

Politechnika Wrocławskiego
Wydział Informatyki i Telekomunikacji

Kierunek: **IST**
Specjalność: **IO**

**PRACA DYPLOMOWA
MAGISTERSKA**

**Opracowanie modelu formalnego oraz implementacja
mechanizmu generowania grafowego rozszerzenia
zestawu zlinearyzowanych zadań fabularnych
wyekstrahowanych z popularnych gier typu MMORPG**

Inż. Marcelli Sokolski

Opiekun pracy
Dr inż. Grzegorz Popek

Slowa kluczowe: proceduralne generowanie treści, generowanie fabuły, projektowanie gier komputerowych, model Dorana

Streszczenie

Celem pracy jest analiza możliwości wyrażania zadań pochodzących z gier RPG przez model zadań fabularnych Dorana i Parberrego oraz rozszerzenie ich modelu, aby mógł reprezentować bardziej skomplikowane zadania. Zamodelowano nim zadania pochodzące z popularnej gry RPG i stwierdzono, że model Dorana nie potrafi przedstawiać zadań nieliniowych. W celu wyeliminowania tej wady rozszerzono go o operatory umożliwiające podejmowanie decyzji przez gracza. Następnie wykorzystano rozszerzony formalny model do implementacji generatora zadań fabularnych, który wykorzystuje autorskie mechanizmy starające się zwiększyć spójność interpretacyjną zadań oraz gwarantujące ich spójność logiczną.

Abstract

The goal of this thesis is to analyze Doran and Parberry's quest model for ability to represent quests from Role Playing Game and to extend the model in order to represent quests of extended complexity. The Doran's model was used to model quests from a popular RPG and it was concluded that the the model is not able to represent nonlinear quests. In order to fix this fault, the model was extended with operators that allow player to make decisions. After that, the newly designed formal model was used to implement a quest generator that uses original mechanisms, in order to guarantee quests completion and make quests feel natural.

Spis treści

Wstęp	1
1 Narracyjne struktury w grach	7
1.1 Fabularne wzory	7
1.2 Sposoby obcowania z fabułą	9
2 Przegląd sposobów generacji fabuły i zadań	11
2.1 Systemy oparte na planowaniu	11
2.1.1 Problem planowania	12
2.1.2 Cechy systemów opartych na planowaniu na przykładzie systemu <i>CONAN</i>	13
2.2 Systemy oparte na modelu fabuły lub modelu zadania wyrażonym w gramatyce bezkontekstowej	17
2.2.1 Gramatyka bezkontekstowa	17
2.2.2 Charakterystyka systemów	18
2.3 Systemy oparte na programowaniu logicznym	19
2.4 Silniki do zarządzania całością historii	19
3 Analiza i rozszerzenie modelu zlinearyzowanych zadań fabularnych Dorana i Parberrego	21
3.1 Notacja Backusa-Naura	21
3.2 Model zadań Dorana	22
3.3 Dualizm językowo-grafowy zadań	28
3.4 Analiza zadań z gry Wiedźmin 3	30
3.4.1 Zadania-zdarzenia	32
3.4.2 Zadania pełne	33
3.4.3 Podsumowanie	35
3.5 Rozszerzona notacja Backusa-Naura	36
3.6 Rozszerzenie modelu zadań	37
3.7 Analiza zadań z gry Wiedźmin 3 z wykorzystaniem rozszerzonego modelu	43
4 SkaldNET – Prototyp generatora zadań fabularnych	47
4.1 Ogólne informacje o systemie	47
4.2 Model świata	47
4.3 Decyzje podejmowane w trakcie generacji	50
4.3.1 Generacja drzewa-zadania w trakcie przechodzenia	50
4.3.2 Kontrola stanu świata gry w trakcie generacji	53

4.3.3	Warunki wstępne akcji złożonych i strategii	54
4.3.4	Mechanizm obostrzeń – utrzymanie spójności zadania	57
4.4	Wysokopoziomowy algorytm generacji zadania	61
4.5	Wygenerowane zadania	63
4.5.1	Wizualizacja drzew-zadań	63
4.5.2	Przykłady wygenerowanych zadań	64
Podsumowanie		67
Bibliografia		69
Spis rysункów		71
Spis tablic		73
A Model Dorana et al. w notacji Backusa-Naura		75
B Zadania z gry Wiedźmin 3 wyrażone modelem Dorana		77
C Zadania z gry Wiedźmin 3 wyrażone rozszerzonym modelem		93

Wstęp

W 2020 roku Brytyjska Akademia Sztuk Filmowych i Telewizyjnych podczas gali British Academy Games Awards nagrodziła nagrodą Gry Roku grę *Hades* wydaną przez niezależne studio Supergiant Games. Nagrodę w analogicznej kategorii ta gra również otrzymała na prestiżowej gali D.I.C.E. Awards. Supergiant Games konkurowało z największymi studiami w branży gier, ale odniósło zwycięstwo pomimo posiadania mniejszych zasobów ludzkich oraz budżetu. Jednym z powodów, dla których wyprodukowana przez nich gra pokonała konkurencję, mogło być zastosowanie proceduralnego generowania treści, które znacząco zwiększyło doznania z ponownego przechodzenia rozgrywki, czyli tak zwane jej *replayability*.

Proceduralna generacja treści (ang. *procedural content generation*, w skrócie PCG) jest jednym z przedmiotów badań naukowców zajmujących się tematyką gier, w szczególności gier wideo. Termin “*generacja treści*” jest jednak mało sprecyzowanym pojęciem. Jak zauważali Togelius et al., powoduje to problemy z określeniem co można zaliczyć do PCG, a czego nie [21]. W trakcie swoich badań zdefiniowali oni PCG jako “*wykorzystanie algorytmów do automatycznej generacji treści*”. Przyjęta przez nich definicja może budzić kontrowersje ze względu na wymóg wykorzystania *algorytmu*, który wiąże się z wykluczeniem losowości z procesu proceduralnej generacji treści. Jak zauważają autorzy, istnieją inne definicje PCG, które są sprzeczne z tym założeniem lub nawet jawnie dopuszczają i promują zastosowanie losowości do PCG. Jedną z takich definicji jest definicja PCG autorstwa A. Doull'a, w której autor podkreślił wielki potencjał wykorzystania losowości w PCG [9]. A. Doull stwierdził, że losowość jest wręcz pożądana w PCG z powodu oczekiwania dużej ilości różnorodnych wyników dla małej ilości wprowadzanych parametrów do generatora. Na potrzeby niniejszej pracy przyjęto, że wykorzystanie elementów losowych nie będzie wykluczało klasyfikacji uzyskanego rozwiązania jako przynależącego do PCG.

PCG jest powszechnie wykorzystywane w branży gier wideo nie tylko do generowania treści w trakcie rozgrywki, ale także jako narzędzie służące jako pomoc przy projektowaniu zawartości. Togelius et al. [17, 21] wyróżniają ze względu na moment generacji dwa rodzaje PCG: *online* i *offline*.

Online PCG to generowanie treści, które odbywa się w trakcie działania aplikacji (gry). Zaletą zastosowania tego rodzaju PCG jest stworzenie modelu “*niekończącej się rozgrywki*” [17] w następstwie możliwości dodatkowego generowania nowej zawartości dla gracza w momencie kiedy on “*skonsumuje*” poprzednią zawartość gry. Przytoczony na początku *Hades* także stosuje ten rodzaj PCG proceduralnie generując lokacje oraz nagrody dla gracza. Gra tworzy układ poziomów łącząc ze sobą lokacje ze zbioru spreddefiniowanych wcześniej lokacji.

W przeciwnieństwie do *Online* PCG, generowanie treści w ramach *Offline* PCG jest wykonywane przed rozpoczęciem rozgrywki. Najczęstszym jego zastosowaniem jest wsparcie w procesie twórczym projektowania gry lub wygenerowania wariantów przedmiotów czy postaci, które później zawarte będą w grze (określanych w nomenklaturze gier komputerowych jako *asset*). Wygenerowane w ten sposób *asset*y są często poddawane ręcznej obróbce w celu poprawy ich jakości i dopiero wówczas zostają umieszczone w grze.

Uprzednio wymienione zastosowania *Offline* i *Online* PCG są tylko jednymi z wielu możliwych. A. Doull wskazuje 7 najpopularniejszych [9]:

- generacja poziomów gry w czasie gry;
- projektowanie zawartości gry;
- dynamiczne generowanie świata gry;
- instancjonowanie obiektów;
- treści udostępniane przez użytkowników;
- dynamiczne systemy AI;
- generacja zagadek oraz fabuły.

Przytoczone stosowane wykorzystania PCG jednoznacznie uwidaczniają potencjał PCG w procesie wytwarzania gier wideo. Jednakże zastosowanie algorytmów do generacji treści, która będzie później konsumowana przez ludzkich odbiorców, może budzić wątpliwości dotyczące jakości otrzymywanych zasobów. Sprawiło to, że znawcy tematyki gier zaczęli badać, czy generowanie zawartości nie ma również negatywnego wpływu na sam produkt końcowy. Badania starające się odpowiedzieć na tak postawione pytanie podsumowane zostały w pracy Rodrigues et al. [17]. Dokonany przez nich przegląd literatury uwzględniający prace porównujące reakcje odbiorców na fragmenty rozgrywki wykorzystujące PCG oraz te, które go nie stosowały, nie pozwolił na obalenie następujących hipotez [17]:

- Poziom zadowolenia graczy może nie różnić się od siebie.
- Poziom chęci do ponownej rozgrywki może nie różnić się od siebie.
- Poziom ciekawości graczy może nie różnić się od siebie.

Na podstawie analizowanych przez nich badań można przyjąć, że przy odpowiednim wykorzystaniu PCG można osiągnąć porównywalne, albo nawet lepsze rezultaty, niż decydując się na ręczne tworzenie całej zawartości gry. Wnioskom tym nie zaprzecza pozytywny odbiór przez graczy oraz krytyków branżowych nagradzanego *Hadesa*.

Spośród wymienionych przez A. Doull'a zastosowań PCG, generacja fabuły do gier lub jej fragmentów sprawia wrażenie szczególnie niszowego. W grach fabuła często przybiera formę zbioru lub listy celów do wykonania przez gracza określanych w nomenklaturze gier jako "zadania"(ang. "quest"). Tematyka generowania fabuły wydaje się być niepopularna zarówno pod względem ilości gier stosujących proceduralną generację zadań, jak i publikacji naukowych poświęconych na jej temat.

Togelius et al. wskazuje, że aby móc zastosować *online* PCG należy mieć szybkie rozwiązanie, które jest w stanie generować treść o możliwej do przewidzenia jakości [17, 21]. Wymóg ten oznacza problem oceny jakościowej wygenerowanych fragmentów fabularnych. Muszą one sprawiać pośród odbiorców **subiektywne** wrażenie bycia "dobrej jakości". Z tego powodu zaistniała potrzeba, aby otrzymywane artefakty były "zadowalające według pewnego kryterium" [21]. O ile takie kryterium w przypadku generacji możliwych do umieszczenia w

grze assetów można powiązać z budzeniem określonych emocji w odbiorcach, to w przypadku generacji zadań fabularnych, czy też całości fabuły, zdefiniowanie takiego kryterium może sprawiać większe problemy. Niezwykle trudno przedstawić numerycznie jakość treści zadania, obliczyć ją za pomocą wzorów matematycznych czy też algorytmów. W celu zdefiniowania takiego kryterium można jednak sięgnąć do badań poświęconych samym grom, czy też zabawie.

Johan Huizinga w dziele pod tytułem “*Homo ludens*” [14] przygląda się grom jako formie zabawy. Współcześni badacze gier przypisują autorowi genezę znanego w branży pojęcia “zaczarowanego koła”. Zaczarowane koło wykorzystywane jest jako opis wagi spójności w grach. Pojęcie to obrazuje granice między światem określonym przez badacza jako *zwyczajny* oraz światem gry, czyli “*miejscem uświeconym, odgraniczonym, sakralnym, na którym obowiązują szczególne, swoiste prawa*”. Te prawa, czy też zasady, mogą różnić się od tych znanych w *zwyczajnym* świecie. Zasady występujące w świecie gry ograniczone są jedynie warunkiem zachowania spójności względem świata gry [19]. Kryterium spójności jest jednym z możliwych pomysłów na ocenę jakościową generowanej fabuły lub zadań fabularnych. Innymi sposobami może być ocena złożoności zadań, ich długości lub różnorodności.

Generację fabuły lub zadań fabularnych można, tak samo jak całość PCG, podzielić na generację *online* i *offline*. Generacja *online* jest tak powszechna, że zastosowanie jej jest uznawane za jedną z głównych cech niektórych gatunków wśród gier wideo, takich jak na przykład *Roguelike* lub *Roguelite*. Generacja fabuły *offline* jest trudniejsza do zaobserwowania w branży gier. Dobrze wykorzystana mogłaby być niezauważalna dla niewtajemniczonego użytkownika, a twórcy gier raczej nie są chętni do przyznawania się do wykorzystania jej przy projektowaniu gry. Nie mniej jednak, nie brakuje programów wykorzystujących generację fabuły *offline*, w celu wsparcia *Game Masterów* lub *Dungeon Masterów*¹ w projektowaniu kampanii dla graczy.

Przykładem gry znanej z wykorzystania PCG do generacji fabuły jest *Dwarf Fortress* (w skrócie DF). Gra generuje cały świat w konwencji średniowiecznego fantasy, wraz z jego historią i zamieszkującymi go postaciami. Generacja świata odbywa się tuż przed pierwszym rozpoczęciem rozgrywki w danym świecie, ale również jej trakcie, kiedy odbywa się symulacja tego świata. W trakcie gry gracz ma możliwość obserwacji zachowania tytułowych krasnoludów i śledzenia ich zachowań oraz przygód (co też jest formą generacji fabuły). Przyglądając się historiom, które potrafi opowiadać DF należy zwrócić uwagę na dzieżace się w nich wydarzenia absurdalne z perspektywy osoby niezaznajomionej z grą. Przykładowo, jedna z historii opowiada o tym, jak zaatakowanie przerośniętej iguany, która zdomowiliła się w pobliżu źródła wody doprowadziło do wojny domowej [7]. Pomimo oczywistych absurdów, jakie się wydarzyły w tej historii, opowiedziane wydarzenia są spójne z zasadami gry i konwencją fantasy, co powoduje, że wygenerowaną fabułę można uznać za pożądaną.

W ramach prac naukowych poświęconych *offline* PCG często powstają generatory zarysu całej fabuły, tylko jej wycinków, czy też zadań (ang. *questów*). Wśród prac naukowych popularne wydaje się być opracowywanie systemów o tak zwanej mieszanej inicjatywie (ang. *mixed-initiative*). Systemy tego rodzaju mogą mieć mają dwa rodzaje użytkowników: projektantów oraz graczy [3]. Gracze są użytkownikami końcowymi wygenerowanej fabuły, a projektanci to osoby nadzorujące proces generacji. Korzystając z takich systemów projektant

¹ *Game Master* lub *Dungeon Master* to określenie na osobę prowadzącą rozgrywkę (kampanię) w klasycznych RPG, takich jak *Dungeons and Dragons*.

opowieści często otrzymuje jedynie podpowiedzi dotyczące kolejnych kroków od generatora. Niektóre systemy o przemiennej inicjatywie wspierają ręczne projektowanie wycinków zadania i wypełnienia wydzielonych części opowieści. Dzięki temu człowiek ma ostateczną pieczę nad wynikową historią i może subiektywnie ją ocenić.

W niniejszej pracy skupiono się na analizie modelu zadań stworzonego z myślą o generacji *offline*, ale charakteryzującego się niską złożonością, pozwalającą na potencjalne wykorzystanie go do generacji *online*. Model ten został opracowany przez Dorana i Parberrego na potrzeby publikacji pod tytułem “*A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs*” [8].

W ramach wspomnianej pracy twórcy wyekstrahowali zestaw zlinearyzowanych zadań fabularnych (ang. *quest*) z popularnych gier typu MMORPG. Następnie zamodelowali je z wykorzystaniem formalnej gramatyki bezkontekstowej przyjmując pewien możliwy zbiór atomowych zadań (ang. *task*) możliwych do wykonania. Otrzymali w ten sposób gramatykę pozwalającą zarówno na generację nowych zadań (ang. *questów*), jak i klasyfikację zadań ze względu na motywację zleceniodawcy.

Rozwiążanie zaproponowane przez Dorana nie jest pozbawione wad. Jest w stanie przedstawiać jedynie zlinearyzowane zadania fabularne. Ponadto, jest bardzo ogólne i nie zakłada wielu warunków zawężających dziedzinę możliwych do uzyskania na wyjściu zadań. Przykładowo, model Dorana pozwoli na wygenerowanie zadania, w którym bohater zabija zleceniodawcę, aby zdobyć przedmiot, o którego przyniesienie ten zleceniodawca poprosił bohatera. Innymi słowy, model Dorana nie posiada mechanizmów zapewniających zadaniom spójność logiczną (wykonywalności) oraz wiarygodność. Powoduje to, że model Dorana lepiej działa jako generator szablonów zadań, które wymagają ingerencji człowieka w celu ich transformacji w zadania o pożądanej jakości.

Zakres pracy

Niniejsza praca została poświęcona tylko jednemu z zastosowań PCG, jakim jest **proceduralna generacja zadań fabularnych**, a konkretniej zadań fabularnych przystosowanych do gier, w których gracz pełni rolę bohatera. W ramach tej pracy nastąpiła dokładna analiza modelu zadań fabularnych zaproponowanego przez Dorana i Parberrego [8].

Model ten został opracowany w oparciu o cztery popularne gry MMORPG. Zadania typowe dla gier tego gatunku są liniowe i przeważają wśród nich zadania o niskim poziomie skomplikowania. Wspomniane cechy charakterystyczne zadań gier MMORPG sprawiły, że model Dorana dobrze opisuje krótkie zadania o niskim poziomie skomplikowania, jednakże jego zdolności dotyczące przedstawiania dłuższych, bardziej skomplikowanych zadań nie zostały wystarczająco przebadane.

W związku ze wspomnianym brakiem oceny ekspresyjności modelu w kontekście skomplikowanych zadań, w niniejszej pracy dokonano analizy pobocznych zadań pochodzących z popularnej gry RPG. W tym celu wykorzystano grę *Wiedźmin 3: Dziki Gon*, określana dalej w pracy jako *Wiedźmin 3*. Przeprowadzone badanie wskazało powtarzalne fragmenty zadań, których model Dorana nie jest w stanie wyrazić.

Dzięki ekstrakcji niewyrażalnych elementów zadań w modelu Dorana, w niniejszej pracy proponuje się rozszerzenie badanego modelu o nowe reguły i operatory umożliwiające przed-

stawianie decyzji gracza. Opracowany model cechuje się możliwością przedstawiania zadań nieliniowych, czyli takich, w których gracz ma możliwość podejmowania wyborów wpływających na zakończenie zadania.

Na potrzeby niniejszej pracy zaimplementowano prototyp generatora zadań fabularnych zgodnego z rozszerzonym modelem. Wraz z implementacją została poruszona problematyka generacji zadań fabularnych, które mogłyby sprawiać wrażenie naturalnych, a co za tym idzie spójnych z zasadami świata gry.

Cele pracy

Niniejsza praca ma na celu:

- Analizę modelu Dorana pod względem możliwości wyrażania zadań pochodzących z gier RPG.
- Rozszerzenie modelu Dorana o reguły umożliwiające reprezentację bardziej skomplikowanych zadań (w szczególności zadań nieliniowych).
- Przygotowanie prototypu generatora zadań fabularnych, które są spójne pod względem logicznym (wykonywalne) oraz interpretacyjnym (racjonalne z fabularnego punktu widzenia).

Struktura pracy

Niniejsza praca dzieli się na 4 rozdziały. W rozdziale [pierwszym](#) zdefiniowane zostało pojęcie zadań fabularnych oraz podkreślona została istotność zawarcia w zadaniach związków przyczynowo-skutkowych. Rozdział [drugi](#) dotyczy różnych sposobów i technik generowania fabuły i zadań. Porównywane są ich zalety, wady oraz potencjalne zastosowania. W [trzecim](#), najważniejszym, rozdziale opracowany przez Dorana i Parberrego model zadań został poddany dogłębnej analizie oraz rozszerzony. W [czwartym](#), ostatnim rozdziale, została zaproponowana implementacja prototypu generatora zadań, zgodnego z opracowanym w ramach tej pracy rozszerzeniem analizowanego modelu. W podsumowaniu zawarto wnioski i spostrzeżenia końcowe oraz wskazano możliwe kierunki rozwoju opracowanego rozwiązania.

1. Narracyjne struktury w grach

Zakres pojęcia fabuła nie powinien sprawić czytelnikowi większych wątpliwości interpretacyjnych. Pojęcie to jest bowiem powszechnie wykorzystywane w kontekście wielu dzieł kultury, w tym gier. Badania pokazują, że fabułę zawartą w grach (i nie tylko) można dzielić na pewne narracyjne struktury, które często są zgodne ze strukturą dramatyczną zdefiniowaną przez Arystotelesa już w IV wieku przed naszą erą [1].

Wspomniane narracyjne struktury występujące w grach często przyjmują formę zadań (ang. *quest*), które dzielą całość fabuły na mniejsze, łatwiejsze do przyjęcia części [1]. Kristen K. Yu et al. na podstawie przeglądu literatury proponują definiować zadanie (ang. *quest*) jako “*częściowo uporządkowany zbiór zadań (ang. task) T, które gracz musi wykonać aby otrzymać nagrodę ze zbioru R*” [24].

A. Alvarez i J. Font zwracają uwagę, że narracyjne struktury, z których złożone są fabuły gier, wykorzystują typowe motywy popularne w dziełach kultury [1]. Wiedza o tych motywach oraz wzorach które mogą formułować będzie istotna w dalszej części pracy, gdyż posłuży do wyciągania wniosków dotyczących siły ekspresji analizowanego modelu formalnego zadań.

1.1. Fabularne wzory

A. Alvarez i J. Font w pracy poświęconej utworzeniu generatora narracyjnych struktur na podstawie popularnych motywów obecnych w dziełach kultury wyszczególnili powszechnie występujące w grach video motywy [1]. Wyniki ich pracy można zobaczyć w tabeli 1.1. Analizując motywy zaobserwowali, że występowanie ich w odpowiednich kontekstach fabuły można grupować w pewne wzory (ang. *patterns*), które podzieliли na [1, 2]:

- “*mikro-wzory*” – atomowe wzory budujące bardziej skomplikowane narracyjne jednostki. Dzielą się na *wzorce postaci, wzory strukturalne* oraz *wzory narzędzi fabuły*. Mikro wzory wyszczególnione w pracy zawarte są wśród popularnych motywów z tabeli 1.1;
- “*meso-wzory*” – wzory złożone z innych *mikro-wzorów* lub *meso-wzorów*. Dzielą się między innymi na:
 - *wzór konfliktu* – łączący motyw konfliktu (będący przykładem *mikro-wzoru*) z jego stronami (także *mikro-wzorami*);
 - *wzór wynikowy* (ang. *derivative pattern*) – obrazujący ciągi przyczynowo skutkowe;
 - *wzór ujawnienia* (ang. *reveal pattern*) – przedstawiający sytuację, kiedy byty A oraz B okazują się tym samym. Jako przykład występowania takiego wzoru można przytoczyć zakończenie filmu “*Fight club*” w którym dwie postaci okazują się tą samą osobą;
 - *plot twist* – punkt fabularny (kluczowe wydarzenie), który zmienia naturalny przebieg lub porządek fabuły.

Tabela 1.1: Typowe motywów w grach wideo [1]

Nazwa	Definicja
Bohater (ang. Hero)	Bohater opowieści
5-osobowa grupa (ang. Five-man band)	Grupa złożona z maksymalnie pięciu archetypowych postaci
Wybraniec (ang. The chosen one)	Konkretny bohater wybrany jako “ten wybraniec”
Superbohater (ang. Superhero)	Konkretny bohater z unikalnymi umiejętnościami
Konflikt (ang. Conflict)	Ogólny problem do przezwyciężenia pomiędzy postaciami
Przeciwnik (ang. Enemy)	Nemezis dla bohatera
Imperium (ang. Empire)	Zbiorowy przeciwnik z ambicją zdobycia świata
Wielki zły (ang. Big bad)	Konkretny przeciwnik, który jest ostatecznie odpowiedzialny za wszystko co złe
Smok (ang. Dragon)	Konkretny przeciwnik będący prawa ręką “Wielkiego złego”
Narzędzie fabuły (ang. Plot device)	Funkcjonalność lub element, który “ <i>pcha</i> ” fabułę do przodu
Chekhov’s gun	Narzędzie fabuły istotne dla konkretnej fabuły
MacGuffin	Narzędzie fabuły o naturze niezwiązańej
Pomoc (ang. May help in quest)	Narzędzie fabuły istotne do rozwiązania konfliktu

W kontekście rozważań zawartych w niniejszej pracy szczególnego znaczenia nabierają wzory wynikowe oraz *plot twisty*. Wzory wynikowe można bowiem bezpośrednio przekształcić na pewną oczekiwanaą cechę, którą powinny charakteryzować się zadania fabularne – powinno dać się w nich wyszczególnić jasne ciągi przyczynowo-skutkowe. Przykładowo, bohater do uratowania krainy przed *Wielkim złym* potrzebuje uwolnić księżniczkę. Dowiaduje się od niej, że potrzebuje klucza (*narzędzie fabuły*), co powoduje, że odszukuje go i uwalnia księżniczkę. Akt poszukiwania klucza bezpośrednio *wynika* z chęci uratowania księżniczki.

Poza wzorami wynikowymi, część zadań powinna zawierać *plot twisty*. Plot twisty stanowią istotny element fabuły, albowiem są wzorami zmieniającymi jej przebieg. Można je zaobserwować, kiedy informacje które odbiorca fabuły arbitralnie przyjmował okazują się nieprawidłowe. Takie sytuacje można również interpretować jako posiadanie informacji o sprzecznym wartościowaniu. Kontynuując przykład dotyczący księżniczki: Księżniczka okazuje się być *Wielkim złym*, a poprzedni “złoczyńca” tak naprawdę ratował krainę przed nią. Teraz bohater musi powstrzymać księżniczkę (związek przyczynowo-skutkowy). Od modelu zadań fabularnych opracowanego w ramach tej pracy oczekiwana jest możliwość przedstawienia obu tych *meso-wzorów*.

1.2. Sposoby obcowania z fabułą

Generowana fabuła może być ściśle związana z rodzajem dzieła kultury, którego ma być częścią. Przykładowo, od historii detektywistycznej odbiorcy mogą oczekwać fabuły o większej zawartości *plot twistów* oraz *wzorów wynikowych*. Zaznaczyć prz tym należy, że gry video wpłynęły na prezentowane w nich historie, ponieważ już samo omawiane medium zakłada interakcyjność. Przedstawiana w grach historia nie musi być liniowa, a bohater może mieć możliwość podejmowania decyzji wpływających na dalszy los opowieści.

Gracz może pełnić różne role w konkretnych grach video. Role te są często charakterystyczne dla różnych gatunków gier i mają bezpośredni wpływ na cechy, jakie są oczekiwane od sposobu prezentacji określonych historii w tychże grach. Martens i Simmons wyróżnili możliwe do przyjęcia w grach role gracza oraz opisali cechy, jakie posiadają historie zawarte w tych grach. Zaproponowali oni następujący podział ról graczy [16]:

- gracz-jako-bohater (ang. player-as-protagonist);
- gracz-jako-współtwórca (ang. player-as-coauthor);
- gracz-jako-detektyw (ang. player-as-detective);
- gracz-jako-reżyser (ang. player-as-director).

Gracz-jako-bohater przedstawia najpopularniejszą rolę, w której gracz wciela się w konkretną postać i kieruje jej losem [16]. Większość gier RPG (Role Playing Games) traktuje tę rolę jako domyślną. Należy zwrócić uwagę, że ta rola zakłada grę jako bohater, co wcale nie wymusza grania pojedynczą postacią. Nie oznacza to również widoku gry z perspektywy pierwszej osoby, czyli gracz nie musi obserwować świata z kamery umieszczonej tak, aby odwzorowywała to, co widzi bohater. Istotnym natomiast jest ograniczenie wiedzy gracza, który powinien wiedzieć tylko to, co aktualnie wie jego bohater lub bohaterowie.

Kolejna rola gracza, czyli gracz-jako-współtwórca, zakłada możliwość kształtowania lub edycji fragmentów historii [16]. W samej treści gier jest to rzadko spotykana perspektywa, jest natomiast częściej obecna w narzędziach służących do tworzenia gier. Warto zwrócić uwagę, że tę rolę przyjmują także użytkownicy systemów o przemiennej inicjatywie (ang. *mixed-initiative*), służących do generacji fabuły pod nadzorem użytkownika. Oferują one często funkcjonalność podpowiadania dalszych możliwych elementów historii. Omawiane systemy można kwalifikować zarówno do generatorów fabuły *online* jak i *offline*. Z perspektywy użytkownika systemu obserwujemy generację w trakcie jego działania (*online*), jednakże odbiorca dzieła kultury, do którego może trafić wygenerowana i dopracowana fabuła, może nie wiedzieć o fakcie, że przedstawione w grze treści zostały wygenerowane (*offline*).

Nastecną możliwą rolą, w którą może wcielić się gracz, jest gracz-jako-detektyw, który z założenia ma przedstawione fragmenty przeszłej historii oraz jej skutków i analizując je musi wydedukować co się wydarzyło [16]. Ten przypadek bardzo przypomina perspektywę gracza-jako-bohatera, w której gracz obserwuje historię jako podmiot zbiorowy złożony z wielu bohaterów będącymi bohaterami wydarzeń, które są przedstawiane często w achronologicznej kolejności.

Ostatnią wymienioną rolą jest gracz-jako-reżyser. Przyjmując ją, gracz wciela się we wszechwiedzącego obserwatora sterującego losami wielu postaci jednocześnie [16]. Gry wykorzystujące tę formę prezentacji fabuły mają wiele cech symulacji, w których gracz wpływa na jej przebieg na różnych poziomach szczegółowości. Przykładem gry, w której gracz wciela się w “reżysera”, może być przytoczony we wstępie DF (*Dwarf Fortress*), gdzie grający może wydawać ogólne polecenia populacji krasnoludów, wpływając tylko pośrednio na historie indywidualnych jednostek. Uczestnik rozgrywki jest w stanie sprawdzać historie poszczególnych krasnoludów i ma wiedzę o całości świata, co rozróżnia rolę gracza-jako-reżysera od gracza-jako-bohatera z bardzo licznym podmiotem zbiorowym jakim byłaby całość populacji fortocy.

Omawiane w dalszej części pracy generatory zadań lub fabuły zostały przystosowane do generacji fragmentów historii przeznaczonych do odbioru z perspektywy gracza-jako-bohatera. Badany w rozdziale 3 model Dorana oraz zaimplementowany na potrzeby niniejszej pracy prototyp generatora jest zgodny z komplementarnym do tej roli sposobem prezentacji fabuły [8].

2. Przegląd sposobów generacji fabuły i zadań

Generacja fabuły powinna dobierać wykorzystywane techniki w zależności od oczekiwanej zastosowania. Techniki wykorzystywane do generacji *online*, czyli podczas użytkowania gry przez gracza, muszą działać szybko [17, 21]. To wymaganie jest uzasadnione niwelacją ryzyka potencjalnego zniechęcania do gry przez długie czasy oczekiwania. Skrócone czasy oczekiwania są wskazane, ale nie niezbędne w momencie, kiedy wygenerowanie treści następuje przed udostępnieniem gry do użytku – czyli w trakcie generacji *offline*. Zmniejszone ograniczenia czasowe pozwalają na stosowanie bardziej zaawansowanych algorytmów, których wynikowe artefakty mogą zostać zweryfikowane przez zespół produkujący grę.

W tym rozdziale zawarty został przegląd wybranych rozwiązań dotyczących generacji fabuły lub zadań fabularnych w celu porównania ich wad oraz zalet. Na potrzeby tej pracy analizowane rozwiązania zostały podzielone na cztery grupy ze względu na technologie stanowiące podstawy generatorów. Podział ten został zaczerpnięty z pracy J. Jonasson et al. [15], aczkolwiek grupa systemów opartych na gramatyce bezkontekstowej została dodatkowo podzielona na systemy wykorzystujące plannery oraz systemy wykorzystujące model zadania lub fabuły wyrażony w gramatyce bezkontekstowej. W ten sposób uzyskano następujący podział systemów:

- systemy oparte na rozwiązywaniu problemu planowania;
- systemy oparte na modelu fabuły lub zadania wyrażonym w gramatyce bezkontekstowej;
- systemy oparte na programowaniu logicznym;
- silniki do zarządzania całością historii.

2.1. Systemy oparte na planowaniu

Badacze narracyjnej teorii oraz badacze gier zauważali pewne podobieństwa pomiędzy badaniami związanymi ze sztuczną inteligencją, a narracyjnymi modelami służącymi między innymi do generowania fabuły [23]. Zdaniem wielu badaczy można doszukiwać się wspólnych cech tych dwóch dziedzin nauki w obszarze rozumowania o czynnoścach i planach. W ramach fabuły każda postać powinna dążyć do osiągnięcia własnych celów, czy inaczej mówiąc planów. Innymi słowy, naturalnym sposobem na generowanie fragmentów fabuły może okazać się rozwiązywanie problemu planowania.

2.1.1. Problem planowania

Problem planowania można zdefiniować jako próbę znalezienia sekwencji akcji, która przetransformuje wejściowy stan do stanu oczekiwanej na wyjściu [23]. Oznacza to, że do określenia konkretnego zagadnienia jako problemu planowania potrzeba [23]:

- stanu początkowego;
- stanu celu lub inaczej oczekiwanej (ang. goal state), określonego również jako opis celu (w skrócie GD – z ang. goal description) [12];
- zbioru możliwych do wykonania akcji.

W ramach badań dotyczących tej poddziedziny sztucznej inteligencji zostało w ciągu ostatnich 50 lat zaproponowanych wiele rozwiązań służących do rozwiązywania wspomnianego problemu. Narzędzia służące do planowania nazwano *plannerami* lub *solverami*. Kolejnym proponowanym przez twórców rozwiązaniom towarzyszyły języki pozwalające na wygodny zapis problemu w postaci, którą dany solver byłby w stanie obsłużyć.

Znaczna część z opracowywanych przez badaczy *plannerów* wykorzystywała te same teorie matematyczne w celu reprezentacji uprzednio wymienionych składowych problemu planowania. Stany początkowy, oczekiwany oraz pośrednie często przedstawiane są za pomocą rachunku predykatów pierwszego stopnia, natomiast akcje przedstawiane były za pomocą par składowej się ze [23]:

- zbioru warunków wstępnych (ang. *preconditions*);
- zbioru efektów, który czasem jest w literaturze określany jako warunki końcowe (ang. *postconditions*).

Jednym z bardziej znanych plannerów, był Stanford Research Institute Problem Solver (znany powszechnie pod skróconą nazwą jako STRIPS). STRIPS wyraża przestrzeń problemu za pomocą modelu początkowego świata, zbioru dozwolonych operatorów wraz z ich efektami na modele świata oraz wyrażenia końcowego [10]. Model świata składa się ze zbioru “*dobrze sformułowanych wyrażeń*” (w skrócie *wffs* – *well-formed formulas*), zgodnych z rachunkiem predykatów pierwszego rzędu. Operatory w STRIPSie składają się z dwóch części – ich wpływu na świat (efektów) oraz warunków wymaganych do zaaplikowania operatora. Efekty operatora to prosty lista *wffs*, które powinny zostać dodane do przedstawionego świata, a warunki wstępne operatora także mogą być w ten sposób wyrażone. Wyrażenie końcowe także jest zdefiniowane jako *wffs*, które plan musi spełnić [10].

Jednym z problemów, z jakimi musiały zmagać się badania poświęcone planowaniu, była niekompatybilność sposobu wyrażenia problemu pomiędzy kolejnymi narzędziami [11]. Dopiero pod koniec XX wieku podjęto udane próby opracowania zunifikowanego języka służącego do definiowania domeny dla problemów planowania. Dzięki wykształconemu w ten sposób językowi PDDL (Planning Domain Definition Language) następne badania mogły ponownie wykorzystywać opracowane na potrzeby poprzednich badań problemy w celu dokładniejszego porównywania rozwiązań [11]. Oczywiście nowo opracowywane rozwiązania nie muszą operować wewnętrznie na reprezentacji problemu wyrażonej w PDDL. Z reguły *plannery* przyjmują

problem opisany w PDDL i dopiero po transformacji do swojej wewnętrznej reprezentacji pracuję nad jego rozwiążaniem.

Język PDDL, podobnie jak STRIPS, pozwala na zdefiniowanie problemu, tak aby zawierał GD (opis celu), stan początkowy oraz domenę zawierającą między innymi zbiór możliwych do wykonania akcji [12]. Oznacza to, że w celu wyrażenia zadania generacji fabuły, jej fragmentu lub zadania fabularnego, niezbędnym jest również określenie tych trzech elementów.

2.1.2. Cechy systemów opartych na planowaniu na przykładzie systemu *CONAN*

Twórcy języka PDDL w pracy opisującej opracowany język wspominają, że sam język służy jedynie do określenia “fizyki” domeny, czyli predykatów, zbioru możliwych do wykonania akcji i ich warunków wymaganych oraz efektów na świat [12]. Powoduje to brak możliwości wyrażania “*podpowiedzi*” dla plannera rozwiązującego problem. Oznacza to, że choć plannery są w stanie przeszukiwać obszerne przestrzenie możliwych rozwiązań, to zdefiniowanie w zaproponowany sposób domeny problemu (jako możliwe do zaistnienia akcje oraz postaci i przedmioty w świecie) prowadzi do nieuchronnego problemu z otrzymywaniem “*wiarygodnych*” rozwiązań [4]. Wspomniana *wiarygodność* jest wraz z przewidywalną jakością rozwiązań jednym z głównych wymagań stawianym systemom do generacji zadań fabularnych [17, 21].

Jednym z generatorów zadań, które starają się zwiększać wiarygodność otrzymywanych narracyjnych struktur jest opracowany przez V. Breault et al. *Creation Of Novel Adventure Narrative*, czyli w skrócie CONAN [4]. CONAN jest systemem przeznaczonym do generacji fabuły *offline* i ma w założeniu dwóch użytkowników: *projektanta* gry korzystającego z systemu do generowania zadań, w które grałby drugi użytkownik tego systemu (*gracz*). CONAN przedstawia generację zadań zgodnie z przedstawionymi w podrozdziale 2.1.1 założeniami, czyli jako problem planowania w postaci stanu początkowego, opisu celu oraz zbioru możliwych do wykorzystania akcji.

System CONAN wykorzystuje dostosowany na potrzeby generatora zbiór możliwych do wykonania akcji, wytypowany na potrzeby pracy Dorana i Parberrego. Zbiór został szczegółowo opisany w rozdziale 3, jednakże w celu zrozumienia działania systemu CONAN czytelnikowi wystarczy wiedza, że zbiór ten zawiera kilkanaście akcji składających się z *preconditions* oraz efektów wywoływanych na świat. Akcje te są następujące: *DoNothing*, *Capture*, *Damage*, *Defend*, *Escort*, *Exchange*, *Experiment*, *Explore*, *Gather*, *Give*, *GoTo*, *Kill*, *Listen*, *Read*, *Repair*, *Report*, *Spy*, *Stealth*, *Take* oraz *Use* [4]. Wynikiem działania systemu jest sekwencja tych akcji, które bohater gry powinien wykonać w celu ukończenia zadania.

Reprezentacja świata przyjęta w systemie CONAN wyróżnia następujące elementy świata: lokacje (miejsc), agentów (postaci) oraz przedmioty. Bohater może przemierzać lokacje, w których znajdują się przedmioty i postaci. Bohater może wchodzić w interakcję z postaciami znajdującymi się w tej samej lokacji co on.

System CONAN zakłada, że zadania są zlecone przez postaci ze świata. Czynnikiem powodującym, że zadania generowane przez system starają się nie zaburzać wiarygodności świata, jest przypisanie poszczególnym postaciom osobowości, które wpływają na szczegóły zlecanych zadań. W opublikowanej pracy autorzy zdefiniowali osobowości jako zbiór par akcji wraz z informacją, czy dana postać jest mniej lub bardziej chętna do zlecenia wykonania danej czynności. Przykładowo dla świata znanego z Aladyna charakterysty postaci można zdefiniować

Tabela 2.1: Preferencje bohaterów świata Aladyna

Postać	Preferencje reprezentowane jako koszt
Aladdin	+kill, -exchange, -use, +escort
Dragon	-damage, -take, -report, +escort, -defend
Genie	-kill, -exchange, -defend, -read
Jasmine	+kill, -spy, -take, -stealth
Jafar	-kill, -spy, -take, -stealth, -move

Tabela 2.2: Przykładowe predykaty dostępne w systemie CONAN [4]

predykat	opis
has?cl?i	postać w lokacji c1 posiada informację i
has?p?o	gracz posiada przedmiot o
has?c?o	postać c posiada przedmiot o
at?c?l	postać c jest w lokacji l
captive?p?c	bohater uwieził postać c
dead?c	postać c jest martwa
explored?l	bohater zwiedził lokację l

tak, jak w tabeli 2.1, gdzie prefix – oznacza akcje mające mniejszy koszt, czyli takie które są bardziej pożądane, a prefix + występuje przy akcjach mniej pożądane zdaniem danej postaci. Akcje niewymienione mają bazowy koszt.

Opis celu (warunki ukończenia zadania) w systemie *CONAN* przyjmuje formę zbioru predykatów, które muszą być spełnione na końcu wynikowego planu, czyli zadania. Na potrzeby systemu przyjęto predykaty pozwalające wyrazić obecność danej postaci w danym miejscu, posiadanie danego przedmiotu przez daną postać, zranienie lub zabicie danej postaci itd. Przykładowe predykaty występujące w systemie *CONAN* można zobaczyć w tabeli 2.2.

System CONAN generuje warunki ukończenia zadania uwzględniając preferencje zleceniodawcy wyrażone poprzez jego osobowość (patrz przykład z tabeli 2.1). Dzięki temu system stara się zapewnić “zwiększenie realistyczności i wiarygodności bohaterów” [4]. Sama zawartość opisów celu generowana jest za pomocą następującego algorytmu:

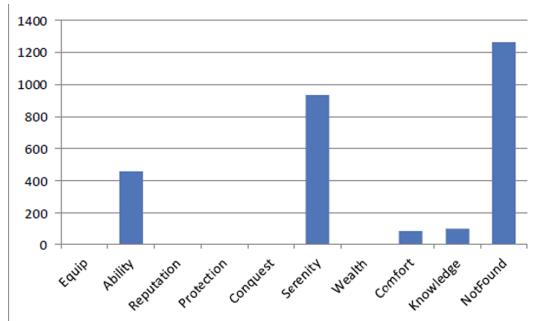
1. Losowy wybór wskazanej przez użytkownika-projektanta liczby predykatów.
2. Przypisanie predykatom wartościowania (np. zastąpienie predykatu has?c?o poprzez has?Aladyn?miecz).
3. Wyznaczenie planów realizujących wybrane predykaty.
4. Obliczenie średniej wagi akcji w planach zgodnie z preferencjami zleceniodawcy (zgodnie z wyjściową domeną, np. z tabelą 2.1).
5. Powtórzenie procesu wskazaną przez użytkownika-projektanta liczbę razy i wybór zbioru predykatów, którego średnia waga planu jest najmniejsza.

W ramach badań poświęconych systemowi *CONAN* autorzy pracy podłączyli system do symulacji, w której użytkownik-gracz wykonywał ruch na zmianę z systemem. Gracz wskazywał akcję którą wykonywał, po czym system generował plan mający na celu doprowadzić zadanie do końca. W ten sposób system przyjmował kolejne stany świata i generował dla nich plany. Sam fakt możliwości przeprowadzenia takiej symulacji gry uwidacznia pewną cechę rozwiązań służących do proceduralnej generacji fabuły lub zadań, opierających się na problemie planowania – systemy je wykorzystujące mogą w łatwy sposób reagować na akcje użytkownika. Powoduje to, że ich wykorzystanie może być skuteczne także do “*online*” PCG.

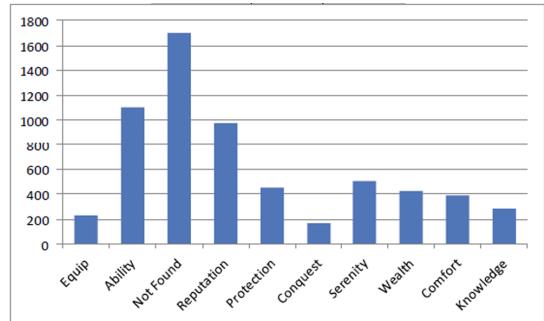
Oczywiście systemy do generacji zadań wykorzystujące problem planowania mogą ograniczać się do generacji *offline*. Wtedy za treść zadania można przyjmować tylko plan wygenerowany przed rozpoczęciem zadania. Tak interpretowane zadania autorzy systemu *CONAN* podają jako przykłady wyników działania ich systemu [4]:

1. (stealth you) (move you village bakery) (take you guard sword village) (giveto you guard sword village) (take you guard sword village) (move you field village) (damage you wheat field sword);
2. (stealth you) (move you village bakery) (take you guard sword village) (experiment you sword) (move you castle village) (spy you king castle secret);
3. (explore you bakery forge) (escort you blacksmith forge castle) (listen you king castle secret) (report you blacksmith secret castle) (getfromlocation you castle spellbook) (escort you blacksmith castle village) (exchange you guard sword spellbook village) (escort you blacksmith village field) (kill you slime gel field sword);
4. (stealth you) (move you cave bakery) (take you troll hammer cave) (use you hammer)
5. (stealth you) (move you cave bakery) (take you troll hammer cave) (move you field cave) (kill you slime gel field hammer) (defend you gel field);
6. (move you castle bakery) (listen you king castle secret) (move you cave castle) (stealth you) (take you troll hammer cave) (move you village cave) (take you guard sword village) (giveto you guard hammer village) (escort you guard village field) (kill you guard hammer field sword);
7. (move you field bakery) (defend you wheat field);
8. (stealth you) (move you cave bakery) (take you troll hammer cave) (move you forest cave) (kill you wolf pelt forest hammer) (move you castle forest) (giveto you king hammer castle);
9. (stealth you) (move you village bakery) (capture you guard village);
10. (stealth you) (explore you bakery village) (take you guard sword village) (explore you village cave).

Po analizie przykładowych zadań można stwierdzić, że system *CONAN* jest w stanie generować nieskomplikowane zadania o zróżnicowanym stopniu wiarygodności. Jednakże wśród



Rysunek 2.1: Dystrybucja motywacji zadań wygenerowanych dla **małego** świata [4]



Rysunek 2.2: Dystrybucja motywacji zadań wygenerowanych dla **dużego** świata [4]

przedstawionych zadań można zauważać pojedyncze akcje, które mogą wydawać się w realiach stereotypowego fantazy dość irracjonalne. Przykładowo w zadaniu 3. gracz ma wymienić miecz strażnika w zamian za księgi czarów, czyli wymienia przedmioty o nieporównywalnej wartości. Podobne problemy można zaobserwować analizując niektóre sekwencje wykonywanych czynności. W zadaniu pierwszym gracz prawdopodobnie okrada strażnika tylko po to, aby oddać mu skradziony przedmiot. Ukaże to brak lub ubogość uwzględniania związków przyczynowo-skutkowych. Wygenerowane zadania są logicznie poprawne i wykonywalne w świecie gry, ale mimo to potrafią burzyć oczekiwana wiarygodność świata gry.

Dodatkowym problemem, który można zaobserwować analizując przykładowe zadania może być zbytnia trudność ustalenia celu zadania lub motywacji zleceniodawcy. Problematyka znalezienia tych motywacji została poruszona przez twórców systemu [4]. Twórcy opracowali klasyfikator motywacji zleceniodawcy przyjmujący na wejściu treść zadania. Wyróżniono 9 motywacji i stwierdzono, że generując zadania z wykorzystaniem systemu *CONAN* dla domeny ze światem o małych rozmiarach około połowy zadań nie da się przyporządkować do żadnej z nich (patrz rys. 2.1). System sprawdza się natomiast lepiej w generacji zadań dla światów o większym rozmiarze, ale wciąż najliczniejszą grupę zadań stanowią te, których zleceniodawcom nie da się przypisać motywacji (patrz rys. 2.2).

Dodatkowym aspektem który należy poruszyć analizując systemy oparte na problemie planowania jest spójność nie tylko w obrębie danego zadania, ale także w ujęciu całościowym, tj. kolejnych generowanych zadań. Systemy tego typu są krytykowane w literaturze za nieumiejętność radzenia sobie samodzielnie z przedstawionym zagadnieniem [15]. Pomimo to twórcy systemu *CONAN* wierzą, że generowanie zadań przy wykorzystaniu stanu świata na końcu poprzedniego zadania jako stanu początkowego przy generacji kolejnego zadania jest w stanie stanowić konkurencyjną alternatywę dla systemów biorących pod uwagę całość fabuły, jednakże zagadnienie to wymaga dalszych badań [4].

Jak widać na przykładzie systemu *CONAN*, wykorzystanie problemu planowania w celu generacji zadań:

- Utrudnia zachowanie związków przyczynowo skutkowych w ramach zadań. Zadania generowane przez system *CONAN* zawierają sekwencje akcji, których cel jest trudny do wywnioskowania.

- **Gwarantuje wykonywalność generowanych zadań i możliwość wykonania poszczególnych czynności w ramach zadania.**
- Pozwala na łatwą zmianę treści zadania w czasie trwania gry (*online PCG*), jeżeli gracz wykonałby czynność sprzeczną z oryginalnym planem (oczywiście pod warunkiem, że istnieje alternatywna sekwencja czynności pozwalająca na spełnienie opisu celu zadania).
- Samo z siebie nie jest w stanie brać pod uwagę innych zadań. Mimo to istnieje możliwość symulacji takiego zachowania, jeżeli zadania będą generowane kolejno i będą przyjmowały jako stan początkowy stan końcowy poprzednio wygenerowanego zadania.

2.2. Systemy oparte na modelu fabuły lub modelu zadania wyrażonym w gramatyce bezkontekstowej

Jak wcześniej zostało wspomniane, oryginalny podział systemów zaproponowany przez Jonasson et al. [15], traktował jako jedną grupę systemy wykorzystujące plannery oraz omawiane w tym podrozdziale systemy. Wiąże się to z faktem, że plannery wykorzystują język PDDL, który z kolei wykorzystuje gramatykę bezkontekstową [12]. W ramach niniejszej pracy zauważono, że omawiana grupa nie jest jednorodna i można dokonać jej dalszego podziału z uwagi na fundamentalne różnice w sposobie generowania zadań lub fabuły. Ponieważ te różnice przekładają się na rozbieżność cech charakteryzujących systemy do generacji fabuły podjęto decyzję o wyróżnieniu systemów opartych na modelu zadania.

2.2.1. Gramatyka bezkontekstowa

Gramatyka bezkontekstowa jest sformalizowaną notacją wyrażającą odpowiadający język bezkontekstowy [13]. Systemy generujące zadania korzystające z wspomnianych gramatyk interpretują zadania, jako poprawne zdania zaczerpnięte z języka definiowanego przez gramatykę.

Na gramatykę bezkontekstową $G = (V, \Sigma, R, S)$ składają się [18]:

- V – skończony zbiór wyrażeń nieterminalnych (zmiennych);
- Σ – skończony zbiór wyrażeń terminalnych, rozłączny z V ;
- R – zbiór reguł będący zmienną oraz wyrażeniem, które można za nią podstawić. To wyrażenie może składać się ze zmiennych i wyrażeń terminalnych;
- $S \in V$ – wyrażenie początkowe.

Zdaniem poprawnym w języku definiowanym przez daną gramatykę jest sekwencja symboli terminalnych Σ , którą da się otrzymać zaczynając od wyrażenia początkowego S i dokonując kolejnych przekształceń znajdujących się w nim wyrażeń nieterminalnych V , zgodnie ze zbiorem reguł R .

2.2.2. Charakterystyka systemów

Omawiana grupa systemów zawiera generatory, które wykorzystują model zadania wyrażony jako gramatyka bezkontekstowa, np. w notacji Backusa-Naura (opisanej szerzej w rozdziale 3.1). Modele te definiują zadania jako sekwencje akcji [8], które powinny zostać wykonane przez bohatera. Zastosowanie gramatyki bezkontekstowej pozwala na opracowanie złożonych akcji, które mogłyby składać się z innych złożonych lub prostych akcji. Przykładowo można zdefiniować zadanie jako sekwencję “<obroń>, porozmawiaj”, a akcję <obroń> jako “idź, zabij”. Umożliwia to otrzymanie zadania “idź, zabij, porozmawiaj”.

Przedstawiony sposób opracowywania modelu zadania ma jednak pewną wadę – ostatecznie po uproszczeniu przedstawia tylko jedną postać dla zadania, co nie jest oczekiwany rezultatem. Aby tego uniknąć należy wprowadzić do modelu akcje, które można rozwinać na różne sposoby. Innymi słowy, w modelu powinny się znaleźć akcje będące alternatywami rozłącznymi różnych sekwencji akcji. Tak otrzymany model zadania, nawet w prostym przypadku losowego wyboru rozwinięcia złożonej akcji, pozwala na generację różnorodnych zadań [8]. Aby to przedstawić, tak samo jak w poprzednim przykładzie, zdefiniujmy zadanie jako sekwencję “<obroń>, porozmawiaj”, ale tym razem akcję <obroń> przedstawmy jako (“idź, zabij” albo “idź, wystrasz”). Dzięki temu zadanie jest w stanie przyjąć dwie postaci: “idź, zabij, porozmawiaj” lub “idź, wystrasz, porozmawiaj”. Pozwala to na znaczne poszerzenie przestrzeni możliwych do zamodelowania zadań.

Przytoczony jako przykład model zadania reprezentuje skończoną przestrzeń zadań, ale modele opracowywane na potrzeby prawdziwych generatorów zadań nie powinny takie być. Dodanie do modelu akcji, które byłyby rekurencyjne (odnoszące się do samych siebie), lub wzajemnie powiązanych akcji, pozwala na uzyskanie złożonych zadań, które nie są ograniczone rozmiarami. Przykładowo, jeżeli akcja “<obroń>” przyjmie postać (“idź, zabij, <obroń>” albo “idź, zabij”), to poprawnym zadaniem jest dowolna ilość par akcji “idź, zabij” zakończona akcją “porozmawiaj”.

Generatory zadań oparte o gramatyki bezkontekstowe mogą pracować poprzez wyjście od głównej reguły swojego modelu, czyli definicji zadania. Następnie rozwijają kolejne złożone akcje, aż do momentu w którym wynikowe zadanie składa się z samych akcji prostych. Różnorodność zadań wynika z występowania w modelu akcji zdefiniowanych jako alternatywy rozłączne innych sekwencji akcji. Podczas próby rozwinięcia takich akcji generator musi wybrać jedną z alternatywnych sekwencji. Decyzja ta mogłaby być podejmowana losowo, w oparciu o kontekst domeny lub stan początkowy świata, który byłby wejściem do generacji zadania.

Wspomniany algorytm *rozwijania* modelu w celu otrzymania zadania ma pewną wadę – sam z siebie nie gwarantuje wykonywalności zadania w kontekście świata gry. W wyniku pracy generatora, który nie uwzględniałby stanu początkowego, można otrzymać tylko szablon struktury zadania. Przytoczone poprzednio przykłady dobrze pokazują tę zależność. Przyjmijmy zadanie “idź, zabij, idź, zabij, porozmawiaj” oraz założmy, że nie da się zabić tej samej postaci dwukrotnie. Innymi słowy, wprowadźmy wymaganie wstępne na akcję “zabij”. Łatwo zauważyc, że istnieje wartościowanie w którym akcja “zabij” wykonywana jest dwukrotnie na tym samym aktorze. Co więcej, tak skonsttuowane zadanie nie daje gwarancji, że istnieje poprawne wartościowanie w danym świecie gry. Aby rozwiązać ten problem generatory stosujące opisany mechanizm do generowania zadań muszą same zadbać o zaspokojenie wymagań

wstępnych akcji. Odróżnia je to od generatorów opartych na planowaniu, gdzie to planner gwarantuje ich zaspokojenie.

Pomimo opisanej wyżej wady omawianego rozwiązania, można zauważać także jego pewną przewagę nad systemami wykorzystującymi plannery. Podczas analizy przykładowych wyników działania takich systemów (patrz podrozdział 2.1.2) można było zauważać trudność ze wskazaniem celu zadania, czy motywacji zleceniodawcy. Generatory oparte na modelu zadania nie mają tego problemu, gdyż można opracować ten model w taki sposób, aby struktura zadania jasno odzwierciedlała motywacje zleceniodawcy, dając przy tym poczucie intencjonalności zadania. Tak też zrobiono w systemie opracowanym przez Dorana i Parberrego [8], który został poddany analizie w dalszej części pracy.

2.3. Systemy oparte na programowaniu logicznym

W pracach Martensa et al. wykorzystanie *Answer Set Programming* (w skrócie ASP) zostało wskazane jako możliwe do zastosowania w celu generacji fragmentów fabuły, jednakże dodatkowe badania są potrzebne w celu wskazania strategii do łączenia ze sobą świata gry oraz modelów opracowanych przez tę grupę systemów [6, 15]. Obecnie jednak, z uwagi na zbyt małą ilość przeprowadzonych badań, nie można jednoznacznie wskazać zalet oraz wad tych systemów. Jednakże istniejące już systemy wydają się obiecujące [15] ze względu na to, że potrafią przypisywać określone role postaciom i nakładać odpowiednie ograniczenia na te role, zwiększaając wiarygodność wynikowej fabuły [5].

2.4. Silniki do zarządzania całością historii

Wielkie systemy zarządzające całością historii cechują się znacznie większą złożonością od poprzednich grup. Często przeznaczone są do generacji *online* i są bardzo spersonalizowane przez co bardzo się od siebie różnią. Dodatkowo często są odpowiedzialne nie tylko za generację fabuły, czy wydarzeń mających miejsce, ale także za generację świata [15].

Tę grupę systemów można podzielić je ze względu na centralizację procesu generowania kolejnych elementów fabuły na systemy silnie autonomiczne oraz na systemy scentralizowane [15]. W pierwszych z nich pojedyncze postaci podejmują decyzje odnośnie kolejnych wykonywanych czynności niezależnie [15]. W systemach silnie scentralizowanych decyzje dotyczące kolejnych akcji podejmowane są przez generatory pełniące rolę “*Drama managerów*” – zarządców fabuły [15].

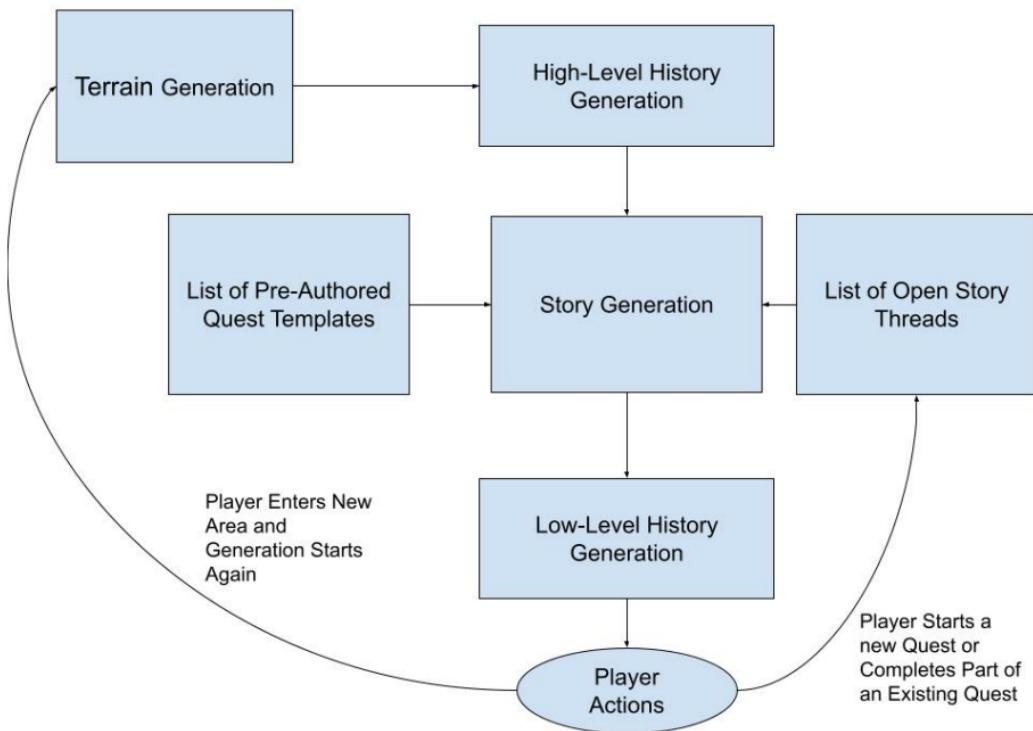
W celu przedstawienia różnic jakie mogą występować pomiędzy silnikami, a systemami omówionymi w ramach podrozdziałów 2.1 i 2.2, zostanie przedstawiony projekt scentralizowanego silnika fabularnego opracowanego przez Martensa et al. Na rysunku 2.3 widać wysokopoziomowy projekt wspomnianego systemu, opracowanego na potrzeby gry mobilnej *Lance a Lot*. Silnik ten “*jest odpowiedzialny za definiowanie fizycznych własności oraz elementów zawartych w każdej lokacji. Dodatkowo generuje historyczne informacje o świecie gry i zapewnia go postaciami.*” [15]. Z uwagi na pełnenie różnych funkcji silnik został podzielony na ścisłe ze sobą współpracujące generatorы: terenu, fabuły wysokiego poziomu, narracji, fabuły niskiego poziomu. Ze względu na tematykę niniejszej pracy zostaną pominięte dwa pierwsze

generatory, których funkcją jest generacja świata.

Generator narracji przyporządkowuje lokacjom wątki fabularne, które mogą być tworzone na nowo na podstawie autoryzowanych szablonów (patrz rys. 2.3) lub wybierane z puli już otwartych wątków (patrz rys. 2.3). Każdy z tych szablonów musi posiadać warunki wstępne, które muszą być spełnione w bieżącym stanie gry [15]. Wątki te reprezentują zadania fabularne, jakie gracz będzie wykonywał, a generator narracji koordynuje je między sobą. Związane jest to z istotną cechą, jaką charakteryzuje się silnik Martensa, w postaci wzajemnego oddziaływania generacji świata i fabuły. W efekcie oznacza to, że generator narracji wpływa nie tylko na sama fabułę, ale i na stan świata.

Zaprezentowany przykładowy silnik Martensa uwidacznia charakterystyczne cechy silników różniące je od generatorów opartych na problemie planowania oraz modelach wyrażonych w gramatyce bezkontekstowej. Są to:

- Uwzględnianie pozostałych zadań podczas otwierania kolejnych wątków fabularnych (generacji kolejnych zadań).
- Ścisłe, wzajemne powiązanie generacji świata i fabuły.



Rysunek 2.3: Projekt silnika fabularnego Martensa et al. opracowanego na potrzeby gry mobilnej *Lance a Lot* [15]

3. Analiza i rozszerzenie modelu zlinearyzowanych zadań fabularnych Dorana i Parberrego

Doran i Parberry zaproponowali model zadania fabularnego wraz z generatorem zadań działającym zgodnie z ideą przedstawioną w podrozdziale 2.2 [8]. W tym rozdziale została dokonana jego analiza. Na podstawie tej analizy dokonano ekstrakcji wad modelu oraz dokonao próby ich naprawienia poprzez opracowanie rozszerzonego modelu. Model ten zostanie wykorzystany w rozdziale 4 do opracowania prototypu generatora zadań.

3.1. Notacja Backusa-Naura

Notacja Backusa-Naura jest metodą zapisu gramatyk bezkontekstowych (patrz rozdział 2.2.1), na której wzorowali się Doran i Parberry przedstawiając opracowany przez nich model zadań. Gramatyka wyrażona w tej notacji składa się ze zbioru reguł w postaci:

$\langle \text{wyrażenie nieterminalne} \rangle ::= \text{wyrażenie}$

Wyrażenie nieterminalne składa się z sekwencji terminalnych lub nieterminalnych wyrażeń. Sekwencje te można oddzielać od siebie za pomocą operatora alternatywy $|$. Wyrażenia terminalne można odróżnić od nieterminalnych poprzez znaki je okalające:

- wokół symboli nieterminalnych znajdują się nawiasy: $\langle \rangle$
- wokół symboli terminalnych znajdują się apostrofy: ”

Aby lepiej zrozumieć gramatykę wyrażoną w BNF przyjrzyjmy się definicji liczby naturalnej (przyjmując, że $0 \notin N$).

```
 $\langle \text{cyfra dodatnia} \rangle ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'$ 
 $\langle \text{cyfra} \rangle ::= '0' | \langle \text{cyfra dodatnia} \rangle$ 
 $\langle \text{ciąg cyfr} \rangle ::= \langle \text{cyfra} \rangle \langle \text{ciąg cyfr} \rangle | ''$ 
 $\langle \text{liczba naturalna} \rangle ::= \langle \text{cyfra dodatnia} \rangle \langle \text{ciąg cyfr} \rangle$ 
```

W powyższej gramatyce cyfrą dodatnią jest dowolna z cyfr 1-9. Sama *cyfra* została natomiast zdefiniowana jako 0, albo *cyfra dodatnia*. *Ciąg cyfr* jest cyfrą i *ciągiem cyfr*, albo niczym (reprezentowanym jako znak pusty ”). W ten sposób dzięki zastosowaniu rekurencji (odwoływanie się do samego siebie) można wyrazić ciąg cyfr o dowolnej długości (w tym ciąg

o długości 0). Ostatecznie przy jego pomocy można zdefiniować liczbę naturalną jako cyfrę dodatnią wraz z ciągiem cyfr o dowolnej długości.

Niektóre źródła [20] dopuszczają zastosowanie w BNF dodatkowych operatorów + i * oraz nawiasów np. {}. Oba operatory pozwalają wyrazić sekwencję powtórzeń poprzedzającego je wyrażenia. Operator * opisuje sekwencję o dowolnej długości, a operator + wymaga, aby sekwencja miała co najmniej jeden element. Z pomocą tych operatorów liczbę naturalną można wyrazić jako cyfrę dodatnią i dowolny ciąg cyfr (w tym ciąg pusty):

$$\langle \text{liczba naturalna} \rangle ::= \langle \text{cyfra dodatnia} \rangle \langle \text{cyfra} \rangle^*$$

3.2. Model zadań Dorana

Doran i Parberry dokonali analizy strukturalnej prawie 3000 zadań pochodzących z czterech popularnych gier MMORPG (Eve Online, World of Warcraft, Everquest, Vanguard: Saga of Heroes) i zauważyli powtarzające się elementy zadań fabularnych [8]. Badacze wyróżnili:

- 20 akcji atomowych, które mogą wykonywać gracze (patrz tabela 3.1);
- 8 akcji złożonych, które są sekwencją akcji złożonych lub atomowych (patrz tabela 3.2);
- 9 głównych motywacji jakimi mogą kierować się NPC (postaci nie będące graczami), które zawarto w tabeli 3.3;
- 39 strategii wykonania zadania (patrz tabela 3.4).

Na podstawie wyróżnionych elementów zadań badacze opracowali model zadania w postaci gramatyki bezkontekstowej i zapisali go z wykorzystaniem notacji przypominającej notację Backusa-Naura. Notacja przyjęta przez badaczy różni się tym, że kolejne redefinicje wyrażeń nieterminalnych traktowane są jako kolejne alternatywne sekwencje wyrażeń. Dodatkowo badacze postanowili nie oznaczać wyrażeń terminalnych znakami "oraz wykorzystali znak ε do oznaczenia pustego wyrażenia. W dodatku A zawarto model Dorana wyrażony zgodnie z Notacją Backusa-Naura.

W modelu Dorana przyjęto, że akcje atomowe wyrażają niepodzielne, proste czynności (np. idź lub wysłuchaj), które bohater musi wykonać, aby pomyślnie zakończyć zadanie. Są one **jedynymi** wyrażeniami terminalnymi występującymi w modelu. Listę wszystkich akcji atomowych można zobaczyć w tabeli 3.1. W niniejszej pracy postanowiono używać ich oryginalnych nazw pochodzących z pracy Dorana.

W przeciwnieństwie do akcji atomowych, akcje złożone reprezentują akcje składające się z kilku prostszych czynności. Powoduje to, że w gramatyce są one wyrażeniami nieterminalnymi, w konsekwencji czego mogą składać się z sekwencji innych akcji. Listę wszystkich akcji złożonych wyrażonych w notacji badaczy zawarto w tabeli 3.2. W niniejszej pracy postanowiono używać ich oryginalnych nazw pochodzących z pracy Dorana.

Motywacje i odpowiadające im strategie zostały zdefiniowane w sposób analogiczny do akcji złożonych. Pełnią one rolę głównego szablonu zadania nadającego mu formę i wrażenie intencjonalności. Lista wszystkich strategii wyrażonych w notacji badaczy znajduje się w tabeli

Tabela 3.1: Atomowe akcje w modelu Dorana [8]

	Akcja	Warunki wstępne	Efekty
1	ε	Brak	Brak
2	capture	Ktoś tu jest	On jest twoim więźniem
3	damage	Ktoś lub coś tu jest	On/to jest bardziej uszkodzone
4	defend	Ktoś lub coś tu jest	On/to nie jest bardziej uszkodzone
5	escort	Ktoś tu jest	On będzie ci towarzyszył
6	exchange	Ktoś tu jest, ma coś i ty masz coś	On ma coś twojego i ty masz jego
7	experiment	Coś tu jest	Być może wiesz do czego to służy
8	explore	Brak	Losowo przemierzasz okolice
9	gather	Coś tu jest	Masz to
10	give	Ktoś tu jest, masz coś	On to ma, a ty nie
11	goto	Wiesz gdzie iść i wiesz jak	Jesteś tam
12	kill	Ktoś tu jest	On jest martwy
13	listen	Ktoś tu jest	Masz ich informacje
14	read	Coś tu jest	Masz informacje z tego
15	repair	Coś tu jest	To jest mniej uszkodzone
16	report	Ktoś tu jest	Ma informacje, które posiadasz
17	spy	Ktoś lub coś tu jest	Masz informacje dotyczące tego/niego
18	stealth	Ktoś tu jest	Skradaj się za nim
19	take	Ktoś tu jest i coś ma.	Masz to, a on nie
20	use	Coś tu jest	To zadziałało na otoczenie

3.4. W niniejszej pracy postanowiono używać ich oryginalnych nazw pochodzących z pracy Dorana.

Ostatnim, najważniejszym elementem modelu, spajającym ze sobą wszystkie motywacje jest sama definicja zadania. Pełni ona rolę wyrażenia początkowego gramatyki bezkontekstowej (patrz rozdział 2.2.1). Zadanie zostało zdefiniowane jako alternatywa rozłączna motywacji. W notacji Backusa-Naura wygląda ona następująco:

```
<QUEST> ::= <Knowledge> | <Comfort> | <Reputation>
          | <Serenity> | <Protection> | <Conquest>
          | <Wealth> | <Ability> | <Equipment>
```

Akcje atomowe

Akcje atomowe modelu Dorana (patrz tabela 3.1) są wyrażeniami terminalnymi, posiadającymi swoją interpretację w kontekście zadania fabularnego. W przypadku większości akcji pokrywa się ona ze znaczeniem ich nazwy, aczkolwiek nie zawsze. Spowodowało to, że każda z nich posiada przyporządkowane warunki końcowe, które definiują efekty akcji na świat w sposób analogiczny do warunków końcowych akcji definiowanych na potrzeby generatorów rozwiązujących problem planowania (patrz rozdziały 2.1.1 i 2.1.2). Podobnie każda akcja może posiadać zbiór warunków wstępnych wymaganych do jej wykonania.

Tabela 3.2: Akcje złożone modelu Dorana [8]

	Reguła	Wyjaśnienie
1	<subquest> ::= <goto>	Po prostu idź gdzieś
2	<subquest> ::= <goto> <QUEST> goto	Idź wykonać zadanie i wróć
3	<goto> ::= ε	Już tu jesteś
4	<goto> ::= explore	Tułaj się
5	<goto> ::= <learn> goto	Dowiedz się gdzie iść i tam idź
6	<learn> ::= ε	Już to wiesz
7	<learn> ::= <goto> <subquest> listen	Idź gdzieś, wykonaj podzadanie w zamian za informacje
8	<learn> ::= <goto> <get> read	Idź gdzieś, zdobądź coś i dowiedz się czegoś
9	<learn> ::= <get> <subquest> give listen	Zdobądź coś, wykonaj podzadanie, i wymień to za informacje
10	<get> ::= ε	Już to masz
11	<get> ::= <steal>	Ukradnij to
12	<get> ::= <goto> gather	Idź gdzieś i podnieś coś co leży
13	<get> ::= <goto> <get> <subquest> <goto> exchange	Idź gdzieś, zdobądź coś, wykonaj podzadanie i wymień
14	<steal> ::= <goto> stealth take	Idź gdzieś, zakradnij się do kogoś i weź coś
15	<steal> ::= <goto> <kill> take	Idź gdzieś, zabij kogoś i zabierz coś
16	<spy> ::= <goto> spy <goto> report	Idź gdzieś, szpieguj kogoś wróć i raportuj
17	<capture> ::= <get> <goto> capture	Weź coś, idź gdzieś i użyj tego aby kogoś porwać
18	<kill> ::= <goto> kill	Idź gdzieś i zabij kogoś

Tabela 3.3: Motywacje modelu Dorana [8]

Motywacja	Opis
Knowledge (pl. wiedza)	Wiedza znana postaci
Comfort (pl. komfort)	Fizyczny komfort
Reputation (pl. reputacja)	Jak inni postrzegają postać
Serenity (pl. spokój)	Spokój ducha
Protection (pl. ochrona)	Ochrona przed zagrożeniami
Conquest (pl. podbój)	Żądza pokonania przeciwnika
Wealth (pl. wealth)	Siła gospodarcza
Ability (pl. umiejętność)	Umiejętności postaci (ang. Skills)
Equipment (pl. wyposażenie)	Używalne przedmioty

Tabela 3.4: Strategie dla motywacji w modelu zadań Dorana [8]

Motywacja	Strategia	Sekwencja akcji
Knowledge	Deliver item for study	<get> <goto> give
	Spy	<spy>
	Interview NPC	<goto> listen <goto> report
	Use an item in the field	<get> <goto> use <goto> <give>
Comfort	Obtain luxuries	<get> <goto> <give>
	Kill pests	<goto> damage <goto> report
Reputation	Obtain rare items	<get> <goto> <give>
	Kill enemies	<goto> <kill> <goto> report
	Visit a dangerous place	<goto> <goto> report
Serenity	Revenge, Justice	<goto> damage
	Capture Criminal(1)	<get> <goto> use <goto> <give>
	Capture Criminal(2)	<get> <goto> use capture <goto> <give>
	Check on NPC(1)	<goto> listen <goto> report
	Check on NPC(2)	<goto> take <goto> give
	Recover lost/stolen item	<get> <goto> <give>
	Rescue captured NPC	<goto> damage escort <goto> report
	Attack threatening entities	<goto> damage <goto> report
Protection	Treat or repair (1)	<get> <goto> use
	Treat or repair (2)	<goto> repair
	Create Diversion	<get> <goto> use
	Create Diversion	<goto> damage
	Assemble fortification	<goto> repair
	Guard Entity	<goto> defend
Conquest	Attack enemy	<goto> damage
	Steal stuff	<goto> <steal> <goto> give
Wealth	Gather raw materials	<goto> <get>
	Steal valuables for resale	<goto> <steal>
	Make valuables for resale	repair
Ability	Assemble tool for new skill	repair use
	Obtain training materials	<get> use
	Use existing tools	use
	Practice combat	damage
	Practice skill	use
	Research a skill(1)	<get> use
	Research a skill(2)	<get> experiment
Equipment	Assemble	repair
	Deliver supplies	<get> <goto> <give>
	Steal supplies	<steal>
	Trade for supplies	<goto> exchange

Warto zwrócić uwagę na fakt, że opracowany zbiór akcji wymusza istnienie pewnych elementów lub zależności w modelu świata gry. Analizując warunki wstępne i końcowe można wyróżnić cztery podstawowe byty, które definiują stan świata gry:

1. bohater;

2. postaci;

3. przedmioty;

4. lokacje;

oraz ich atrybuty i relacje między nimi

1. posiadanie – postaci mogą posiadać przedmioty;

2. przebywanie – postaci i przedmioty przebywają w lokacjach;

3. pojmanie – postaci mogą być pojmane przez inne postaci lub bohatera;

4. wiedza – postaci i bohater mogą posiadać informacje na temat innych postaci oraz ich miejsca przebywania;

5. towarzyszenie – postaci mogą towarzyszyć bohaterowi;

6. bycie martwym;

7. bycie uszkodzonym.

Zdefiniowane warunki wstępne analizowanego zbioru akcji są mało precyzyjne i opisują jedynie to co musi być spełnione, aby dało się logicznie wykonać daną akcję. W większości przypadków ogranicza się to do potrzeby przebywania w okolicy elementu świata, z którym oddziaływało bohater. Niewątpliwie, jeżeli bohater ma wymienić się przedmiotami z jakąś postacią, to ta postać powinna znajdować się w tym samym miejscu co bohater i posiadać przedmiot, którym się ma wymienić. Jednakże warunki wstępne nic nie mówią o tym, czy ta postać w ogóle będzie chciała przeprowadzić taką wymianę.

Sam brak precyzyjności związany z brakiem zdefiniowania warunków wstępnych zwiększających wiarygodność zadań nie musi być traktowany jako wada modelu. Chociaż nieopłacalna wymiana nie jest do końca wiarygodna, to fizycznie mogłaby się odbyć. Nadając kolejne warunki wstępne na akcje modelu podejmuje się ryzyko, że wiele ciekawych i pożądanych zadań nie będzie zgodnych z opracowywanym modelem. Z tego powodu sam model powinien zawierać warunki wstępne, które będą odrzucały zadania nierozwiązywalne w świecie gry, np. nakazujące zabicie pewnej postaci, a następnie porozmawianie z nią.

Akcje złożone

Akcje złożone w modelu Dorana (tabela 3.2) są wyrażeniami nieterminalnymi, złożonymi najczęściej z kilku alternatywnych sekwencji akcji. Warto zwrócić uwagę, że konkretne zadania są zdaniami w języku definiowanym przez gramatykę Dorana. Oznacza to, że są sekwencją akcji atomowych, niezawierającą zadań złożonych. Akcje złożone są sposobem na zaprezentowanie związków przyczynowo-skutkowych pomiędzy kolejno wykonywanymi czynnościami w ramach zadania.

Każda z akcji złożonych posiada wyjaśnienie, które prezentuje przykładowe znaczenie akcji i przechowuje metadane sugerujące restrykcje dotyczące późniejszego ich wykorzystania. Przykładowo reguła 8. (patrz tabela 3.2) opisuje jeden z możliwych sposobów zdobycia wiedzy. W ramach niej występują wyrażenia *<get>* i *read*, których znaczenie jest przekazane tylko w ramach wyjaśnienia do reguły.

Na potrzeby niniejszej pracy w definicji reguły 13. została zamieniona kolejność akcji składowych w stosunku do odpowiednika z oryginalnej tabeli zawartej w pracy Dorana. Zmiana została spowodowana wielokrotnym wykorzystaniem przez Dorana zmodyfikowanej wersji w przykładach dołączonych do jego pracy. W niniejszej pracy ta nieścisłość została uznana za pomyłkę. Poniżej można zobaczyć kolejno oryginalną i zmienioną regułę.

```
<get> ::= <goto> <get> <goto> <subquest> exchange  
<get> ::= <goto> <get> <subquest> <goto> exchange
```

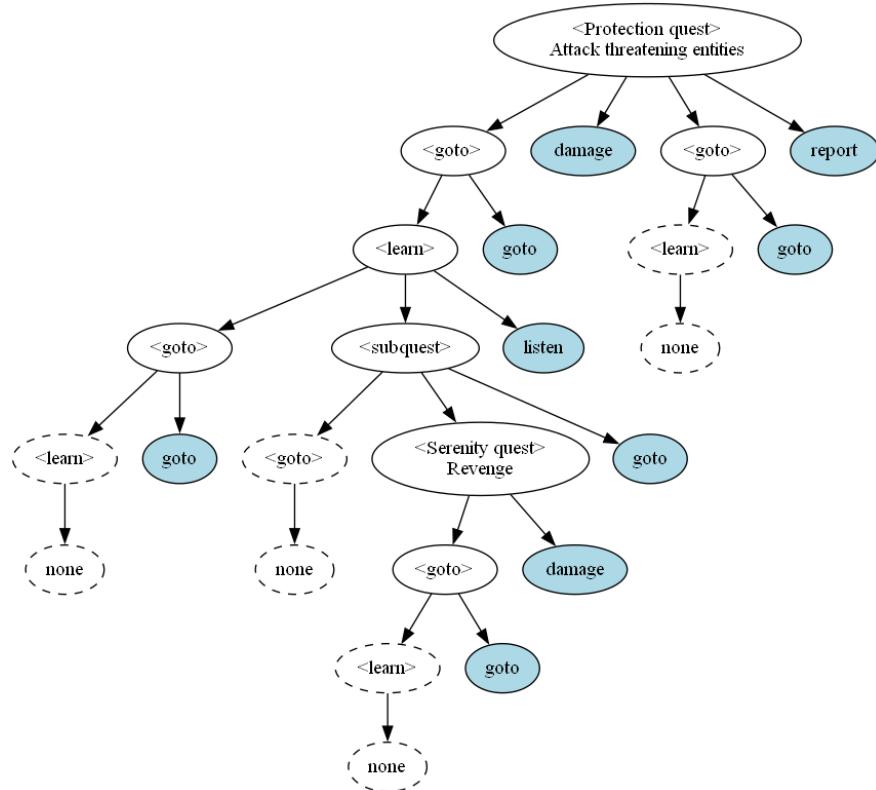
Motywacje

Strategie przyporządkowane motywacjom (tabela 3.4) pełnią rolę archetypów typowych zadań, które udało się wyekstrahować z badanych gier MMORPG. Ich celem jest zapewnienie spójności zadania, nadanie mu głębszego celu i struktury zawierającej w sobie związki przyczynowo-skutkowe. Uwzględnienie w nich motywacji postaci zwiększa wiarygodność zleceniodawcy oraz nadaje wrażenia intencjonalności zlecanego zadania.

Doran i Parberry poddali analizieczęstość występowania danych motywacji w zadaniach z badanych gier. Zaobserwowali, że motywacje *Reputation*, *Wealth*, *Comfort* oraz *Ability* stanowią łącznie 11.2% badanych zadań, a reszta motywacji posiada podobnączęstość mieszczącą się w przedziale od 18% do 20.5%. Wyniki nie dziwią zważywszy, że za wyjątkiem motywacji *Ability*, motywacje rzadziej występujące w grach mają mniej przyporządkowanych sobie strategii (patrz tabela 3.4). Te odstępstwo od reguły można wytłumaczyć tym, że zadania związane z motywacją *Ability* związane są z sekwencjami wprowadzającymi w grę, które wyjaśniają jej mechaniki i sterowanie.

Przyglądając się motywacjom w tabeli 3.4 można zauważyć, że niektóre z nich posiadają strategie, które mają identyczne przypisane im sekwencje akcji. Przykładowo sekwencja *<goto> damage* jest równoważna ze strategią *Attack enemy (Conquest)*, *Create Diversion (Protection)* oraz *Revenge, justice (Serenity)*. Konsekwencją kolidujących ze sobą strategii jest niemożliwość odtworzenia pierwotnej motywacji zleceniodawcy tylko na podstawie sekwencji czynności składających się na określone zadanie. Więcej o tym problemie i jego rozwiązaniu można przeczytać w podrozdziale 3.3.

3.3. Dualizm językowo-grafowy zadań



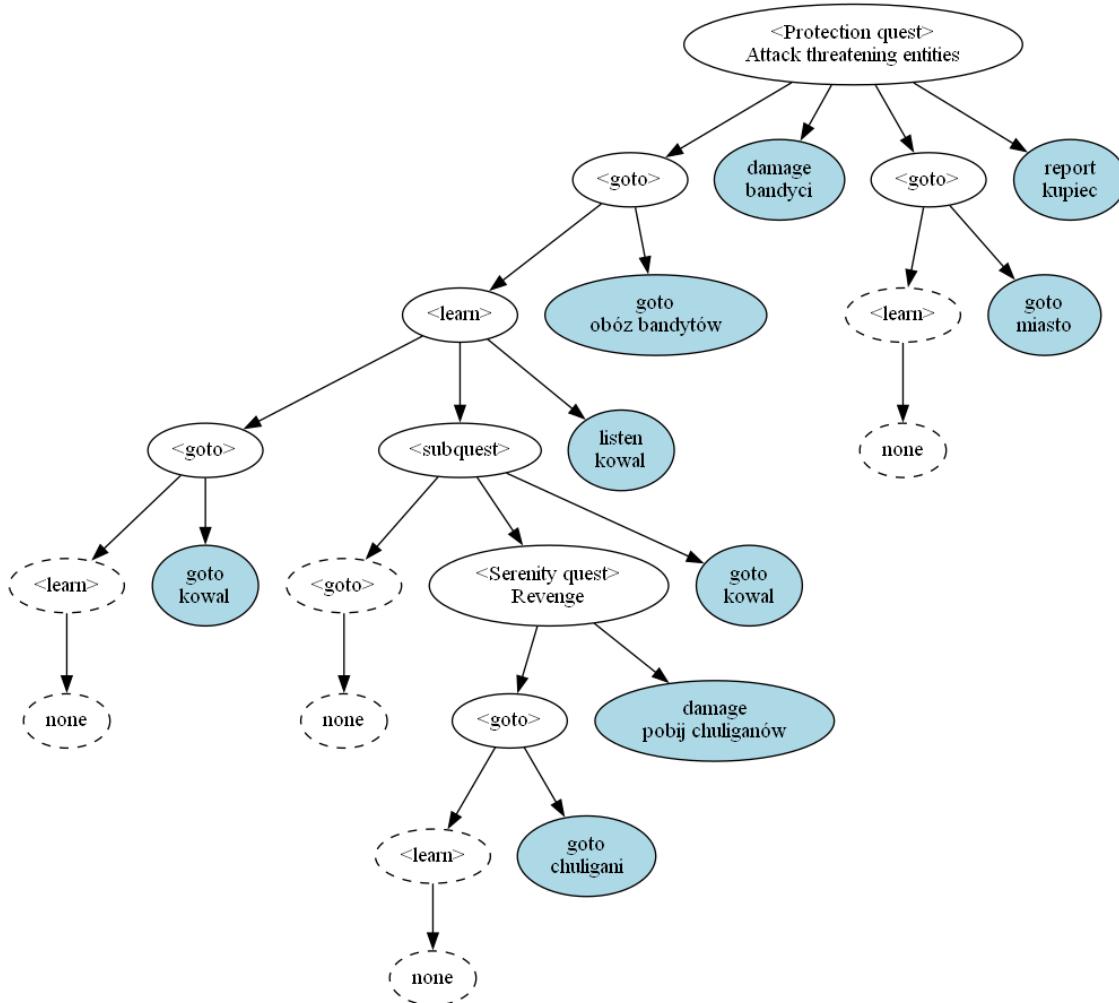
Rysunek 3.1: Zadanie jako graf-drzewo sekwencji akcji

W modelu Dorana zadania fabularne są sekwencjami czynności, jakie musi wykonać gracz, aby zakończyć pomyślnie zadanie. Taka forma wynika z zaproponowanej gramatyki i jest zgodna z definicją przyjętą w rozdziale 2. Jednakże sposób prezentacji zadania tylko i wyłącznie w postaci zdania poprawnego w języku definiowanym przez zaproponowaną gramatykę bezkontekstową wydaje się być niedostateczny.

Zadanie *goto goto damage goto listen goto damage goto report* jest całkowicie zgodne z gramatyką zaproponowaną przez Dorana, aczkolwiek nie przekazuje wystarczającej ilości wiedzy. W takiej formie wyrażenia zadania wszystkie związki przyczynowo-skutkowe zostały ukryte przed czytelnikiem, a samo zadanie przypomina bardziej zbitek losowych czynności, niż narrację.

Rozwiązaniem tego problemu jest zaproponowana przez Dorana i Parberregre struktura drzewa reprezentująca zadanie. Struktura przechowuje informacje dotyczące reguł gramatyki (tabele 3.1, 3.2, 3.4), których kolejne zastosowanie doprowadzi do określonego zdania-zadania w języku definiowanym przez gramatykę.

Budowę drzewa rozpoczyna się od korzenia będącego przyjętą strategią zadania, a jego dziećmi są kolejne akcje z odpowiadającej sekwencji (zawartej w tabeli 3.4). Wśród nich mogą występować wierzchołki reprezentujące wyrażenia nieterminalne. Do wszystkich takich węzłów dołącza się dzieci reprezentujące akcje przynależące do jednej z możliwych do przyjęcia sekwencji akcji. Takie *rozwijanie* drzewa powtarza się dopóki wszystkie jego liście nie są akcjami pustymi lub atomowymi.



Rysunek 3.2: Zadanie jako graf-drzewo sekwencji akcji z metadanymi

Na rysunku 3.1 widać zadanie *goto goto damage listen goto damage goto report* wyrażone w formie drzewa. Drzewo należy przehodzić zgodnie z zasadą preorder, czyli wchodząc jak najgłębiej się da do kolejnych poddrzew (kolejnych akcji składowych, graficznie rozmieszczanych od lewej do prawej) i wracając kiedy się nie da iść głębiej. Zaznaczone na niebiesko liście drzewa są niepustymi akcjami atomowymi. Dodatkowo oznaczono przerywaną linią poddrzewa którego wszystkimi liśćmi są akcje puste.

Wierzchołki drzewa mogą dodatkowo przechowywać informacje dotyczące szczegółów odpowiadającej akcji. Na rysunku 3.2 widać zadanie z rysunku 3.1 zawierające dodatkowe metadane dotyczące elementów świata, z którymi bohater wchodzi w interakcje. Ostatecznie w ten sposób otrzymano czytelne drzewo-zadanie odnoszące się do konkretnego świata gry.

Zastosowanie formy drzewa pozwala na wyrażenie zadań z zachowaniem ich struktury. Samo zastosowanie formy tekstu jest niejednoznaczne, ponieważ da się wykorzystując różne reguły otrzymać to samo zdanie-zadanie. Wykorzystanie postaci drzewa nie posiada tej wady.

Opracowane na potrzeby niniejszej pracy drzewa przedstawiające zadania z gier potrafią

być dużych rozmiarów. W celu zwiększenia ich czytelności przyjęto dodatkowe dwie reguły pełniące funkcję skróconego zapisu większej ilości innych reguł.

$$<\text{subquest}> ::= \varepsilon \quad (3.1)$$

$$<\text{subquest}> ::= <\text{QUEST}> \quad (3.2)$$

3.4. Analiza zadań z gry Wiedźmin 3

Model zadań Dorana został opracowany na podstawie zadań pochodzących z gier MMORPG. Jedną z cech charakterystycznych tego gatunku gier jest występowanie w nich prostych, liniowych zadań. Pomimo tego, niniejsza praca stawia tezę, że model ten jest w stanie także przedstawiać zadania pochodzące z gier RPG, które posiadają więcej złożonych zadań.

W celu potwierdzenia postawionej tezy, w niniejszej pracy dokonano analizy wszystkich pobocznych zadań fabularnych pochodzących z jednej lokacji (*Velen*) z gry *Wiedźmin 3*. Analizę wykonano na podstawie manualnego wykonania zadań w grze oraz wspierano się opisami zawartymi na stronie agregującej zadania z tej gry [22].

Ręczna analiza zadań wiąże się z podejmowaniem przez człowieka decyzji dotyczących translacji zadań do postaci zgodnej z modelem Dorana. Decyzje te często dotyczyły motywacji, której dotyczyło zadanie (niektóre strategie się ze sobą pokrywają, patrz podrozdział 3.2), ale także wyboru reguł, które zostaną zastosowane przy interpretacji zadania. Aby umożliwić reprodukcję przeprowadzonych badań, wszystkie wynikowe zadania w postaci drzewa zostały umieszczone w dodatku B.

Wyniki analizy zadań z *Wiedźmina 3* znajdują się w tabeli 3.5. Każdy wiersz tabeli zawiera wyniki dotyczące pojedynczego zadania. Tabela została podzielona na 6 kolumn:

- nr – numer porządkowy zadania przyjęty na potrzeby tej pracy;
- nazwa – nazwa zadania pochodząca ze strony zawierającej opisy zadań [22].
- motywacja – motywacja zadania przyjęta w wyniku manualnej analizy zadania. Jeżeli zadania nie udało się przedstawić przy pomocy modelu Dorana przyjmuje wartość nd;
- akcje atomowe – liczba akcji atomowych składających się na zadanie;
- typ – rodzaj zadania;
- uwagi – zawierają powód dlaczego nie udało się zamodelować zadania.

Badane zadania postanowiono podzielić na trzy grupy ze względu na ich ogólny charakter – *zadania-zdarzenia*, *zadania-aktywności* oraz *zadania pełne*. *Zadania-zdarzenia* (w skrócie *zdarzenia*) są niewielkimi zadaniami pobocznymi, w których bohater szybko reaguje na wydarzenie, które odbywa się w jego pobliżu. Ważną cechą tych zadań jest to, że zamiast z pełnoprawnym zleceniodawcą możemy mieć do czynienia z beneficjentem otrzymującym pomoc od przypadkowo znajdującego się w odpowiednim miejscu i czasie bohatera.

W kolejnej grupie zadań pobocznych *zadaniach-aktywnościach* (w skrócie *aktywnościach*) mieścią się zadania przypominające agregacje podobnych tematycznie czynności do wykonania. Wśród badanych zadań znaleziono 3 tego rodzaju zadania: Q4, Q12 i Q43 (tabela 3.5),

Tabela 3.5: Analiza możliwości zamodelowania zadań z *Wiedźmina 3* modelem Dorana

Nr	Nazwa	Motywacja	Akcje atomowe	Typ	Uwagi
Q1	Atak na karawanę	Protection	2	zdarzenie	
Q2	Dręcenie trolla	Protection	2	zdarzenie	
Q3	Fałszywe papiery	Equipment	2	zdarzenie	
Q4	Gwint: Rozgrywki w Velen	nd.	nd.	aktywności	Zbiór pobocznych aktywności
Q5.1	Na lasce obcych część 1	Protection	2	zdarzenie	
Q5.2	Na lasce obcych część 2	nd.	nd.	zdarzenie	Możliwość wyboru
Q6	Najlepszy przyjaciel człowieka	Protection	2	zdarzenie	
Q7	Napad na trakcie	Serenity	2	zdarzenie	
Q8	Niebezpieczny ładunek	Protection	2	zdarzenie	
Q9.1	Szabrownicy część 1	nd.	nd.	zdarzenie	Możliwość wyboru
Q9.2	Szabrownicy część 2	Protection	2	zdarzenie	Możliwość wyboru
Q9.3	Szabrownicy część 3	nd.	nd.	zdarzenie	Możliwość wyboru
Q10	Śmiertelna przeprawa	nd.	nd.	zdarzenie	Niezwiązane ze sobą czynności
Q11	Wróżbita prawdę ci powie	Knowledge	3	pełne	
Q12	Wyścigi: Wrońce	nd.	nd.	aktywności	Zbiór pobocznych aktywności
Q13	Zmierz się ze mną!	Protection	9	zdarzenie	Skomplikowana sekwencja zdarzeń
Q14	Stosy pogrzebowe	nd.	nd.	pełne	Zleceniodawca oczekuje wielu czynności I kłamie
Q15	Pieskie życie	Serenity	4	zdarzenie	
Q16	Pokój Ciri	Knowledge	7	pełne	Bohater jest zleceniodawcą
Q17	Duchy przeszłości	nd.	nd.	pełne	Skomplikowana sekwencja czynności
Q18	Magiczny kaganek	Serenity	7	pełne	
Q19	Mysia wieża	nd.	nd.	pełne	Wysoki poziom skomplikowania, nieliniowość
Q20	Przyjacielska przysługa	Serenity	5	pełne	
Q21	Szepczące Wzgórze	nd.	nd.	pełne	Nieliniowe, zawiera plot twist
Q22	Zagłada domu Reardonów	Comfort	4	pełne	
Q23	Zaprośenie od Keiry Metz	Serenity	3	pełne	Bohater jest zleceniodawcą
Q24	Zlecenie: Potwór z lasu	nd.	nd.	pełne	nieliniowe
Q25	Złoto głupców	Protection	8	pełne	
Q26	Dziady	nd.	nd.	pełne	Nieliniowe, zawiera plot twist
Q27	Dzikość serca	nd.	nd.	pełne	nieliniowe, zleceniodawca kłamie
Q28	Opium dla ludu	nd.	nd.	pełne	nieliniowe
Q29	Przejścia nie ma	nd.	nd.	nie	bohater otrzymuje przedmiot w zamian za przysługę
Q30	Samosąd	nd.	nd.	pełne	nieliniowe
Q31	Zlecenie: Zaginiony patrol	Serenity	4	pełne	
Q32	Dla dobra nauki	nd.	nd.	pełne	Nieszablonowe, jeden z możliwych przebiegów odbiega motywacją od innych
Q33	Zlecenie: Skrzekacz	Protection	10	pełne	Podobne do innych zleceń
Q34	Hieny cmentarne	Protection	2	pełne	Może zostać zarówno pełne, jak i być zdarzeniem
Q35	Ostatnia posługa	nd.	nd.	pełne	Nieszablonowe
Q36	Powrót na Krzywuchowe Moczary	Protection	6	pełne	
Q37	Dama w opałach	Serenity	4	pełne	
Q38	Obrońca wiary	nd.	nd.	pełne	Wymaga wykonania kilku czynności, nieliniowe
Q39	Śmierć w ogniu	nd.	nd.	zdarzenie	Nieliniowe
Q40	Wiedźmin jak malowany	nd.	nd.	pełne	nieszablonowe
Q41	Zlecenie: Leśnica	Protection	13	pełne	Podobne do innych zleceń
Q42	Zlecenie: Kłopoty grabarza	Comfort	9	pełne	Podobne do innych zleceń
Q43	Wściekłe pięści: Velen	nd.	nd.	aktywności	Zbiór побocznych aktywności
Q44	Więzy krwi	Serenity	6	pełne	
Q45	Zlecenie: Potwór z bagien	Comfort	6	pełne	Podobne do innych zleceń
Q46	Ochotnik	Comfort	4	pełne	Handel jest nieoczywisty do zamodelowania
Q47	Zlecenie: Tajemnicze ślady	nd.	nd.	pełne	nieszablonowe, skomplikowane
Q48	Zlecenie: Tajemnica wsi Stegny	Comfort	8	pełne	
Q49	Zlecenie: Zmora kupieckiego traktu	Comfort	8	pełne	Podobne do innych zleceń
Q50	Mistrz płatnerstwa	nd.	nd.	pełne	Wymaga możliwości zdobycia kilku przedmiotów z różnych miejsc
Q51	Zlecenie: Gryf na wzgórzach	Comfort	6	pełne	Podobne do innych zleceń
Q52	Bierz co chcesz	Wealth	2	pełne	
Q53	Jak kot z wilkiem	nd.	nd.	pełne	Bardzo nieszablonowe i skomplikowane
Q54	Zlecenie: Bestia z Dobrowa	nd.	nd.	pełne	Nieszablonowe, urywa się w niespodziewanym miejscu
Q55	Zlecenie: Zaginiony brat	Comfort	6	pełne	Podobne do innych zleceń

które pokazują gdzie można grać w karty, ścigać się konno oraz bić na pięści. Samo zaliczenie ich do grona zadań fabularnych może być kwestionowane, w związku z czym zadania te zostały pominięte przy analizie.

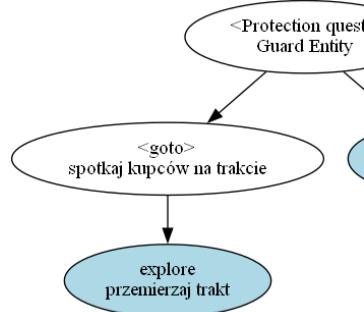
Ostatnią, najliczniejszą grupą badanych zadań są *zadania pełne*. Są to zadania typowe dla gier RPG. Wśród nich znajdują się zarówno proste, jak i bardziej rozbudowane zadania.

3.4.1. Zadania-zdarzenia

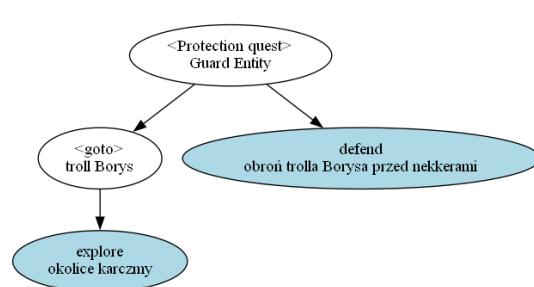
Model Dorana dobrze opisuje krótkie, nieskomplikowane zadania. Spośród badanych 15 *zadań-zdarzeń*, 10 zostało zamodelowanych poprawnie, co daje wynik 66.6% zamodelowanych *zadań-zdarzeń* o średniej wielkości 2.9 niepustych akcji atomowych.

Większość z analizowanych *zadań-zdarzeń* jest do siebie podobna ze względu na wykorzystanie tych samych lub podobnych strategii oraz akcji atomowych. Znaczną część z nich fabularnie polega po prostu na uratowaniu zaatakowanej przez potwory lub bandytów postaci. Wszystkie *zadania-zdarzenia* z definicji zakładają przypadkowe spotkanie z osobą potrzebującą pomocy, co jest odzwierciedlone poprzez rozpoczętą ją akcją atomową *explore*.

Przykładowe *zadania-zdarzenia* Q1 i Q2 (patrz tabela 3.5) zostały przedstawione w formie drzew na rysunkach 3.3 i 3.4. *Zdarzenia* te posiadają ten sam schemat, różnią się jedynie danymi dotyczącymi szczegółów zadania. Zdaje się to potwierdzać tezę Dorana i Parberrego, którzy twierdzą, że można generować zadania niskim kosztem, wykorzystując wielokrotnie jeden szablon, zmieniając przy tym jedynie obiekty wchodzące w interakcje z bohaterem [8].



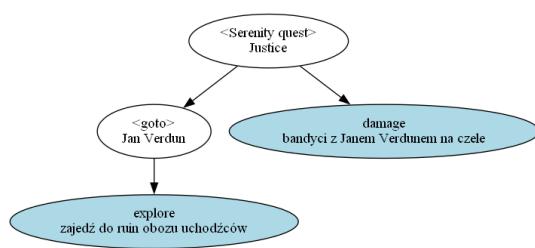
Rysunek 3.3: Q1 – Atak na karawanę (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



Rysunek 3.4: Q2 – Dręczenie trolla (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

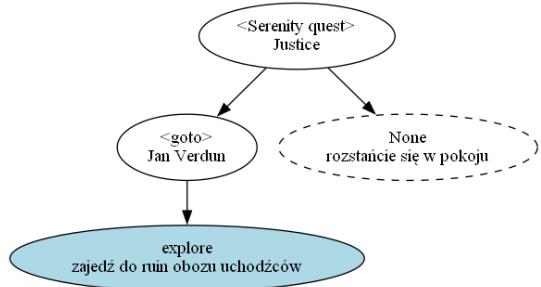
Spośród badanych *zadań-zdarzeń* wyróżnia pod względem wielkości zadanie Q13 widoczne na rysunku 3.7. Zadanie to składa się aż z 9 akcji atomowych. Polega ono na kolejnych spotkaniach z rycerzem Ronvidem, który rzuca bohaterowi wyzwanie. Można by zatem rozbić te zadanie na trzy kolejne *zdarzenia*. Taki zapis spowodowałby jednak utratę części informacji przekazywanych w pierwotnej formie zadania, albowiem Ronvida można spotkać przy karczmie dopiero jak pokonamy go we Wrońcach. Rozdzielając zadania stracilibyśmy informację o konieczności spotykania rycerza po kolei w określonych miejscowościach.

Próba zamodelowania 5 *zadań-zdarzeń* zakończyła się jednak niepowodzeniem. Główną przyczyną takiego stanu rzeczy jest brak możliwości reprezentacji decyzji gracza wpływających na zakończenie zadania. Rysunki 3.5 i 3.6 przedstawiają dwa alternatywne przebiegi *zdarzenia*



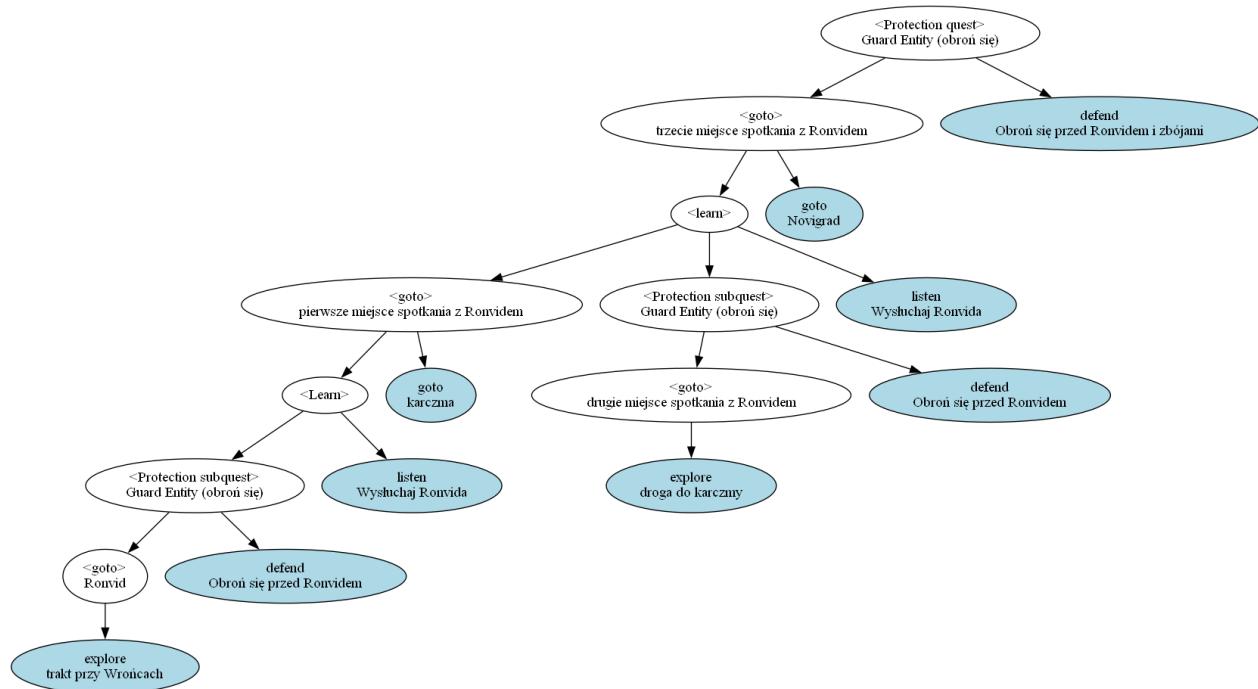
Rysunek 3.5: Q5.2 – *Na łasce obcych*, pierwszy przebieg zadania (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

Q5.2. Podczas niego bohater ma możliwość rozejść się w pokoju z bandytami lub powstrzymać ich przed czynieniem dalszego zła.



Rysunek 3.6: Q5.2 – *Na łasce obcych*, alternatywny przebieg zadania (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

Q5.2. Podczas niego bohater ma możliwość rozejść się w pokoju z bandytami lub powstrzymać ich przed czynieniem dalszego zła.



Rysunek 3.7: Q13 – *Zmierz się ze mną!* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

3.4.2. Zadania pełne

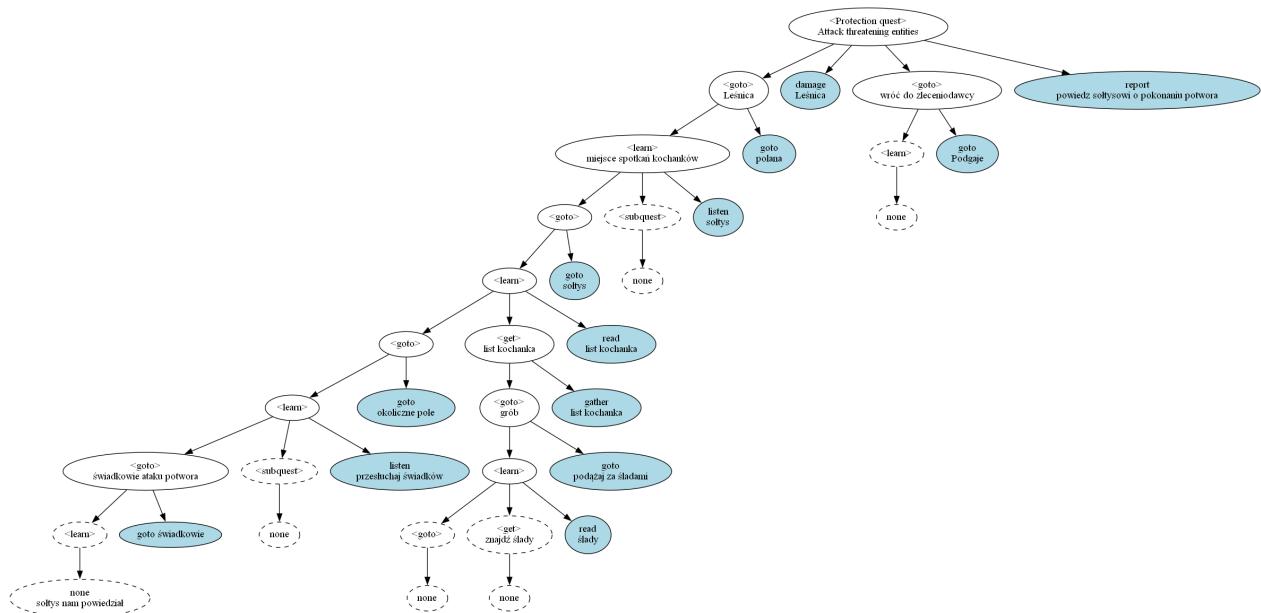
Zadaniami pełnymi są typowe dla gier fabularnych zadania poboczne. Spośród badanych 40 zadań pełnych udało się zamodelować 22 zadania, co stanowi 55% całej analizowanej grupy.

Analiza zadań pełnych z *Wiedźmina 3* pokazała, że model Dorana jest w stanie przedstawić nie tylko proste zadania, ale także te o wyższym stopniu skomplikowania, który objawia

się między innymi większym rozmiarem zadań oraz występowaniem w nich rozbudowanych związków przyczynowo-skutkowych. Średni rozmiar możliwych do zamodelowania *pełnych zadań* wynosi 5.9 niepustych akcji atomowych, a najdłuższe z nich zawiera aż 13 takich akcji.

Zdolność modelu do przedstawiania dużych związków przyczynowo-skutkowych została szczególnie wyeksponowana w trakcie modelowania powszechnych w grze *zadań-zleceń* (Q31, Q33, Q41, Q42, Q45, Q48, Q49 oraz Q51). Zadania te polegają na przeprowadzeniu śledztwa mającego na celu znalezienie i zabicie potwora zagrażającego zleceniodawcy.

Na rysunku 3.8 można zobaczyć typowe zadanie-zlecenie. Zadania te posiadają wysoce rozbudowaną akcję *<goto>*, która pełni rolę śledztwa, w trakcie którego bohater dowiaduje się co zagraża zleceniodawcy i gdzie się znajduje. Kolejne kroki tego śledztwa wyrażone są poprzez akcje złożone *<learn>*. Po wykonaniu śledztwa, w punkcie kulminacyjnym zadania, bohater pokonuje potwora (akcja *damage*) po czym wraca do zleceniodawcy i odbiera nagrodę.



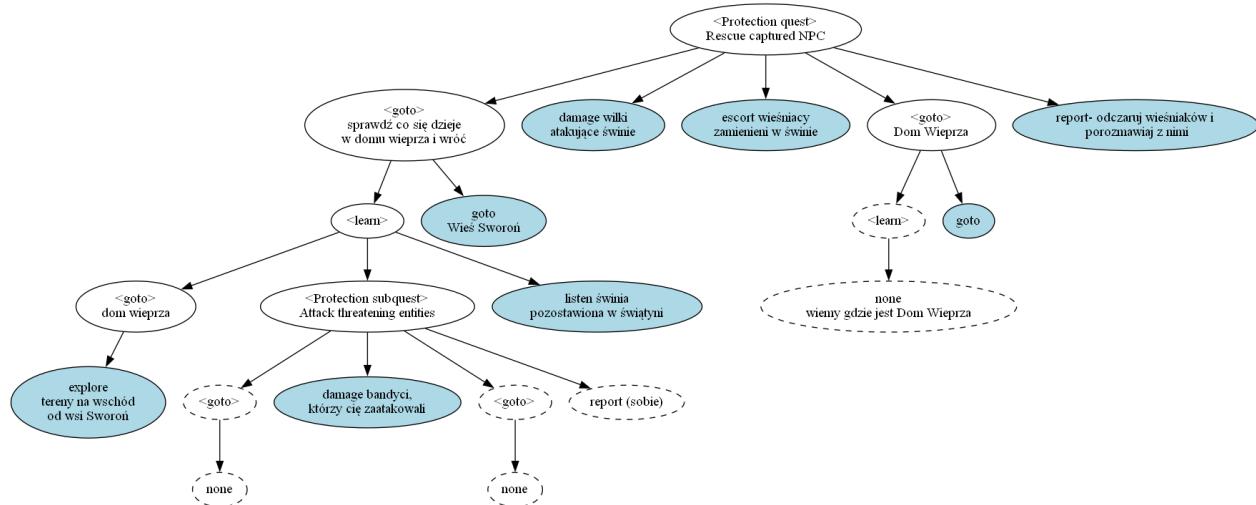
Rysunek 3.8: Q41 – Zlecenie: Leśnica (opracowanie własne na podstawie opisu zadania [22] i gry Wiedźmin 3)

Nie wszystkie zвязki przyczynowo-skutkowe znajdujące się w zamodelowanych zadaniach z *Wiedźmina 3* polegają na wytropieniu potwora. Z przeprowadzonej analizy zadań wynika, że model Dorana jest w stanie przedstawić zadania, w których bohater musi dowiedzieć się najpierw jak pomóc zleceniodawcy, a dopiero potem wykonać odpowiednie czynności.

Przykładem zadania o opisanych uprzednio cechach jest zadanie zadanie Q25 *Złoto głupców* widoczne na rysunku 3.9. Jest to jedno z bardziej rozbudowanych zadań pobocznych w *Wiedźminie 3*. W zadaniu tym bohater odwiedza wieś której mieszkańców zostali zamienieni w świnie. Aby ich odczarować bohater:

1. Udaje się do domu wieprza. Wie mniej więcej gdzie się on znajduje (akcja *explore* na rysunku 3.9).
2. W domu wieprza bohater spotyka bandytów i musi się przed nimi obronić (subquest: Atak threatening entities).

3. Po pokonaniu bandytów bohater znajduje świnię i “rozmawia” z nią dowiadując się jak odczynić kłatwę.
4. Bohater wraca do wsi, pokonuje zwierzęta, które ją zaatakowały.
5. Bohater eskortuje świnie do domu wieprza i odczynia kłatwę.



Rysunek 3.9: Q25 – Złoto głupców (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

Model Dorana jest w stanie modelować część typowych czynności wchodzących w skład zadań fabularnych, a nie mających bezpośredniego odzwierciedlenia w zbiorze *akcji atomowych*. Przykładem takiego powtarzalnego elementu jest kupno zakup jakiegoś przedmiotu. Sposób zamodelowania takiej czynności możemy zobaczyć na rysunku 3.10, przedstawiającym zadanie w którym bohater ma zakupić farby dla trolla. Sam sposób prezentacji zakupu farb jest dość skomplikowany, ale pokazuje zdolności modelu do prezentacji czynności tego rodzaju.

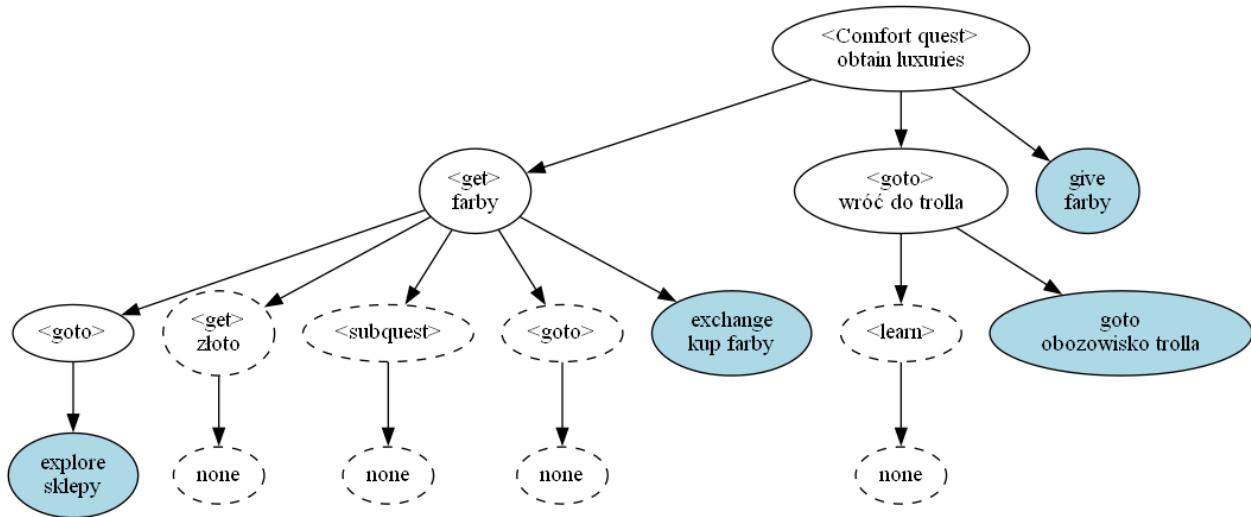
3.4.3. Podsumowanie

Pomimo faktu, że model potrafi przedstawiać wiele motywów występujących w zadaniach fabularnych, to nie udało mu się wyrazić 45% badanych *zadań pełnych*. Model ma problemy z przedstawianiem bardzo skomplikowanych zadań, które mocno odbiegają od typowych zadań z gier MMORPG. Do tej grupy zadań należy zaliczyć te, w których zleceniodawca próbował oszukać bohatera, miał nieszczere intencje. Ponadto analizowany model nie jest w stanie wyrazić wybór bohatera, które mają wpływać na przebieg zadania.

W *zadaniach pełnych* wybory podejmowane przez bohatera nie muszą być tak proste, jak w *zadaniach-zdarzeniach* i mogą być związane z plot-twistami. Doskonale obrazuje to zadanie Q21 *Szepczące Wzgórze*, którego nie udało się zamodelować. Podczas tego zadania bohater ma pozbyć się ducha zaklętego w drzewie. Jednakże kiedy bohater dociera do ducha, ten stara się wzbudzić litość bohatera i prosi o uwolnienie. Gracz staje przed wyborem, może stoczyć z nim walkę zgodnie z oryginalnym zlecienniem, albo pomóc duchowi nie wypełniając woli

zleceniodawcy. Ta druga opcja wymaga wykonania dodatkowego podzadania i zignorowania reszty podstawowego przebiegu zadania.

Sumując, należy stwierdzić, że model Dorana posiada niedopatrzenie powodujące, że nie udało zamodelować się kilku zadań. Modelowi temu brakuje możliwości wyrażenia potrzeby zdobycia kilku przedmiotów. Reguły 10-13 z tabeli 3.2 nie dopuszczają na zdobycie kilku przedmiotów, które znajdują się w innych miejscach. Jest to jedna z przyczyn niemożliwości zamodelowania zadań Q21 oraz Q50.



Rysunek 3.10: Q46 – *Ochotnik* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

3.5. Rozszerzona notacja Backusa-Naura

Do zapisu gramatyki rozszerzonego modelu zadań w niniejszej pracy została wykorzystana opisana w *ISO/IEC 14977* rozszerzona notacja Backusa-Naura (w skrócie EBNF) z dodatkowym, przyjętym na potrzeby tej pracy jej rozszerzeniem. EBNF cechuje się rozszerzalnością, co stało się powodem wykorzystania tej własności w ramach rozszerzenia notacji, które ułatwiło zapis operatorów zmieniających sposób przechodzenia zadania w formie drzewa.

EBNF różni się od BNF sposobem zapisu wyrażeń (terminalnych oraz nieterminalnych). Poniżej znajdują się wszystkie informacje potrzebne do przeczytania znajdujących się w niniejszej pracy reguł:

- Symbole nieterminalne **nie** są zawarte pomiędzy nawiasami $\langle \rangle$ i mogą składać się z wielu słów.
 - Symbole terminalne są zawarte w cudzysłowie “”.
 - Reguły składają się z wyrażenia nieterminального, znaku równości = i sekwencji wyrażeń oddzielonych od siebie przecinkiem lub operatorem | oznaczającym alternatywę. W BNF wyrażenia nie musiały być oddzielane przecinkami.

- Reguła zakończona jest średnikiem.
- Nawiąsy () służą do grupowania wyrażeń.
- W nawiasach klamrowych znajdują się wyrażenia, które można powtórzyć 0 lub więcej razy.

Dodatkowo na potrzeby niniejszej pracy rozszerzono EBNF o następujące dwa zapisy przedstawiające operatory zmieniające sposób przechodzenia drzewa-zadania:

- or(wyrażenie1)(wyrażenie2) – równoważny z: “or”, “(”, wyrażenie1, “)”, “(”, wyrażenie2, “)”
- twist(wyrażenie) – równoważny z: “twist”, “(”, wyrażenie, “)”

Aby lepiej zrozumieć gramatykę wyrażoną w EBNF przyjrzyjmy się przykładowi na podstawie którego została przedstawiona notacja BNF w podrozdziale 3.1:

cyfra dodatnia = “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”;
cyfra = “0” | cyfra dodatnia;
ciąg cyfr = (cyfra, ciąg cyfr) | “”;
liczba naturalna = cyfra dodatnia, ciąg cyfr;

3.6. Rozszerzenie modelu zadań

Podczas analizy zadań z Wiedźmina 3 zaobserwowano następujące wady modelu Dorana:

- W1: Model nie uwzględnia możliwości wprowadzania głównego bohatera w błąd przez zleceniodawcę.
- W2: Model nie pozwala na przedstawienie potrzeby zdobycia kilku przedmiotów.
- W3: Model nie pozwala na zdobycie w pokojowy sposób przedmiotu posiadanego przez kogoś bez dokonania wymiany.
- W4: Model pozwala tylko na przedstawianie liniowych zadań, czyli takich, w których bohater nie ma możliwości podejmowania decyzji.

Przyczyną wady W1 jest dobór zadań na podstawie, których model był opracowywany. Były to liniowe zadania pochodzące z gier typu MMORPG, w których zadania są proste i raczej nie wprowadzają bohatera w błąd. Aby naprawić tę wadę należało by wyekstrahować z zadań z innych gier dodatkowe motywacje i opracować dla nich nowe strategie. Można uznać, że model Dorana nie jest przeznaczony do przedstawiania tego typu zadań, a rozszerzenie go o motywacje związane z próbą oszukania bohatera nie wchodzi w zakres niniejszej pracy. Niemniej jednak jest to możliwy kierunek dalszych badań.

Wada W2, polega na braku zdolności modelu do wyrażenia czynności zdobycia kilku przedmiotów znajdujących się w **różnych** miejscach. Jeżeli znajdują się one w tym samym

miejscu to można po prostu wykonać akcje *<get>*, *take*, *gather* na kilku przedmiotach jednocześnie. Aby naprawić wadę W2 wystarczy zastąpić większość dotychczasowych wystąpień akcji *<get>* w regułach zawartych w tabelach 3.2 i 3.4 nową regułą:

get many = {get};

Regułę należy czytać następująco: akcja *get many* składa się z dowolnie liczby akcji *get*. W postaci drzewa będzie ona przedstawiana jak na rysunkach 3.11 i 3.12 z zastrzeżeniem, że w przypadku akcji *get many*, która składa się z pojedynczej akcji *get*, akcja *get many* będzie pomijana w postaci drzewa.



Rysunek 3.12: Pusta akcja get many

Rysunek 3.11: Akcja get many reprezentująca zdobycie 2 przedmiotów

Wadę W3, czyli problem niemożliwości zdobycia posiadanej przez kogoś przedmiotu uprzednio bez wymiany, może zostać naprawiony poprzez dodanie nowej sekwencji akcji do definicji akcji *<get>* wyrażonej regułami 10-13 w tabeli 3.2. Dodatkowa sekwencja akcji wygląda następująco:

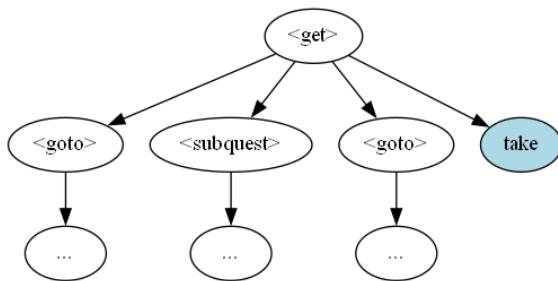
goto, subquest, goto, "take"

Dodanie jej sprawia, że cała zapisana w EBNF reguła definiująca wyrażenie *get* ma postać:

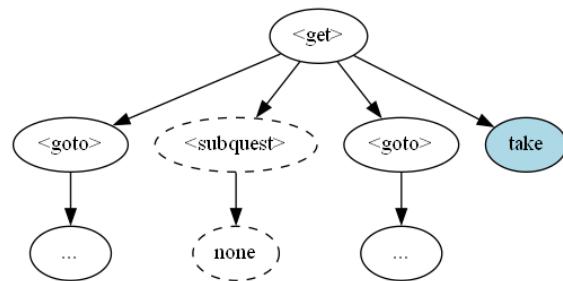
```
get = "none"
    | steal
    | goto, "gather"
    | goto, get many, subquest, goto, "exchange"
    | goto, subquest, goto, "take";
```

Dzięki dodaniu alternatywnej sekwencji akcji możemy przedstawić dwie dodatkowe, często spotykane w zadaniach, sytuacje. Pierwsza z nich została przedstawiona na rysunku 3.13 i zakłada, że bohater otrzymuje przedmiot w zamian za wykonanie jakieś przysługi dla posiadacza potrzebnego obiektu. Druga sytuacja występuje wtedy, kiedy wyrażenie *subquest* zostanie rozwinięte do pustej akcji atomowej. W takiej sytuacji bohater otrzymuje przedmiot bezinteresownie (patrz rysunek 3.14).

Przypomnijmy sobie algorytm poruszania się po drzewach-zadaniach w modelu Dorana, gdyż to z niego wynika źródło wady W4. Polega on na przechodzeniu drzew *preorder*, czyli



Rysunek 3.13: Akcja get, w której posiadacz przedmiotu oczekuje wykonania przyługi



Rysunek 3.14: Akcja get, w której bohater otrzymuje przedmiot bez wykonywania dodatkowych czynności

wchodzeniu jak najgębiej się da w **kolejne** poddrzewa i zawracanie, w momencie gdy znajdujemy się w liściu, aż do przejścia całego drzewa. Stosując taki algorytm odwiedzamy wszystkie węzły drzewa, a co za tym idzie wykonujemy wszystkie akcje zdefiniowane w zadaniu. Oznacza to, że aby wprowadzić możliwość reprezentacji wyborów gracza, potrzebujemy wprowadzić zmiany w sposobie poruszania się po drzewach-zadaniach.

Istotnym jest, aby zmiany w sposobie przechodzenia drzew-zadań odzwierciedlone zostały także w zadaniach w postaci zdania języka definiowanego przez gramatykę bezkontekstową. Aby to zapewnić można do języka wprowadzić wyrażenia terminalne, które pełniłyby role słów kluczowych i byłyby w odpowiedni sposób interpretowane.

Pierwszym z dodanych słów kluczowych jest *or*, które reprezentować będzie wybór. W gramatyce zapis:

or(wyrażenie1)(wyrażenie2)...(wyrażenieN)

będzie oznaczać wykonanie **jednego**, dowolnie wybranego przez gracza wyrażenia spośród wyrażeń zawartych w kolejnych nawiasach.

Warto zwrócić uwagę na jednoznaczność przyjętego zapisu. Przyjrzyjmy się przykładowym regułom:

reguła 1 = *or("damage", "goto")("")*;
 reguła 2 = *or("damage")("")*, "goto";

Reguła pierwsza reprezentuje akcję, w której bohater ma wykonać akcję "damage" **i** "goto", ewentualnie nic nie. Czyli może wykonać 2 lub 0 akcji atomowych. Reguła 2 przedstawia akcję, w której bohater wybiera, czy wykonać akcję "damage", czy niczego nie robić. Następnie, niezależnie od wcześniej podjętego wyboru, wykonuje akcję "goto". Innymi słowy wykonuje albo jedną, albo dwie akcje atomowe.

Kolejne zaproponowane słowo kluczowe "twist" ma na celu reprezentację tych części zadania, w których sprzeciwiamy się zleceniodawcy. W gramatyce zapis:

twist(wyrażenie)

będzie oznaczać wykonanie wyrażenia zawartego w nawiasie i **przerwanie dalszego wykonywania zadania**.

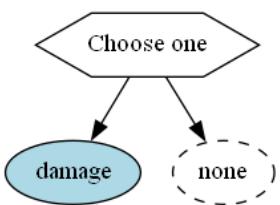
Jak można zauważać słowo kluczowe “*twist*” nabiera znaczenia tylko w połączeniu ze słowem kluczowym “*or*”. Założmy scenariusz, w którym zleceniodawca chciał, abyśmy dokonali zabójstwa pewnej postaci, a następnie wrócili zameldować o wynikach działania. Tymczasem podczas konfrontacji z postacią bohater może alternatywnie dać jej dowody na zlecenie zabójstwa działając na szkodę zleceniodawcy.

reguła 1 = $\text{twist}(\text{"give"}), \text{"kill"}, \text{"goto"}, , \text{"report"}$

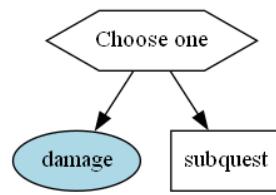
reguła 2 = $\text{or}(\text{twist}(\text{"give"}))(\text{"kill"}), \text{"damage"}, \text{"goto"}, \text{"report"}$

Z powyższych reguł pozwalającą na poprawne zamodelowanie przedstawionego scenariusza jest reguła 2. Reguła pierwsza pozwala tylko na stworzenie zadań sprzeciwiających się zleceniodawcy.

Nowe słowa kluczowe wymagają wprowadzenia sposobu na ich ilustrację w zadaniach-drzewach. Na rysunkach 3.15 i 3.16 można zobaczyć kolejno reprezentacje graficzne wyrażeń zawierających nowo dodane słowa kluczowe. Hexagonalny węzeł *Choose one* (odpowiadający wyrażeniu *or*) oznacza wybór jednego z poddrzew do przejścia. Prostokątny węzeł odpowiadający wyrażeniu *twist* oznacza, że przechodzenie drzewa zakończy się w momencie, kiedy przejdziemy całe poddrzewo którego ten węzeł jest korzeniem.



Rysunek 3.15: Zapis wyrażenia $\text{or}(\text{"damage"})("")$ w postaci drzewa



Rysunek 3.16: Zapis wyrażenia $\text{or}(\text{"damage"})(\text{twist}(\text{subquest}))$ w postaci drzewa

Analiza zadań z Wiedźmina 3 pozwoliła na znalezienie często występujących motywów, których nie udało się zapisać stosując model Dorana. Takim motywem jest wybór pomiędzy wtrąceniem się i obroną postaci przed zagrożeniem, lub zignorowanie zaistniałej sytuacji. Może to być odzwierciedlone za pomocą sekwencji akcji:

$\text{or}(\text{"damage"})("")$

wykorzystanej w nowo dodanych strategiach *Justice(2)* i *Attack threatening entities(2)* widocznych w tabeli 3.6

W analizowanych *zadaniach pełnych* można było zauważać występowanie bardziej rozbudowanej alternatywy dla akcji *damage*, która oferuje alternatywny przebieg zadania. Zamiast wykonania zleconego ataku wykonuje się inne zadanie dla poprzedniego celu ataku. Reprezentowane będzie to poprzez sekwencje akcji

goto, get many, subquest, goto, "exchange";

występującą w nowych strategiach *Attack threatening entities(3)* i *Kill enemies(2)* widocznych w tabeli 3.6.

Tabela 3.6: Nowo dodane strategie

Motywacja	Strategia	Sekwencja akcji
Serenity	Justice(2)	goto, or("damage")("")
Protection	Attack threatening entities(2)	goto, or("damage")(""), goto, report
Protection	Attack threatening entities(3)	goto, or("damage")(twist(subquest)), goto, report
Reputation	Kill enemies(2)	goto, or("kill")(twist(subquest)), goto, report

Zbiór reguł rozszerzonego modelu został dodatkowo poprawiony względem oryginału. Poprawa ta polega na usunięciu nieużywanych reguł (np. reguły 17. z tabeli 3.2) i innych nie wpływających na siłę ekspresji modelu modyfikacjach.

Gramatyka bezkontekstowa wyrażająca rozszerzony model składa się ze:

- Zbioru wyrażeń terminalnych: "or", "(", ")", "twist", "capture", "damage", "defend", "escort", "exchange", "experiment", "explore", "gather", "give", "goto", "kill", "listen", "read", "repair", "report", "spy", "stealth", "take", "use".
- Zbioru wyrażeń nieterminalnych: *QUEST*, *subquest*, *goto*, *learn*, *get*, *steal*, *spy*, *capture*, *kill*, *Knowledge*, *Reputation*, *Serenity*, *Protection*, *Conquest*, *Wealth*, *Ability*, *Equipment*.
- Wyrażenia początkowego *quest*.
- Poniższego zbioru reguł:

```


$$\begin{aligned} \text{QUEST} &= \text{Knowledge} \mid \text{Comfort} \mid \text{Reputation} \\ &\quad \mid \text{Serenity} \mid \text{Protection} \mid \text{Conquest} \\ &\quad \mid \text{Wealth} \mid \text{Ability} \mid \text{Equipment}; \\ \text{subquest} &= \text{goto} \\ &\quad \mid \text{goto, QUEST, "goto"} \\ &\quad \mid ""; \\ \text{goto} &= "" \\ &\quad \mid \text{"explore"} \\ &\quad \mid \text{learn, "goto"}; \\ \text{learn} &= "" \\ &\quad \mid \text{goto, subquest, "listen"} \\ &\quad \mid \text{goto, get, "read"} \\ &\quad \mid \text{get many, goto, "give", "listen"}; \\ \text{get many} &= \{\text{get}\}; \\ \text{get} &= "" \\ &\quad \mid \text{goto, subquest, goto, "take"} \\ &\quad \mid \text{steal} \end{aligned}$$


```

```

| goto, "gather"
| goto, get many, subquest, goto, "exchange";
steal = goto, "stealth", "take"
| goto, kill, "take";
spy = goto, "spy", goto, "report";
kill = goto, "kill";

Knowledge = get many, goto, "give"
| spy
| goto, "listen", goto, "report"
| get many, goto, "use", goto, "give";

Comfort = get many, goto, "give"
| goto, "damage", goto, "report";

Reputation = get many, goto, "give"
| goto, kill, goto, "report"
| goto, or(kill)(twist(subquest)), goto, "report"
| goto, goto, "report";

Serenity = goto, "damage"
| goto, or("damage")("")
| get many, goto, "use", goto, "give"
| get many, goto, "use", "capture", goto, "give"
| goto, "listen", goto, "report"
| goto, "subquest", goto, "report
| goto, "take", goto, "give"
| get many, goto, "give"
| goto, "damage", "escort", goto, "report";

Protection = goto, "damage", goto, "report"
| goto, or("damage")(""), goto, "report"
| goto, or(kill)(twist(subquest)), goto, "report"
| get many, goto, "use"
| goto, "repair"
| get many, goto, "use"
| goto, "damage"
| goto, "repair"
| goto, "defend";

Conquest = goto, "damage"
| goto, steal, goto, "give";

Wealth = goto, get many

```

```
| goto, steal  
| "repair";  
Ability = "repair", "use"  
| get many "use"  
| "use"  
| "damage"  
| "use"  
| get many, "use"  
| get many, "experiment"  
Equipment = "repair";  
| get many, goto, "give"  
| steal  
| goto, "exchange";
```

3.7. Analiza zadań z gry Wiedźmin 3 z wykorzystaniem rozszerzonego modelu

Wyniki ponownego badania zadań z *Wiedźmina 3* znajdują się w tabeli 3.7, a opracowane modele zadań zawarte są w dodatkach B i C w celu umożliwienia weryfikacji badań. Z analizy zadań wynika, że przy pomocy rozszerzonego modelu udało się przedstawić:

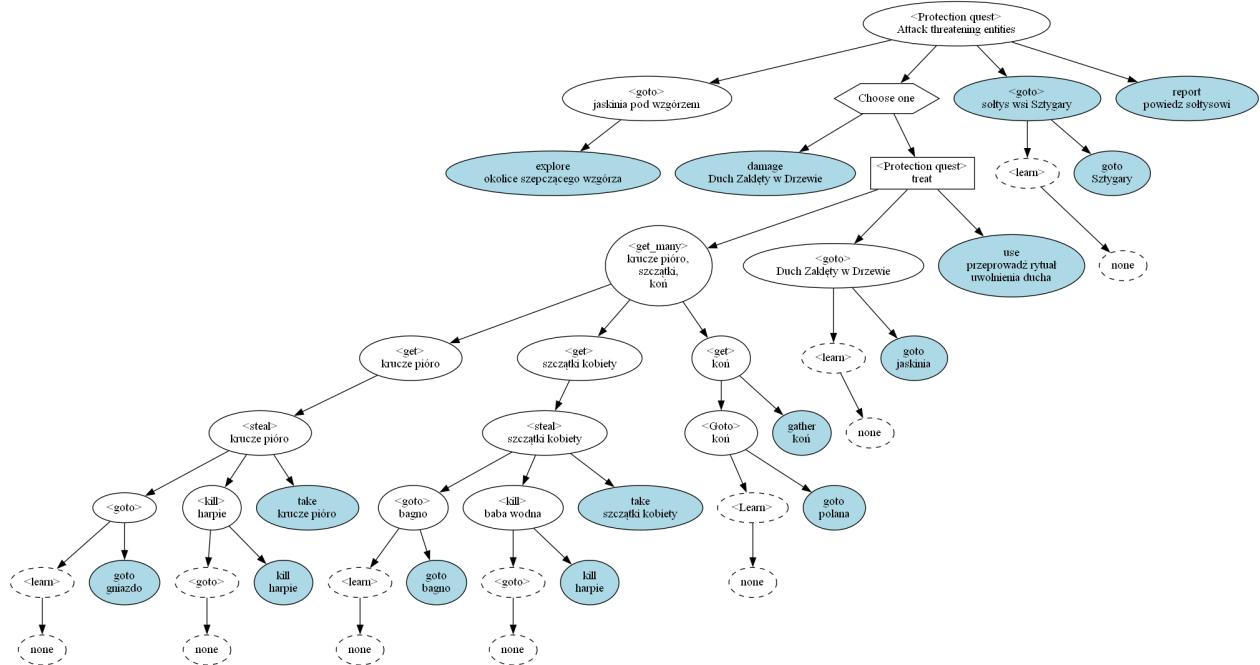
- 14 z 15 zadań-zdarzeń (93.3%), co daje wynik wyższy o 26.7 punktów procentowych w stosunku do wyniku uzyskanego przez model Dorana;
- 30 z 40 zadań pełnych (75%), co daje wynik wyższy o 20 punktów procentowych w stosunku do wyniku uzyskanego przez model Dorana.

Nowo zamodelowane zadania pełne charakteryzują się większym poziomem skomplikowania od zadań uprzednio zamodelowanych przez model Dorana. Przejawia się to zarówno w możliwości dokonywania decyzji przez gracza, jak i zwiększonej średniej długości zadań. Wszystkie zamodelowane zadania pełne przez rozszerzony model zawierały średnio 6.1 akcji atomowych, a największe zadanie zawierało ich aż 15.

Największe z zamodelowanych zadań można zobaczyć w postaci drzewa na rysunku 3.17. Zadanie to jest najbardziej rozbudowane ze wszystkich analizowanych zadań i zawiera w sobie zarówno podejmowanie decyzji przez gracza (operator *or*), jak i alternatywne zakończenie (operator *twist*) oraz potrzebę zdobycia kilku przedmiotów z różnych miejsc (akcja złożona *get many*). Zaprezentowany przykład uwidacznia możliwości rozszerzonego modelu do reprezentacji dużych i nieliniowych zadań fabularnych.

Tabela 3.7: Analiza możliwości zamodelowania zadań z *Wiedźmina 3* rozszerzonym modelem

Nr	Nazwa	Motywacja	Akcje atomowe	Typ	Uwagi
Q1	Atak na karawanę	Protection	2	zdarzenie	
Q2	Dręczenie trolla	Protection	2	zdarzenie	
Q3	Fałszywe papiery	Equipment	2	zdarzenie	
Q4	Gwint: Rozgrywki w Velen	nd.	0	aktywności	
Q5	Na łasce obcych część 1	Protection	2	zdarzenie	
Q5	Na łasce obcych część 2	Serenity	2	zdarzenie	
Q6	Najlepszy przyjaciel człowieka	Protection	2	zdarzenie	
Q7	Napad na trakcie	Serenity	2	zdarzenie	
Q8	Niebezpieczny ładunek	Protection	2	zdarzenie	
Q9	Szabrownicy część 1	Serenity	2	zdarzenie	
Q9	Szabrownicy część 2	Protection	2	zdarzenie	
Q9	Szabrownicy część 3	Serenity	2	zdarzenie	
Q10	Śmiertelna przeprawa	nd.	0	zdarzenie	
Q11	Wróżbita prawdę ci powie	Knowledge	3	pełne	
Q12	Wyścigi: Wrońce	nd.	0	aktywności	
Q13	Zmierz się ze mną!	Protection	9	zdarzenie	
Q14	Stosy pogrzebowe	nd.	0	pełne	Zleceniodawca kłamie
Q15	Pieskie życie	Serenity	4	zdarzenie	
Q16	Pokój Ciri	Knowledge	7	pełne	
Q17	Duchy przeszłości	nd.	0	pełne	Nieszablonowe
Q18	Magiczny kaganek	Serenity	7	pełne	
Q19	Mysia wieża	nd.	0	pełne	Nieszablonowe
Q20	Przyjacielska prysząca	Serenity	5	pełne	
Q21	Szepczące Wzgórze	Protection	15	pełne	
Q22	Zagłada domu Reardonów	Comfort	4	pełne	
Q23	Zaproszenie od Keiry Metz	Serenity	3	pełne	
Q24	Zlecenie: Potwór z lasu	Protection	5	pełne	
Q25	Złoto głupców	Protection	8	pełne	
Q26	Dziady	Protection	11	pełne	
Q27	Dzikosć serca	nd.	0	pełne	Zleceniodawca kłamie
Q28	Opium dla ludu	Protection	4	pełne	
Q29	Przejścia nie ma	Serenity	5	pełne	
Q30	Samosąd	Serenity	2	pełne	
Q31	Zlecenie: Zaginiony patrol	Serenity	4	pełne	
Q32	Dla dobra nauki	nd.	0	pełne	Nieszablonowe
Q33	Zlecenie: Skrzekacz	Protection	10	pełne	
Q34	Hieny cmentarne	Protection	2	pełne	
Q35	Ostatnia posługa	nd.	0	pełne	Nieszablonowe
Q36	Powrót na Krzywuchowe Moczary	Protection	6	pełne	
Q37	Dama w opałach	Serenity	4	pełne	
Q38	Obrońca wiary	nd.	0	pełne	Wymaga wykonania kilku czynności
Q39	Śmierć w ogniu	Serenity	2	zdarzenie	
Q40	Wiedźmin jak malowany	Serenity	3	pełne	
Q41	Zlecenie: Leśnica	Protection	13	pełne	
Q42	Zlecenie: Kłopoty grabarza	Comfort	9	pełne	
Q43	Wściekłe pięści: Velen	nd.	0	aktywności	
Q44	Więzy krwi	Serenity	6	pełne	
Q45	Zlecenie: Potwór z bagien	Comfort	6	pełne	
Q46	Ochotnik	Comfort	4	pełne	
Q47	Zlecenie: Tajemnicze ślady	nd.	0	pełne	Nieszablonowe
Q48	Zlecenie: Tajemnica wsi Stegny	Comfort	7	pełne	
Q49	Zlecenie: Zmora kupieckiego traktu	Comfort	8	pełne	
Q50	Mistrz płatnerstwa	Comfort	8	pełne	
Q51	Zlecenie Gryf na wzgórzach	Comfort	6	pełne	
Q52	Bierz co chcesz	Wealth	2	pełne	
Q53	Jak kot z wilkiem	nd.	0	pełne	Nieszablonowe
Q54	Zlecenie: Bestia z Dobrowa	nd.	0	pełne	Nieszablonowe
Q55	Zlecenie: Zaginiony brat	Comfort	6	pełne	



Rysunek 3.17: Q21 – Szepczące Wzgórze (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

4. SkaldNET – Prototyp generatora zadań fabularnych

W poprzednim rozdziale zostało zaproponowane rozszerzenie modelu Dorana o możliwość reprezentacji bardziej skomplikowanych, nieliniowych zadań (nazywane w dalszej części pracy rozszerzonym modelem). Wykonane badania dotyczące analizy zadań z *Wiedźmina 3* udowodniły, że model jest w stanie wyrażać zadania pochodzące z gier RPG i można go wykorzystywać do analizy istniejących już zadań. Ten rozdział został poświęcony kolejnemu zastosowaniu modelu – generacji zadań fabularnych. W ramach tego rozdziału zaprojektowany został prototyp systemu *SkaldNET*, który wykorzystuje rozszerzony model zadań aby generować zadania fabularne.

4.1. Ogólne informacje o systemie

System *SkaldNET* jest biblioteką przeznaczoną do wykorzystania w środowisku .NET. Biblioteka ta pozwala na wygenerowanie zadań fabularnych zgodnych z rozszerzonym modelem. Wraz z biblioteką powstała komplementarna aplikacja testowa wykorzystująca ją do wygenerowania i wizualizacji przykładowych zadań fabularnych.

W trakcie projektowania systemu *SkaldNET* inspirowano się systemem *CONAN* (omówionym w rozdziale 2.1.2). Oba systemy przyjmują podobne założenia wobec ogólnego sposobu wykorzystania systemu. Użytkownik sam projektuje świat gry, który będzie wejściem do procesu generacji. Na podstawie wprowadzonego stanu początkowego świata oraz wybranej strategii (rodzaju zadania) system generuje zadanie fabularne i zwraca je użytkownikowi.

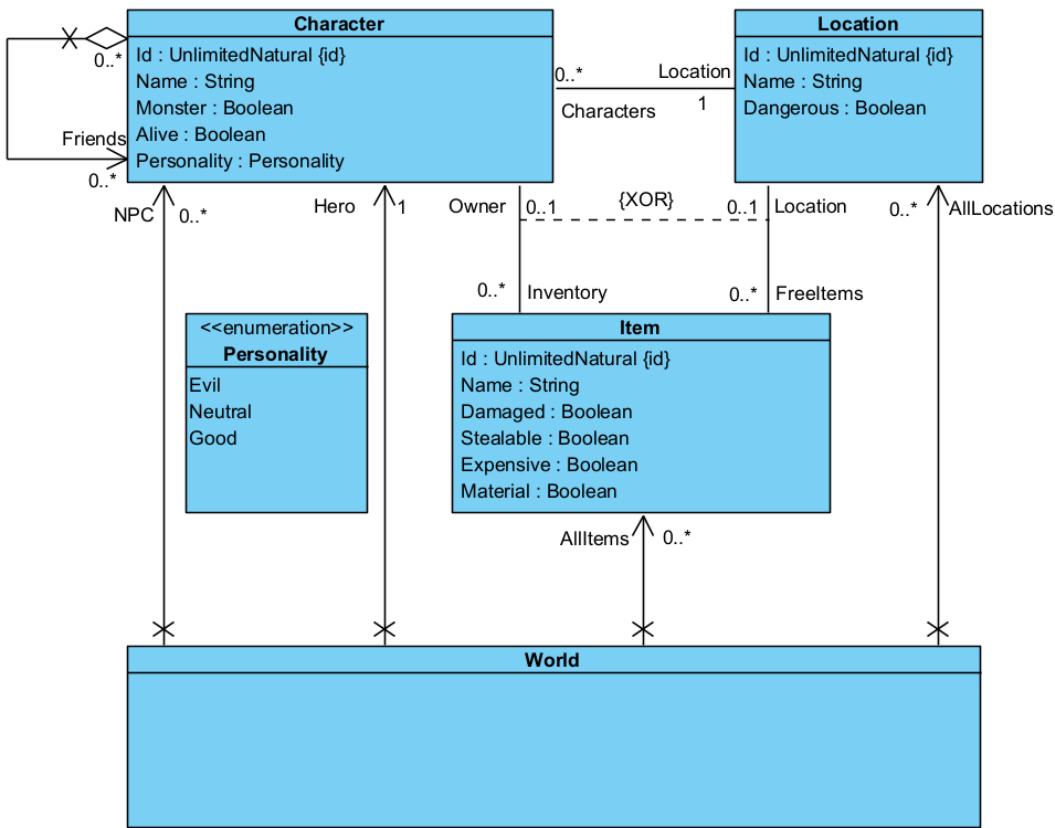
Wygenerowane przez system zadania wyrażone są jako obiekty zdefiniowane w bibliotece, a nie jako obrazki png, co jest dużo bardziej elastyczne w porównaniu do zwracania obrazków w określonym formacie. Pozwala to bowiem użytkownikowi na zdefiniowanie własnego sposobu wizualizacji zadań oraz ich obróbki lub analizy programistycznej. Zgodnie z tą ideą, to aplikacja testująca bibliotekę konwertuje drzewa-zadania z postaci obiektów biblioteki do formatu dokumentów wspieranych przez narzędzie do wizualizacji grafów *Graphviz*¹.

4.2. Model świata

Na potrzeby systemu *SkaldNET* został opracowany model świata widoczny na rysunku 4.1. Wyróżniono trzy rodzaje elementów świata gry:

- lokacje (ang. *location*);
- postaci (ang. *characters*);

¹<https://graphviz.org/>

Rysunek 4.1: Model świata w systemie *SkaldNET*

- przedmioty (ang. *items*).

Lokacje są elementami świata reprezentującymi miejsca, w których może znaleźć się bohater lub inne postaci w świecie gry. Ich cechy zostały opisane w tabeli 4.1. Prezentowany model świata zakłada, że po świecie bohater może poruszać się bez ograniczeń, tj. nie istnieje para lokacji pomiędzy którymi bohater nie może się bezpośrednio przemieścić. Innymi słowy połączenia pomiędzy lokacjami tworzą graf pełny.

Postaci to elementy świata reprezentujące bohatera lub postaci niezależne (NPC), z którymi bohater może wejść w interakcję. Postaci zawsze znajdują się w jakiejś lokacji i mogą posiadać przedmioty w tzw. ekwipunku (ang. *inventory*). Dodatkowo potrafią uznawać za przyjaciół inne postaci, co sprawi, że będą niechętne do zlecania niektórych rodzajów zadań przeciwko nim. Wszystkie cechy postaci zostały opisane w tabeli 4.2.

Przedmiotami nazywamy elementy świata reprezentujące przedmioty, z którymi bohater może wejść w interakcję. Przedmioty mogą znajdować się w jakiejś lokacji albo (alternatywna rozłączna) być w ekwipunku postaci należącej do świata gry. Wszystkie cechy przedmiotów zostały opisane w tabeli 4.3.

Stan świata gry składa się z listy lokacji, postaci niezależnych (NPC), bohatera oraz

Tabela 4.1: Cechy lokacji w systemie *SkaldNET*

Cecha	Opis
Id	Unikalny identyfikator obiektu.
Name (nazwa)	Nazwa lokacji
Dangerous (niebezpieczna)	Czy lokacja jest niebezpieczna
FreeItems (wolne przedmioty)	Przedmioty bez właściciela znajdujące się w danej lokacji
Characters (postaci)	Postaci znajdujące się w danej lokacji

Tabela 4.2: Cechy postaci w systemie *SkaldNET*

Cecha	Opis
Id	Unikalny identyfikator obiektu.
Name (nazwa)	Imię postaci
Alive (żywy)	Określa czy postać jest żywa
Personality (osobowość)	Określa typy zadań jakie może zlecać dana postać (patrz tabela ??)
Friends (przyjaciele)	Postaci, których krzywdzenia nie będzie zlecała dana postać
Location (lokacja)	Miejsce przebywania postaci
Inventory (ekwipunek)	Przedmioty posiadane przez daną postać

przedmiotów. Jest to wystarczająca zawartość, gdyż każdy z elementów świata zawiera cechy opisujące go i jego relacje z innymi istniejącymi równolegle elementami świata (zgodnie z diagramem na rysunku 4.1).

Nie wszystkie cechy przedmiotów, lokacji oraz postaci zostały zdefiniowane w trakcie opracowywania rozszerzonego modelu w rozdziale 3. Podczas opracowywania systemu *SkaldNET* wprowadzono dodatkowe cechy do modelu świata, takie jak np. określenie czy postać jest potworem. Dzięki temu *SkaldNET*, w porównaniu do systemu *CONAN* (patrz rozdział 2.1.2), nie wybiera elementów świata w sposób całkowicie losowy, ale najpierw je filtryje. Pozwala to na selekcję bardziej odpowiednich elementów świata w danym kontekście i otrzymanie dzięki temu bardziej wiarygodnych zadań.

Prototyp systemu *SkaldNET* został opracowany z myślą o niewielkich światach, tj. takich, w których jest możliwa wiarygodna interakcja ze sobą większości elementów świata. Znacząca ilość

Tabela 4.3: Cechy przedmiotów w systemie *SkaldNET*

Cecha	Opis
Id	Unikalny identyfikator obiektu.
Name (nazwa)	Nazwa przedmiotu
Damaged (uszkodzony)	Czy przedmiot jest uszkodzony
Stealable (możliwy do ukradnięcia)	Czy przedmiot jest możliwy do ukradnięcia
Expensive (drogi)	Czy przedmiot posiada dużą wartość
Material (zasób)	Czy przedmiot jest zasobem
Location (lokacja)	Miejsce przebywania przedmiotu jeżeli nie ma właściciela
Owner (właściciel)	Postać w której ekwipunku znajduje się przedmiot.

popularnych gier RPG wykorzystuje światy niespełniające tego wymagania (np. *Wiedźmin 3*). Nie oznacza to jednak, że *SkaldNET* nie może zostać wykorzystany w kontekście generowania zadań do gry z dużym światem. Wystarczy na wejściu do systemu podawać wycinki świata odpowiednich rozmiarów.

Na potrzeby prezentacji wyników działania systemu *SkaldNET* opracowany został przykładowy wycinek świata utrzymanego w klimatach średniowiecznego fantasy. Świat ten składa się z:

- twierdzy dobrych (pod względem osobowości, która zgodnie z tabelą 4.5 zawęża ilość możliwych do zlecenia rodzajów zadań) paladynów (*Ravenskeep*);
- miasteczka neutralnych mieszkańców (*Golden Village*);
- obozowiska neutralnych myśliwych (*Hunters' camp*);
- obozu złych gołębów (*Goblin camp*);
- niebezpiecznego lasu zawierającego potwory (pająki) oraz mieszkańca miasta *Elfhaima* i gołębina *Grombocka*.

Wymienione lokacje zawierają przedmioty oraz postaci, a każdy ze zdefiniowanych elementów świata ma określone cechy widoczne w tabelach 4.1, 4.2, 4.3, tak aby utworzyć spójny świat.

4.3. Decyzje podejmowane w trakcie generacji

System *SkaldNET* przyjmuje na wejściu stan początkowy świata oraz strategię i na ich podstawie generuje zadania. Generacja odbywa się w oparciu o reguły rozszerzonego modelu zdefiniowanego w rozdziale 3.6. Opracowany prototyp systemu wykorzystuje tylko widoczny w tabeli 4.4 podzióbior strategii (rodzajów zadań) rozszerzonego modelu. Podzióbior ten jest wystarczający do zaprezentowania koncepcji działania systemu i pokazania, że jest on w stanie generować rozbudowane zadania fabularne.

4.3.1. Generacja drzewa-zadania w trakcie przechodzenia

Proces generacji zadania odbywa się w trakcie przechodzenia *preorder*² drzewa-zadania, czyli zgodnie ze sposobem czytania zadania (nie uwzględniając wprowadzonych w rozdziale 3.6 operatorów *or* i *twist*).

W trakcie przechodzenia powstającego drzewa-zadania *SkaldNET* rozwija odwiedzane węzły, czyli tworzy i dodaje do poddrzewa ich dzieci. W trakcie tego procesu podejmuje decyzje dotyczące:

- selekcji dzieci węzła, w jakim się znajduje w trakcie przechodzenia;
- selekcji elementów świata wchodzących w interakcję z bohaterem w ramach akcji reprezentowanej przez węzły.

²Kolejność węzłów jest wyznaczana poprzez kolejność odpowiadających ich akcji w *rozwijanej* sekwencji akcji. Graficznie będzie przedstawiana od lewej do prawej.

Tabela 4.4: Strategie wspierane przez Skald-NET

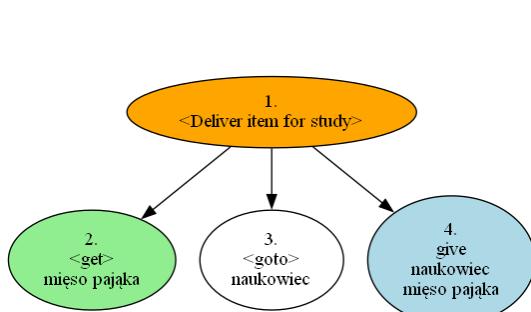
Id	Motywacja	Strategia modelu	Skrócona nazwa	Sekwencja akcji
S01	Knowledge	Deliver item for study	Deliver	get many, goto, "give"
S02	Knowledge	Spy	Spy	spy
S03	Knowledge	Interview NPC	Interview	goto, "listen", goto, "report"
S04	Comfort	Obtain luxuries	ObtainLuxuries	get many, goto, "give"
S05	Comfort	Kill pests	KillPests	goto, "damage", goto, "report"
S06	Reputation	Obtain rare items	ObtainRare	get many, goto, "give"
S07	Reputation	Kill enemies	KillEnemies	goto, kill, goto, "report"
S08	Reputation	Kill enemies (2)	KillEnemies2	goto, or(kill)(twist(subquest)), goto, "report"
S09	Reputation	Visit a dangerous place	VisitDangerous	goto, goto, "report"
S10	Serenity	Revenge, Justice	Revenge	goto, "damage"
S11	Serenity	Revenge, Justice (2)	Revenge2	goto, or("damage")("")
S12	Serenity	Check on NPC(1)	Check1	goto, "listen", goto, "report"
S13	Serenity	Check on NPC(2)	Check2	goto, "subquest", goto, "report"
S14	Serenity	Check on NPC(3)	Check3	goto, "take", goto, "give"
S15	Serenity	Recover lost/stolen item	Recover	get many, goto, "give"
S16	Protection	Attack threatening entities	AttackThreat	goto, "damage", goto, "report"
S17	Protection	Treat or repair	Repair	goto, "repair"
S18	Protection	Attack threatening entities (2)	AttackThreat2	goto, or("damage")("", goto, "report")
S19	Protection	Attack threatening entities (3)	AttackThreat3	goto, or(kill)(twist(subquest)), goto, "report"
S20	Protection	Guard Entity	Guard	goto, "defend"
S21	Conquest	Attack enemy	AttackEnemy	goto, "damage"
S22	Conquest	Steal stuf	StealStuff	goto, steal, goto, "give"
S23	Wealth	Gather raw materials	GatherMaterials	goto, get many
S24	Wealth	Steal valuables for resale	StealValuables	goto, steal

Tabela 4.5: Przyporządkowanie strategii osobowościom

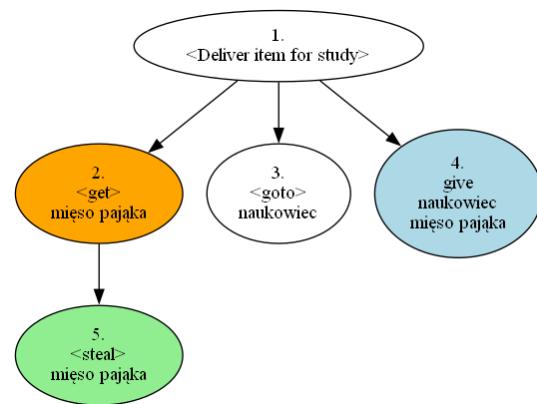
Id	Skrócona nazwa	Osobowość zleceniodawcy		
		dobry (good)	neutralny (neutral)	zły (evil)
S01	Deliver	✓	✓	✗
S02	Spy	✗	✓	✓
S03	Interview	✓	✓	✗
S04	ObtainLuxuries	✗	✓	✓
S05	KillPests	✓	✓	✓
S06	ObtainRare	✗	✓	✓
S07	KillEnemies	✗	✓	✓
S08	KillEnemies2	✗	✓	✓
S09	VisitDangerous	✗	✓	✓
S10	Revenge	✗	✓	✓
S11	Revenge2	✗	✓	✓
S12	Check1	✓	✓	✓
S13	Check2	✓	✓	✓
S14	Check3	✓	✓	✓
S15	Recover	✓	✓	✓
S16	AttackThreat	✓	✓	✓
S17	Repair	✓	✓	✗
S18	AttackThreat2	✓	✓	✓
S19	AttackThreat3	✓	✓	✓
S20	Guard	✓	✓	✗
S21	AttackEnemy	✗	✗	✓
S22	StealStuff	✗	✗	✓
S23	GatherMaterials	✓	✓	✗
S24	StealValuables	✗	✓	✓

Selekcja dzieci polega na wyborze jednej spośród alternatywnych sekwencji akcji wyrażenia nieterminalnego odpowiadającego węzłowi, który jest *rozwijany* przez generator. Razem z nią **od razu** następuje selekcja spośród elementów świata. Na rysunkach 4.2 i 4.3 widzimy kolejne kroki generacji przykładowego zadania. Pomarańczowym kolorem oznaczono poprzednio *rozwijany* przez generator węzeł, a zielonym nastepny węzeł, który będzie *rozwijany*.

W każdym momencie pracy generatora wszystkie dotychczas utworzone węzły drzewa-zadania posiadają przypisane elementy świata. Oznacza to, że posiadają swoją interpretację w świecie gry. Jednakże interpretacja jest pozbawiona szczegółów, które zostają dodane w momencie ich *rozwijania*. Dla przytoczonego przykładu generacji zadania, po wykonaniu pierwszego kroku (rysunek 4.2) węzły 2, 3 i 4 posiadają już przypisane elementy świata gry. Wiadomo, że w zadaniu bohater zdobędzie mięso pajaka oraz pójdzie do naukowca. Nie wiadomo jednak w jaki sposób tego dokona. Te detale zostaną wypełnione dopiero jak generator *rozwinięte* kolejne węzły. W kolejnym kroku (rys. 4.3) widzimy, że generator podjął decyzję, że mięso zostanie zdobyte dzięki wykonaniu akcji złożonej *steal*.



Rysunek 4.2: Początek generacji zadania o strategii Deliver item for study

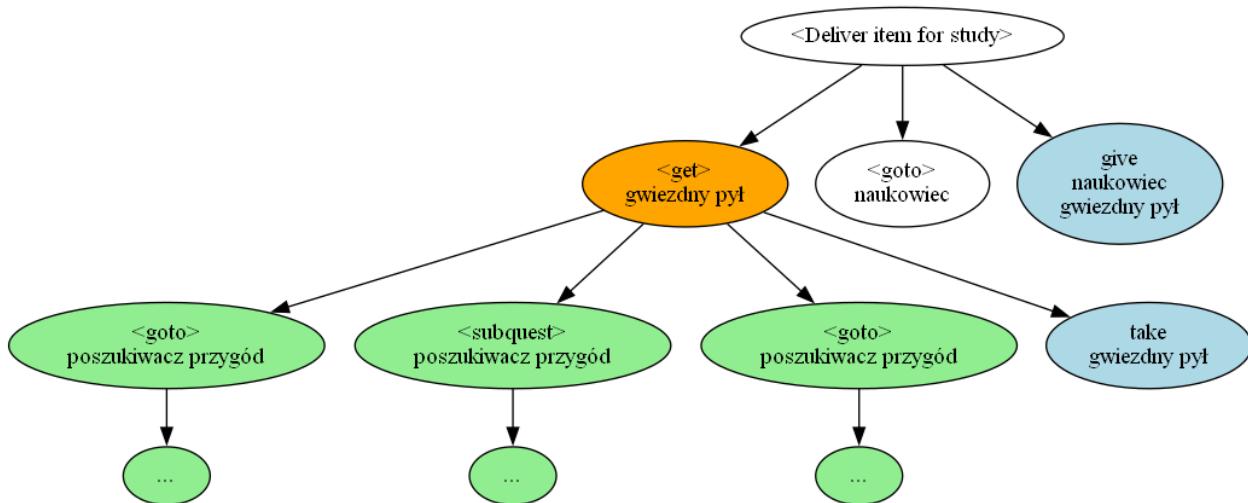


Rysunek 4.3: Drugi krok generacji zadania o strategii Deliver item for study

Selekcja elementów świata dla dzieci jest podejmowana *wysoko* w drzewie, podczas *rozwijania* węzłów odpowiadających strategiom i niektórym akcjom złożonym. Dzięki temu wybory generatora mogą być wspierane poprzez wysokopoziomowy kontekst czynności, jaki nadają im akcje złożone i strategie. Zobrazowane to zostało na rysunku 4.4, na którym widać, że dokonując wyborów wchodząc do pomarańczowego wierzchołka możemy mieć pośredni wpływ na to, jak zostaną *rozwinięte* zielone poddrzewa.

W modelu rozszerzonym na akcje atomowe (znajdujące się tylko w liściach drzewa-zadania) zostały nałożone warunki wstępne dotyczące elementów świata (patrz tabela 3.1). Ponieważ *SkaldNET* dokonuje decyzji elementów świata *wysoko* w drzewie, to muszą być jednocześnie wprowadzone dodatkowe mechanizmy zapewniające spełnienie warunków wstępnych *niżej* w drzewie. W tym celu zaproponowano trzy mechanizmy, które korzystają z siebie wzajemnie w celu zapewnienia spójności (wykonywalności) i wiarygodności generowanych zadań fabularnych:

- Mechanizm aktualizacji stanu świata gry w trakcie przechodzenia drzewa.
- Mechanizm warunków wstępnych akcji złożonych.
- Mechanizm obostrzeń podejmowanych decyzji w poddrzewach.



Rysunek 4.4: Decyzje podejmowane wyżej w drzewie (węzeł pomarańczowy) mają wpływ na całe poddrzewa (węzły zielone).

4.3.2. Kontrola stanu świata gry w trakcie generacji

Wykorzystanie tego samego sposobu przechodzenia drzewa-zadania do jego generacji oraz późniejszego odczytu przez użytkownika ma zasadniczą zaletę. Pozwala bowiem na przechowywanie i aktualizowanie *aktualnego* stanu świata gry w trakcie generacji. Powoduje to, że w trakcie generacji przez cały czas znany jest *aktualny* stan świata, czyli równoważny jego stanowi w danym węźle podczas czytania wynikowego drzewa-zadania.

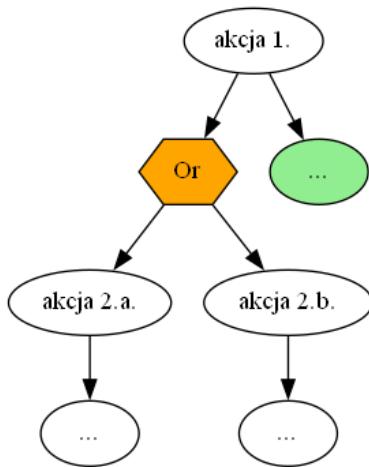
Znajomość *aktualnego* stanu świata gry otrzymywana jest poprzez wykonanie na nim efektów końcowych akcji aktualnie odwiedzanego węzła. Należy zwrócić uwagę na to, że w rozszerzonym modelu tylko liście drzewa mają efekty końcowe, a tymi liśćmi mogą być tylko akcje atomowe.

Wraz z rozszerzeniem modelu zaproponowane zostały akcje zmieniające sposób przemieszczania się po drzewie (patrz rozdział 3.6). Operator *or* sprawia, że użytkownik wybiera tylko jedno z dzieci danego węzła. Zwiększa to ilość rozpatrywanych *aktualnych* stanów świata gry. Natomiast operator *twist* zmniejsza liczbę *aktualnych* stanów świata gry.

Wchodząc do węzła *or* należy wykonać kopię *aktualnego* stanu świata, po jednej na każdy alternatywny przebieg zadania. *Rozwijając* kolejne dzieci węzła *or* należy analizować wykonaną uprzednio kopię. Po *rozwinieciu* ich wszystkich *aktualny* stanem świata gry są wszystkie zmodyfikowane przez dzieci kopie. Przykład opisanej sytuacji znajduje się na rysunku 4.5, gdzie po ukończeniu *rozwijania* całego poddrzewa *or* należy rozpatrywać dwa stany świata – jeden po wyborze akcji 2.a. i drugi po wyborze akcji 2.b.

Operator *twist* sprawia, że zadanie kończy się w momencie, kiedy wracalibyśmy z odpowiadającemu mu węzła. Występuje on tylko w połączeniu z operatorem *or* i powoduje, że kopia świata rozpatrywana w jego poddrzewie może zostać odrzucona. Obrazuje to przykład na rysunku 4.6, w którym po *rozwinieciu* całego poddrzewa *or* analizowany jest tylko stan świata po przejściu akcji 2.a.

Warto zauważyć, że tworzenie kopii świata nie musi być wykonywane dla pustych poddrzew (zawierających tylko akcję *none*). Ten przypadek został zobrazowany na rysunku 4.7.



Rysunek 4.5: W węźle *or* wykonuje się kopie stanu dla każdego dziecka. Po powrocie z węzła *or* wszystkie kolejne *rozwijane* akcje (zielone) muszą brać wszystkie kopie pod uwagę.

Zaprezentowany mechanizm kontroli stanu świata można uprościć ze względu na specyfikę wszystkich zdefiniowanych w rozszerzonym modelu sekwencji akcji wykorzystujących operatory *or* i *twist*. Ponieważ operator *or* jest wykorzystywany tylko w połączeniu z akcją *none* (patrz rysunek 4.7) albo z operatorem *twist*, to generując zadania można zawsze rozpatrywać tylko jeden *aktualny stan świata*. Jednakże **aktualny stan świata musi zostać zapamiętany podczas wejścia do węzła *twist* i przywrócony po rozpatrzeniu całego poddrzewa, którego jest korzeniem**.

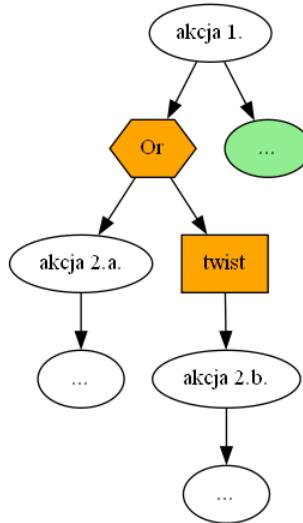
4.3.3. Warunki wstępne akcji złożonych i strategii

Mechanizm warunków wstępnych dzieli się na dwie części. Pierwsza z nich przyporządkowuje warunki wstępne wymagane, aby *rozwinać* dany węzeł przy pomocy odpowiedniej sekwencji akcji (wybór dzieci). Druga część odnosi się do wyboru elementów świata. W ramach niej przypisano warunki wstępne wszystkim węzłom, w trakcie których *rozwijania* dokonywana jest selekcja elementów świata.

W tabeli 4.6 znajduje się opis wszystkich możliwych wyborów, jakie mogą być podejmowane w ramach wyboru dzieci dla węzła akcji złożonej. Wybór ten jest podejmowany na podstawie widocznych w tabeli elementów świata, które zostały już wybrane *wyżej* w drzewie. Przypisane sekwencjom akcji warunki wstępne muszą zostać spełnione, aby dana sekwencja mogła posłużyć do *rozwijania* węzła. Warunki te nie są wzajemnie wykluczające się, a w przypadku możliwości wybrania kilku sekwencji akcji wybór jest podejmowany losowo.

Podejmowanie selekcji elementów świata *wysoko* w drzewie pozwala na wykorzystanie wysokopoziomowego kontekstu *rozwijanego* węzła w celu selekcji bardziej wiarygodnych elementów stanu świata.

Druga część mechanizmu warunków wstępnych przyporządkowała węzłom warunki wstępne odnoszące się do wybieranych elementów świata. Warunki te stanowią zestaw cech, jakie muszą być spełnione w *aktualnym* świecie gry, aby dany element mógł zostać wybrany. Selekcja zatem polega polega na filtracji wybieranych elementów świata i losowym wyborze spośród elementów spełniających wszystkie warunki wstępne (z uwzględnieniem mechanizmu obostrzeń,



Rysunek 4.6: W węźle *or* wykonuje się kopie stanu dla każdego dziecka. Po powrocie z węzła *or* wszystkie kolejne *rozwijane* akcje (zielone) muszą brać pod uwagę tylko kopie, które nie trafiły do węzła *twist*.

omówionego w podrozdziale 4.3.4).

Zdecydowana większość występujących selekcji elementów świata podejmowana jest podczas *rozwijania* strategii zgodnie z tabelą 4.7³. Tabela ta zawiera przyporządkowane kolejnym strategiom warunki wstępne, które odwołują się do cech wybieranych elementów z modelu świata (patrz rysunek 4.1).

Widoczne w tabeli 4.7 warunki wstępne odnoszą się także do dodanych podczas opracowywania systemu *SkaldNET* cech elementów świata. Daje to możliwość wykorzystywania ich do wyboru bardziej wiarygodnych w danym kontekście elementów świata. Przykładowo widzimy dla strategii S16 *AttackThreat*, że atakowana postać nie może być przyjacielem zleceniodawcy. Innym przykładem jest wykorzystanie w strategii S09 *VisitDangerousPlace* tylko niebezpiecznych lokacji.

Rozwijanie strategii następuje podczas *rozwijania* akcji złożonej *subquest*. W tabeli 4.6 możemy zobaczyć, że sekwencja akcji

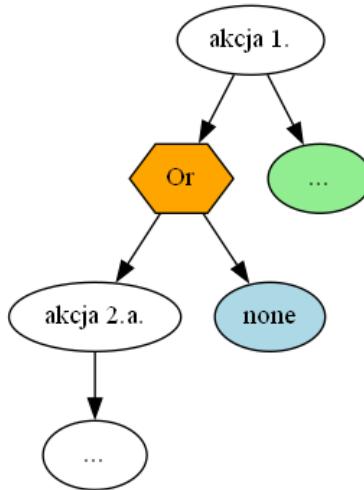
goto, QUEST, “goto”

jest uwarunkowana istnieniem możliwego do wygenerowania podzadania. Generator sprawdza to poprzez próbę *rozwinienia* w losowej kolejności wszystkich strategii zgodnych z osobowością postaci npc (patrz tabela 4.5). W momencie kiedy ktoś z nich uda się *rozwinąć*, to jest nią zastępowany węzeł *QUEST*. To, czy uda się jakąś strategię *rozwinąć* jest równoznaczne ze znalezieniem elementów *aktualnego* świata spełniających warunki wstępne z tabeli 4.7 oraz mechanizmu obostrzeń opisanego w podrozdziale 4.3.4.

Poza strategiami występują jeszcze dwie akcje złożone, w trakcie *rozwijania* których dokonywana jest selekcja elementów świata. Jest to sekwencja rozwijająca akcję złożoną *get*:

goto, get, subquest, goto, “exchange”

³Przyjęto uproszczenie *rozwijające* akcję *get many* pojedynczą akcją *get*.



Rysunek 4.7: W węźle *or* nie trzeba wykonywać kopii stanu dla węzłów pustych.

W trakcie jej *rozwijania* wybierany jest Przedmiot p2, który musi spełniać warunki:

- p2 to nie p.
- p2 nie należy do p.owner.

W analogiczny sposób wprowadzono wymagania wstępne co do postaci npc lub przedmiotu p2 wybieranych wraz z wyborem poszczególnych sekwencji akcji dla akcji złożonej *learn*:

goto, subquest, “listen”
get, goto, give, “listen”

która ma przyporządkowane następujące warunki wstępne:

- npc nie jest potworem.
- npc żyje.
- p2 nie należy do npc.

W momencie, w którym dokonujemy filtracji spośród elementów składowych świata, należy rozpatrzyć sytuację, w której nie istnieją elementy spełniające naszych oczekiwani. W przypadku rozpatrywanych strategii sprawa jest dość prosta, ponieważ można po prostu spróbować wybrać inną strategię. W przypadku, kiedy żadna strategia nie może zostać wykorzystana, można *rozwinąć* akcję *subquest* wykorzystując sekwencję akcji *goto*, co jest zgodne z rozszerzonym modelem. Sytuacja, w której nie da się wybrać żadnej strategii w korzeniu zadania jest mało prawdopodobna i może oznaczać zbyt mały rozmiar świata wejściowego. W przypadku niemożliwości *rozszerzenia* akcji złożonej można przyjąć taktykę rozpoczęcia generacji całego zadania od nowa ze względu na szybki czas generacji zadań.

Tabela 4.6: Sposób wyboru sekwencji akcji dla akcji złożonych

Akcja	Wybrane wcześniej elementy świata	Sekwencja akcji	Warunki wstępne
goto	Lokacja l albo Postać npc albo Przedmiot p	“”	bohater jest w l, npc.location, p.location lub p.owner.location w <i>aktualnym</i> świecie
		“explore”	pierwsze rozwinięcie akcji goto w całym zadaniu
		learn, “goto”	bohater nie jest ani w l, ani w npc.location, ani w p.location, ani w p.owner.location w aktualnym świecie
subquest	Postać npc	goto	wygenerowano już maksymalną ilość podzadań lub nie istnieje możliwe do wygenerowania podzadanie dla npc
		goto, QUEST, “goto”	nie wygenerowano jeszcze maksymalnej ilości podzadań oraz istnieje możliwe do wygenerowania podzadanie dla npc
		“”	sekwenca nigdy nie jest wybierana
learn	Lokacja l albo Postać npc albo Przedmiot p	“”	wcześniej rozwinięto learn do tego samego elementu świata gry
		goto, subquest, “listen”	Nie rozwinięto wcześniej ani akcji goto, ani akcji learn do tego elementu świata gry
		goto, get, “read”	
get	Przedmiot p	get, goto, “give”, “listen”	
		“”	bohater posiada p
		goto, subquest, goto, “take”	
		steal	bohater nie posiada p, p ma posiadacza
		goto, “get”, subquest, goto, “exchange”	
steal	Przedmiot p	goto, “gather”	bohater nie posiada p, p nie ma posiadacza
		goto, “stealth”, “take”	p da się ukraść,
		goto, kill, “take”	zawsze można wybrać tę sekwencję

4.3.4. Mechanizm obostrzeń – utrzymanie spójności zadania

Opisany w podrozdziale 4.3.3 mechanizm warunków wstępnych nie wystarcza do stworzenia spójnego i wykonywalnego zadania. Można to zaobserwować analizując fragmenty zadania znajdujące się na rysunku 4.8. Przedstawiają one sytuację, w której bohater atakuje zleceniodawcę, aby otrzymać przedmiot, który później ma dostarczyć zleceniodawcy. Decyzja podjęta podczas *rozwijania* akcji oznaczonej kolorem zielonym spowodowała, że wynikowe drzewo-zadanie posiada dwie pomarańczowe akcje, które są sprzeczne ze sobą.

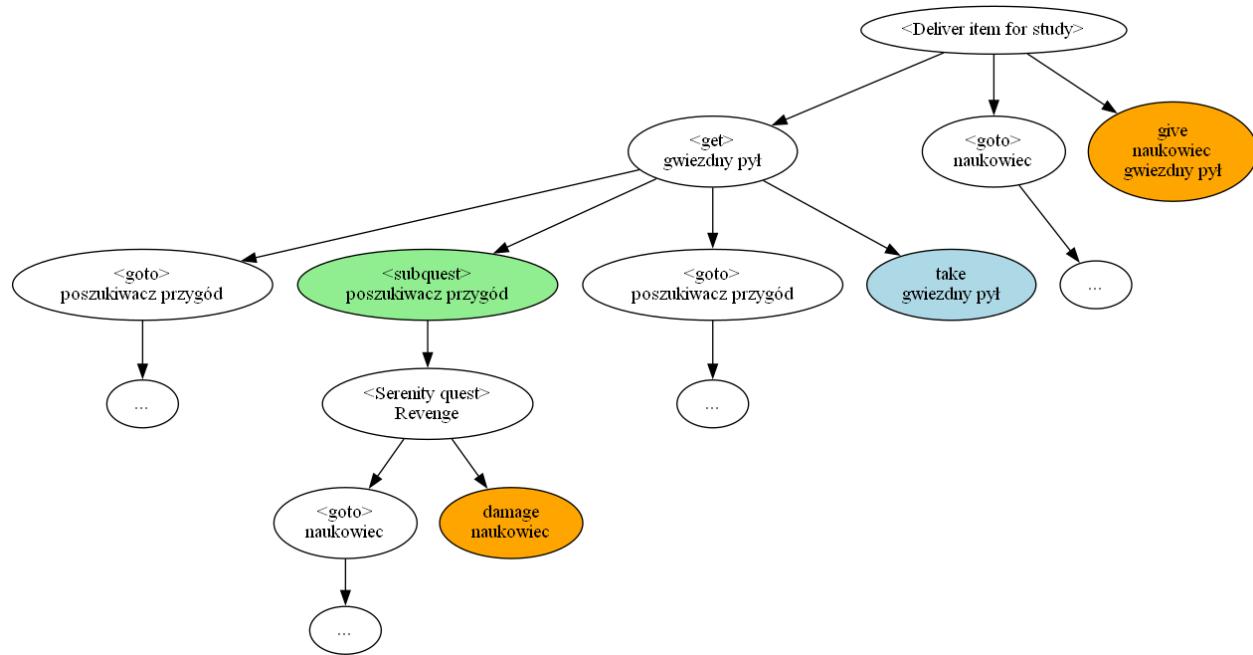
Mechanizm obostrzeń ma na celu zapobieganiu sytuacjom analogicznym do opisanej wyżej. Pozwala on na zabronienie podejmowania decyzji w poddrzewach, które miałyby określony skutek. Skutek ten jest definiowany jako wystąpienie węzła odpowiadającego zabronionej akcji z danym elementem świata gry. Innymi słowy, mechanizm obostrzeń przechowuje zbiory par (element świata, zabronione rodzaje akcji), które nie mogą zostać dodane do zadania w odpowiadających poddrzewach na skutek nowo podjętych decyzji.

Aby nie dopuścić do wygenerowania się wadliwego zadania z przytoczonego przykładu zasadnym jest zastosowanie mechanizmu obostrzeń jak na rysunku 4.9. Widzimy na nim kolorowe linie wprowadzające obostrzenia na wyznaczonych przez nich poddrzewach. Poniżej fioletowej linii zabronione zostało zaatakowanie naukowca (naukowiec, “damage” | “kill” | “kill”⁴) oraz zmiana właściciela gwiazdnego pyłu (gwiazdny pył, “get” | “gather” | “give” |

⁴Zapis akcja 1 | akcja 2 należy rozumieć jako alternatywę akcji 1 oraz akcji 2

Tabela 4.7: Warunki wstępne strategii

Id	Skrócona nazwa	Elementy świata	Warunki wstępne
S01	DeliverForStudy	Zleceniodawca z Przedmiot p	z żyje, z nie jest potworem, posiadaczem p nie jest z
S02	Spy	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z
S03	Interview	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z
S04	ObtainLuxuries	Zleceniodawca z Przedmiot p	z żyje, z nie jest potworem, p nie należy do z
S05	KillPests	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc potworem, npc nie jest przyjacielem z
S06	ObtainRareItems	Zleceniodawca z Przedmiot p	z żyje, z nie jest potworem, p nie należy do z, p jest wartościowy
S07	KillEnemies	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc nie jest przyjacielem z
S08	KillEnemies2	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc nie jest przyjacielem z
S09	VisitDangerousPlace	Zleceniodawca z Miejsce l	z żyje, z nie jest potworem, l jest niebezpieczne, z nie jest w l
S10	Revenge	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc nie jest przyjacielem z
S11	Revenge2	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc nie jest przyjacielem z
S12	Check1	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc jest przyjacielem z
S13	Check2	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc jest przyjacielem z
S14	Check3	Zleceniodawca z Postać npc Przedmiot p	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc jest przyjacielem z, p należy do npc, p nie należy do z
S15	Recover	Zleceniodawca z Przedmiot p	z żyje, z nie jest potworem, p nie należy do z
S16	AttackThreat	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest przyjacielem z
S17	Repair	Zleceniodawca z Przedmiot p	z żyje, z nie jest potworem, p należy do z lub jest w lokacji w której jest z, p jest uszkodzony
S18	AttackThreat2	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc nie jest przyjacielem z
S19	AttackThreat3	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc nie jest przyjacielem z
S20	Guard	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc nie jest potworem, npc jest przyjacielem z, npc jest w lokacji niebezpiecznej
S21	AttackEnemy	Zleceniodawca z Postać npc	z żyje, z nie jest potworem, npc żyje, npc to nie z, npc nie jest potworem, npc nie jest przyjacielem z
S22	StealStuff	Zleceniodawca z Przedmiot p	z żyje, z nie jest potworem, p nie należy do z, p ma właściciela, p da się ukrąść
S23	GatherMaterials	Zleceniodawca z Przedmiot p	z żyje, z nie jest potworem, p nie ma właściciela, p jest zasobem
S24	StealValuables	Zleceniodawca z Przedmiot p	z żyje, z nie jest potworem, p nie należy do z, p ma właściciela, p da się ukrąść, właściciel p nie jest przyjacielem z



Rysunek 4.8: Przykład niepoprawnego zadania, które można otrzymać stosując tylko mechanizm aktualizacji stanu świata oraz mechanizm warunków wstępnych

“take”). Poniżej niebieskiej linii zabroniono zaatakowania poszukiwacza przygód, a poniżej czerwonej – zaatakowania bandyty. Warto zauważyć, że poddrzewa oznaczone czerwoną linią są składowymi poddrzew oznażonych niebieską i fioletową linią, co oznacza, że obowiązują w nich obostrzenia wszystkich trzech linii.

Przyglądając się opisanemu przykładowi można dojść do błędного wniosku, że przedstawia on łamiące mechanizm obostrzeń zadanie. Wnioskowanie to jednak nie jest poprawne, ponieważ **obostrzenia wpływają na podejmowane w poddrzewach decyzje dotyczące jedynie selekcji elementów świata**. Na rysunku 4.9 w poddrzewach oddzielonych czerwoną linią występuje węzeł *damage bandyta* pomimo zakazu ataku bandyty. Jest to zgodne z omawianym mechanizmem, ponieważ decyzja o dołączeniu węzła *damage bandyta* została dokonana w trakcie *rozwijania* akcji *subquest* znajdującej się nad czerwoną linią. W analogiczny sposób zakaz nie został naruszony poprzez dodanie akcji *take gwiazdny pył*, ponieważ selekcja tego elementu została dokonana już w węźle *Deliver for study* i spropagowana niżej przez węzeł *<get> gwiazdny pył*.

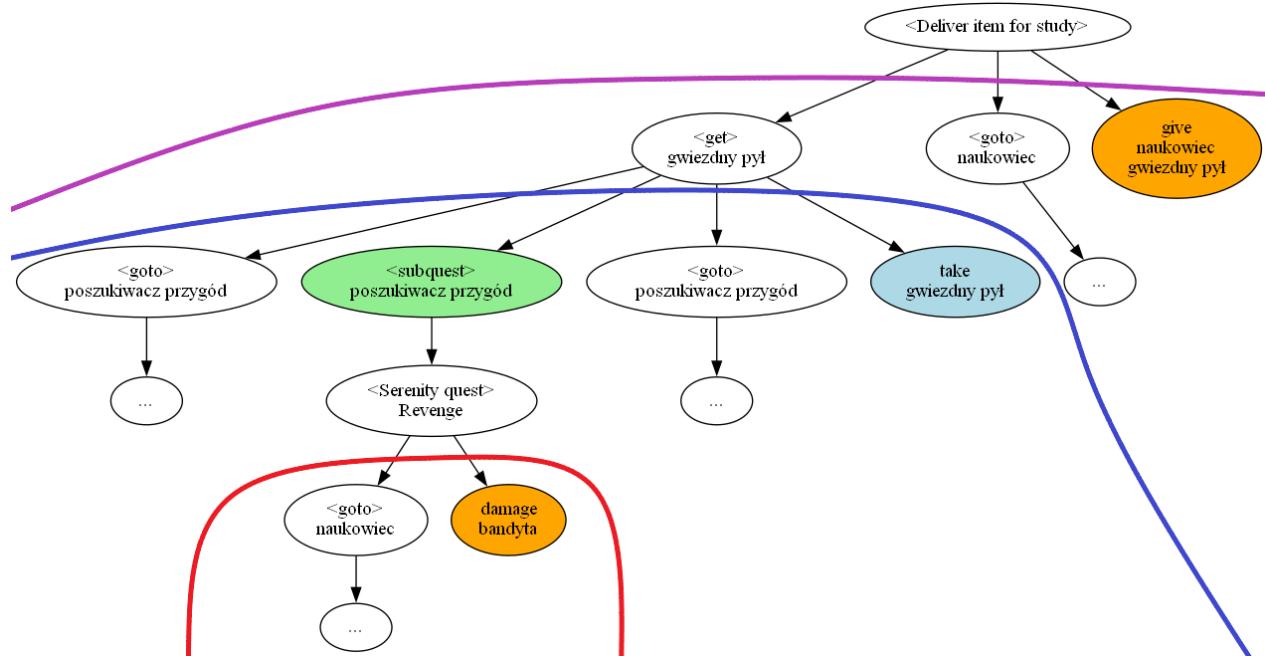
Opracowując system *SkaldNET* zdefiniowano i przyporządkowano obostrzenia wszystkim węzłom dokonującym selekcji zgodnie z tabelą 4.8. W celu uproszczenia zapisu wyekstrahowano ze względu na logiczny skutek następujące obostrzenia:

- *Attack* = “*damage*” | “*kill*” | *kill*
- *ChangeOwner* = *get* | “*gather*” | “*give*” | “*take*” | *steal*

Głównym celem mechanizmu obostrzeń jest zapewnienie, że *nizej* w drzewie będą dodawane tylko takie węzły, które nie spowodują konfliktów *wyżej* w drzewie. Powoduje to, że operator

Tabela 4.8: Obostrzenia wprowadzane przez węzły dokonujące selekcji elementów świata

Id	nazwa lub ciąg	Elementy świata	Obostrzenia na poddrzewa
S01	DeliverForStudy	Zleceniodawca z Przedmiot p	(z, <i>Attack</i>), (p, <i>ChangeOwner</i>)
S02	Spy	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S03	Interview	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S04	ObtainLuxuries	Zleceniodawca z Przedmiot p	(z, <i>Attack</i>), (p, <i>ChangeOwner</i>)
S05	KillPests	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S06	ObtainRareItems	Zleceniodawca z Przedmiot p	(z, <i>Attack</i>), (p, <i>ChangeOwner</i>)
S07	KillEnemies	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S08	KillEnemies2	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S09	VisitDangerousPlace	Zleceniodawca z Miejsce l	(z, <i>Attack</i>),
S10	Revenge	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S11	Revenge2	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S12	Check1	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S13	Check2	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S14	Check3	Zleceniodawca z Postać npc Przedmiot p	(z, <i>Attack</i>), (npc, <i>Attack</i>), (p, <i>ChangeOwner</i>)
S15	Recover	Zleceniodawca z Przedmiot p	(z, <i>Attack</i>), (p, <i>ChangeOwner</i>)
S16	AttackThreat	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S17	Repair	Zleceniodawca z Przedmiot p	(z, <i>Attack</i>), (p, <i>ChangeOwner</i> “repair”)
S18	AttackThreat2	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S19	AttackThreat3	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S20	Guard	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S21	AttackEnemy	Zleceniodawca z Postać npc	(z, <i>Attack</i>), (npc, <i>Attack</i>)
S22	StealStuff	Zleceniodawca z Przedmiot p	(z, <i>Attack</i>), (p, <i>ChangeOwner</i>)
S23	GatherMaterials	Zleceniodawca z Przedmiot p	(z, <i>Attack</i>), (p, <i>ChangeOwner</i>)
S24	StealValuables	Zleceniodawca z Przedmiot p	(z, <i>Attack</i>), (p, <i>ChangeOwner</i>)
Get	goto, get, subquest, goto, “exchange”	Przedmiot p	(p, <i>ChangeOwner</i>)
Learn1	goto, subquest, “listen”	Postać npc	(npc, <i>Attack</i>)
Learn2	get, goto, give, “listen”	Postać npc Przedmiot p	(npc, <i>Attack</i>), (p, <i>ChangeOwner</i>)



Rysunek 4.9: Mechanizm obostrzeń pozwala wprowadzać restrykcje na wybierane elementy świata w poddrzewach.

twist, który powoduje podczas przechodzenia drzewa brak powrotu w górę drzewa, może odrzucić dotychczasowe obostrzenia narzucone wyżej w drzewie.

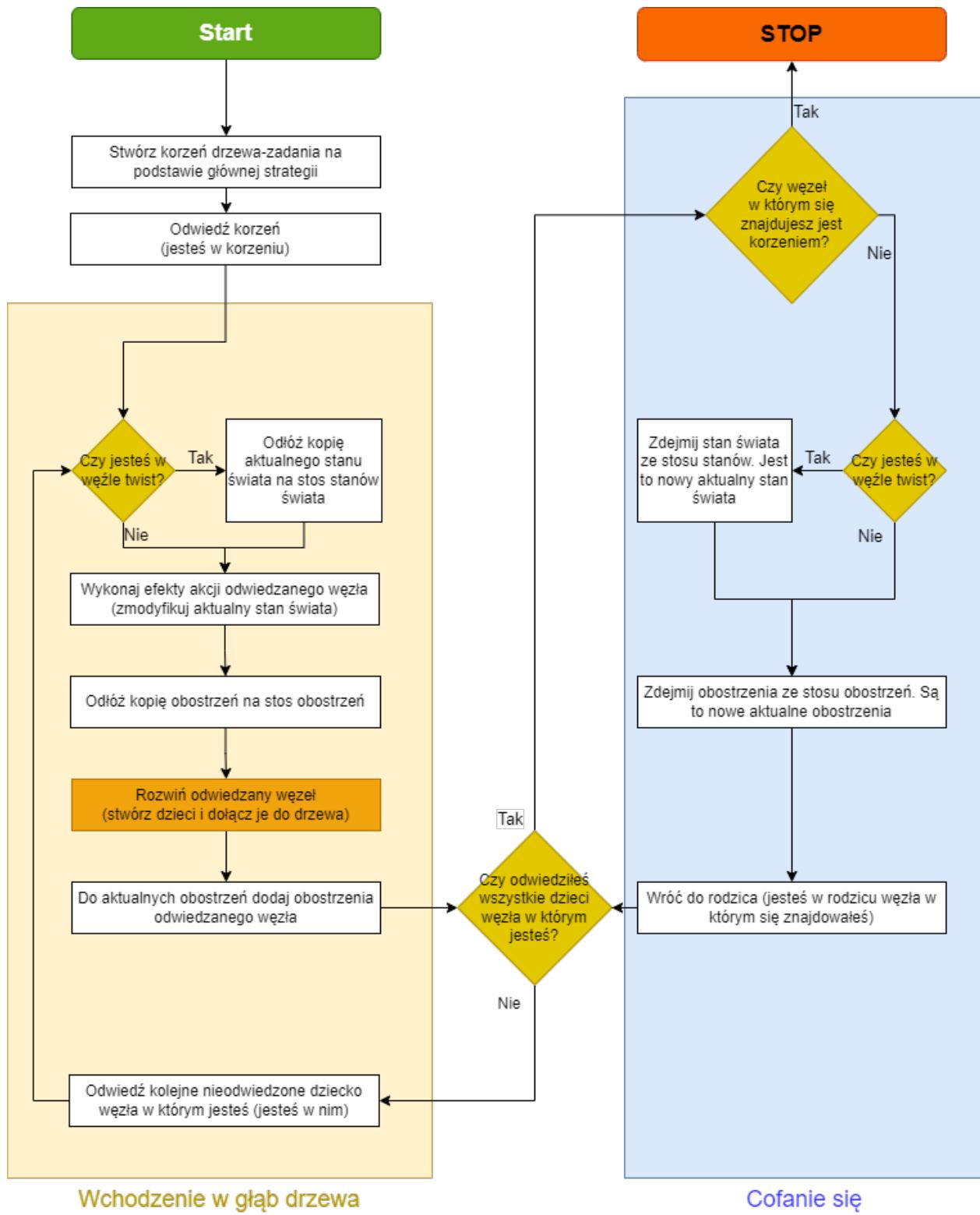
4.4. Wysokopoziomowy algorytm generacji zadania

Ogólna idea, w oparciu o którą zaprojektowany został wykorzystywany przez system *SkaldNET* algorytm generacji zadań, została opisana w podrozdziale 4.3.1. Algorytm ten wykorzystuje wszystkie trzy mechanizmy w celu generacji **spójnych**, czyli nie zawierających sprzeczności, zadań.

Algorytm na wejściu przyjmuje:

- początkowy stan świata (patrz rysunek 4.1);
- maksymalną ilość akcji *learn* oraz *subquest* (parametry wpływające na spełnienie warunków wstępnych z tabeli 4.6);
- główną strategię zadania (czyli strategię wykorzystaną do utworzenia korzenia i jego *rozwinienia*).

Algorytm generacji zadań wykorzystywany przez system *SkaldNET* znajduje się na rysunku 4.10. Algorytm rozpoczyna się od stworzenia korzenia na podstawie głównej strategii. Następnie generator *rozwija* kolejne węzły zadania wchodząc wgłąb drzewa (co oznaczono kolorem żółtym na diagramie). Podczas tego procesu przez cały czas kontrolowany i aktualizowany jest *aktualny* stan świata oraz obostrzenia. Jeżeli generator znajdzie się w liściu drzewa to następuje wycofywanie się (kolor niebieski na diagramie), po czym odwiedza kolejne, nieodwiedzone



Rysunek 4.10: Algorytm generowania zadań

poddzewa. Algorytm kończy się w momencie, w którym generator po przejściu całego drzewa wróci do korzenia.

W opisany algorytmie warto zwrócić szczególną uwagę na krok *rozwijający* odwiedzany węzeł. W tym oznaczonym kolorem pomarańczowym na diagramie (rysunek 4.10) kroku podejmowane są przez generator opisane wcześniej decyzje:

- Wybór sekwencji akcji, której będą odpowiadać dzieci odwiedzanego węzła (patrz podrozdział 4.3.3).
- Selekcja elementów świata na podstawie *aktualnego* stanu świata oraz *aktualnych obstrzeń* (patrz podrozdziały 4.3.3 i 4.3.4).

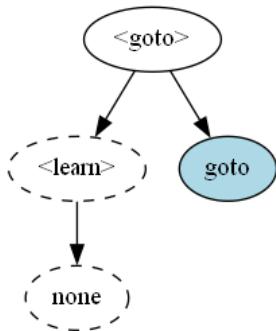
4.5. Wygenerowane zadania

Biblioteka *SkaldNET* zwraca użytkownikowi wygenerowane zadania w postaci obiektów reprezentujących poszczególne węzły drzewa-zadania. Obiekty te zawierają opis słowny węzła, rodzaj akcji oraz listę dzieci. Dzięki takiej formie zwracanych wyników użytkownik aplikacji jest w stanie dostosować sposób wizualizacji zadań do swoich potrzeb.

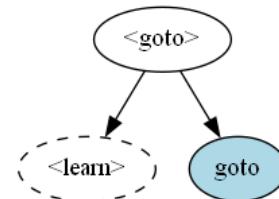
4.5.1. Wizualizacja drzew-zadań

Testowa aplikacja wykorzystująca bibliotekę *SkaldNET* wykorzystuje narzędzie *Graphviz* do generacji obrazków png przedstawiających otrzymywane zadania. Dokonuje tego przechodząc drzewo-zadanie, konwertując kolejne wierzchołki do formatu .dot (wykorzystywanego przez *Graphviz*).

Podczas konwersji do formatu .dot zastosowano kilka dodatkowych zasad pozwalających na otrzymanie czytelniejszych drzew-zadań. Pierwszą z nich jest zaznaczenie przerywaną krawędzią poddrzew, których wszystkie liście zawierają same akcje *none*. Następnie z tych poddrzew usunięto wszystkie wierzchołki poza ich korzeniami. Przykładowe uproszczenie tego rodzaju znajduje się na rysunkach 4.11 i 4.12.

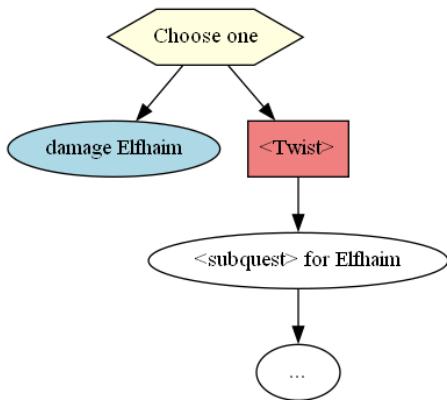


Rysunek 4.11: Puste poddrzewo przed uproszczeniem

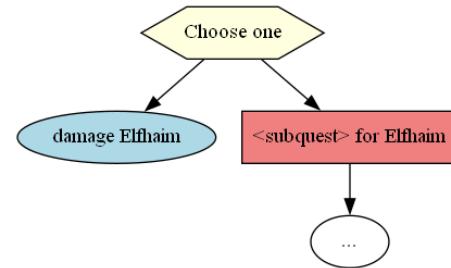


Rysunek 4.12: Puste poddrzewo po uproszczeniu

Kolejnym uproszczeniem jest złączenie węzła *twist* z jego poddrzewem. Złączenie to odbywa się poprzez oznaczenie dziecka węzła *twist* jako czerwonego prostokąta oraz podpięcie go w miejsce węzła *twist* jak na rysunkach 4.13 i 4.14



Rysunek 4.13: Węzeł twist przed złączeniem



Rysunek 4.14: Węzeł twist po złączeniu

4.5.2. Przykłady wygenerowanych zadań

W celu prezentacji możliwości systemu *SkaldNET* wygenerowano przy użyciu aplikacji testowej dwa zadania o strategii głównej *AttackThreat3*. Wybrano tę strategię ze względu na wykorzystanie w niej nowo wprowadzonych operatorów *twist* i *or*.

Poniższe opracowania wygenerowanych zadań zostały opatrzone interpretacją, jaką mógłby im nadać użytkownik systemu *SkaldNET*. Wynikiem działania aplikacji testowej są wyłącznie rysunki 4.15, 4.16. Jednakże poniższe opisy zadań są tym, co mógłby opracować użytkownik systemu chcący wykorzystać *SkaldNET* do offline⁵ PCG zadań.

Pierwsze wygenerowane zadanie widoczne na rysunku 4.15 polega na zaatakowaniu elfa *Elfheima* na polecenie goblinia *Kerklerka*. Bohater musi najpierw wytropić elfa (*gather tracks*), a następnie pójść do niego i pokonać go. Alternatywnie bohater ma możliwość zamiast atakować elfa, pomóc mu pogodzić się z goblinem (<*Interview quest*> for *Elfhaim Interview Kerklerk*).

Kolejne wygenerowane zadanie widoczne na rysunku 4.16 także wykonywane jest dla goblinia *Kerklerka*, który tym razem zleca bohaterowi zabicie Alberta. Bohater może to bezmyślnie wykonać polecenia goblinia, albo porozmawiać z Albertem i dowiedzieć się, że jest on jedyną osobą zdolną do przywrócenia mocy zaginionemu Świętemu Graalowi (ang. *Holy Grail*). Bohater udaje się zatem do obozu goblinów porozmawiać z ich przywódcą (który dalej myśli, że bohater jest po ich stronie), aby dowiedzieć się gdzie zginął Graal. Okazało się, że paladyni zdążyli już odzyskać skradziony przez gobliny przedmiot. Zaopatrzony w tę wiedzę bohater musi zatem udać się do paladynów i przywrócić moc artefaktowi.

Wygenerowane zadania posiadają kolejno 9 i 11 niepustych akcji atomowych. Są to wyniki lekko odbiegające od średniej, która wyniosła 14.75 niepustych akcji atomowych dla kolejno wygenerowanych 100 zadań o strategii *AttackThreat3*. Jest to jednak wartość zbliżona do długości większych zadań pełnych spośród badanych zadań z gry *Wiedźmin 3* (największe miało 15 niepustych akcji, patrz rozdział 3.7). Średnia długość otrzymanych zadań nie jest przypadkowa, albowiem zastosowano mechanizm ograniczający długość wynikowych zadań. Wykorzystano w tym celu maksymalne ilości niepustych rozwinień akcji *subquest* oraz *learn*

⁵Generacja offline polega na wygenerowaniu treści do gry i manualnym dostosowaniu jej przed wprowadzeniem do gry.Więcej informacji o niej można przeczytać we wstępie niniejszej pracy.

(patrz podrozdział 4.3.3).

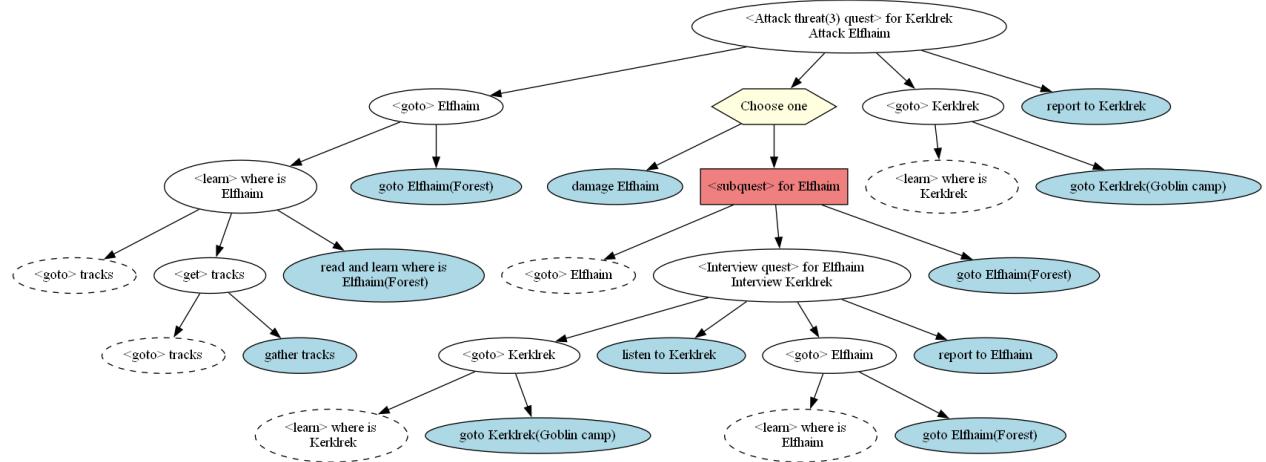
Zadania generowane przez *SkaldNET* reprezentują związki przyczynowo-skutkowe w sposób czytelny dla użytkownika. Związki te są rozbudowane w porównaniu do zadań generowanych przez system *CONAN* (opisany w rozdziale 2.1.2). Wynika to z interpretacji kolejnych węzłów w drzewie-zadaniu. Struktura drzewa sprawia, że w celu wykonania czynności reprezentowanej przez dany węzeł należy wykonać czynności reprezentowane przez jego dzieci.

Graficzna reprezentacja zadania sprawia, że zadanie jest czytelne dla ludzkiego użytkownika. Dzięki temu człowiek jest w stanie szybko czytać i interpretować treść zadania. Ta przejrzystość połączona z wyraźnym przedstawianiem związków przyczynowo-skutkowych powoduje, że użytkownik jest w stanie łatwo wymyślać własną narrację do zadań (podobną do powyższych opisów wygenerowanych zadań). Sprawia to, że wygenerowane przez system *SkaldNET* zadania mogą znaleźć zastosowanie w *offline PCG* zadań fabularnych.

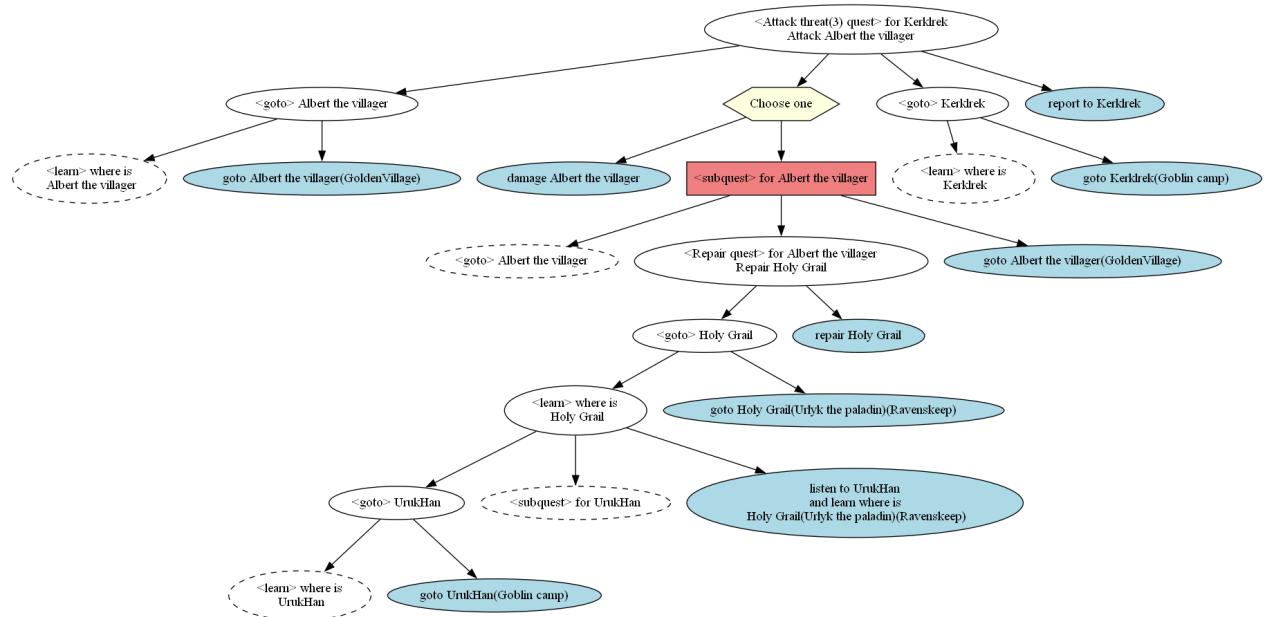
SkaldNET, w przeciwieństwie do systemu *CONAN*, stara się wykorzystywać elementy świata pasujące w danym kontekście zadania. W drugim przykładowym zadaniu, podzadanie *Repair quest* wybrało jako przedmiot wymagający naprawy przedmiot uszkodzony w *aktualnym* stanie świata gry (patrz mechanizm warunków wstępnych opisany w podrozdziale 4.3.3). Co więcej, generator pilnował, aby przedmiot nie został naprawiony podczas próby dostania się do niego (poddrzewo *<goto> holy Grail*) za pomocą mechanizmu obostrzeń (patrz podrozdział 4.3.4).

Zwróćmy uwagę na akcje *learn* występujące w przykładowych zadaniach. Podczas ich *rozwijania* generator sprawdzał, czy bohater już wcześniej nie dowiedział się gdzie znajduje się dany element świata. Opisany mechanizm wykorzystywany jest to do zapewnienia, że bohater nie będzie wielokrotnie poszukiwał tego samego elementu świata. Na przykładowych zadaniach (rysunki 4.15 i 4.16) uwidacznia się działanie tego mechanizmu w ostatniej akcji *<goto> Kerklerk*. W ramach niej akcja *<learn> where is Kerklerk* musiała zostać *rozwinięta* jako pusta. Spowodowane było to faktem, że *Kerklerk* jest zleceniodawcą zadania, więc bohater wie gdzie się on znajduje (musiał odbyć z nim wcześniejszą interakcję).

Zaimplementowany prototyp systemu zademonstrował sposób wykorzystania rozszerzonego modelu do generacji nieliniowych zadań fabularnych (dzięki operacjom *twist* i *or*). Przykładowe wygenerowane przez niego zadania pokazują, że model ten można wykorzystywać nie tylko do analizy istniejących zadań, ale także do generacji nowych. Zadania otrzymywane dzięki opracowanemu w niniejszej pracy algorytmowi, są spójne (wykonywalne), a wykorzystane mechanizmy starają się sprawić by były wiarygodne.



Rysunek 4.15: Przykład zadania wygenerowanego przez *SkaldNET*



Rysunek 4.16: Drugi przykład zadania wygenerowanego przez SkaldNET

Podsumowanie

W dobie powszechnej cyfryzacji problematyka generacji treści zyskuje coraz większe znaczenie, gdyż pozwala małym studiom produkującym gry wideo konkurować z gigantami tej branży. Znalazło to odwzorowanie w niniejszej pracy, która została poświęcona jednemu z działów proceduralnej generacji treści (PCG) – generacji zadań fabularnych.

Cel niniejszej pracy został osiągnięty. W rozdziale 3 przeanalizowano i rozszerzono model zlinearyzowanych zadań fabularnych opracowany przez Dorana i Parberrego. Następnie, w rozdziale 4, wykonano prototyp generatora zadań fabularnych, który wykorzystuje opracowany rozszerzony model zadań fabularnych do generacji spójnych zadań.

Podczas analizy modelu Dorana (patrz podrozdział 3.4) zaobserwowano, że jest on w stanie reprezentować znaczącą część zadań fabularnych o cechach typowych dla gier RPG. Badanie polegało na próbie zamodelowania wybranych zadań pobocznych z gry RPG pod tytułem *Wiedźmin 3: Dziki Gon*. Udało się zamodelować 66.6% spośród mniejszych *zadań-zdarzeń* oraz 55% badanych złożonych zadań *pełnych*, co uznano za pozytywny wynik.

Podczas analizy modelu Dorana zwrócono szczególną uwagę na przyczyny niepowodzeń zamodelowania niektórych zadań. Wyróżniono cztery wady tego modelu:

- W1: Model nie uwzględnia możliwości wprowadzania głównego bohatera w błąd przez zleceniodawcę.
- W2: Model nie pozwala na przedstawienie potrzeby zdobycia kilku przedmiotów.
- W3: Model nie pozwala na zdobycie w pokojowy sposób przedmiotu posiadanego przez kogoś bez dokonania wymiany.
- W4: Model pozwala tylko na przedstawianie liniowych zadań, czyli takich w których bohater nie ma możliwości podejmowania decyzji.

W związku z powyższym zaproponowano w podrozdziale 3.6 rozszerzenie modelu mające na celu wyeliminowanie wad W2, W3 oraz W4.

Rozszerzenie, poza wprowadzeniem nowych reguł eliminujących wady W2 oraz W3, dodaje do modelu dwa nowe operatory zmieniające sposób poruszania się po drzewie-zadaniu. Operator *or* reprezentuje wybór gracza, a operator *twist* pozwala na wcześniejsze zakończenie zadania. Wprowadzenie tych dwóch operatorów pozwoliło rozszerzonemu modelowi na reprezentację zadań nielinowych, rekompensując w ten sposób wadę W4.

Aby zademonstrować zwiększoną siłę ekspresji rozszerzonego modelu powtórzono wykonane wcześniej badania. Zaobserwowano:

- zwiększenie ilości możliwych do zamodelowania *zadań-zdarzeń* z 66.6% do 93.3% (26.7 p.p.);
- zwiększenie ilości możliwych do zamodelowania zadań *pełnych* z 55% do 75% (20 p.p.).

W celu demonstracji sposobu wykorzystania rozszerzonego modelu do generacji zadań zaimplementowano prototyp generatora zadań fabularnych. Generator ten przyjmuje na wejściu stan początkowy świata gry i na podstawie niego generuje zadania. Wykorzystuje w tym celu autorski algorytm (patrz podrozdział 4.4), który zawiera mechanizmy gwarantujące logiczną spójność otrzymywanych zadań (wykonywalność). Dodatkowo generator stara się zapewnić wiarygodność zadań poprzez wykorzystanie mechanizmów wspierających selekcję odpowiednich (kontekście zadania) elementów świata.

Należy również zwrócić uwagę na możliwości dalszego rozwoju zarówno samego generatora, jak i opracowanego rozszerzonego modelu zadań. Opracowany model można bowiem, w zależności od potrzeb, rozszerzyć o kolejne typy zadań (strategie) i motywacje zleceniodawców, w szczególności reprezentujące próbę wprowadzenia w błąd bohatera (naprawienie wady W1 modelu). Dodatkowo można usprawnić sposób generacji zadań obsługując więcej cech elementów świata gry. Generator mógłby je wykorzystywać do usprawnienia selekcji elementów świata gry, a co za tym idzie, zwiększenia wiarygodności zadań.

Ostatecznie wynikiem niniejszej pracy jest formalny model zadań, zdolny do przedstawiania złożonych zadań o cechach typowych dla gier RPG. Ponadto udało się opracować generator zdolny do generowania nieliniowych, spójnych oraz wiarygodnych drzew-zadań, który można wykorzystywać do generacji zadań nadzorowanej przez projektanta gry (*offline*).

Bibliografia

- [1] A. Alvarez and J. Font. Tropetwist: Trope-based narrative structure generation. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, FDG '22, New York, NY, USA, 2022. Association for Computing Machinery.
- [2] A. Alvarez, J. Font, and J. Togelius. Story designer: Towards a mixed-initiative tool to create narrative structures. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, FDG '22, New York, NY, USA, 2022. Association for Computing Machinery.
- [3] V. Breault, S. Ouellet, and J. Davies. Let conan tell you a story: Procedural quest generation. *Entertainment Computing*, 38:100422, 2021.
- [4] V. Breault, S. Ouellet, and J. Davies. Let conan tell you a story: Procedural quest generation. *Entertainment Computing*, 38:100422, 2021.
- [5] S. Chen, A. Smith, A. Jhala, N. Wardrip-Fruin, and M. Mateas. Rolemodel: Towards a formal model of dramatic roles for story generation. *Proceedings of the Intelligent Narrative Technologies III Workshop, INT3 '10*, page 17, 06 2010.
- [6] C. Dabral and C. Martens. Generating explorable narrative spaces with answer set programming. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 16(1):45–51, Oct. 2020.
- [7] T. Denee. www.timdenee.com, 2017. Dostep 5.04 2023.
- [8] J. Doran and I. Parberry. A prototype quest generator based on a structural analysis of quests from four mmorpgs. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, PCCGames '11, New York, NY, USA, 2011. Association for Computing Machinery.
- [9] A. Doull. The death of the level designer, 2008.
- [10] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971.
- [11] M. Fox and D. Long. Pddl+ : Modelling continuous time-dependent effects. 04 2003.
- [12] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld. Pddl - the planning domain definition language. 08 1998.
- [13] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 3 edition, 2006.

- [14] J. Huzinga. *Homo ludens. Zabawa jako źródło kultury*. Wydawnictwo Aletheia, Warszawa, 2007.
- [15] B. Lech, S. Azad, J. Wellnitz, J. Jonasson, and C. Martens. Designing a combined world and story procedural content generation engine. In B. Harrison, editor, *Joint Proceedings of the AIIDE 2021 Workshops co-located with 17th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2021)(AAAI 2021), Virtual Event, Lexington, KY, USA, October 11-12, 2021*, volume 3217 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.
- [16] C. Martens and R. J. Simmons. Inbox games: Poetics and authoring support. In *Interactive Storytelling: 14th International Conference on Interactive Digital Storytelling, ICIDS 2021, Tallinn, Estonia, December 7–10, 2021, Proceedings*, page 94–106, Berlin, Heidelberg, 2021. Springer-Verlag.
- [17] L. Rodrigues, R. Bonidia, and J. Brancher. Procedural versus human level generation: Two sides of the same coin? *International Journal of Human-Computer Studies*, 141:102465, 2020.
- [18] M. Sipser. *Introduction to Automata Theory, Languages, and Computation*. Cengage Learning, 3 edition, 2012.
- [19] M. Sokolski. Projekt i implementacja modułu potyczek strategicznych w grze typu dungeon crawler w unity, 2021.
- [20] Sun Microsystems Inc. Federated naming service programming guide. Copyright 2002 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A.
- [21] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis. What is procedural content generation? mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, PCGames ’11, New York, NY, USA, 2011. Association for Computing Machinery.
- [22] wiedzmin.fandom.com. Zadania w velen. Ostatni dostęp 3.03.2023.
- [23] M. Young, S. Ware, B. Cassell, and J. Robertson. Plans and planning in narrative generation: A review of plan-based approaches to the generation of story, discourse and interactivity in narratives. *SDV: Sprache und Datenverarbeitung: International Journal of Language Processing*, 37:41–64, 01 2013.
- [24] K. Yu, N. R. Sturtevant, and M. Guzdial. What is a quest. In *Proceedings of the Intelligent Narrative Technologies Workshop at AIIDE*. <http://www.cs.ualberta.ca/~nathanst/papers/yu2020quest.pdf>, 2020.

Spis rysunków

2.1	Dystrybucja motywacji zadań wygenerowanych dla małego świata [4]	16
2.2	Dystrybucja motywacji zadań wygenerowanych dla dużego świata [4]	16
2.3	Projekt silnika fabularnego Martensa et al. opracowanego na potrzeby gry mobilnej <i>Lance a Lot</i> [15]	20
3.1	Zadanie jako graf-drzewo sekwencji akcji	28
3.2	Zadanie jako graf-drzewo sekwencji akcji z metadanymi	29
3.3	Q1 – <i>Atak na karawanę</i> (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	32
3.4	Q2 – <i>Dręczenie trolla</i> (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	32
3.5	Q5.2 – <i>Na łasce obcych</i> , pierwszy przebieg zadania (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	33
3.6	Q5.2 – <i>Na łasce obcych</i> , alternatywny przebieg zadania (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	33
3.7	Q13 – <i>Zmierz się ze mną!</i> (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	33
3.8	Q41 – <i>Zlecenie: Leśnica</i> (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	34
3.9	Q25 – <i>Złoto głupców</i> (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	35
3.10	Q46 – <i>Ochotnik</i> (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	36
3.11	Akcja get many reprezentująca zdobycie 2 przedmiotów	38
3.12	Pusta akcja get many	38
3.13	Akcja get, w której posiadacz przedmiotu oczekuje wykonania przysługi	39
3.14	Akcja get, w której bohater otrzymuje przedmiot bez wykonywania dodatkowych czynności	39
3.15	Zapis wyrażenia or(“damage”)(“”) w postaci drzewa	40
3.16	Zapis wyrażenia or(“damage”)(twist(subquest)) w postaci drzewa	40
3.17	Q21 – <i>Szepczące Wzgórze</i> (opracowanie własne na podstawie opisu zadania [22] i gry <i>Wiedźmin 3</i>)	45
4.1	Model świata w systemie <i>SkaldNET</i>	48
4.2	Początek generacji zadania o strategii Deliver item for study	52
4.3	Drugi krok generacji zadania o strategii Deliver item for study	52

4.4 Decyzje podejmowane wyżej w drzewie (węzeł pomarańczowy) mają wpływ na całe poddrzewa (węzły zielone)	53
4.5 W węźle <i>or</i> wykonuje się kopie stanu dla każdego dziecka. Po powrocie z węzła <i>or</i> wszystkie kolejne <i>rozwijane</i> akcje(zielone) muszą brać wszystkie kopie pod uwagę.	54
4.6 W węźle <i>or</i> wykonuje się kopie stanu dla każdego dziecka. Po powrocie z węzła <i>or</i> wszystkie kolejne <i>rozwijane</i> akcje(zielone) muszą brać pod uwagę tylko kopie, które nie trafiły do węzła <i>twist</i>	55
4.7 W węźle <i>or</i> nie trzeba wykonywać kopii stanu dla węzłów pustych.	56
4.8 Przykład niepoprawnego zadania, które można otrzymać stosując tylko mechanizm aktualizacji stanu świata oraz mechanizm warunków wstępnych	59
4.9 Mechanizm obostrzeń pozwala wprowadzać restrykcje na wybierane elementy świata w poddrzewach.	61
4.10 Algorytm generowania zadań	62
4.11 Puste poddrzewo przed uproszczeniem	63
4.12 Puste poddrzewo po uproszczeniu	63
4.13 Węzeł twist przed złączeniem	64
4.14 Węzeł twist po złączeniu	64
4.15 Przykład zadania wygenerowanego przez <i>SkaldNET</i>	66
4.16 Drugi przykład zadania wygenerowanego przez <i>SkaldNET</i>	66

Spis tabelic

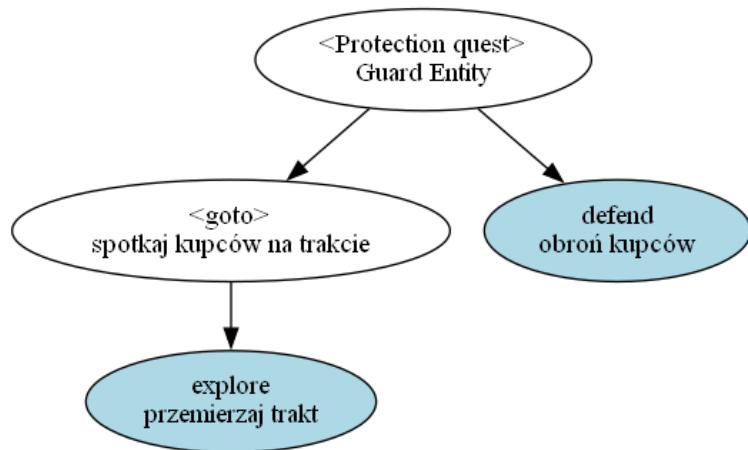
1.1	Typowe motywów w grach wideo [1]	8
2.1	Preferencje bohaterów świata Aladyna	14
2.2	Przykładowe predykaty dostępne w systemie CONAN [4]	14
3.1	Atomowe akcje w modelu Dorana [8]	23
3.2	Akcje złożone modelu Dorana [8]	24
3.3	Motywacje modelu Dorana [8]	24
3.4	Strategie dla motywacji w modelu zadań Dorana [8]	25
3.5	Analiza możliwości zamodelowania zadań z <i>Wiedźmina 3</i> modelem Dorana	31
3.6	Nowo dodane strategie	41
3.7	Analiza możliwości zamodelowania zadań z <i>Wiedźmina 3</i> rozszerzonym modelem	44
4.1	Cechy lokacji w systemie <i>SkaldNET</i>	49
4.2	Cechy postaci w systemie <i>SkaldNET</i>	49
4.3	Cechy przedmiotów w systemie <i>SkaldNET</i>	49
4.4	Strategie wspierane przez Skald-NET	51
4.5	Przyporządkowanie strategii osobowościom	51
4.6	Sposób wyboru sekwencji akcji dla akcji złożonych	57
4.7	Warunki wstępne strategii	58
4.8	Obostrzenia wprowadzane przez węzły dokonujące selekcji elementów świata	60

A. Model Dorana et al. w notacji Backusa-Naura

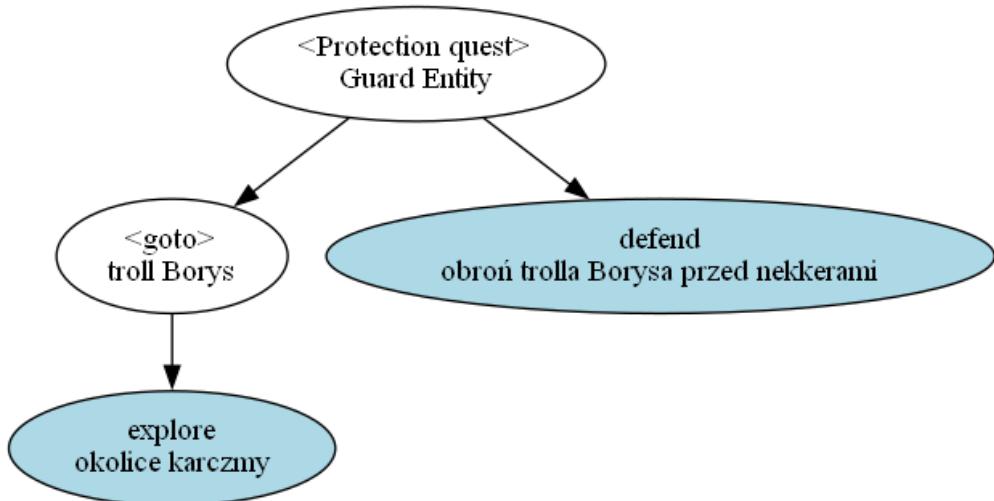
```
<QUEST> ::= <Knowledge> | <Comfort> | <Reputation>
           | <Serenity> | <Protection> | <Conquest>
           | <Wealth> | <Ability> | <Equipment>
<subquest> ::= <goto>
              | <goto> <QUEST> 'goto'
<goto> ::= "
              | 'explore'
              | <learn> 'goto'
<learn> ::= "
              | <goto> <subquest> 'listen'
              | <goto> <get> 'read'
              | <get> <subquest> 'give' 'listen'
<get> ::= "
              | <steal>
              | <goto> 'gather'
              | <goto> <get> <subquest> <goto> 'exchange'
<steal> ::= <goto> 'stealth' 'take'
              | <goto> <kill> 'take'
<spy> ::= <goto> 'spy' <goto> 'report'
<capture> ::= <get> <goto> 'capture'
<kill> ::= <goto> 'kill'
<Knowledge> ::= <get> <goto> 'give'
                | <spy>
                | <goto> 'listen' <goto> 'report'
                | <get> <goto> 'use' <goto> <give>
<Comfort> ::= <get> <goto> <give>
                | <goto> 'damage' <goto> 'report'
<Reputation> ::= <get> <goto> <give>
                | <goto> <kill> <goto> 'report'
                | <goto> <goto> 'report'
```

```
<Serenity> ::= <goto> 'damage'  
    | <get> <goto> 'use' <goto> <give>  
    | <get> <goto> 'use' 'capture' <goto> <give>  
    | <goto> 'listen' <goto> 'report'  
    | <goto> 'take' <goto> 'give'  
    | <get> <goto> <give>  
    | <goto> 'damage' 'escort' <goto> 'report'  
<Protection> ::= <goto> 'damage' <goto> 'report'  
    | <get> <goto> 'use'  
    | <goto> 'repair'  
    | <get> <goto> 'use'  
    | <goto> 'damage'  
    | <goto> 'repair'  
    | <goto> 'defend'  
<Conquest> ::= <goto> 'damage'  
    | <goto> <steal> <goto> 'give'  
<Wealth> ::= <goto> <get>  
    | <goto> <steal>  
    | 'repair'  
<Ability> ::= 'repair' 'use'  
    | <get> 'use'  
    | 'use'  
    | 'damage'  
    | 'use'  
    | <get> 'use'  
    | <get> 'experiment'  
<Equipment> ::= 'repair'  
    | <get> <goto> <give>  
    | <steal>  
    | <goto> 'exchange'
```

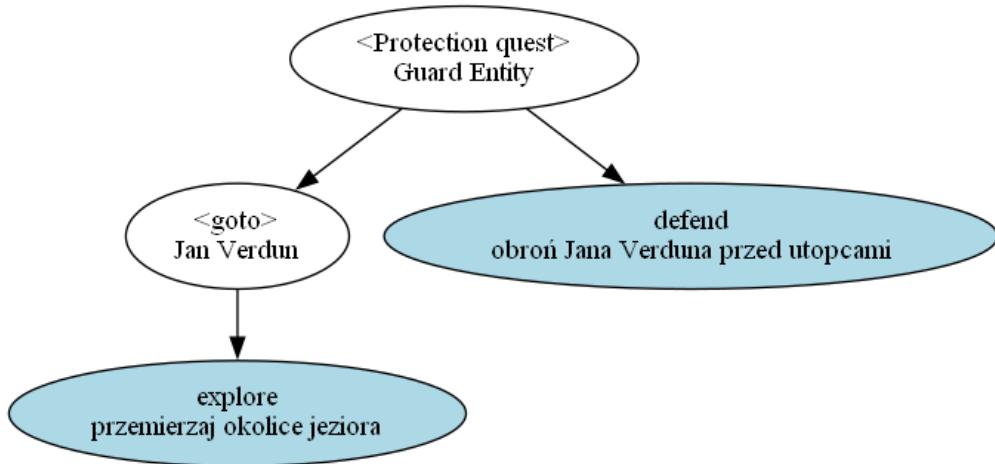
B. Zadania z gry Wiedźmin 3 wyrażone modelem Dorana



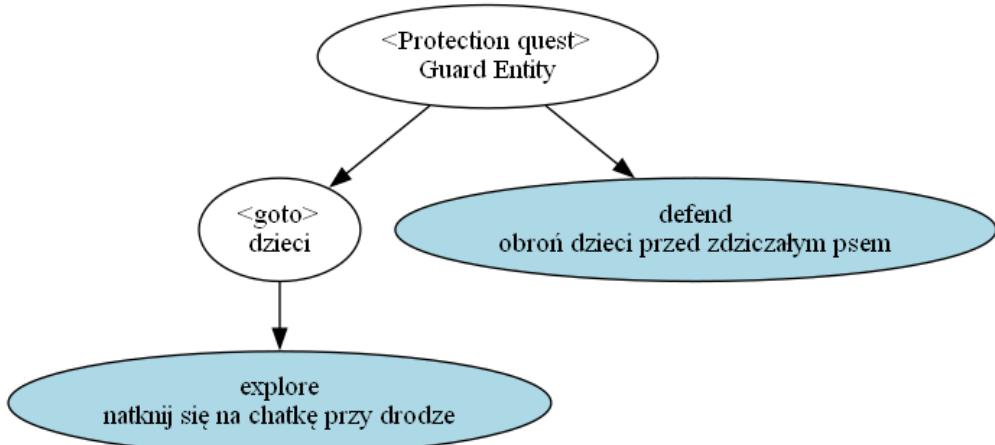
Rysunek B.1: *Atak na karawane* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



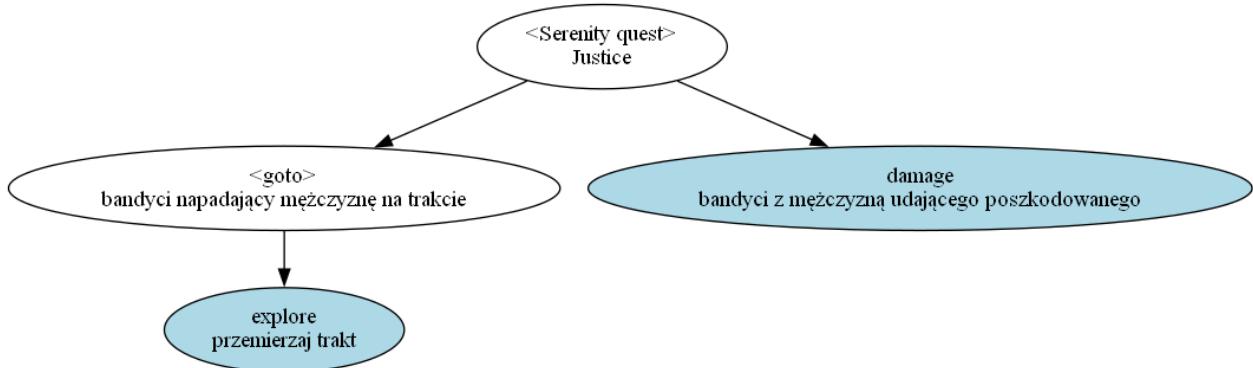
Rysunek B.2: *Dręczenie trolla* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



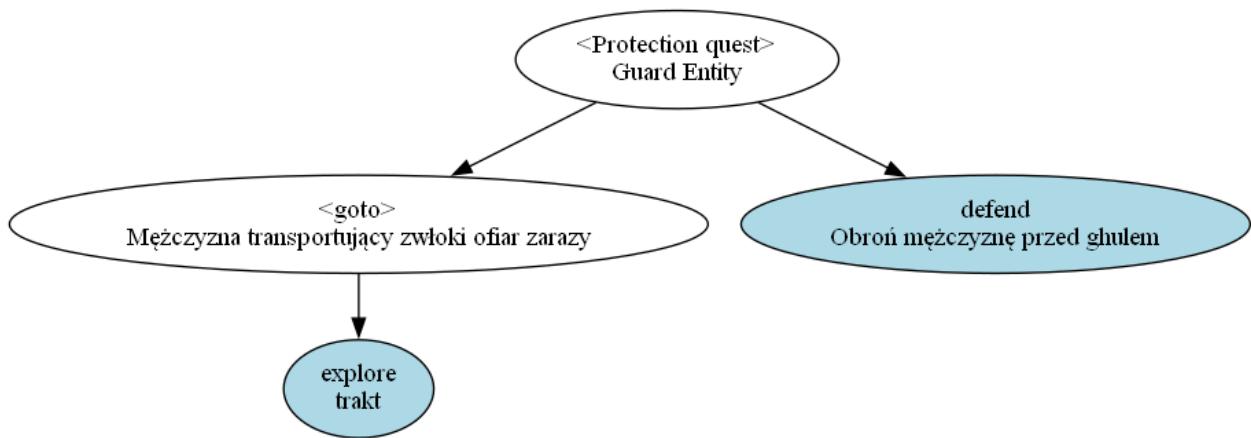
Rysunek B.3: *Na łasce obcych* – pierwsze spotkanie z Janem Verdunem (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



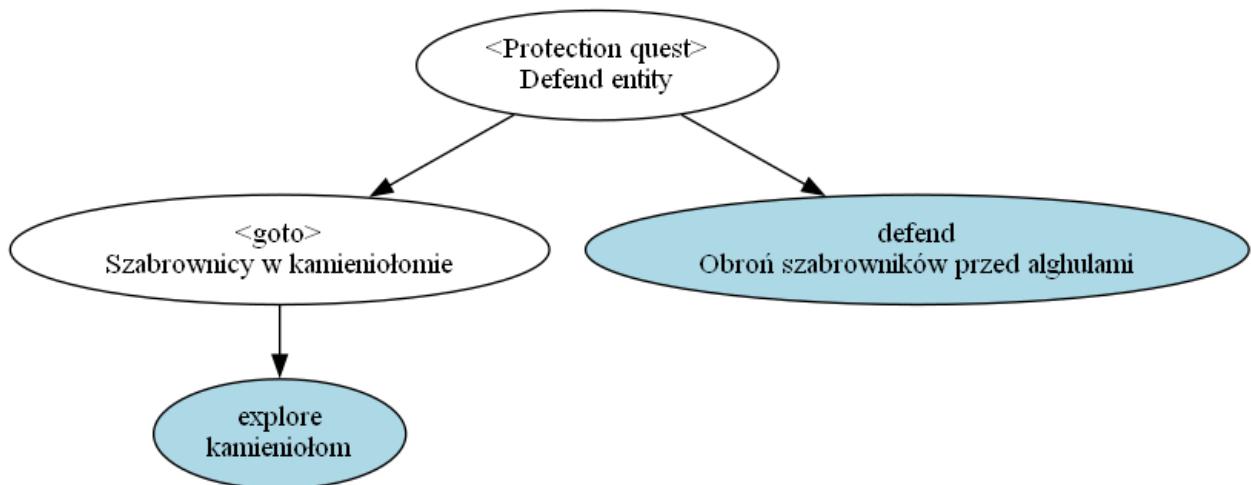
Rysunek B.4: *Najlepszy przyjaciel człowieka* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



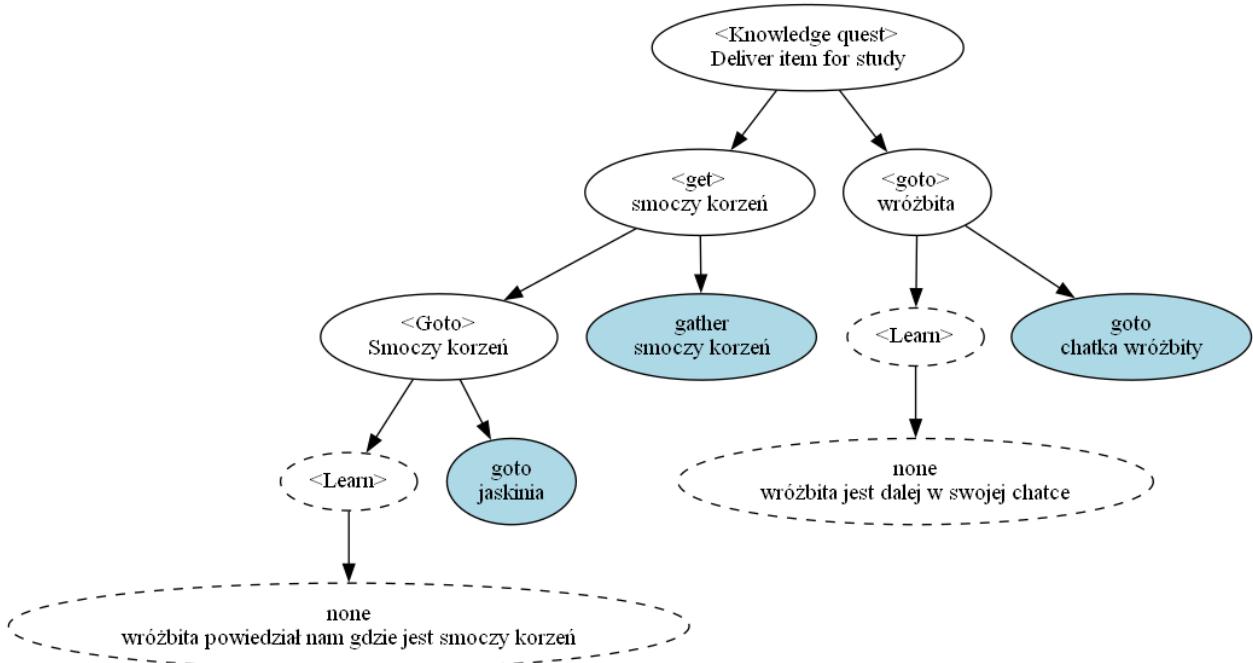
Rysunek B.5: *Napad na trakcie* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



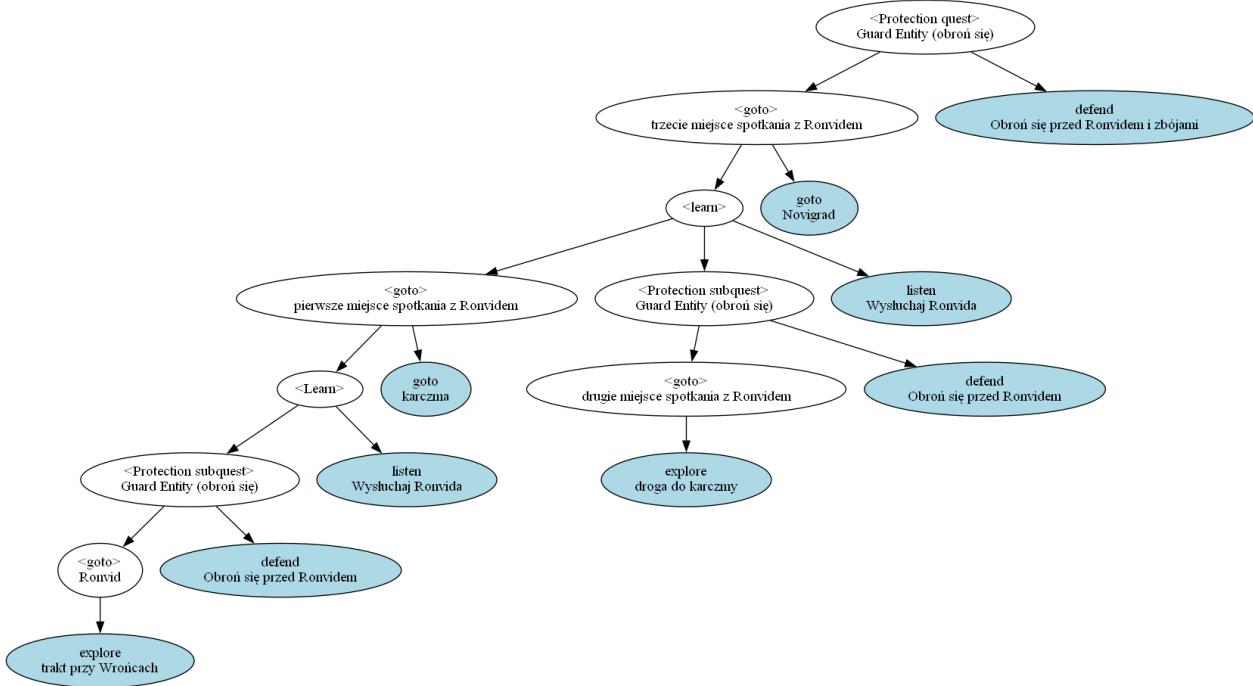
Rysunek B.6: *Niebezpieczny ładunek* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



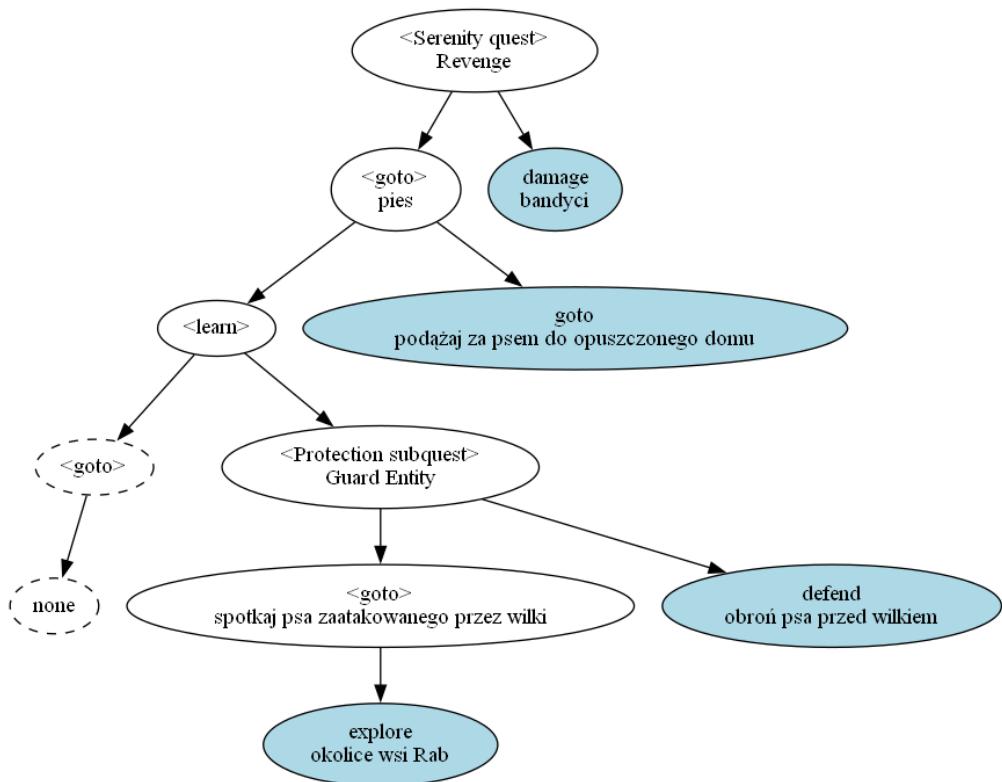
Rysunek B.7: *Szabrownicy część druga* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



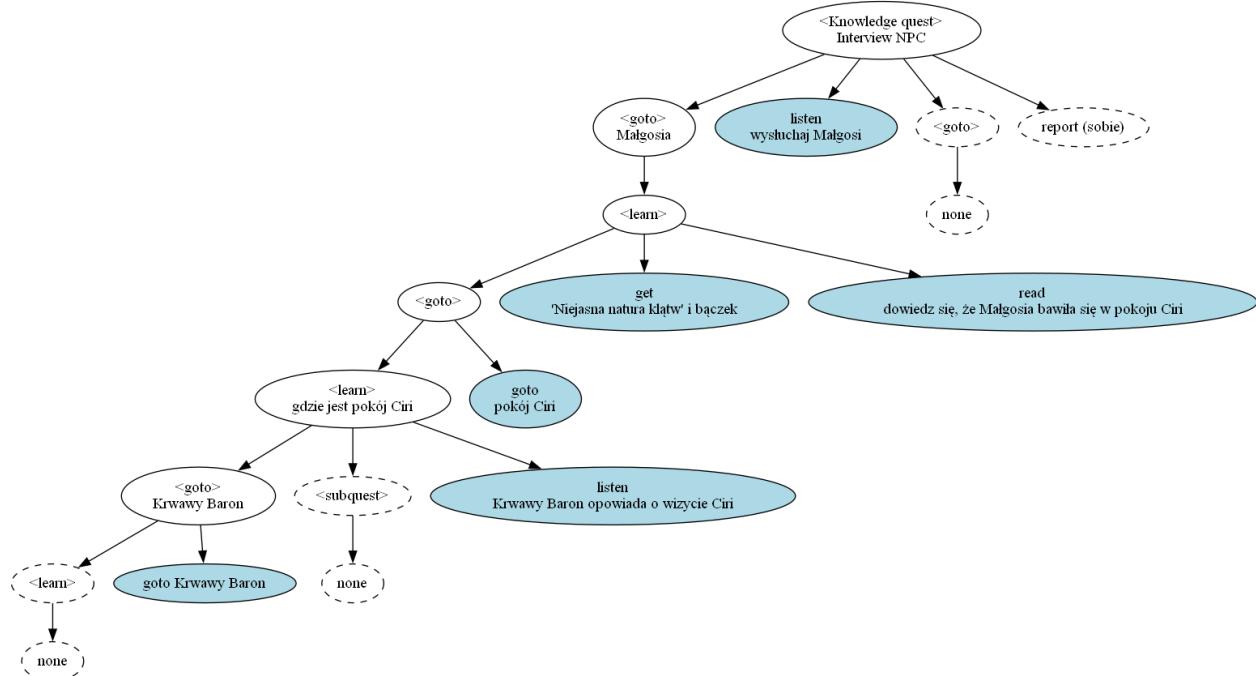
Rysunek B.8: Wróżbita prawdę Ci powie (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



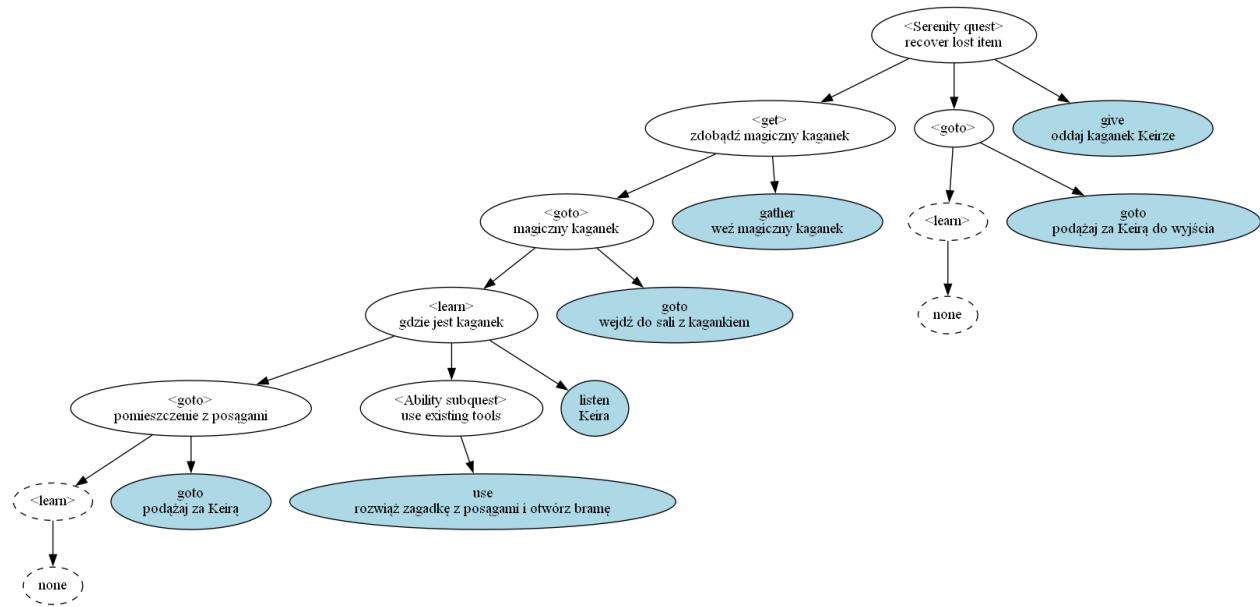
Rysunek B.9: Zmierz się ze mną! (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



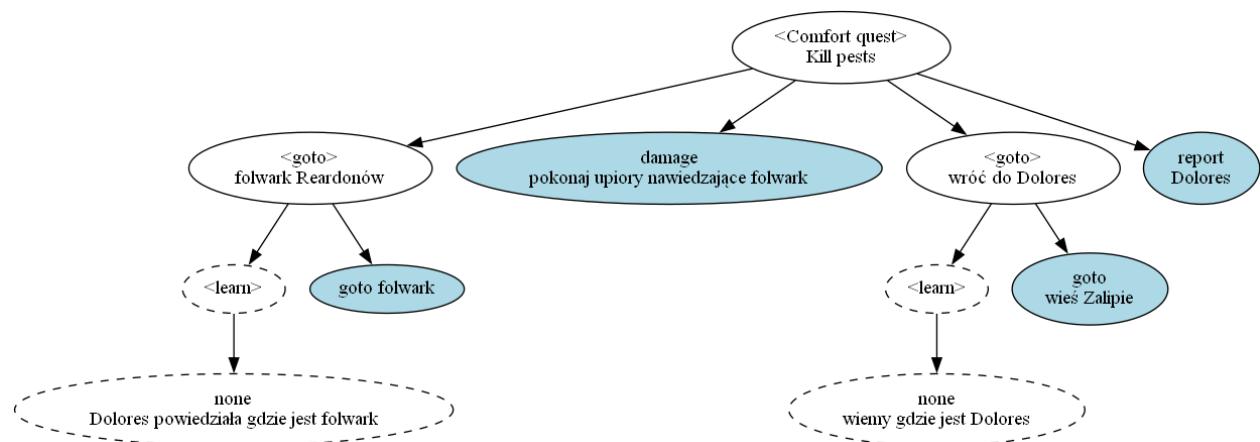
Rysunek B.10: *Pieskie życie* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



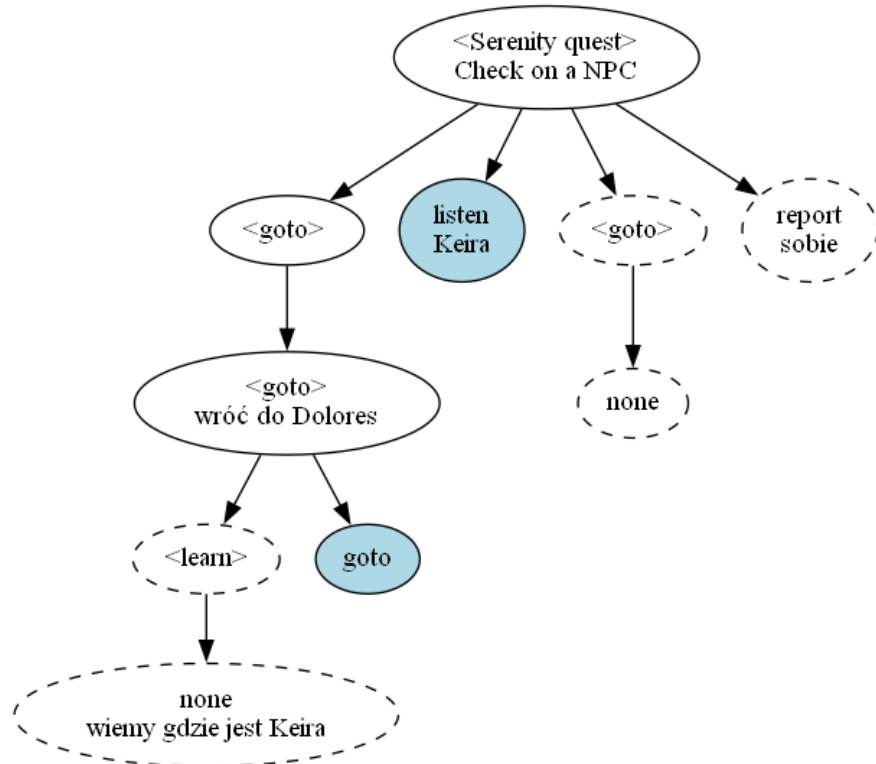
Rysunek B.11: *Pokój Ciri* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



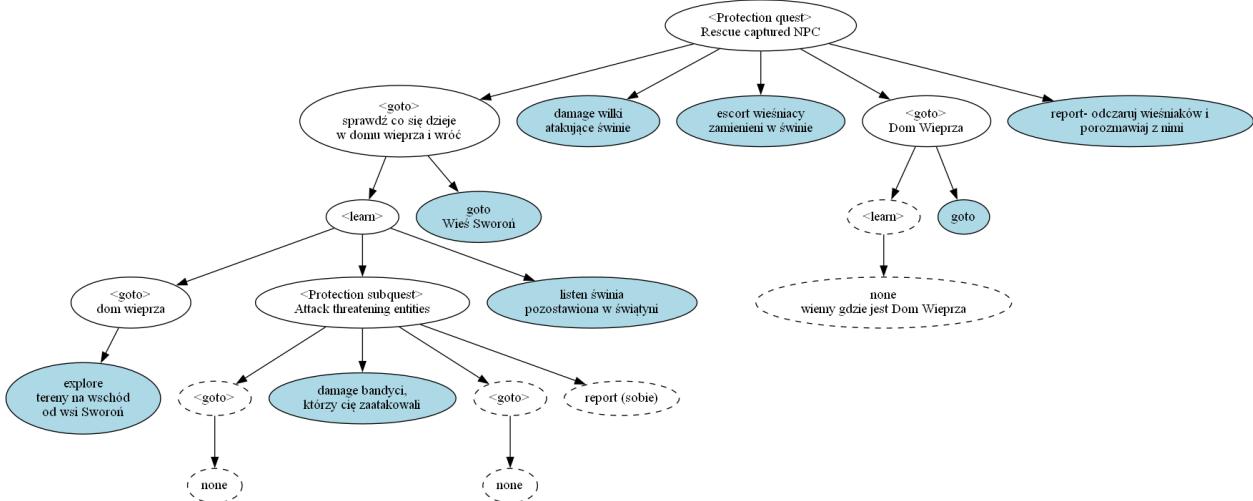
Rysunek B.12: *Magiczny kaganek* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



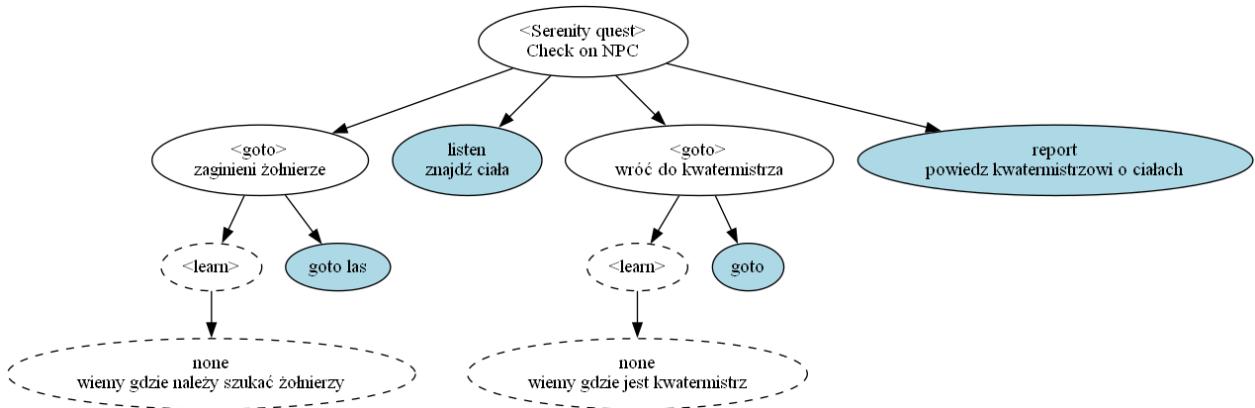
Rysunek B.13: *Zagłada domu Reardonów* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



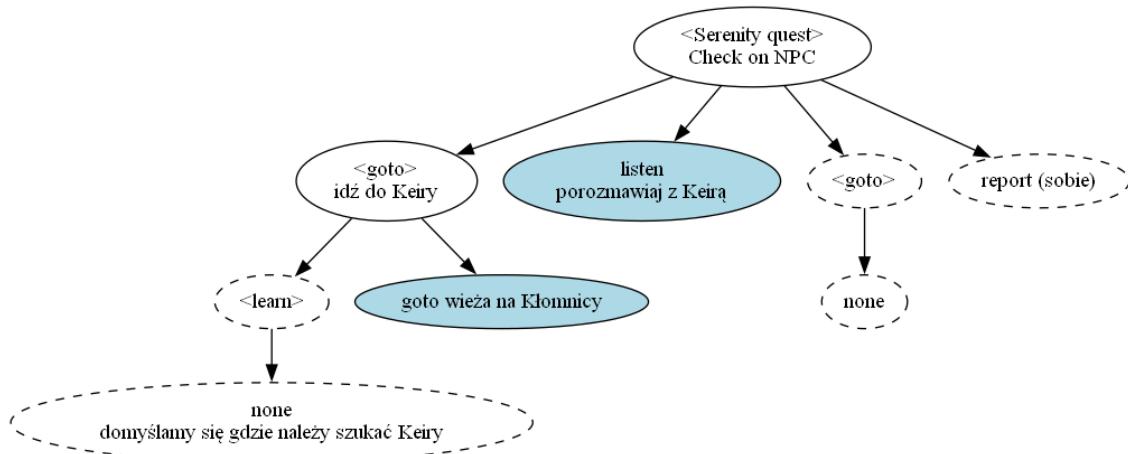
Rysunek B.14: *Zaproszenie od Keiry Metz* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



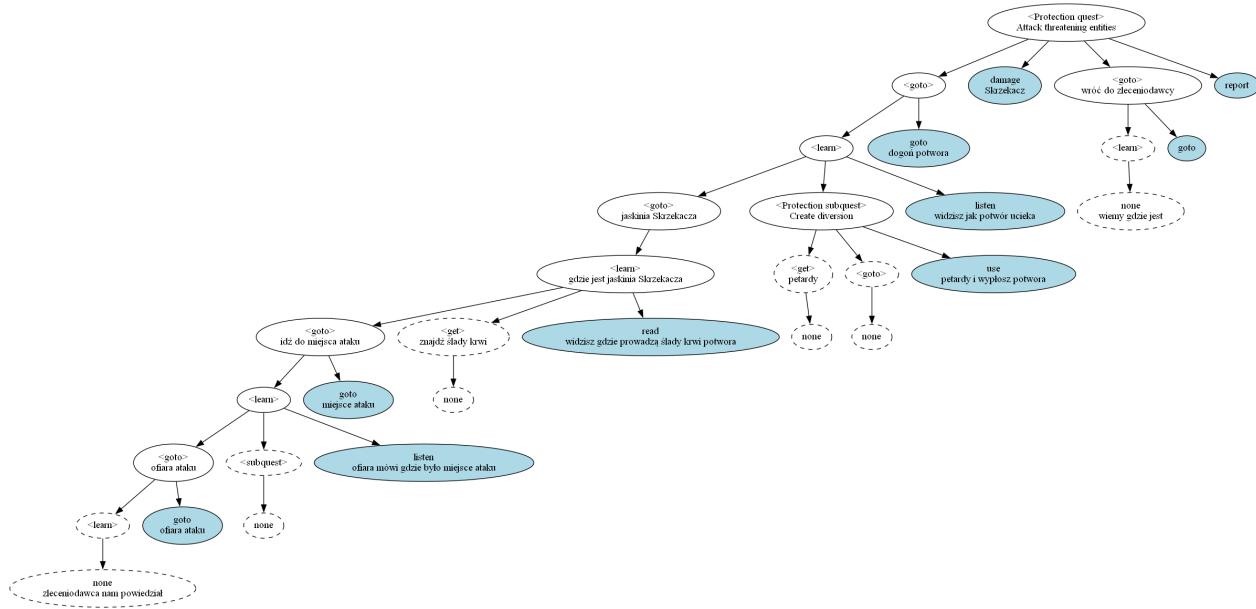
Rysunek B.15: *Złoto głupców* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



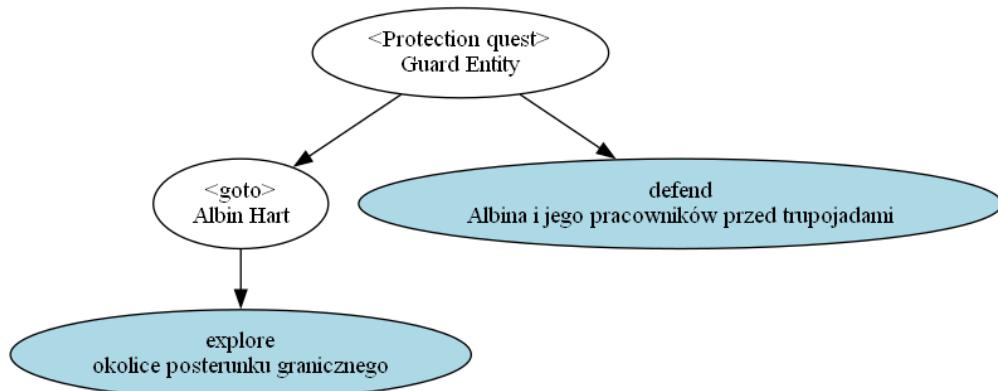
Rysunek B.16: Zlecenie: Zaginiony patrol (opracowanie własne na podstawie opisu zadania [22] i gry Wiedźmin 3)



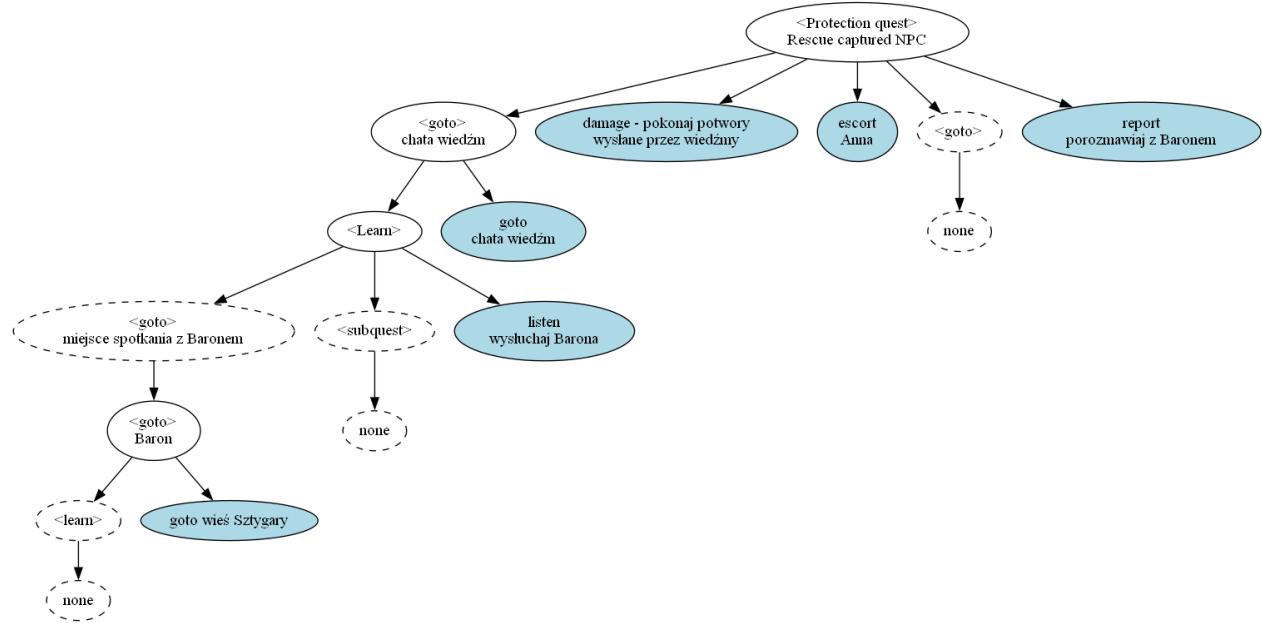
Rysunek B.17: Podstawowy przebieg zadania Dla dobra nauki (opracowanie własne na podstawie opisu zadania [22] i gry Wiedźmin 3)



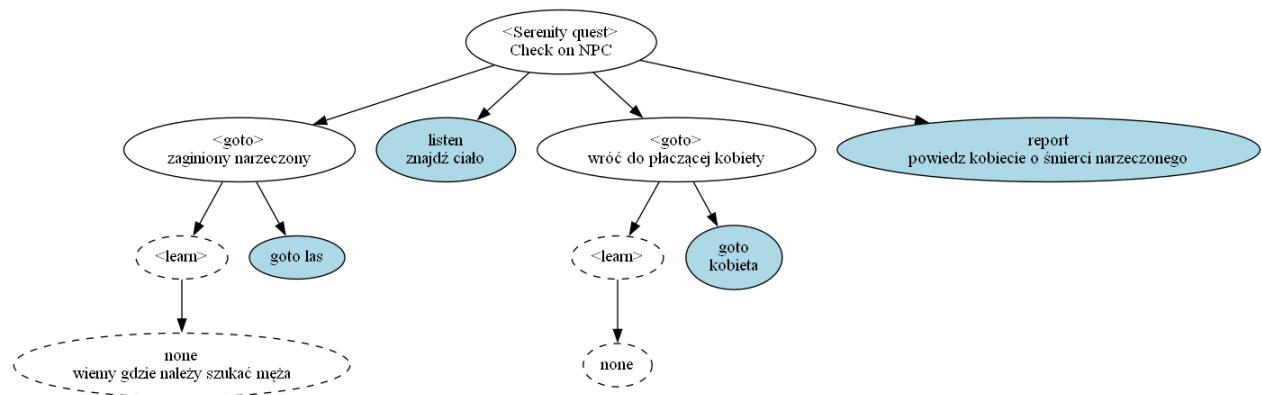
Rysunek B.18: Zlecenie: Skrzekacz (opracowanie własne na podstawie opisu zadania [22] i gry Wiedźmin 3)



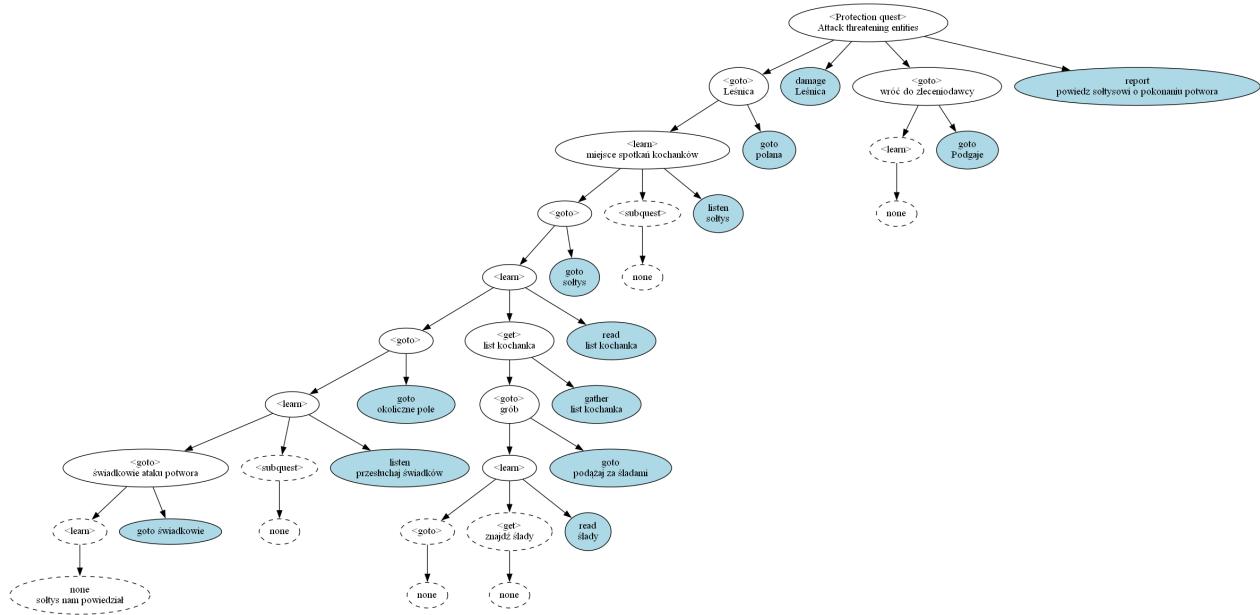
Rysunek B.19: Hieny cmentarne (opracowanie własne na podstawie opisu zadania [22] i gry Wiedźmin 3)



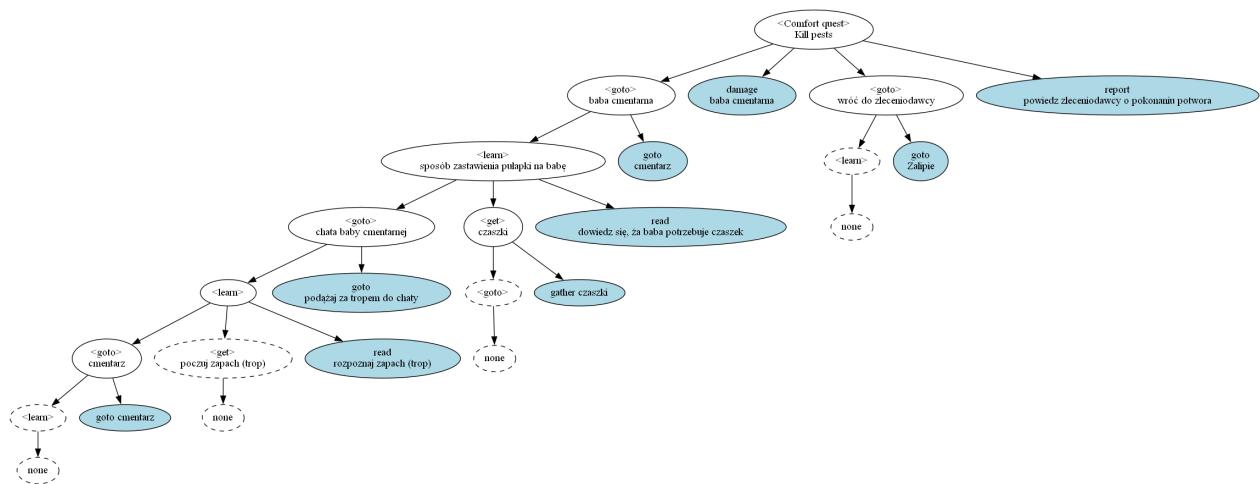
Rysunek B.20: *Powrót na Krzywuchowe Moczary* (wersja zadania po wykonaniu podstawowego przebiegu zadania Szepczące wzgórze) (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



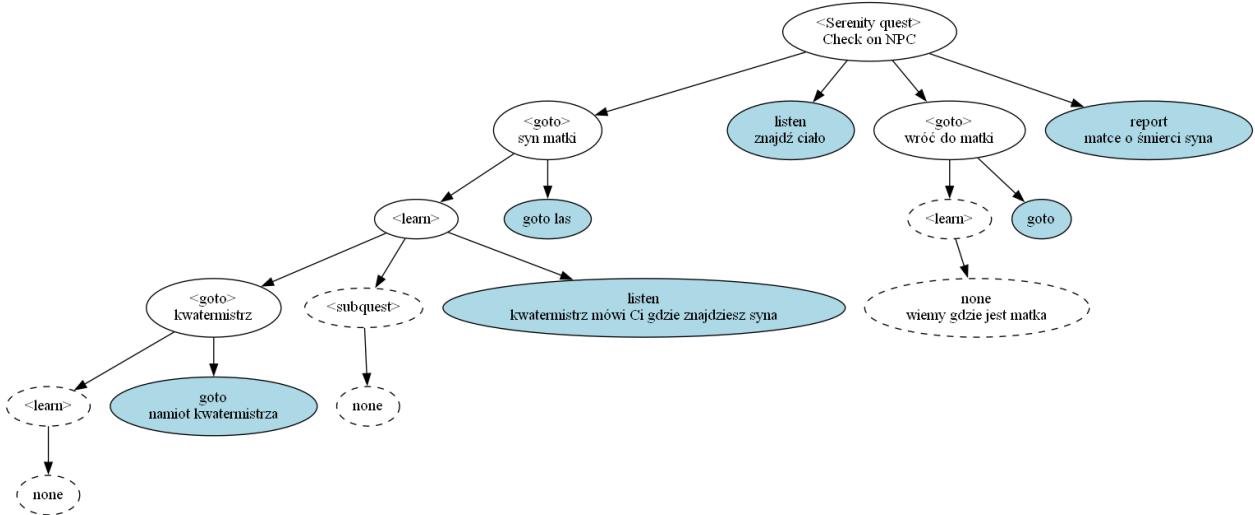
Rysunek B.21: *Dama w opałach* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



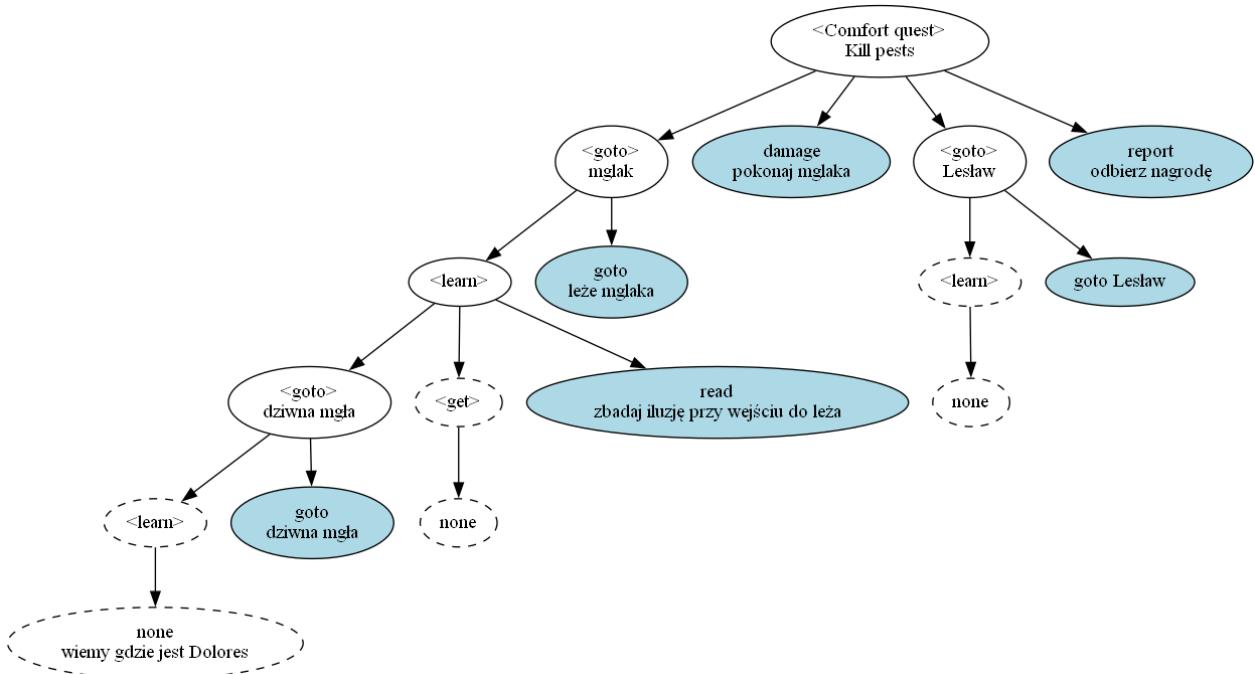
Rysunek B.22: Zlecenie: Leśnica (opracowanie własne na podstawie opisu zadania [22] i gry Wiedźmin 3)



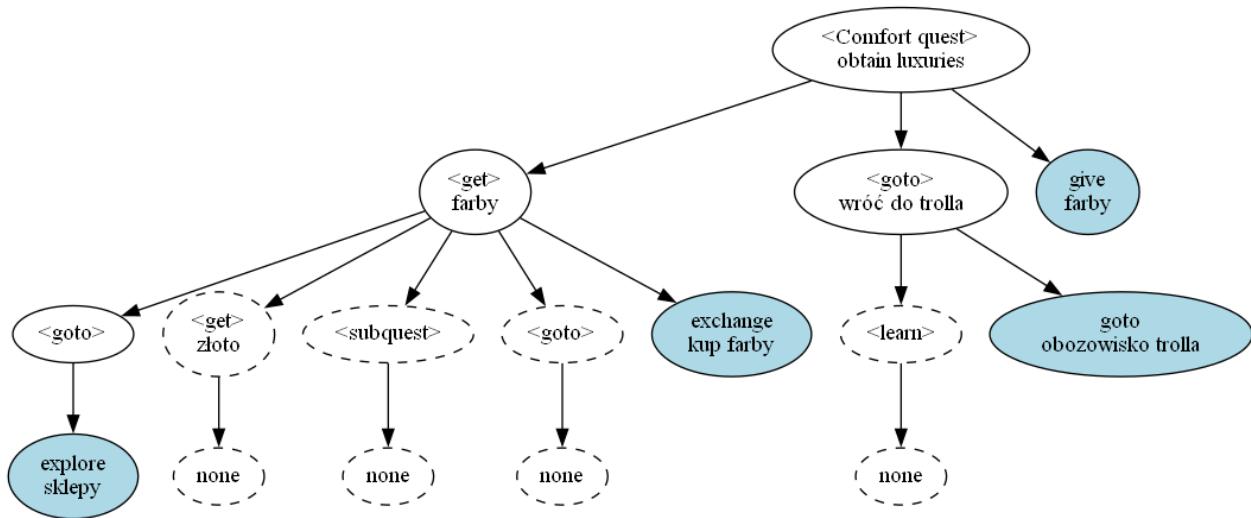
Rysunek B.23: Zlecenie: *Kłopoty grabarza* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



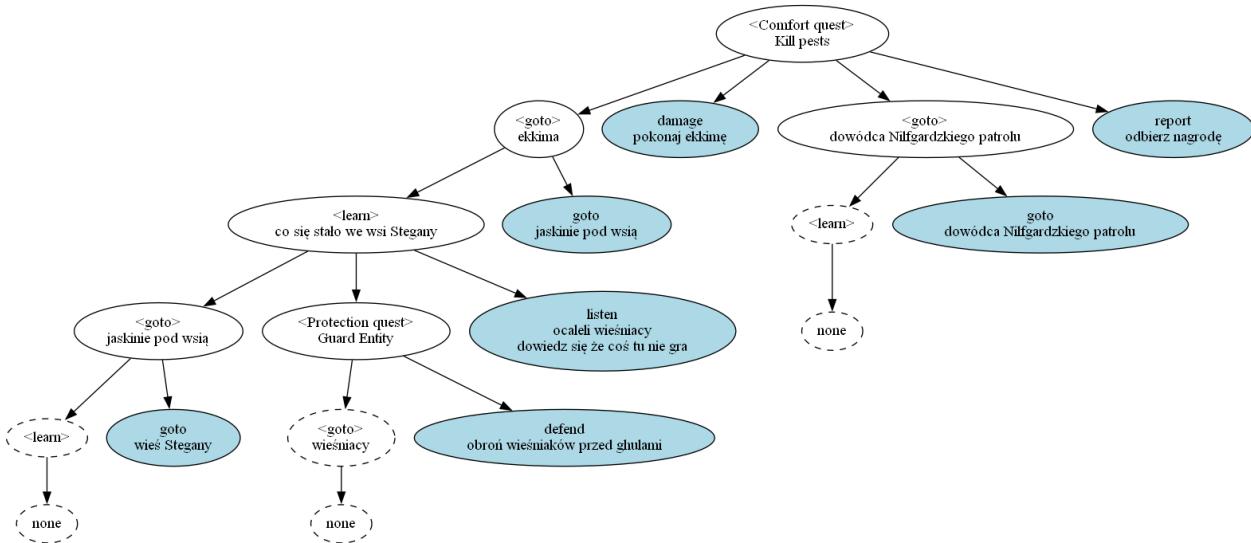
Rysunek B.24: *Wiezy krwi* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



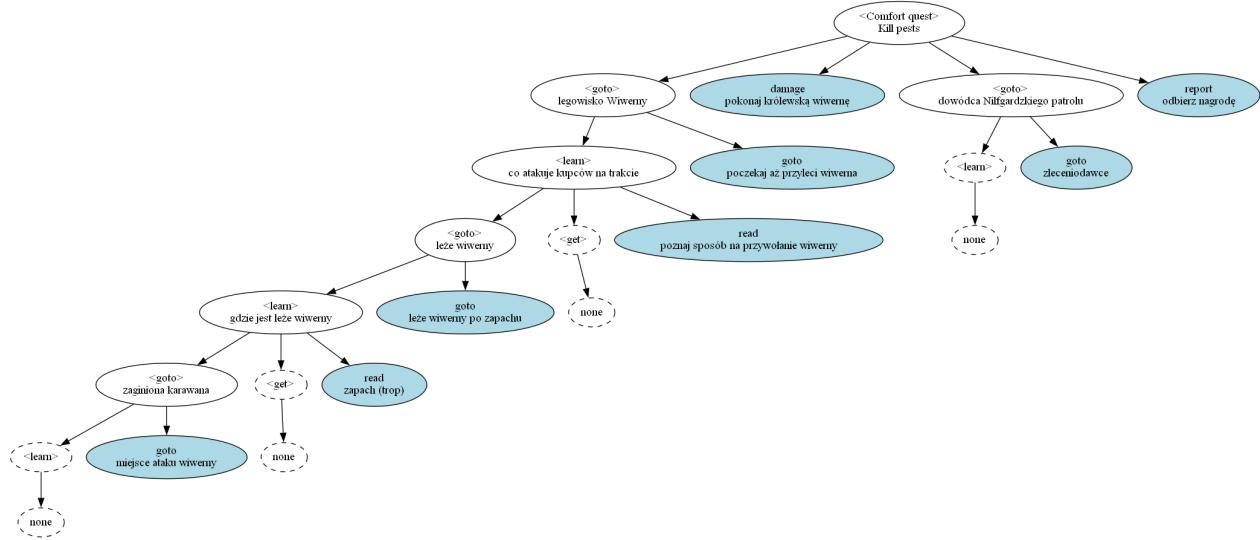
Rysunek B.25: *Zlecenie: Potwór z bagien* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



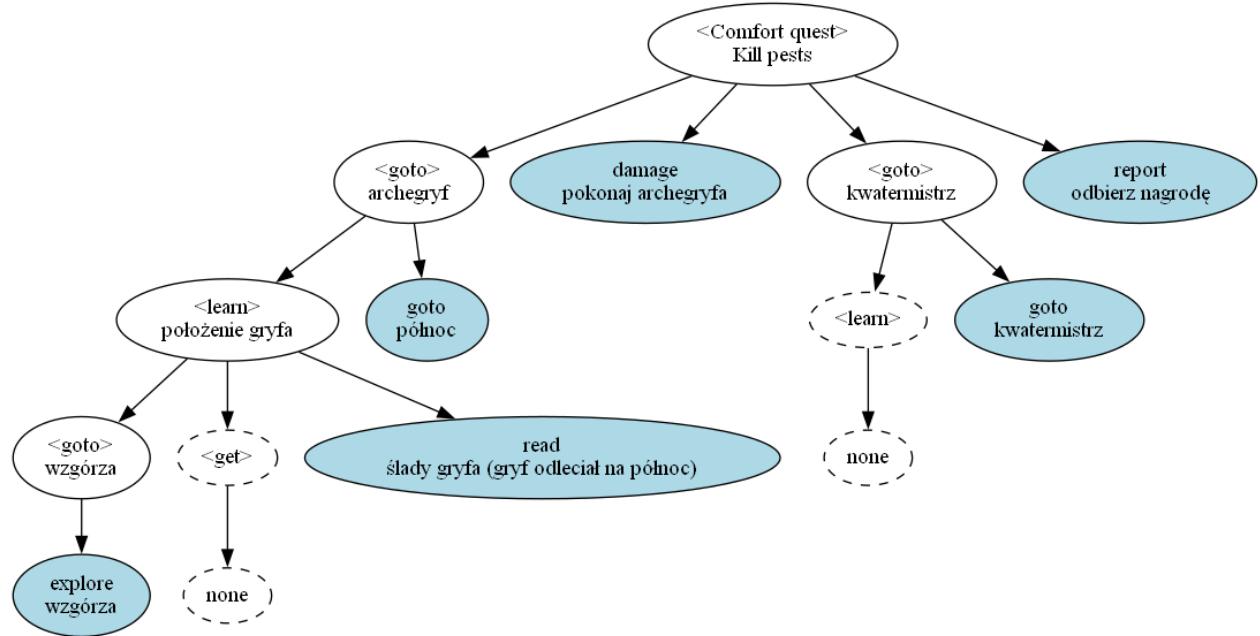
Rysunek B.26: *Ochotnik* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



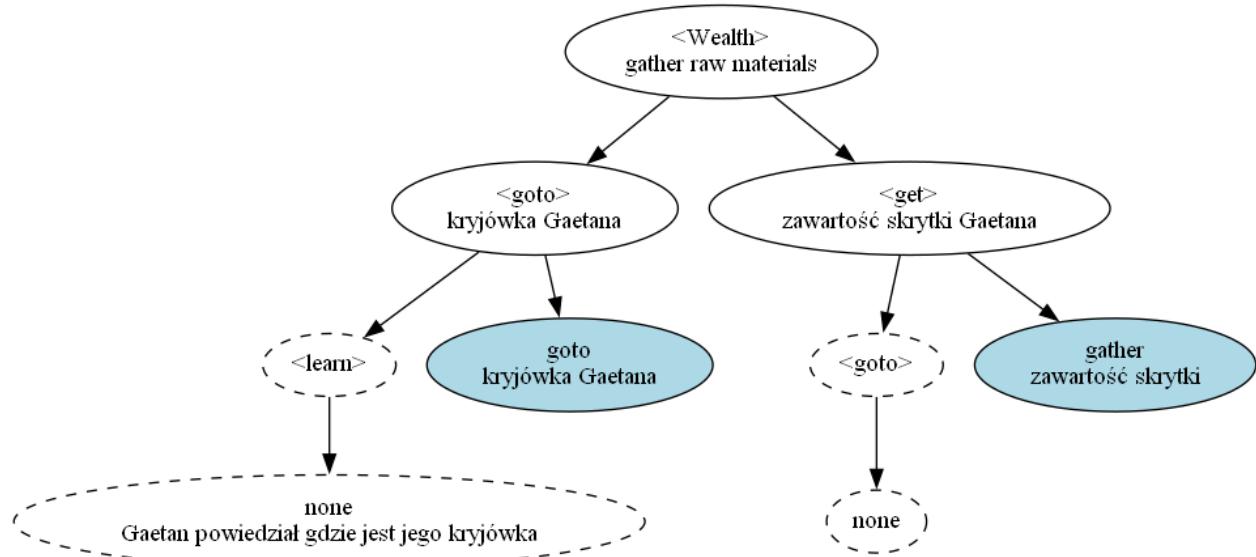
Rysunek B.27: *Zlecenie: Tajemnica wsi Stegny* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



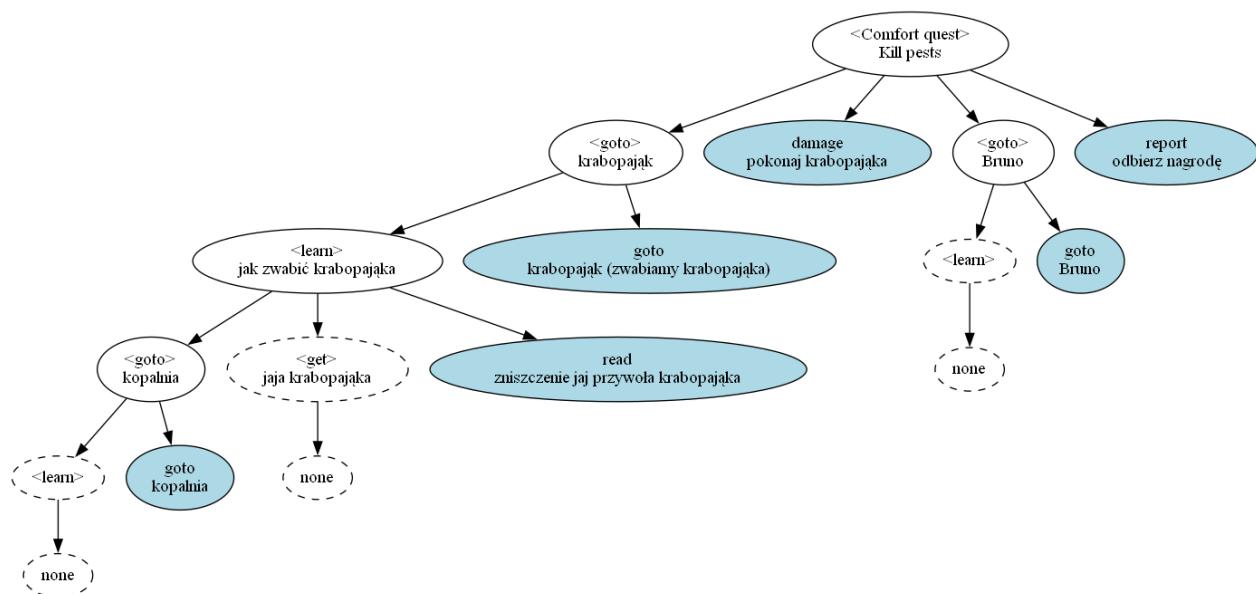
Rysunek B.28: Zlecenie: Zmora kupieckiego traktu (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



Rysunek B.29: Zlecenie: Gryf na wzgórzach (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

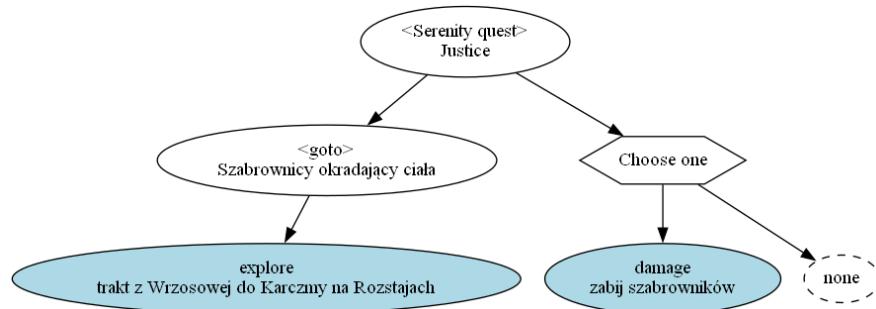


Rysunek B.30: *Bierz co chcesz* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

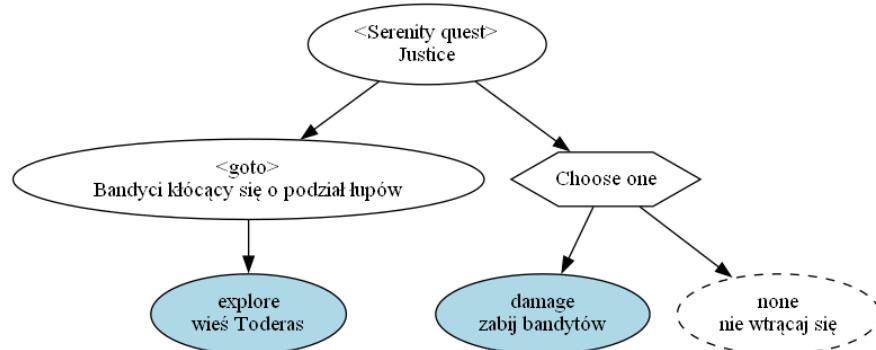


Rysunek B.31: *Zlecenie: Zaginiony brat* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)

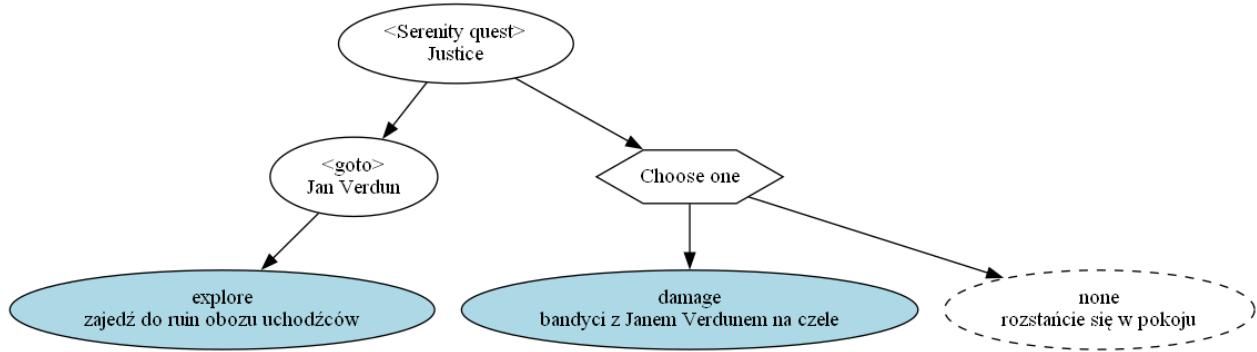
C. Zadania z gry Wiedźmin 3 wyrażone rozszerzonym modelem



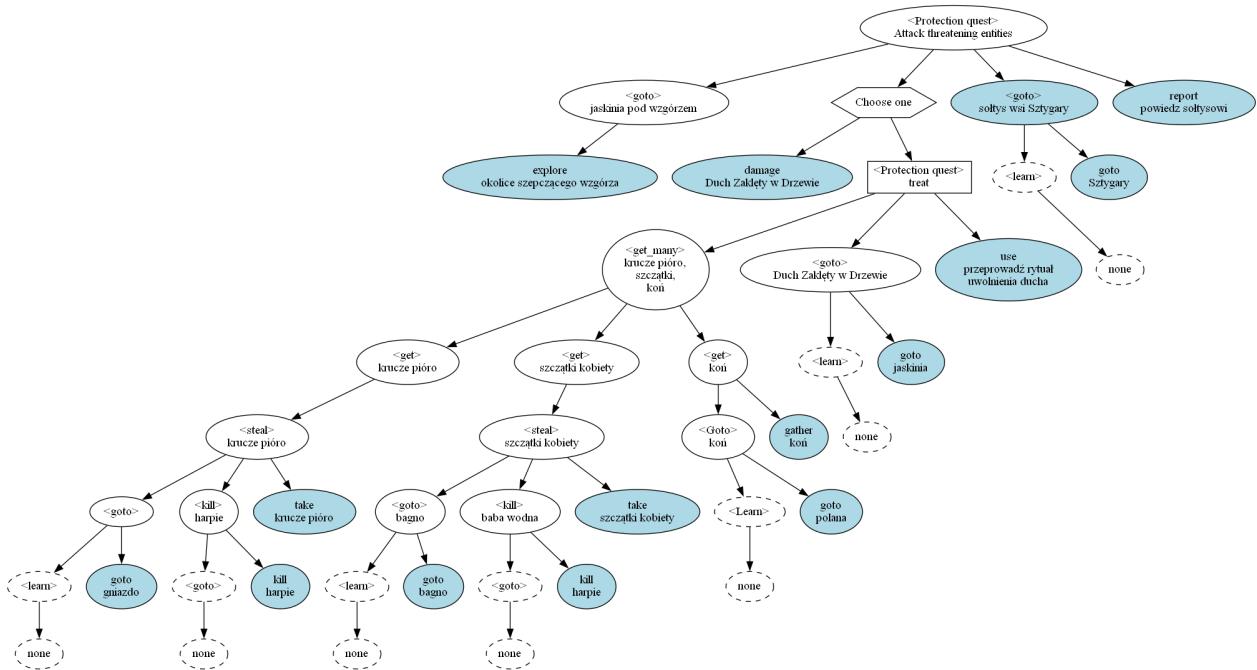
Rysunek C.1: *Szabrownicy* część pierwsza (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



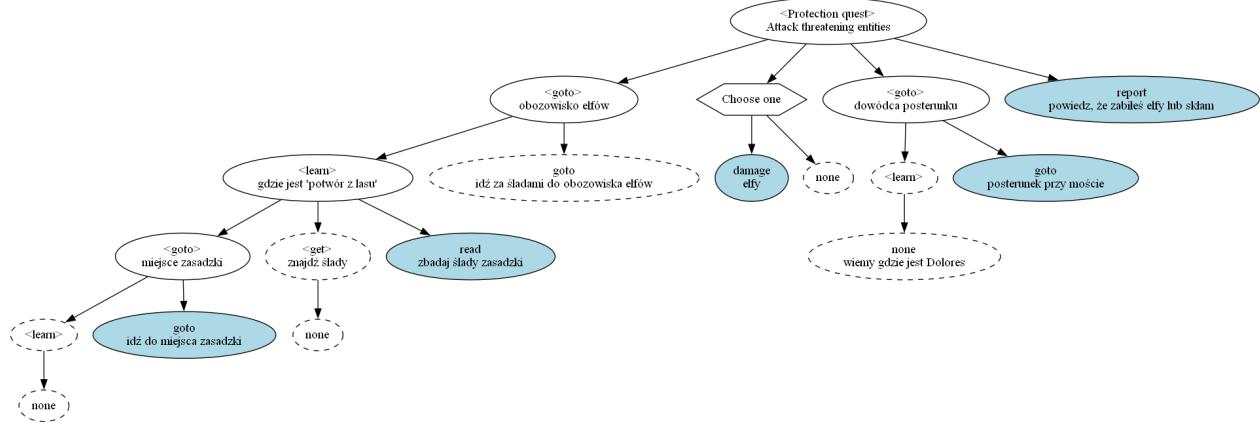
Rysunek C.2: *Szabrownicy* część trzecia (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



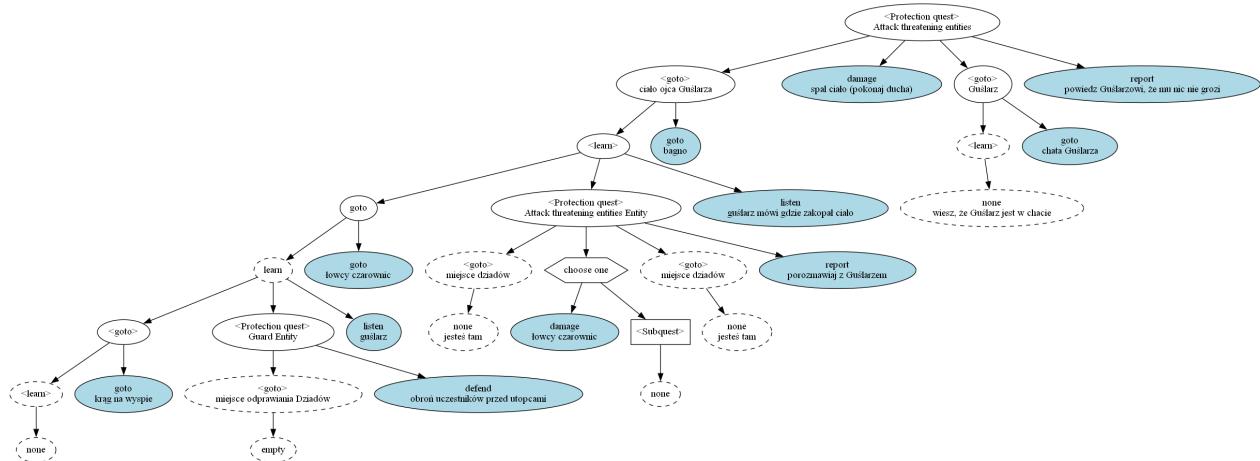
Rysunek C.3: *Na łasce obcych – drugie spotkanie z Janem Verdunem* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



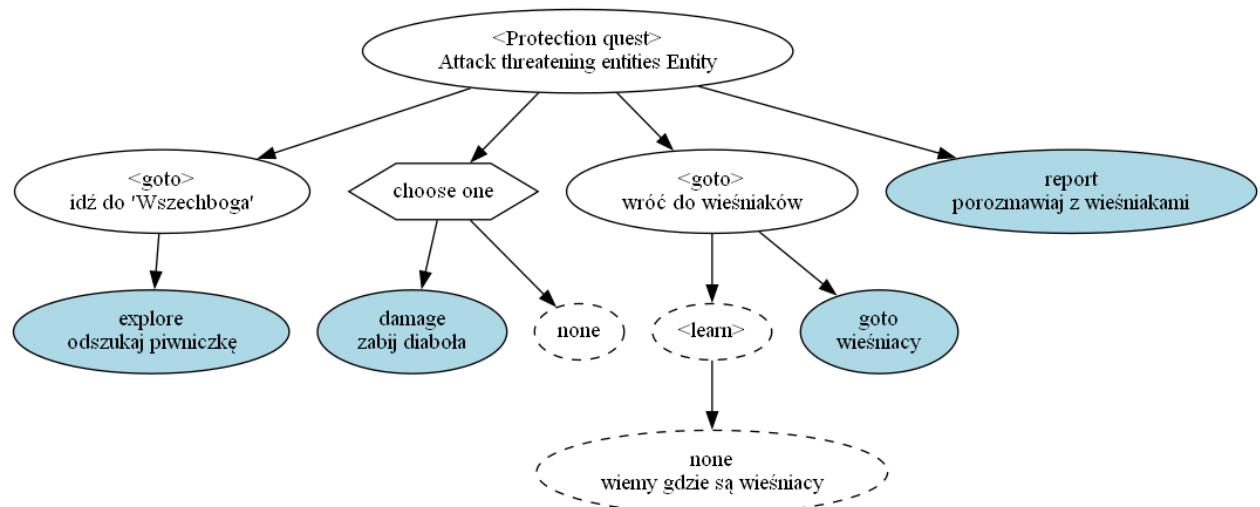
Rysunek C.4: *Szepczące wzgórze* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



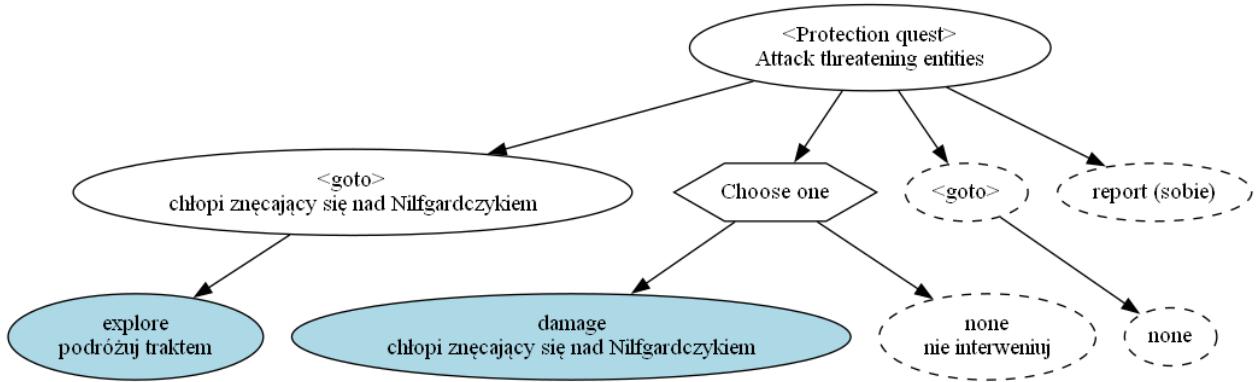
Rysunek C.5: *Zlecenie: Potwór z lasu* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



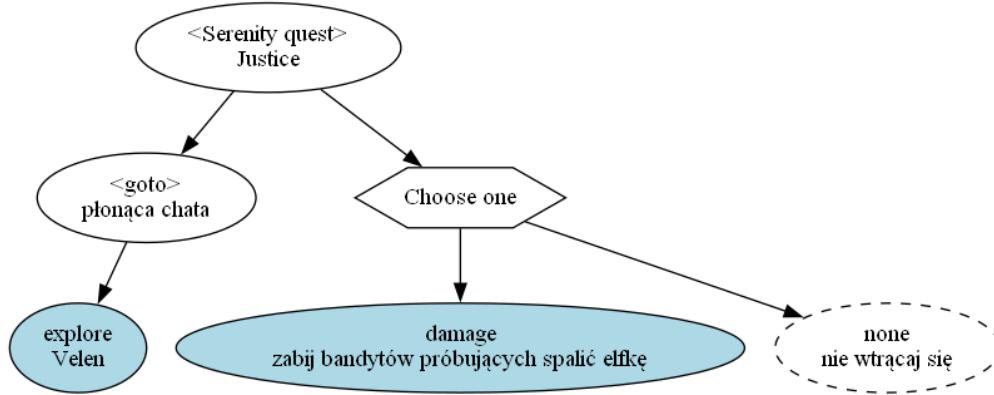
Rysunek C.6: *Dziady* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



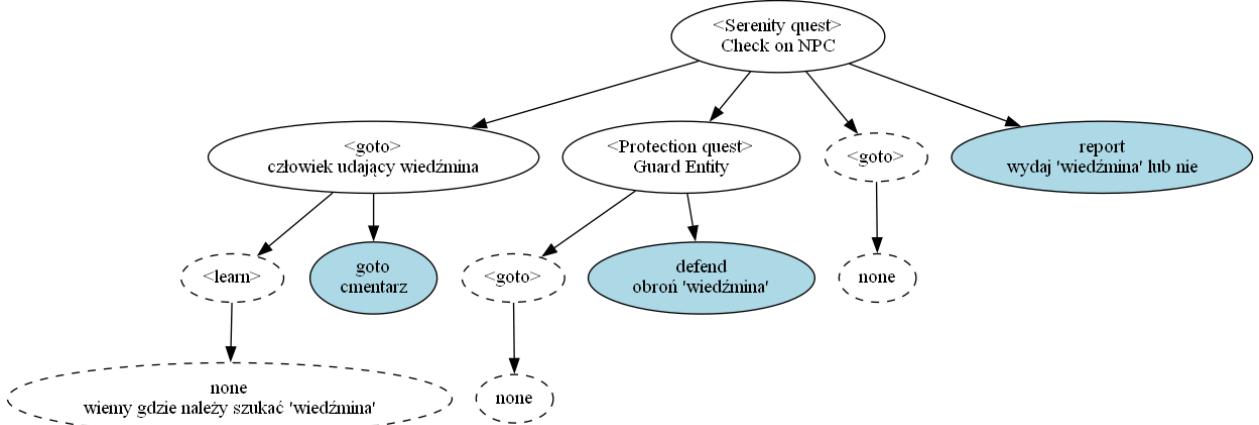
Rysunek C.7: *Opium dla ludu* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



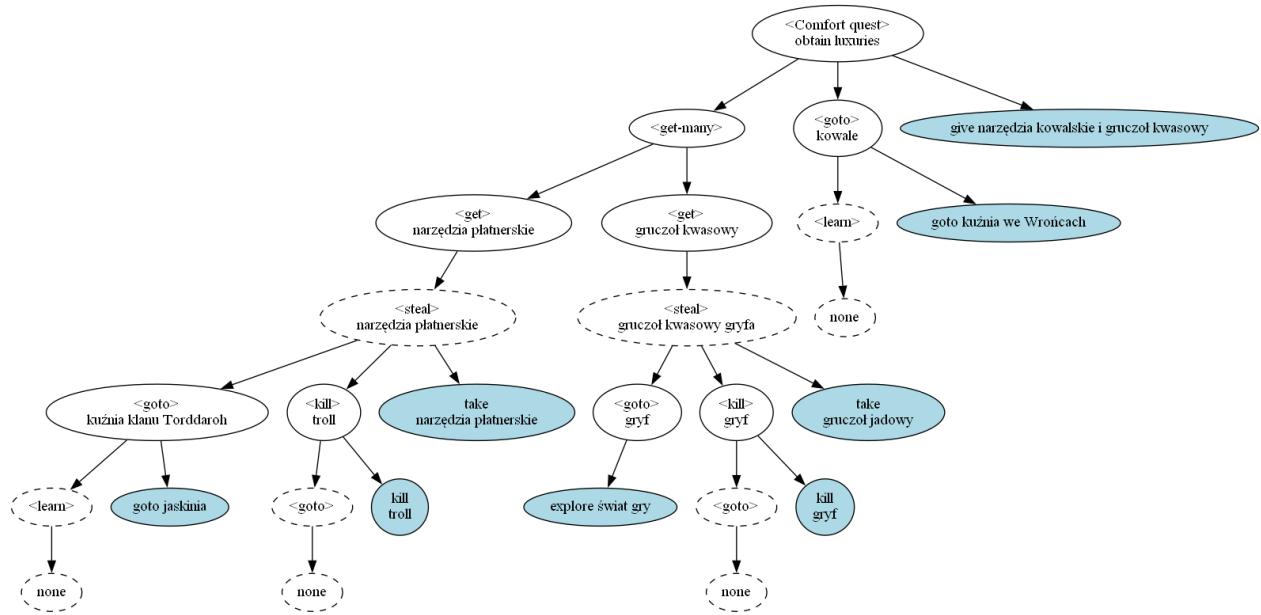
Rysunek C.8: *Samosąd* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



Rysunek C.9: *Śmierć w ogniu* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



Rysunek C.10: *Wiedźmin jak malowany* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)



Rysunek C.11: *Mistrz płatnerstwa* (opracowanie własne na podstawie opisu zadania [22] i gry *Wiedźmin 3*)