otwarcia drzwi.



Rys. 2.2: Wygląd podpowiedzi dla gracza do pierwszego poziomu

Aby wydostać się z więzienia gracz musi przekształcić wyrytą liczbę na ciąg cyfr w systemie binarnym. Położenie poszczególnych pinów odzwierciedla '0' bądź '1'. Gdy użytkownikowi wcielającemu się w postać króla wikingów uda się rozwiązać zagadkę, drzwi się otwierają. W korytarzu Björn znajduje topór, który będzie mu służył do obrony przed napotkanymi przeciwnikami.

SVARTAL

Szczęśliwie, królowi Kattegat udaje się wydostać z cytadeli, w której był więziony. Trafia on do królestwa *Mrocznych elfów* (zob. rysunek 2.3). Są to bardzo niebezpieczne istoty zamieszkujące ciemne miejsca, charakteryzujące się posiadaniem czerwonych oczu oraz szarym odcieniem skóry, co odróżnia je od innych elfów. Żyją one bezpieczne w podziemiach twierdzy *Jormunganda*.



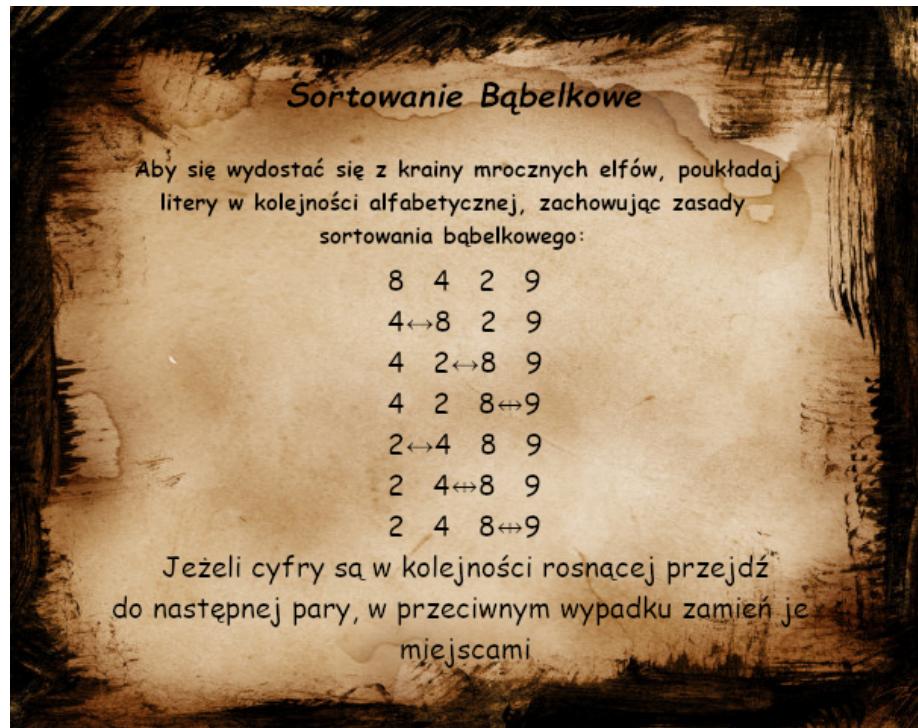
Rys. 2.3: Przykładowy wygląd poziomu w lokacji "SVARTAL"

Björn musi się wydostać z tego królestwa. Aby tego dokonać musi skakać po półkach skalnych, na których może napotkać wrogo nastawione do siebie elfy, które mogą przejawiać jeden z trzech stanów:

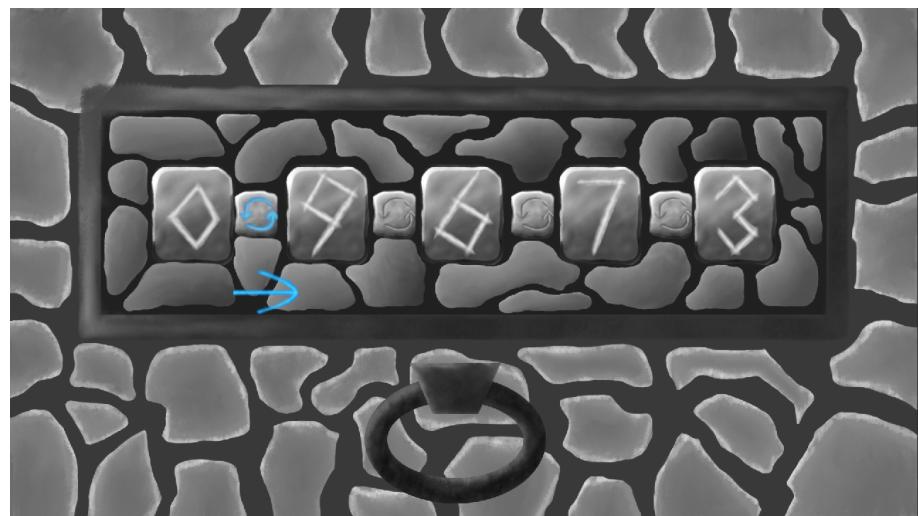
- Zdziwienie - postacie te krążą tylko po platformie, nie umieją strzelać, zazwyczaj można je unieszkodliwić jednym bądź dwoma strzałami z łuku
- Zdenerwowanie - osobniki te potrafią strzelać, w porównaniu do opisanych wyżej gracz będzie potrzebował oddać większą ilość strzałów, aby je unieszkodliwić
- Rozwścieczenie - jednostki te bardzo szybko oddają swoje strzały, posiadają najczęściej dużą "ilość" życia, gracz musi się natrudzić, aby je pokonać

Na swojej drodze gracz może zbierać dodatkowe życie oraz wskazówkę (zob. rysunek 2.4), która pomoże mu pokonać następną łamigłówkę, aby przejść do kolejnej lokalizacji.

Po dotarciu do drzwi, król Kattegat musi je otworzyć układając cyfry w odpowiedniej kolejności (zob. rysunek 2.5). Mroczne elfy chcąc złapać postaci, które dostają się do ich królestwa, oraz przewidując, że strażnicy będący na skalnych półkach mogą sobie nie poradzić z intruzem, stworzyły mechanizm blokujący drzwi, który znają tylko one. Björn podczas swojej wędrówki poprzez tę krainę może znaleźć instrukcję, opisującą w jaki sposób może otworzyć drzwi. W przeciwnym wypadku będzie musiał próbować odgadnąć algorytm rozwiązuający tę sekwencję.



Rys. 2.4: Przykładowa podpowiedź do poziomu Bubble Sort



Rys. 2.5: Wygląd mechanizmu wyjścia z krainy elfów

ANCIENT FOREST

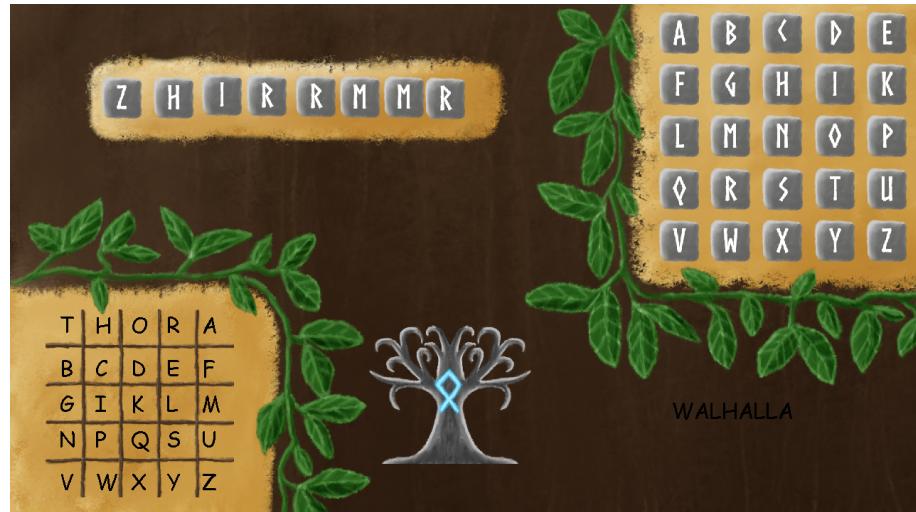
Po wydostaniu się z królestwa mrocznych elfów główny bohater trafia do starożytnego lasu (zob. rysunek 2.6). Na swojej drodze spotyka wysłane na poszukiwania przez *Jormunganda* Draugry. Tym razem nie mają one zamiaru złapać króla Kattegat - chcą go zabić, aby nie osiągnął on celu swojej podróży - *Drzewa Życia*.



Rys. 2.6: Przykładowy wygląd poziomu w lokacji "ANCIENT FOREST"

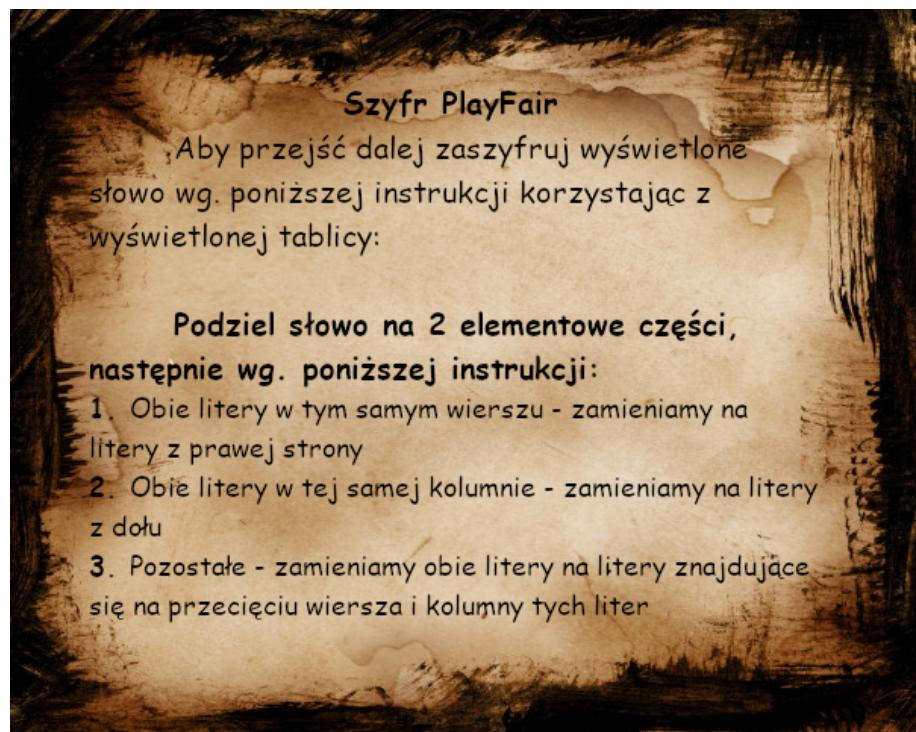
Podobnie jak w królestwie mrocznych elfów, Björn musi skacząc po platformach, pokonać przeciwników oraz znaleźć wskazówkę. Tym razem jego celem nie jest wydostanie się z królestwa, chce on dostać się do drzewa, aby zdobyć lek dla swojej córki. Lokalizacja, w której się znajduje jest nieprzewidywalna i pełna dziwnych mocy. Na każdym kroku na gracza czeka niebezpieczeństwo w postaci przeciwników bądź dziwnych przeszkód.

Mijając platformy u podnóża starodrzewa, oczom gracza ukazuje się ogromna brama. Aby przez nią przejść trzeba zaszyfrować słowo w odpowiedni sposób. Żelaznoboki musi łapać klocki i przeciągać je w przeznaczone do tego miejsce, tak aby powstał *szyfrogram*, czyli ciąg znaków, który po odpowiednim przekształceniu da użytkownikowi tekst jawnny, otrzymany na początku. Aby rozwiązać łamigłówkę gracz musi znać zasadę szyfrowania według *szyfru PlayFair 2.7*. Król Kattegat musi się śpieszyć, ponieważ w każdej chwili mogą na niego napaść Draugry bądź Mroczne Elfy.



Rys. 2.7: Przykładowy wygląd mechanizmu bramy

Podczas trwania "części platformowej" poziomu król wikingów może znaleźć wskazówkę (zob. rysunek 2.8), która może mu pomóc w rozwiązaniu łamigłówki. Jeżeli jej nie zabierze, będzie musiał próbując różnych kombinacji odnaleźć odpowiedni ciąg znaków umożliwiający dostanie się do *Yggdrasila*.



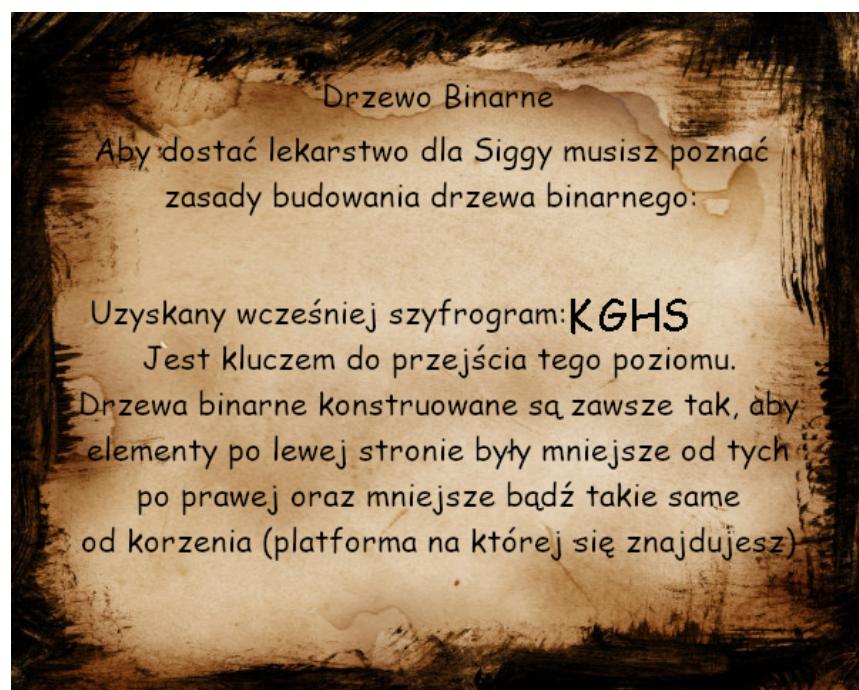
Rys. 2.8: Przykładowa podpowiedź do poziomu PlayFair

Tree of Life

Ostatnim wyzwaniem dla Björna jest przejście po odpowiednich gałęziach *Drzewa Życia* (zob. rysunek 2.9). Musi on wykorzystać ciąg liter użyty do otwarcia bramy tak, aby skacząc po odpowiednich platformach dostał się do upragnionego lekarstwa dla swojej córki (zob. rysunek 2.10). Tutaj nie czekają już na niego przeciwnicy, ponieważ wszystkie złe istoty boją się energii, która promieniuje od starodrzewa.



Rys. 2.9: Przykładowy wygląd gałęzi w lokacji "Tree Of Life"



Rys. 2.10: Przykładowy wygląd podpowiedzi w poziomie Tree Of Life

Na każdej gałęzi jest znak z literą, jaką ta odrośl reprezentuje. Po prawidłowy przejściu po odpowiedniej ścieżce Björn dostaje w swoje ręce lekarstwo, dzięki któremu może uratować życie *Siggy*. Natomiast, jeżeli gracz w trakcie pokonywania poziomu pomyli się i skoczy na złą gałąź, spadnie na ziemię i będzie musiał rozpocząć swoją przygodę z pokonywaniem drzewa od początku.

2.2 Pomysły niezrealizowane

W fazie preprodukcyjnej pomysłem było utworzenie większej ilości lokalizacji, w której gracz mógłby się poruszać oraz dodanie bardziej zróżnicowanych przeciwników, jak również schematów, dzięki którym inaczej odbywałyby się walki z napotkanyimi podczas rozgrywki postaciami. Przedstawione tutaj lokalizacje są wybranymi z wymyślonych wcześniej, ponieważ gra która zawierałaby w sobie to wszystko co zostanie poniżej opisane byłaby niedopracowana, pełna różnych błędów. Podczas tworzenia *Głównego Dokumentu Koncepcyjnego* przewidziane były następujące lokalizacje:

1. *CONDEMNED CITADEL* - lokacja użyta w finalnej wersji gry, w zamыśle miała być podzielona na kilka mniejszych:
 - *PRISON* - więzienna część cytadeli, lochy, z których nikt nigdy nie powinien się wydostać,
 - *VALLEY OF SHADOW* - mroczna ścieżka prowadząca z cytadeli do podziemnej krainy mrocznych elfów,
2. *SVARTAL* - królestwo mrocznych elfów, lokalizacja użyta w finalnej wersji gry.
 - *MINES* - tereny kopalniane, gdzie elfy wydobywają cenne surowce,
 - *FORGES* - kuźnie, w których metal zamieniany jest w oręż,
 - *CAVES* - system jaskiń nad kuźniami prowadzący na powierzchnię,
3. *ANCIENT FOREST* - ogromny starożytny las, w środku którego stoi drzewo życia - cel podróży bohatera
 - *SWAMPS* - mroczne i nieprzyjazne bagna mające odstraszyć wszystkich próbujących wejść do lasu,
 - *WILDERNESS* - gęsta puszcza u podnóża drzewa życia,

- *TREE OF LIFE* - droga wiodąca przez drzewo na jego szczyt (pomysł zrealizowany),

Głównym zamysłem było to, aby gra była w jak największym stopniu grywalna, żeby każdy mechanizm w niej zawarty działał bezbłędnie, a powiększanie gry o dodatkowe, opisane lokalizacje powodowało by generowanie większej liczby błędów.

Kolejnym elementem, który nie został do końca zrealizowany jest mechanizm walk. W *GDD* zaplanowany został oprócz mechanizmu strzelania (walki na odległość), system walk bezpośrednich za pomocą np.: miecza. Faza produkcyjna pokazała, że projekt tworzenia gry komputerowej jest dużo bardziej złożony niż zakładano na początku i 2 osoby nie są w stanie stworzyć wystarczającego zespołu, który byłby w stanie przygotować grę z kilkoma mechanizmami walk.

Ostatnią rzeczą zaplanowaną, której realizacja nie powiodła się, jest liczba typów przeciwników. W *GDD* opracowani zostali:

1. *GRAUGR* - nieumarli wojownicy walczący wręcz, znajdujący się we wszystkich lokalizacjach, posiadają 1 punkt życia,
2. *DARK ELVES* - mroczne elfy zamieszkujące *SVARTAL*, wojownicy walczący wręcz oraz z dystansu za pomocą rzucanych sztyletów, posiadają 2 punkty życia,
3. *TROLLS* - postacie zamieszkujące starożytny las, duże i twardy, walczą wręcz oraz z dystansu: rzucają głazami. Posiadają 3 punkty życia

Przedstawione postacie przeciwników zostały zmodyfikowane tak, aby walczyły tylko z dystansu: rzucając różnymi przedmiotami. Z racji zmniejszenia liczby lokalizacji usunięta została postać *Trolli*.

2.3 Scenariusze rozwoju gry *Tree Of Life*

Gra *Tree Of Life* ma w sobie duży potencjał na wydłużenie czasu rozgrywki. Struktura plików została stworzona w sposób ułatwiający dalszą rozbudowę. Wykorzystana biblioteka *pygame* jest bardzo prostą biblioteką, która zmusza programistę do samodzielnego implementacji podstawowych metod. Można stwierdzić, że powstały do tej pory kod jest bardzo prostym silnikiem gry.

W przyszłości gra zostanie rozbudowana o kolejne lokalizacje, korzystając ze stworzonych do tej pory funkcjonalności, a na które z powodu małego zespołu tworzącego

ten tytuł nie wystarczyło czasu. Zostaną również dodani nowi przeciwnicy, do których stworzenia zostanie użyta *Sztuczna Inteligencja*.

Język *Python* oraz biblioteka *pygame* nie są narzędziami, które są stworzone do produkcji dużych gier, przez co dalsza rozbudowa tego tytułu w obecnej formie może być problematyczna. Rozwiązaniem tego problemu jest przeniesienie tej gry na silnik *Unity*, który jest lepiej przystosowany do takich projektów.

Rozdział 3

Grafika

W tym rozdziale opisane zostaną zagadnienia związane z częścią artystyczną produkcji. Ukażane zostaną rodzaje grafiki oraz narzędzia, dzięki którym można je tworzyć. Przedstawione zostaną również sposoby tworzenia grafiki na potrzeby tejże produkcji.

3.1 Rodzaje grafiki

Słowo *grafika* kojarzy się najczęściej z obrazkiem, prostym obiektem widzianym na płaszczyźnie ekranu monitora, jednak jej istota w rzeczywistości nie jest tak trywialna. Grafikę można podzielić na różne kategorie, według wielu kryteriów, między innymi:

- sposób tworzenia,
- sposób prezentacji,
- zastosowanie

W dzisiejszych czasach obiekty graficzne znajdują zastosowanie w każdej dziedzinie naszego życia, a zwłaszcza w informatyce. Wszystko co widzi użytkownik podczas korzystania z komputera czy smartfona jest pewnego rodzaju graficznym elementem. Każdy obrazek, interfejs, a nawet tekst jest swoistą graficzną prezentacją, jednak często bardzo różną od siebie nawzajem.

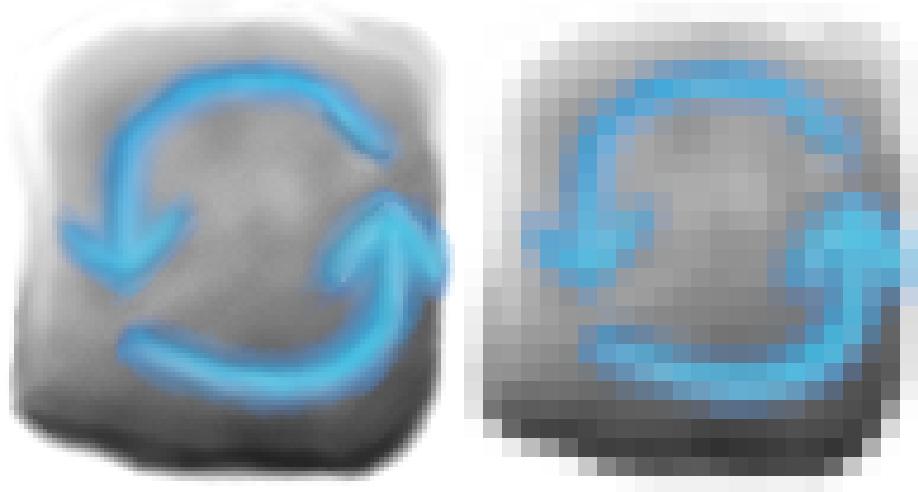
3.1.1 Grafika 2D

Dla przeciętnego użytkownika najbardziej popularnym rodzajem tworów graficznych są obrazy 2D. Z uwagi na ogromne możliwości ich zastosowania są wykorzysty-

wane w bardzo wielu projektach. Grafiki 2D można podzielić na wiele kategorii na przykład ze względu na ich przeznaczenie, jednak z praktycznego punktu widzenia lepszym podziałem jest rozróżnienie ich ze względu na sposób tworzenia.

Grafika rastrowa

W grafice rastrowej wszelkiego rodzaju obiekty tworzy się za pomocą pikseli, ponieważ z nich składa się całe "płótno", na którym powstaje grafika. Nawet okręgi oraz inne krzywe są kreowane za pomocą odpowiednio rozłożonych kwadratów - pikseli. Z takim rozwiązaniem wiąże się pewna poważna wada: im mniejsze wymiary obrazu, tym gorsza jego jakość. Oczywiście jeśli grafika ma odpowiednią ilość pikseli jest to praktycznie niezauważalne, ale przy mniejszych rozdzielczościach można dostrzec, że krzywe składają się ze schodkowo ułożonych, małych kwadratów lub prostokątów. Przez taką budowę obrazy rastrowe nie są "odporne" na *skalowanie* (zob. rysunek 3.1), ponieważ przy powiększaniu obrazu jeden piksel zamienia się w kilka, przez co kąty proste są bardziej widoczne. Przy pomniejszaniu obrazu mechanizm działa podobnie, tylko że liczba pikseli zostaje zredukowana, a to prowadzi do utraty danych związanych z kolorem. Dzieje się tak ponieważ każdy z pikseli przed skalowaniem może mieć inną kolorystykę, a po skalowaniu barwa wyjściowego piksela jest uśrednioną wartością barw początkowych pikseli - mówimy o tak zwanej *interpolacji*.



Rys. 3.1: Przykład obrazu rastrowego po skalowaniu:

Po lewej stronie - oryginalny obraz,

Po prawej stronie - obraz zmniejszony czterokrotnie

Poza kwestią skalowania, możliwość edycji nawet pojedynczych pikseli niesie za sobą wiele zalet. Dzięki takiemu rozwiązaniu można narysować praktycznie wszystko, ponieważ w programach do edycji grafiki rastrowej takich jak na przykład *Photoshop* czy *Gimp 2*, istnieją dziesiątki narzędzi, które dają praktycznie nieograniczone możliwości. Jednym z najprostrzyszych narzędzi jest pędzel - podstawy przyrząd - za którego pomocą powstaje większość grafiki. Wysoką jakość obrazu można osiągnąć używając tzw. miękkich krawędzi pędzla (zob. rysunek 3.2). Jeśli pędzel rysuje obiekty w kształcie koła, miękkimi krawędziami będą zewnętrzne okręgi koła o odpowiednim stopniu przeźroczystości. Częściowa przeźroczystość pozwala na "wtapianie" w siebie różnych elementów obrazu, maskując przy tym ostre krawędzie tworzone przez piksele. Kolejną techniką pozwalającą na osiągnięcie podobnego efektu jest używanie wszelakich narzędzi wygładzających, które powodują stopniowe wyrównywanie wartości barw. Takie wyrównywanie powoduje większe ujednolicenie kolorów, co owocuje gładszym przejściem pomiędzy pikselami.



Rys. 3.2: Przykład zastosowania miękkich krawędzi pędzla - po prawej stronie,
oraz twardych krawędzi - po lewej stronie

Grafika wektorowa

W przeciwieństwie do grafiki rastrowej, wektorowa nie jest tworzona za pomocą pikseli. Obiekty kreowane za pomocą grafiki wektorowej opisywane są za pomocą

równań geometrycznych. Dzięki takiej budowie grafiki wektorowe są odporne na utratę jakości podczas skalowania (zob. rysunek 3.3), ponieważ sama budowa obiektów się nie zmienia, a jedynie ich wyjściowe wymiary. Z pewnego punktu widzenia grafika rastrowa przypomina malowanie czegoś nowego, natomiast grafika wektorowa budowanie z istniejących już elementów. Dzięki takiej konstrukcji obrazy wektorowe znajdują szczególną zastosowanie na przykład w dziedzinie web development'u, gdzie logo firmy umieszczone na stronie zawsze jest w dobrej jakości, a przy odpowiednim dostosowaniu *responsywności*¹⁾, również zawsze w adekwatnym rozmiarze.



Rys. 3.3: Różnica w skalowaniu grafik wektorowej i rastrowej. Źródło:

[10]

3.1.2 Grafika 3D

Grafika 3D łączy w sobie aspekty zarówno grafiki rastrowej jak i wektorowej. Przestrzeń w jakiej są tworzone obiekty jest przestrzenią trójwymiarową (zob. rysunek 3.4), zamiast dwuwymiarowego "płotna", ale zasada działania jest podobna do tej wykorzystywanej w grafice wektorowej. Obiekty również są opisywane równaniami matematycznymi, a ich składowe takie jak wierzchołki, ściany czy krawędzie posiadają swoje parametry. Dzięki tym parametrom można określić wymiary oraz położenie w przestrzeni zarówno całych obiektów jak i ich składowych. Pomimo przestrzeni posiadającej trzy wymiary jesteśmy w stanie tworzyć obiekty dwuwymiarowe, służące na przykład jako tło scenerii. Najczęściej jednak grafika rastrowa w trójwymiarze znajduje zastosowanie w postaci tekstur dla obiektów 3D. W przypadku grafiki 3D bardzo ważnym aspektem jest również oświetlenie - coś nieobecnego w grafice 2D. Odpowiednio ustalone oświetlenie daje niesamowite możliwości

¹⁾Dostosowanie obrazu do wielkości wyświetlacza na którym jest prezentowana

oraz znacznie poprawia jakość finalnego produktu, dzięki bardziej realistycznemu wyglądowi.



Rys. 3.4: Przykłady obiektów trójwymiarowych

3.2 Grafika w grach komputerowych

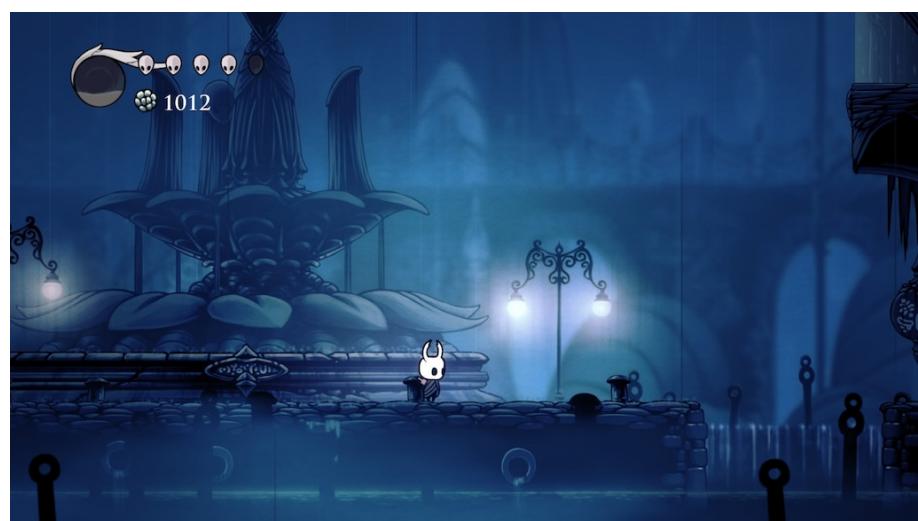
3.2.1 Graficzne aspekty gier

W grach komputerowych grafika jest praktycznie wszechobecna, wykorzystuje się ją w zasadzie na każdym etapie produkcji, czy promocji gry. Zależnie od rodzaju gry, może być wykorzystywana grafika 2D lub 3D, najczęściej jednak obie. Więcej zastosowań znajduje grafika 2D, jednak to nie umniejsza znaczenia obiektów 3D, które bardzo często zajmują więcej czasu podczas tworzenia, przez swoją wysoką złożoność. Obrazy dwuwymiarowe występują w grach między innymi jako:

- *Tekstury* (zob. rysunek 3.5) - w przypadku gier 2D są to obrazy tworzące elementy otoczenia, postacie oraz przedmioty. W grafice 3D tekstury są obrazami specjalnie rozłożonymi na płaszczyźnie tak, aby po nałożeniu ich na obiekt 3D poprawnie go pokrywały.
- *Tła* (zob. rysunek 3.6) - są to obrazy mające na celu nadanie głębi grom, takim jak gry platformowe; zależnie od wielu czynników mogą być bardzo złożone jak i bardzo proste, ponieważ nie są czymś co gracz widzi na pierwszym planie, są pewnego rodzaju wypełnieniem, akcentem.
- *Ekrany lądowania* (zob. rysunek 3.7) - niezależnie od rodzaju gry, ekrany lądowania mają na celu umilenie czasu, jaki gracz musi odczekać przed za-



Rys. 3.5: Przykładowe tekstury i przedmioty. Źródło: [11]



Rys. 3.6: Przykład tła w grze. Źródło: [12]

dowaniem się poziomu. Często przedstawiają ciekawe widoki ze świata gry, potwory z jakimi przyjdzie graczowi się zmierzyć, albo różne ciekawostki, czy wskazówki dotyczące rozgrywki. W niektórych grach za pomocą ekranów ładowania prowadzona jest narracja - tekst, często w towarzystwie czytającego go lektora, lub filmy czy animacje.



Rys. 3.7: Przykład ekranu ładowania. Źródło: [13]

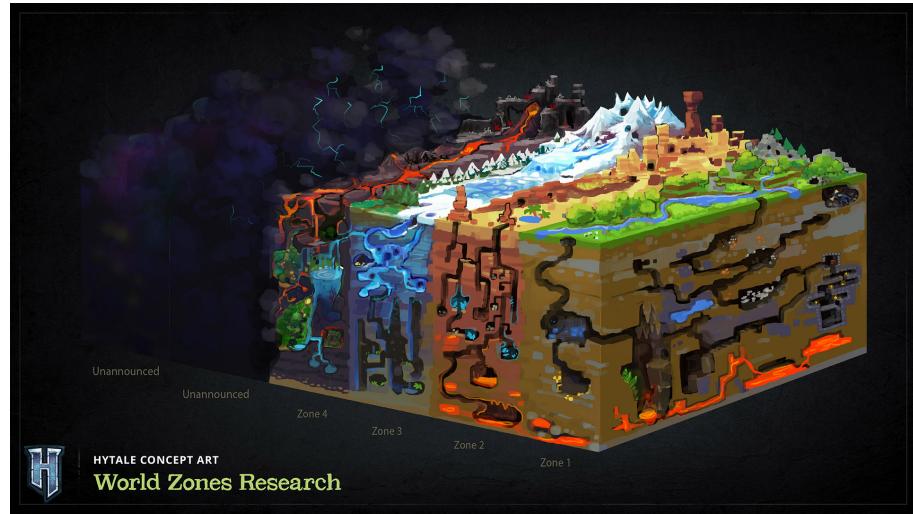
- *Elementy interfejsu* (zob. rysunek 3.8) - do tej kategorii elementów graficznych należą różnego rodzaju przyciski, obramowania okien, kursory, a nawet specjalnie robione na potrzeby gry czcionki.



Rys. 3.8: Przykładowy interfejs. Źródło: [14]

Poza szerokim zakresem zastosowań grafik 2D podczas produkcji gier, istnieje również inny obszar, który tak naprawdę opiera się na grafikach i animacjach związanych z grą.

zanych z grą - promocja gry. Do promocji gry najczęściej wykorzystywane spośród grafik są tak zwane *Concept Arty* (zob. rysunek 3.9), czyli grafiki koncepcyjne przedstawiające to, jak gra ma docelowo wyglądać lub co ma oferować. Warto również wspomnieć, że wszystkie grafiki używane podczas produkcji, czy promocji gier są grafikami rastrowymi, wyjątki stanowią jedynie obrazy, takie jak logo gry zamieszczone na stronie dotyczącej produktu.



Rys. 3.9: Przykład grafiki koncepcyjnej. Źródło: [15]

3.2.2 Animacje

Animacje są bardzo ważnym aspektem w produkcji gier komputerowych. Tak jak zwykłe grafiki, mogą być bardzo zaawansowane, jak i bardzo proste. Najprostszymi animacjami są *animacje poklatkowe*, które są po prostu zbiorem grafik przedstawiających kilka stadiów ruchu. Takie grafiki są wyświetlane po sobie bardzo szybko, co powoduje złudzenie optyczne wyglądające jak ruch. Do uzyskania takiego efektu można wykorzystać zaledwie trzy grafiki, ale można użyć ich o wiele więcej, co znacząco zwiększy płynność animacji. Innym rodzajem animacji wykorzystywanym w grach są animacje tworzone przy użyciu programów do grafiki 3D takich jak *3Ds max*, czy *Blender*. Na potrzeby animacji tworzy się całe sceny, a następnie przy pomocy odpowiednich narzędzi wyznacza się tak zwane *klatki kluczowe*, w których zmienia się na przykład położenie, czy kształt obiektów. Po wyznaczeniu klatek kluczowych program sam wylicza parametry obiektów dla klatek pośrednich, tak żeby przejście pomiędzy jedną klatką kluczową, a drugą było płynne. Takie rozwiązanie wydaje się być prostsze i mniej czasochłonne od klasycznych animacji poklatkowych, ale za to wymaga o wiele większej mocy obliczeniowej.

3.3 Grafika w grze "Tree of Life"

"Tree of Life" jest dwuwymiarową grą platformową, dlatego do jej stworzenia została wykorzystana jedynie grafika 2D, a konkretnie rastrowa. Wszystkie obrazy zostały wykonane w programie *Gimp 2* oraz wyeksportowane w formacie *PNG*.

Gry platformowe często posiadają poziomy o różnej tematyce, co oznacza, że lokacje różnią się motywem przewodnim jak np. las, jaskinia, czy zima. Pomimo tych różnic między lokacjami, budowane one są w ten sam sposób - z modułowych platform (zob. rysunek 3.10). Wspomniane platformy tworzy się za pomocą pojedynczych bloków, które pokrywa ta sama tekstura. W "Tree of Life" platformy zbudowane są przy użyciu 4 wariantów tej samej tekstury - lewa i prawa krawędź, środek oraz pojedynczy, samodzielnny blok. Model postaci (zob. rysunek 3.11) został utworzony na podobnej zasadzie co platformy - zbiór kilku grafik, używanych zależnie od stanu postaci, w jaki wprawia ją gracz. Ruchy postaci osiągane są przy użyciu najprostszej animacji poklatkowej - szybka zmiana obrazów przedstawiająca różne pozy bohatera.



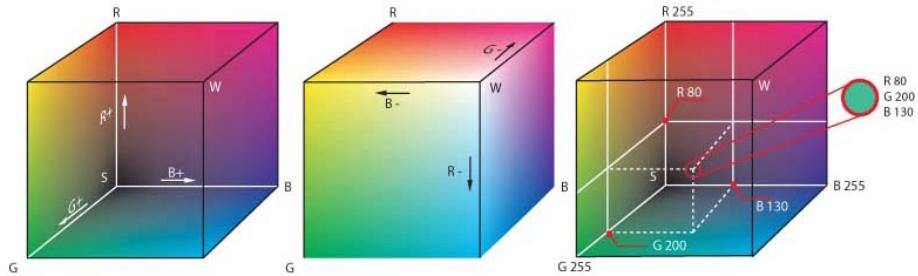
Rys. 3.10: Przykład modułów platformowych.



Rys. 3.11: Przykład różnych pozycji bohatera.

Model RGB

W modelu barw *RGB - Red Green Blue* (zob. rysunek 3.12) kolory powstają poprzez połączenie świateł o kolorach czerwonym, zielonym oraz niebieskim. Każdemu z kolorów początkowych można przypisać wartość od 0 do 255, a ponieważ model *RGB* jest modelem *addytywnym*, przypisanie każdej składowej wartości równej 0 da kolor czarny - brak światła, natomiast po przypisaniu wartości 255 dla każdej składowej, otrzymamy kolor biały - pełne światło. Dzięki tak dużemu zakresowi przypadającemu na każdy z kolorów bazowych można uzyskać ponad 16 milionów barw, co daje ogromne możliwości podczas tworzenia obrazów.



Rys. 3.12: Graficzna prezentacja modelu RGB. Źródło: [16]

Format PNG

Format *PNG* charakteryzuje bezstratna kompresja danych, która pozwala na wykorzystanie pełnej gamy kolorów modelu *RGB* bez straty na jej jakości. W *PNG* dostępny jest również *Kanał Alfa*, który umożliwia zastosowanie przeźroczystości. Dzięki jej wykorzystaniu postać w grze nie przypomina prostokąta, pomimo iż "plotno", na którym zastała narysowana jest w kształcie prostokąta. Takie rozwiązanie pozwala również na gładkie wtapianie w siebie różnych grafik, a tym samym mniejszą widoczność pojedynczych pikseli.

Gimp 2

Gimp 2 to zaawansowany program do edycji grafiki rastrowej, posiadający dziesiątki narzędzi dających niemal nieograniczone możliwości. Do tworzenia grafik użytych w grze "Tree of Life" zostały wykorzystane między innymi następujące przybory:

- *Pędzel* - Najbardziej podstawowe narzędzie w programach do grafiki rastrowej.

- *Rozsmarowywanie* - Narzędzie służące do rozmywania pikseli ze sobą sąsiadujących, w odpowiednim kierunku.
- *Rozjaśnianie/Przyciemnianie* - Za pomocą tego przyboru można rozjaśnić lub przyciemniać punktowo obszary obrazu, dzięki czemu można uzyskać na przykład efekt cienia.

Poza powyższymi narzędziami, w dużym stopniu została wykorzystana budowa warstwowa obrazu, dzięki której można pracować na jednym elemencie obrazu, nie ingerując w pozostałe.

Rozdział 4

Techniczne aspekty tworzenia gier komputerowych

Rozdział ten porusza kwestie związane z częścią programistyczną gier komputerowych. Opisane zostaną silniki oraz biblioteki służące do ich tworzenia oraz przedstawione w tej produkcji algorytmy. Przedstawiona zostanie również struktura projektu oraz sposoby testowania gry.

4.1 Silniki gier komputerowych

Obecnie na rynku istnieje wiele firm specjalizujących się w tworzeniu gier komputerowych. Wykorzystują one najróżniejsze technologie, aby ich produkcje były chętnie nabywane przez użytkowników. Jednym z ważniejszych elementów powstania dobrej produkcji jest dobranie odpowiedniego *silnika gry*, czyli głównej części kodu gry komputerowej, który jest dostępny razem z odpowiednim środowiskiem programistycznym. Jego głównym zadaniem jest ułatwienie pracy podczas tworzenia poszczególnych elementów oraz interakcję między nimi. Czasami zdarza się, że silnik gry ma wbudowane w siebie moduły do obsługi grafik, sieci oraz sztucznej inteligencji.

Istnieje bardzo duża ilość dostępnych silników umożliwiających programistom tworzenie gier komputerowych. Można tutaj wyróżnić:

- *Unity* - jest to jeden z najpopularniejszych silników. Stworzony przez firmę *Unity Technologies* i opublikowany w 2005 roku. Jego zaletą jest obsługa większości platform dostępnych na rynku. Jest on doskonały dla początkujących twórców, ponieważ jest bardzo intuicyjny. Dla twórców, których dochody nie przekraczają 100 tysięcy dolarów rocznie jest on darmowy.

- *Source Engine* - jest to silnik objęty prawami zamkniętego oprogramowania, stworzony przez firmę *Valve*. Swoją premierę miał w 2004 roku wraz z wydaniem kultowej gry *Half-Life 2*. Z biegiem lat był on ulepszany o coraz to nowe funkcjonalności, w efekcie czego w oparciu o niego powstało wiele popularnych tytułów: *Counter-Strike: Global Offensive*, *Dota 2* oraz wiele innych.
- *Unreal Engine* - wydany w 1998 roku przez firmę *Epic Games* wraz z grą *Unreal* silnik *Unreal Engine* jest jednym z najpopularniejszych silników gier na świecie. Wprowadzono w nim *schematy*, które umożliwiają pracę z nim nawet osobom nieznającym języka programowania *C++*.
- *REDengine* - jest to silnik opracowany przez polskie studio *CD Projekt RED* na potrzeby ich własnych produkcji. Wykorzystany został w grach z serii *Wiedźmin* oraz w grze *Cyberpunk 2077*. Jego główną zaletą jest możliwość projektowania złożonej oraz nielinowej fabuły.

4.2 Biblioteki do tworzenia gier komputerowych

Obok silników gier komputerowych na rynku dostępne są również biblioteki służące do ich tworzenia. Mimo ich małej popularności oraz większego skomplikowania podczas procesu powstawania gry, na ich podstawie również powstało kilka produkcji, które zawładnęły rynkiem gier.

4.2.1 The Lightweight Java Game Library

Czołowym tytułem, który powstał dzięki bibliotece służącej do produkcji gier jest najpopularniejsza produkcja na świecie *Minecraft Java Edition*(zob. rys. 4.1). Tytuł ten został napisany w oparciu o bibliotekę *The Lightweight Java Game Library*, w skrócie *LWJGL*. Jest to biblioteka dla języka *Java*, która zawiera w sobie biblioteki powszechnie stosowane podczas tworzenia gier takie jak: *OpenGL*, *Vulkan*, *OpenAL* oraz *OpenCL*. Zapewnia również dostęp do urządzeń periferyjnych umożliwiających sterowanie grą.



Rys. 4.1: Przykładowy zrzut ekranu z gry Minecraft Java Edition.

Źródło: [17]

4.2.2 Biblioteka Pygame

W projekcie *TreeOfLife* wykorzystana została biblioteka *pygame*. Stworzona została przez *Pete Shimmersa* w oparciu o biblioteki *SDL*. Jest ona udostępniona na zasadach wolnego oprogramowania i dystrybuowana na licencji *LGPL*¹⁾. Została ona napisana w języku programowania *Python*, który obecnie jest jednym z popularniejszych na rynku. Pozwala ona na wyświetlanie na ekranie grafik 2D oraz obsługę klawiatury czy też myszy. Jest ona bardzo prosta w obsłudze, co jest jej dużą zaletą. W przeciwieństwie do innych tego typu bibliotek, nie ma aż tak dużej liczby wbudowanych funkcji, więc aby osiągnąć zamierzony efekt programista musi zdać się na swoje umiejętności i wyobraźnię.

4.3 Algorytmy i struktury użyte w grze "Tree Of Life"

Podczas procesu tworzenia gry *Tree Of Life* wykorzystany został system kontroli wersji - git. Dzięki zastosowaniu tego narzędzia możliwe było wersjonowanie powstającego kodu. Pełny kod programu oraz przedstawione algorytmy znajdują się w repozytorium Github[18]. Mimo niewielkiej ilości poziomów w grze wykorzystanych zostało około 160 grafik. Za działanie całego programu odpowiada 32 klasy

¹⁾*LGPL* - licencja bardzo podobna do GPL, jednak w przeciwieństwie do niej nakłada ograniczenia tylko na poszczególne pliki źródłowe, a nie na cały program

oraz około 3000 linii kodu.

4.3.1 Przeliczanie Binarne

Przeliczanie Binarne jest podstawową operacją używaną w informatyce. W systemie dwójkowym liczby zapisujemy za pomocą cyfr 0 i 1. *System dwójkowy* jest używany w implementacji sprzętowej, która odpowiada stanom "włączony" i "wyłączony", jak również w celu zminimalizowania przekłamania danych.

4.3.2 Sortowanie bąbelkowe

Sortowanie bąbelkowe jest jednym z najstarszych algorytmów służących do sortowania. Jest to nieefektywny algorytm, ponieważ jego czasowa złożoność obliczeniowa jest rzędu $O(n^2)$, gdzie n oznacza rozmiar sortowanej tablicy. Polega on na zamianie sąsiadujących ze sobą elementów do czasu napotkania większego od niego elementu. W pesymistycznym przypadku program ten z każdym kolejnym krokiem będzie miał o jeden element mniej do porównania. Sytuacja ta będzie miała miejsce, gdy danymi wejściowymi będzie otrzymana tablica posortowana w sposób odwrotny do oczekiwanej.

4.3.3 Szyfr PlayFair

Szyfr PlayFair jest jednym z prostszych algorytmów szyfrujących. Został on wynaleziony w 1854 roku przez sir Charlesa Wheatstone'a, spopularyzował go baron Lyon Playfair, skąd wzięła się jego nazwa. Był on używany podczas I wojny światowej przez armię brytyjską. Polega on na podziale tekstu wejściowego na pary liter - gdy długość tekstu jest nieparzysta, na koniec ciągu znaków należy dodać literę na przykład 'z' - oraz zamianie ich na inną parę liter według ścisłe określonych zasad:

- Jeżeli obie litery znajdują się w tym samym wierszu, zamieniane zostają na litery sąsiadujące z prawej strony według tablicy szyfrującej - jeżeli litera będzie znajdować się w ostatniej kolumnie należy zastąpić ją znakiem z pierwszej kolumny,
- Jeżeli obie litery znajdują się w tej samej kolumnie, zamieniane zostają na litery sąsiadujące z dołu według tablicy szyfrującej - jeżeli litera będzie znajdować się w ostatnim wierszu, należy zastąpić ją znakiem z pierwszego wiersza,
- w pozostałych przypadkach daną parę liter należy zastąpić znakami leżącymi na przecięciu wierszy i kolumn, w jakich dane wejściowe się znajdują.

Do szyfrowania użyta zostaje również tablica szyfrująca, która składa się z niepowtarzających się liter alfabetu. Może zostać zastosowany również klucz szyfrowania - słowo, które umieszcza się na początku danej tablicy szyfrowania, a resztę znaków ustawia się zgodnie z zasadą niepowtarzalności w kolejności alfabetycznej na dalszych miejscach tablicy (zob. rysunek 4.2).

Proces odszyfrowania danego szyfrogramu polega na odwróceniu działań związanych z szyfrowaniem, czyli: litery, które zamieniane były na te sąsiadujące z prawej- zamieniane są na te z lewej, te zamieniane na te z dołu, na te z góry.



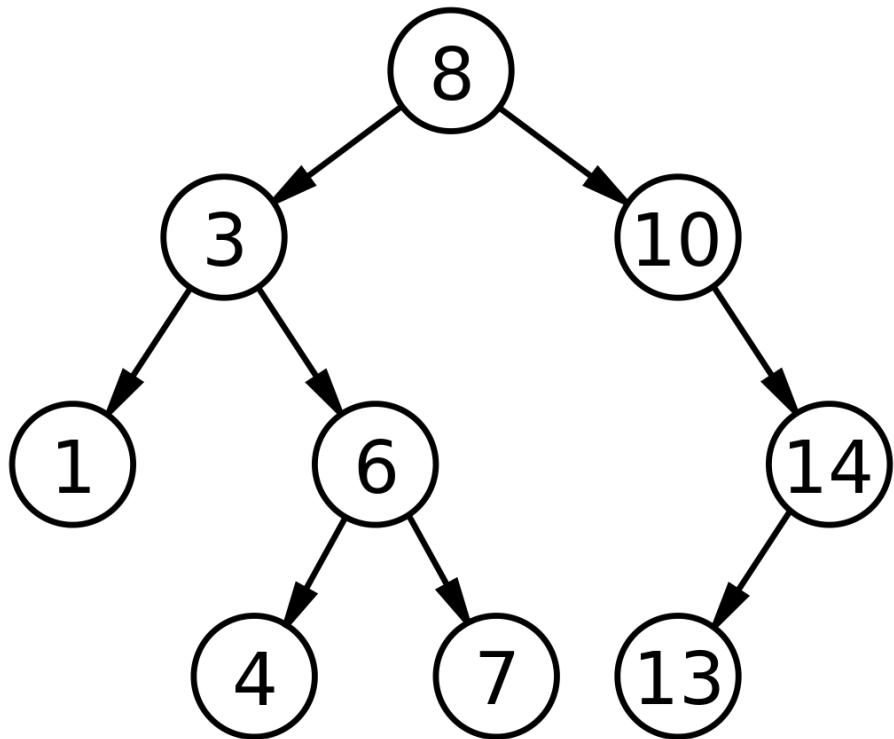
Rys. 4.2: Przykładowy proces szyfrowania za pomocą szyfry PlayFair z kluczem

4.3.4 Drzewa Binarne

Drzewa Binarne (zob. rysunek 4.3) są dynamicznymi strukturami danych dworzonymi zgodnie z zasadą, że każde lewe poddrzewo zawiera elementy mniejsze od *węzła*²⁾, natomiast prawe poddrzewo zawiera elementy większe od tegoż elementu. Każdy z węzłów oprócz wartości (klucza), przechowuje również wskaźniki na swojego lewego i prawego potomka jak również i na swojego przodka³⁾. Każde drzewo binarne posiada tzw. *korzeń* - pierwszy węzeł, który nie posiada elementu nadzędnego. W przeciwieństwie do innych typów drzew, drzewo binarne może posiadać co najwyżej dwoje dzieci - poddrzew.

²⁾ *węzeł* - element nadzędny dla poddrzew

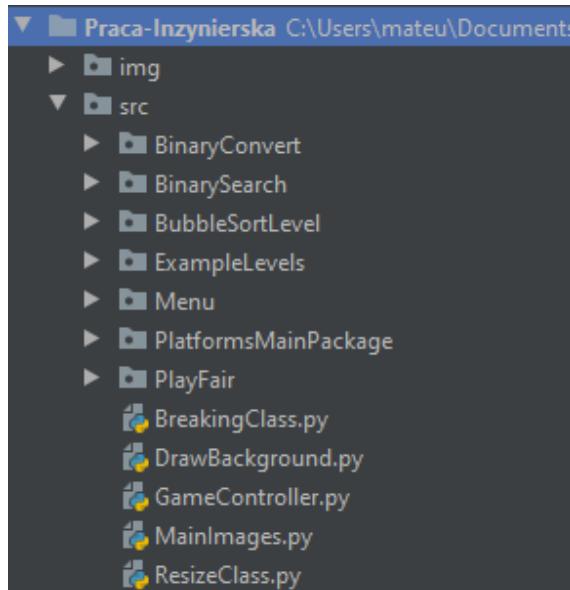
³⁾ *przodek* - element nadzędny dla danego węzła



Rys. 4.3: Przykładowe drzewo binarne o wysokości 3. Źródło: [19]

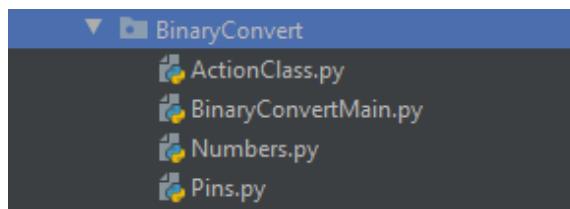
4.4 Struktura projektu

W projekcie *Tree Of Life* (zob. rysunek 4.4) można wyodrębnić następujące funkcjonalne części:



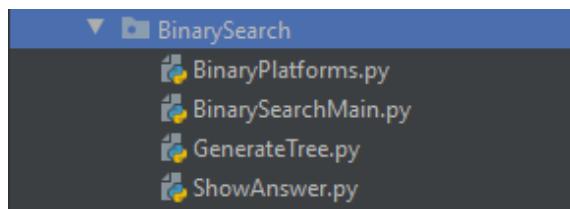
Rys. 4.4: Struktura katalogów Projektu

- Moduł *BinaryConvert* (zob. rysunek 4.5) zawierający operacje związane z poziomem przeliczania binarnego. Wyróżnia się tu:
 - *ActionClass.py* - klasa zawierająca zdefiniowane akcje dozwolone dla gracza,
 - *BinaryConvertMain.py* - główna klasa poziomu,
 - *Numbers.py* - klasa, w której zdefiniowane są operacje sprawdzające poprawność rozwiązania,
 - *Pins.py* - klasa reprezentująca wyświetlane ”piny” na ekranie,



Rys. 4.5: Struktura Modułu *BinaryConvert*

- Moduł *BinarySearch* (zob. rysunek 4.6) zawierający operacje związane z ostatnim poziomem w grze. Wyróżnia się tu:
 - *BinaryPlatforms.py* - zawiera klasę reprezentującą platformy,
 - *BinarySearchMain.py* - główna klasa poziomu,
 - *GenerateTree.py* - klasa odpowiedzialna za tworzenie drzewa binarnego,
 - *ShowAnswer.py* - klasa mająca na celu wyświetlanie przebytej drogi przez użytkownika,



Rys. 4.6: Struktura Modułu *BinarySearch*

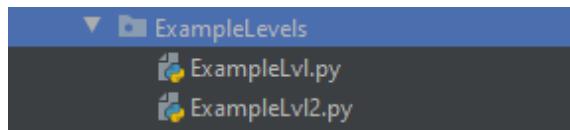
- Moduł *BubbleSortLevel* (zob. rysunek 4.7) zawierający operacje związane z poziomem sortowania bąbelkowego. Wyróżnia się tu:
 - *BubbleSortLevelMain.py* - główna klasa poziomu,

- *SwitchingButtons.py* - klasa odpowiedzialna za wyświetlane na ekranie przyciski,
- *SwitchingNumbers.py* - klasa zawierająca operacje związane z cyframi wyświetlonymi na ekranie,



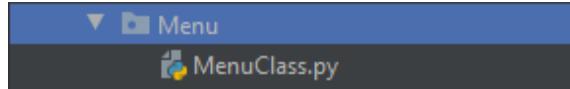
Rys. 4.7: Struktura Modułu BubbleSortLevel

- Moduł *ExampleLevels* (zob. rysunek 4.8) zawierający definicje poszczególnych poziomów platformowych. Wyróżnia się tu:
 - *ExampleLvl.py* - definicja poziomu w lokacji *SVARTAL*,
 - *ExampleLvl2.py* - definicja poziomu w lokacji *ANCIENT FOREST*,



Rys. 4.8: Struktura Modułu ExampleLevels

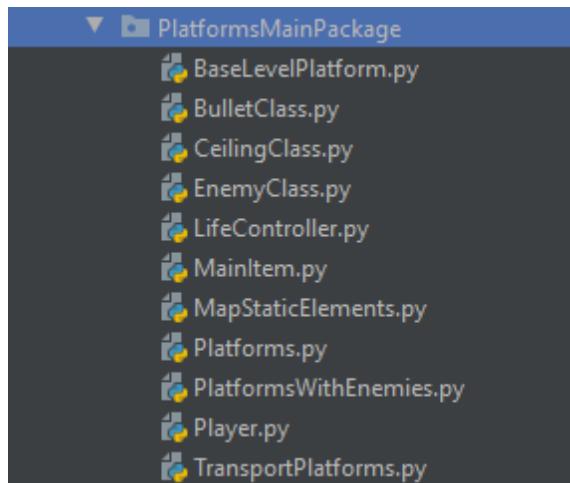
- Moduł *Menu* (zob. rysunek 4.9) zawierający klasę odpowiedzialną za ekran rozpoczęcia i zakończenia rozgrywki. Wyróżnia się tu:
 - *Menu.py* - klasa służąca do obsługi rozpoczęcia i zakończenia rozgrywki,



Rys. 4.9: Struktura Modułu Menu

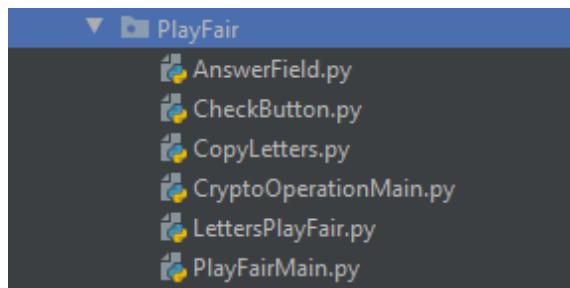
- Moduł *PlatformsMainPackage* (zob. rysunek 4.10) zawierający pliki odpowiedzialne za poszczególne elementy w poziomach platformowych. Wyróżnia się tu:
 - *BaseLevelPlatform.py* - główna klasa poziomów platformowych,
 - *BulletClass.py* - klasa odpowiadająca za pociski tworzone przez gracza bądź przeciwników,

- *CeilingClass.py* - klasa, która użyta będzie podczas rozwijania gry - odpowiada za sufit - zachowanie gracza, aby postać nie zniknęła poza ekranem gry,
- *EnemyClass.py* - klasa definiująca przeciwników,
- *LifeController.py* - klasa kontrolująca ilość życia bohatera,
- *MainItem.py* - klasa, w której zdefiniowane są wszystkie elementy spotykane na mapie przez gracza,
- *MapStaticElements.py* - klasa, odpowiadająca za zakończenie poziomu przez gracza - w przyszłości również za elementy statyczne spotykane na mapie - przeszkody,
- *Platforms.py* - główna klasa definiująca platformy wyświetlane na ekranie,
- *PlatformsWithEnemies.py* - klasa odpowiadająca za platformy, na których poruszają się przeciwnicy,
- *Player.py* - główna klasa gracza,
- *TransportPlatforms.py* - klasa reprezentująca ruchome platformy,



Rys. 4.10: Struktura Modułu PlatformsMainPackage

- Moduł *PlayFair* (zob. rysunek 4.11) zawierający operacje związane z poziomem kryptograficznym:
 - *AnswerField.py* - klasa odpowiadająca za pobieranie odpowiedzi ułożonej przez gracza,
 - *CheckButton.py* - klasa reprezentująca przycisk sprawdzenia poprawności rozwiązania,
 - *CopyLetters.py* - klasa odpowiadająca za kopie liter tworzonych przez użytkownika, w celu rozwiązania zagadki,
 - *CryptoOperationMain.py* - klasa zawierająca operacje sprawdzające rozwiązanie oraz generowanie klucza szyfrującego,
 - *LettersPlayFair.py* - klasa odpowiadająca za rysowanie na ekranie liter służących do tworzenia odpowiedzi,
 - *PlayFairMain.py* - główna klasa poziomu,



Rys. 4.11: Struktura Modułu PlayFair

- *BreakingClass.py* - klasa zawierająca obsługę przerywników w grze
- *DrawBackground.py* - klasa odpowiadająca za wyświetlanie tła dla poziomów
- *GameController.py* - klasa zarządzająca rozgrywką
- *MainImages.py* - zawiera w sobie wszystkie wymagane w grze grafiki
- *ResizeClass.py* - zawiera w sobie funkcje służące do skalowania wyświetlanych obrazków

4.5 Testowanie Gry

Gra *Tree Of Life* została przetestowana za pomocą testów manualnych. Są to testy bazujące głównie na zasobach ludzkich. Jest to czasochłonny proces, ponieważ tester musi sprawdzać każdą funkcjonalność ręcznie. Są one również bardziej podatne na błędy w porównaniu do testów automatycznych, gdzie proces sprawdzania programu odbywa się za pomocą wcześniej przygotowanych skryptów - scenariuszy.

Testerami gry *Tree of Life* byli potencjalni gracze, jak również zawodowi testerzy. Każda osoba po zakończonej rozgrywce odpowidała na następujące pytania:

1. *Czy podczas grania w grę TreeOfLife natrafiliś na jakieś błędy (tzw. bugi)?*
2. *Czy znalezione błędy przeszkadzały w rozgrywce?*
3. *Czy gra jest grywalna?*
4. *Czy sposób przedstawienia algorytmów w grze był jasny/zrozumiały?*

Podczas testowania testerzy natrafili na różnego rodzaju błędy. Były one głównie związane z poziomami platformowymi, natomiast poziomy, w których użytkownik musiał rozwiązać zagadkę algorytmiczną działały bez zarzutów. Większość z napotkanych błędów dotyczyła generowania przeciwników: nie poruszali się oni po całej dostępnej platformie, jedynie po jej niewielkiej części. Część z graczy zauważała również problem z wykrywaniem kolizji. Jednak te błędy nie przeszkadzały w rozgrywce.

Algorytmy w produkcji według oceny testerów, przedstawione były w sposób zrozumiały.

Zakończenie

Pomimo faktu rozpoczęcia pracy w marcu 2020 roku, gra nie została w pełni skończona. Jest ona ciągle udoskonalana oraz rozbudowywana o coraz to nowe elementy. Bazując na doświadczeniu przy tworzeniu gry *Tree Of Life*, która była niekomercyjnym, dwuosobowym projektem, można wyobrazić sobie jak wielki wysiłek muszą włożyć firmy zajmujące się profesjonalnym tworzeniem gier, aby ich tytuły cieszyły się niesłabnącą popularnością. *Tree Of Life* mimo, że nie jest dużą produkcją zawiera w sobie kilka niedociągnięć - które będą eliminowane, aby przyszli użytkownicy mogli choć w niewielkim stopniu poznać podstawy świata informatyki, bądź sprawdzić swoją wiedzę na jego temat. Po zakończeniu prac nad grą można stwierdzić, że dwuosobowy zespół to stanowczo za mało na stworzenie rozbudowanej produkcji, która nie będzie miała błędów utrudniających bądź ułatwiających rozgrywkę.

W przyszłości planowane jest przeniesienie gry na silnik *Unity* oraz rozbudowanie jej o większą liczbę poziomów, a także dodanie wyboru poziomu trudności rozgrywki, tak aby użytkownik mógł wybrać, na jakim stopniu zaawansowania chce grać.

Bibliografia

- [1] <https://sjp.pwn.pl/slowniki/gra%20komputerowa.html>. Dostęp: 2020-12-05.
- [2] <https://pl.wikipedia.org/wiki/Pong>. Dostęp: 2020-12-05.
- [3] https://pl.wikipedia.org/wiki/Automat_do_gry. Dostęp: 2020-12-05.
- [4] <https://www.telepolis.pl/images/2020/07/super-mario-bros.jpg>. Dostęp: 2020-01-12.
- [5] https://www.50gameslike.com/sites/default/files/styles/screen_large/public/images/h/hollow_knight_godmaster/hollow_knight_godmaster_1.jpg. Dostęp: 2020-01-12.
- [6] <https://sjp.pl/cutscena>. Dostęp: 2020-12-05.
- [7] <https://testerzy.pl/baza-wiedzy/testowanie-akceptacyjne>. Dostęp: 2020-12-05.
- [8] www.steampowered.com. Dostęp: 2020-12-05.
- [9] <https://historiamiejznanaizapomniana.wordpress.com/2015/12/01/draugr-nieumarly-wiking/>. Dostęp: 2020-12-05.
- [10] <http://roznice.com/technologia/komputery/roznice-miedzy-grafika-rastrowa-a-grafika-wektorowa/>. Dostęp: 2020-01-12.
- [11] <https://craftportal.pl/paczki-zasobow-tworzyc-modele-3d-czesc-1/28718>. Dostęp: 2020-01-12.
- [12] <https://steamcommunity.com/sharedfiles/filedetails/?id=1368519389>. Dostęp: 2020-01-12.

- [13] <http://www.ps2hardcore.net/forum/viewtopic.php?t=3852&p=122218>. Dostęp: 2020-01-12.
- [14] <https://www.artstation.com/artwork/2AYqe>. Dostęp: 2020-01-12.
- [15] <https://hytalegame.pl/wprowadzenie-do-generatora-swiatow-hytale/>. Dostęp: 2020-01-12.
- [16] https://pl.wikipedia.org/wiki/Plik:RGB_farbuerfel.jpg. Dostęp: 2020-01-13.
- [17] https://cdn.benchmark.pl/uploads/backend_img/c/recenzje/2020_04/10323_Co_to_Minecraft_poradnik/Minecraft_2.jpg. Dostęp: 2020-01-12.
- [18] <https://github.com/Paarzivall/Praca-Inzynierska>.
- [19] https://pl.wikipedia.org/wiki/Drzewo_binarne#/media/Plik:Binary_tree.svg. Dostęp: 2020-01-12.
- [20] Rafał Nowocień. *Tworzenie gier komputerowych. Kompendium producenta*. Helion, Poland, 2019.

Spis rysunków

1.1	Gra "Super Mario Bros" z 1986. Źródło: [4]	8
1.2	Gra "Hollow Knight" z 2017 Źródło: [5]	8
2.1	Wygląd mechanizmu zamka	17
2.2	Wygląd podpowiedzi dla gracza do pierwszego poziomu	18
2.3	Przykładowy wygląd poziomu w lokacji "SVARTAL"	19
2.4	Przykładowa podpowiedź do poziomu Bubble Sort	20
2.5	Wygląd mechanizmu wyjścia z krainy elfów	20
2.6	Przykładowy wygląd poziomu w lokacji "ANCIENT FOREST" . . .	21
2.7	Przykładowy wygląd mechanizmu bramy	22
2.8	Przykładowa podpowiedź do poziomu PlayFair	22
2.9	Przykładowy wygląd gałęzi w lokacji "Tree Of Life"	23
2.10	Przykładowy wygląd podpowiedzi w poziomie Tree Of Life	23
3.1	Przykład obrazu rastrowego po skalowaniu: Po lewej stronie - oryginalny obraz, Po prawej stronie - obraz zmniejszony czterokrotnie . . .	28
3.2	Przykład zastosowania miękkich krawędzi pędzla - po prawej stronie, oraz twardych krawędzi - po lewej stronie	29
3.3	Różnica w skalowaniu grafik wektorowej i rastrowej. Źródło: [10] . . .	30
3.4	Przykłady obiektów trójwymiarowych	31
3.5	Przykładowe tekstury i przedmioty. Źródło: [11]	32
3.6	Przykład tła w grze. Źródło: [12]	32
3.7	Przykład ekranu ładowania. Źródło: [13]	33
3.8	Przykładowy interfejs. Źródło: [14]	33
3.9	Przykład grafiki koncepcyjnej. Źródło: [15]	34
3.10	Przykład modułów platformowych.	35
3.11	Przykład różnych pozycji bohatera.	35
3.12	Graficzna prezentacja modelu RGB. Źródło: [16]	36
4.1	Przykładowy zrzut ekranu z gry Minecraft Java Edition. Źródło: [17]	40

4.2	Przykładowy proces szyfrowania za pomocą szyfry PlayFair z kluczem	42
4.3	Przykładowe drzewo binarne o wysokości 3. Źródło: [19]	43
4.4	Struktura katalogów Projektu	43
4.5	Struktura Modułu BinaryConvert	44
4.6	Struktura Modułu BinarySearch	44
4.7	Struktura Modułu BubbleSortLevel	45
4.8	Struktura Modułu ExampleLevels	45
4.9	Struktura Modułu Menu	45
4.10	Struktura Modułu PlatformsMainPackage	46
4.11	Struktura Modułu PlayFair	47