Specyfikacja systemu StoryGraph

v. 1.2

Storygraph to system pozwalający zapisać strukturę narracyjną gry fabularnej w postaci modelu grafowego. Fabuła gry przedstawiona jest w postaci grafu stanu świata i produkcji pozwalających go modyfikować. Celem jego stworzenia było umożliwienie tworzenia i testowania fabuł.

Dla kogo przeznaczona jest specyfikacja:

- dla programistów, którzy chcą wykorzystać system StoryGraph w swoim projekcie,
- w połączeniu z pracą *Tworzenie narracji komputerowej gry fabularnej z użyciem transformacji grafowych* (Grabska-Gradzińska, 2022) dla teoretyków, którzy chcą poznać ograniczenia implementacji.

Informacje potrzebne do wykorzystania formatu SHEAF do zapisu stanu świata znajdują się w rozdziałach 1-3. Informacje potrzebne do zaimplementowania transformacji grafowych w rozdziałach 4-5. Informacje potrzebne do wykorzystania systemu StoryGraph jako silnika narracyjnego gry w rozdziale 7.

Spis treści

1. Założenia systemu StoryGraph 4
Obiekty i relacje w świecie gry 4
Format zapisu danych 5
Związek między modelem formalnym a specyfikacją 5
2. Struktury danych opisu świata 6
Świat gry 7
Lokacja z zawartością 7
Obiekt wraz z obiektami podrzędnymi 9
Uwagi 9
3. Opis stanu świata 10
Przykład 10
4. Modyfikacja stanu świata gry 12
Wskazanie fragmentu grafu, którego istnienie jest podstawą akcji 13
Wybór wierzchołków do modyfikacji w oparciu o lewą stronę produkcji 13
Wskazywanie wierzchołków za pomocą referencji tekstowych 14
Odwoływanie się do węzłów lub tablic bezpośrednio wskazanych 14
Odwoływanie się do węzłów ze snopków 14
Instrukcje modyfikujące graf 15
Warunki dodatkowe stosowania produkcji 16
5. Struktury danych w produkcji grafowej 17
Lewa strona produkcji 18
Predykaty stosowalności 18
Instrukcje 19
Typy instrukcji 20
6. Przykładowe produkcje 25
Transakcja kupna-sprzedaży 26
Zabicie smoka, wersja pierwsza 28
Zabicie smoka, wersja druga 29

Zabicie smoka, wersja trzecia 30
Przechodzenie pomiędzy połączonymi lokacjami 31
Zabijanie komarów 31
Zamach skrytobójczy 32
Walka z opcjonalnymi konsekwencjami 33
7. Wczytywanie i zapisywanie świata z i do pliku JSON 34
Poprawność zapisu JSON 34
Serializacja i deserializacja danych 36
8. Stosowanie produkcji 37
Wykorzystanie implementacji referencyjnej jako silnika fabularnego gry 37
Implementacja algorytmu dopasowania lewej strony produkcji do świata gry 38
9. Dobre praktyki nazewnicze 39
Tworzenie opisu produkcji 39

1. Założenia systemu StoryGraph

System StoryGraph służy do definiowania fabuły gry komputerowej (gry wideo) z gatunku gier przygodowych lub innej gry o analogicznych założeniach fabularnych. Umożliwia zdefiniowanie struktury świata gry oraz akcji modyfikujących ten świat dostępnych dla gracza i postaci niezależnych lub wykonywanych automatycznie po zaistnieniu zadanych warunków. Jest to system grafowy, tzn. aktualny stan świata gry wyrażony jest strukturą grafową a wszystkie jego modyfikacje są transformacjami grafowymi.

Specyfikacja obejmuje w szczególności definicję struktur danych i ich zastosowanie do opisu i modyfikacji świata gry. W szczególności istotna jest struktura grafu wielosnopkowego (konstrukcji teoretycznej stworzonej na potrzeby niniejszego systemu), która wykorzystywana jest zarówno w opisie świata jak i jego modyfikacji.

W pierwszej części specyfikacji najpierw przedstawiona jest ogólna definicja struktury reprezentującej graf wielosnopkowy, a potem jest ona doprecyzowana dla poszczególnych zastosowań. Następnie wyjaśniony jest zapis struktur w formacie JSON. Informacje zawarte w tej części wystarcza do stosowania formatu danych opartych o StoryGraph jako formy zapisu informacji o świecie gry. Kolejne dwie części specyfikacji przedstawiają sposób modyfikacji tak zapisanego świata zgodnie z systemem StoryGraph. W drugiej części opisane są operacje na strukturach i produkcje grafowe. Trzecia część to gramatyki opisujące fabuły gier oraz przesłanianie produkcji w ramach danej gramatyki.

System może być wykorzystywany w grach turowych oraz grach czasu rzeczywistego, ale specyfikacja nie obejmuje zasad priorytetyzowania wykonania akcji.

Obiekty i relacje w świecie gry

Wszystkie obiekty w świecie gry przypisujemy do jednej z czterech grup: lokacji (ang. *locations*), postaci (ang. *characters*), przedmiotów (ang. *items*) i informacji fabularnych (ang. *plot elements*). Cechy obiektów są reprezentowane jako atrybuty wierzchołków. Relacje pomiędzy obiektami dzielimy na **relację połączenia**, dotyczącą pary lokacja-lokacja i relacje zależności obiektu podrzędnego od nadrzędnego, dotyczące wszystkich innych par obiektów.

Podstawowe **relacje zależności** to:

- Relacja przynależności dotycząca pary postać-lokacja oraz przedmiot-lokacja. Tyczy się wszystkich obiektów znajdujących się bezpośrednio w lokacji,
- Relacja własności dotycząca pary przedmiot-postać,

- Relacja podporządkowania dotycząca pary postać-postać,
- Relacja zawierania dotycząca pary przedmiot-przedmiot lub postać-przedmiot.

Wszystkie relacje reprezentowane są hierarchiczne z wyjątkiem relacji połączenia. W przyszłości możliwe jest rozszerzenie specyfikacji o inne relacje rozróżniane od powyższych np. za pomocą atrybutów.

Lokacje różnią się od innych obiektów tym, że mogą być elementami relacji połączenia i celem relacji przynależności. Postaci od przedmiotów różnią się tym, że mogą być podmiotami (inicjatorami) produkcji. Aby wejść w bezpośrednią interakcję z innym obiektem, postać musi być połączona relacją przynależności z lokacją, w której ten obiekt się znajduje. Informacje o świecie wynikające z obserwacji świata przez gracza ograniczają się do obiektów znajdujących się w tej samej lokacji.

Format zapisu danych

Wszystkie struktury danych mogą być zapisane do pliku jako obiekty JSON. Obiekt zgodnie ze specyfikacją formatu[1] składa się z par: *nazwa: wartość.* Nazwy muszą być unikatowym łańcuchem znaków. Wartość może być dowolnym typem JSON: łańcuchem tekstowym (ang. *string*), liczbą (ang. *number*), wartością logiczną (ang. *boolean*), tablicą (ang. *array*), obiektem (ang. *object*), wartością nieokreśloną (ang. *null*).

Obiekty zawierające odwołania cykliczne (połączone ze sobą lokacje) przed zapisem zostają zserializowane a przy wczytywaniu a deserializowane. Proces ten opisany jest w rozdziale 7 w podrozdziale *Serializacja i deserializacja danych*.

Związek między modelem formalnym a specyfikacją

System opiera się na teoretycznym modelu grafowym, którego definicje formalne dołączone są w *Dodatku A*, a założenia i szczegółowy opis znajdują się w pracy *Tworzenie narracji komputerowej gry fabularnej z użyciem transformacji grafowych* (I. Grabska-Gradzińska, 2022).

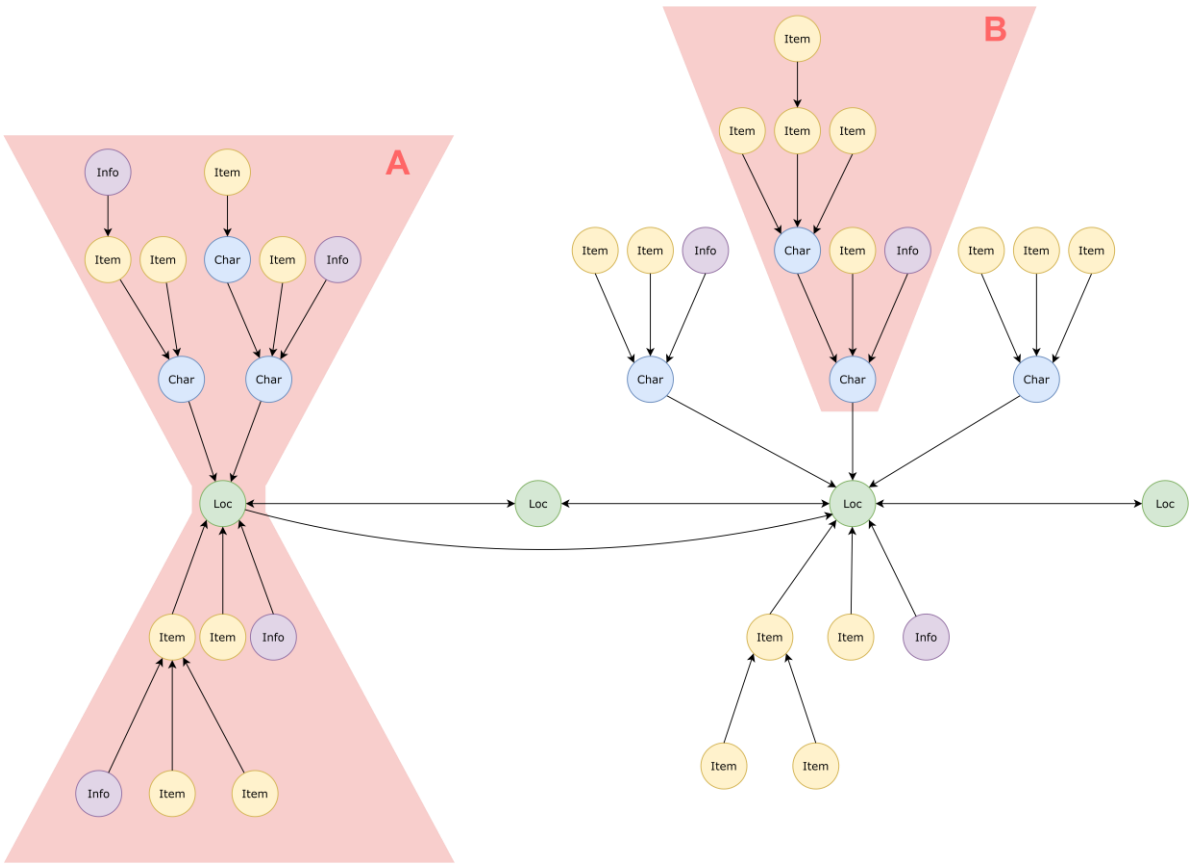
Specyfikacja standardu implementacji obejmuje definicję struktury odpowiadającej aktualnemu stanowi świata (GWSG, instancji grafu czterowarstwowego) oraz definicję struktury odpowiadającej akcji w grze w rozumieniu produkcji grafowej, czyli trójce (*M, C, P*), gdzie *M* jest grafem czterowarstwowym z częściowym bądź pełnym etykietowaniem, *C* jest procedurą sprawdzania predykatów stosowalności, a *P* jest procedurą modyfikującą przekształcany graf.

[1] RFC8259

Struktury odpowiadające aktualnemu stanowi świata (instancja grafu czterowarstwowego wielosnopkowego z pełnym etykietowaniem) i lewej stronie produkcji (czterowarstwowy wielosnopkowy graf z częściowym etykietowaniem) zostały na poziomie specyfikacji uwspólnione, zaproponowano jeden typ danych SHEAF zawierający elementy pozwalające wyrazić oba typy obiektów modelu teoretycznego. Dzięki temu ułatwiony jest proces implementacyjny, w szczególności procedury walidacji mogą opierać się na jednym schemacie JSON Schema, co ułatwia utrzymanie spójności danych.

2. Struktury danych opisu świata

Ze względu na kierunek relacji między obiektami można świat gry wyobrazić sobie jako las podgrafów, których korzeniem jest lokacja. Każdy taki podgraf możemy podzielić na dwie części: postaci, znajdujące się w lokacji i przedmioty znajdujące się w lokacji i wyobrazić sobie na podobieństwo snopka zboża z lokacją w najwęższym miejscu. Z tego powodu główna struktura danych nazywa się SHEAF (snopek).



Rys. 1. Las snopków (graf wielosnopkowy) z wyróżnioną strukturą snopka (lokacja z zawartością) i półsnopka (obiekt wraz z obiektami podrzędnymi).

Świat gry

Typ danych SHEAF, odpowiadający teoretycznej koncepcji grafu wielosnopkowego (por. *Dodatek A, Definicja 8*), zdefiniowany jest w oparciu o powszechnie znane typy danych: łańcuchy (ang. *strings*), liczby (ang. *numbers*), wartości logiczne (ang. *boolean*), tablice (ang. *arrays*) i słowniki[2] (ang. *associative arrays, dictionaries*). Kluczami słowników zawsze są łańcuchy, natomiast związane z tymi kluczami wartości to *null* lub łańcuchy, liczby, tablice i słowniki.

Typ danych SHEAF służy w systemie StoryGraph do opisu kilku typów obiektów, między innymi do wyrażenia **całości świata gry ze wszystkimi obiektami**.

SHEAF to słownik z kluczem `Locations` i związaną z tym kluczem tablicą zawierającą wartości typu LOC-DICT. W przypadku grafu pustego tablica jest pusta.

Kod 1. Przykład struktury SHEAF

```
"Locations": [
                { obiekt typu LOC-DICT }
                            …
            ]
```

Lokacja z zawartością

LOC-DICT to słownik odpowiadający teoretycznej koncepcji podgrafu snopkowego, który służy do odwzorowania **lokacji z zawartością**, może zawierać:

- Klucz `Name`, z którym związany jest łańcuch będący etykietą wierzchołka.
- Klucz `Attributes` i związaną z nim wartość typu ATTR-DICT, czyli słownik o wartościach *null*, tekstowych, logicznych lub liczbowych.
- Klucze `Characters`, `Items` i `Narration`. Związane z nimi są tablice z wartościami, odpowiednio, typu CHAR-DICT, ITEM-DICT lub NARR-DICT. Te znajdujące się w tablicach słowniki reprezentują węzły z warstwy odpowiednio postaci, przedmiotów i informacji fabularnych. Wystąpienie węzła w tablicy oznacza istnienie relacji między obiektem znajdującym się w tablicy a obiektem nadrzędnym.
- Klucz `Connections`. Związana z nim jest tablica z wartościami typu CONN-DICT. Odwzorowuje relacje połączenia pomiędzy lokacjami. Nie jest to klucz związany z definicją grafu snopkowego jako takiego. CONN-DICT to słownik zawierający klucz `Destination`

[2] Terminu „słownik” będę używała na określenie struktury danych, która przechowuje pary klucz – wartość, niezależnie od tego, czy będę odnosiła się do języka python (*dictionary*), czy formatu JSON (*JSON object*) czy innych struktur danych (*associative array*).

i związaną z nim referencję wskazującą na którąś z lokacji w danym SHEAF. Referencją może być klucz `Name` obiektu połączonego, klucz `Id` obiektu lub adres obiektu w pamięci podczas działania aplikacji. W zapisie do pliku adresy obiektu w pamięci zamieniane są na wartość tekstową, identyfikator obiektu istniejący lub definiowany na bieżąco. Występowanie takiego elementu w `Connections` oznacza, że istnieje jednokierunkowe połączenie z lokacji omawianego obiektu do lokacji wymienionej jako wartość klucza `Destination`. Żeby połączenie było dwustronne, to w obu obiektach lokacji musi znajdować się odpowiedni zapis w tablicy `Connections`. Złożenie wszystkich słowników CONN-DICT daje nam pełną informację o połączeniach między lokacjami. Słownik CONN-DICT może też zawierać klucz `Attributes` i związaną z nim wartość typu ATTR-DICT.

- Klucz `Id`, z którym związany jest łańcuch będący poprawnym identyfikatorem w sensie popularnych języków programowania. Te łańcuchy muszą być unikalne w ramach danego SHEAF. Dla uniknięcia niejednoznaczności powinny być też różne od nazw (etykiet)[3]. Klucz ten nie odwzorowuje żadnej własności modelu grafowego, wprowadzony został dla ułatwienia implementacji instrukcji i zapisu struktury w pliku tekstowym.

Kod 2. Przykład struktury LOC-DICT

```
{
  "Id": "140686910713664",
  "Name": "Road",
  "Attributes": { },
  "Characters": [obiekty typu CHAR-DICT],
  "Items": [obiekty typu ITEM-DICT],
  "Narration": [obiekty typu NARR-DICT],
  "Connections": [
    {
      "Destination": "140686910710080"
    },
    {
      "Destination": "140686911964288"
    }
  ]
}
```

[3] Por. rozdział: „Dobre praktyki nazewnicze”

Obiekt wraz z obiektami podrzędnymi

CHAR-DICT oraz ITEM-DICT to słowniki odpowiadające koncepcji podgrafu półsnopkowego, reprezentujący **obiekt wraz z obiektami podrzędnymi**, zbudowane analogiczne jak LOC-DICT, ale nie zawierające klucza `Connections`. CHAR-DICT dodatkowo może zawierać klucz `IsObject` wskazujący daną postać jako podmiot produkcji, czyli postać inicjującą jej wykonanie. Klucz ten ma wartość *true*/*false* i ma zastosowanie wyłącznie do produkcji.

NARR-DICT to słownik taki jak LOC-DICT, ale mogący zawierać co najwyżej klucze `Name`, `Id` i `Attributes`, ponieważ na bazie węzła z warstwy narracyjnej nie można zbudować nietrywialnego podgrafu półsnopkowego. Nie jest to ograniczenie związane z modelem formalnym, zostało narzucone dopiero w specyfikacji systemu, być może zostanie zmienione w kolejnych wersjach specyfikacji.

Kod 3. Przykład struktury CHAR-DICT

```
{
  "Name": "Main_hero",
  "Attributes": { },
  "Characters": [obiekty typu CHAR-DICT ],
  "Items": [obiekty typu ITEM-DICT],
  "Narration": [obiekty typu NARR-DICT],
}
```

Uwagi

- Zbiory etykiety obiektów różnych kategorii muszą być rozłączne, czyli w szczególności nie możemy mieć postaci o etykiecie identycznej co przedmiot. Może za to wystąpić wiele obiektów z jednej kategorii o tych samych nazwach (np. 78 wierzchołków z etykietą „Droga”). Wierzchołek może mieć zagnieżdżonych kilka podwierzchołków o takich samych etykietach (np. „Oddział” i w nim 5 × „Żołnierz”).
- Słownik najwyższego poziomu z pojedynczym kluczem `Locations` nie jest konieczny (na najwyższym poziomie mogłaby po prostu być tablica), ale warto go zostawić dla zwiększenia czytelności i jako ubezpieczenie na przyszłość. Jeśli kiedyś będą potrzebne globalne atrybuty grafu, to w tym słowniku doda się klucz `Attributes` i już.

3. Opis stanu świata

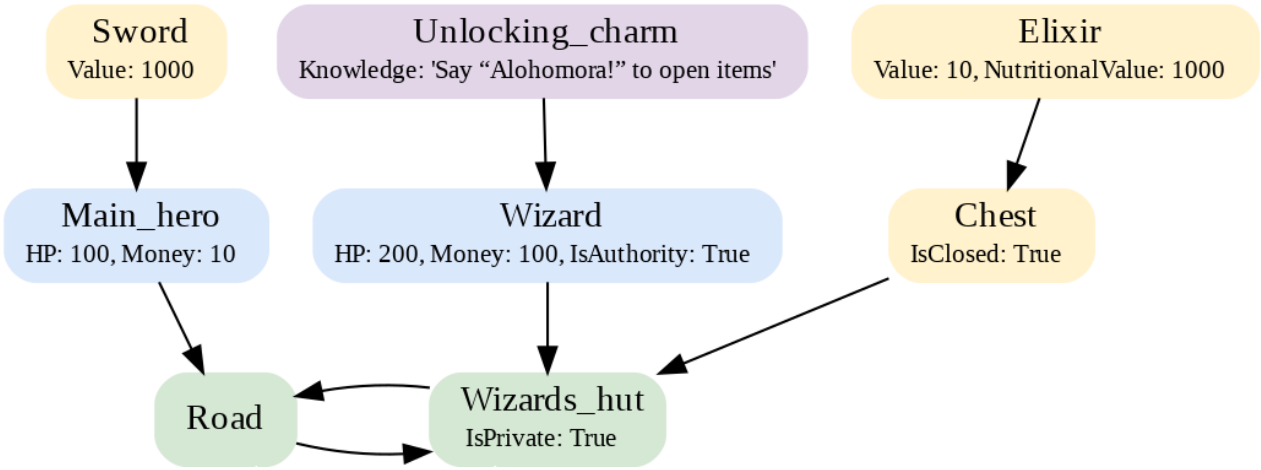
Świat opisywany jest przez instancję grafu wielosnopkowego (por. *Dodatek A, Definicja 05*). Reprezentowany jest przez strukturę SHEAF, narzucone jest jednak na nią kilka dodatkowych ograniczeń wynikających definicji instancji grafu:

- Klucze `Name` są obowiązkowe (każdy wierzchołek musi mieć etykietę).
- Kluczowi `Attributes` musi być przypisana wartość typu ATTR-DICT, ale nie jest dozwolona wartość *null*.
- Klucze `Id` w odniesieniu do świata w trakcie rozgrywki nie są używane, przypisywane są więc obiektom podczas serializacji a usuwane podczas deserializacji po tym jak odniesienia do `Id` w słowniku `Destination` zamienione zostaną na wskaźniki do obiektów.

Siłą rzeczy wszystkie przykłady w tekście pokazują obiekty zserializowane.

Przykład

Przykładowy świat zawierający dwie lokacje i w każdej po jednej postaci. Pierwsza postać, Main_hero, posiada jeden przedmiot, miecz. Druga postać, Wizard, nie posiada przedmiotów, ale w lokacji, w której przebywa, znajduje się przedmiot Chest zawierający przedmiot Elixir.



Rys. 2. Wizualizacja przykładowego świata.

Kod 4. Struktura JSON świata z rysunku 2.

```
"Locations": [
                {
                    "Id": "140686910713664",
                    "Name": "Road",
                    "Characters": [
                        {
                            "Name": "Main_hero",
                            "Attributes": {
                                "HP": 100,
                                "Money": 10
                            },
                            "Items": [
                                {
                                    "Name": "Sword",
                                    "Attributes": {
                                        "Value": 1000
                                    }
                                }
                            ],
                        }
                    ],
                    "Connections": [
                        {
                            "Destination": "140686910710080"
                        }
                    ]
                },
                {
                    "Id": "140686910710080",
                    "Name": "Wizards_hut",
                    "Attributes": {
                        "IsPrivate": true
                    },
                    "Characters": [
                        {
                            "Name": "Wizard",
                            "Attributes": {
                                "HP": 200,
                                "Money": 100,
                                "IsAuthority": true
                            },
```

```
                        "Narration": [
                            {
                                "Name": "Unlocking_charm",
                                "Attributes": {
                                    "Knowledge": "Say “Alohomora!” to open items."
                                }
                            }
                        ]
                    }
                ],
                "Items": [
                    {
                        "Name": "Chest",
                        "Attributes": {
                            "IsClosed": true
                        },
                        "Items": [
                            {
                            "Name": "Elixir",
                            "Attributes": {
                                "Value": 10,
                                "NutritionalValue": 1000
                                }
                            }
                        ]
                    }
                ],
                "Connections": [
                    {
                        "Destination": "140686910713664"
                    }
                ]
            }
        ]
```

Czytelnik niezainteresowany modyfikacją stanu świata gry poprzez produkcje proszony jest o przejście do rozdziału 7.

4. Modyfikacja stanu świata gry

Podstawą rozgrywki są wydarzenia dziejące się w świecie gry. Wydarzenia mogą być inicjowane przez gracza, przez postaci niezależne lub wynikać z logiki funkcjonowania świata (zmiana

pór dnia, utrata sił życiowych przez bohaterów etc.). Typy możliwych wydarzeń narzuca mechanika gry. Ponieważ system StoryGraph oparty jest o formalny model grafowy, działania w grze wyrażone są w sposób bazujący na idei transformacji grafowych, czyli poprzez produkcje[4].

Ideą działania produkcji grafowej jest wskazanie fragmentu grafu (podgrafu), którego istnienie jest podstawą do wykonania akcji a następnie modyfikacja węzłów i krawędzi grafu.

Wskazanie fragmentu grafu, którego istnienie jest podstawą akcji

Wskazanie podgrafu odbywa się poprzez zdefiniowanie grafu, którego odpowiednika szukamy w grafie źródłowym (problem znajdowania podgrafu). W systemie StoryGraph opisywana jest przez graf wielosnopkowy z częściowym bądź pełnym etykietowaniem (por. *Dodatek A*, *Definicja 08* i *Definicja 13*). Nazywamy go lewą stroną produkcji, ponieważ w tradycyjnym modelu produkcji grafowych usuwamy go z grafu źródłowego i zastępujemy innym grafem, który nazywamy prawą stroną produkcji. W systemie StoryGraph zasada działania produkcji jest trochę inna, ale nazewnictwo pozostało, dlatego graf, którego odpowiednika szukamy w świecie gry, nazywamy lewą strona produkcji.[5]

W systemie StoryGraf po znalezieniu odpowiednika lewej strony produkcji w grafie wskazujemy – za pomocą instrukcji – jakim modyfikacjom poddajemy wybrane węzły grafu.

Wybór wierzchołków do modyfikacji w oparciu o lewą stronę produkcji

Po znalezieniu odpowiednika lewej strony produkcji w grafie wszystkie wierzchołki grafu możemy podzielić na trzy grupy: 1) węzły znajdujące się bezpośrednio w dopasowanym podgrafie, 2) węzły będące w relacji podrzędności do węzłów znajdujących się bezpośrednio w dopasowanym podgrafie, 3) pozostałe węzły.

Węzły z pierwszej i drugiej grupy możemy wskazać poprzez referencje tekstowe przedstawione w następnym rozdziale. Modyfikacji węzłów trzeciej grupy specyfikacja w wersji 1.2 nie definiujemy, nie jest możliwa ich zmiana. Dopuszczalne jest definiowanie sposobów ich wskazania i modyfikacji w konkretnych implementacjach, ale w sposób niezakłócający działania mechanizmów modyfikowania węzłów z pierwszej i drugiej grupy wskazany w specyfikacji.

[4] Zauważmy, że mówiąc o modyfikacji świata za pomocą produkcji mamy na myśli modyfikacje inicjowane przez gracza lub system w trakcie rozgrywki. Nie dotyczy to działań na grafie w momencie jego tworzenia przez projektanta.
[5] Różnica między sposobem tradycyjnym a zdefiniowanym w systemie StoryGraph znajduje się w dodatku B

Wskazywanie wierzchołków za pomocą referencji tekstowych

W trakcie modyfikacji świata gry niezbędne jest wskazywanie wierzchołków, które mają wziąć udział w procesie modyfikacji. W tym celu powstała koncepcja referencji do wierzchołków, czyli łańcuchów tekstowych identyfikujących wierzchołek w ramach danej struktury SHEAF.

Odwoływanie się do węzłów lub tablic bezpośrednio wskazanych

Referencja to łańcuch, który w ramach danej struktury SHEAF w jednoznaczny sposób wskazuje słownik reprezentujący wierzchołek albo krawędź grafu (czyli lokację, postać, przedmiot, element fabuły, połączenie pomiędzy lokacjami). Jako referencji można użyć:

- Etykiety wskazywanego słownika, czyli wartości związanej z jego kluczem `Name`. Aby taka forma referencji była jednoznaczna etykieta musi być unikatowa w ramach danego SHEAF.
- Identyfikatora wskazywanego słownika, czyli wartości związanej z jego kluczem `Id`. Jest to mechanizm umożliwiający rozróżnianie węzłów o identycznych etykietach i z definicji wartość klucza `Id` jest jednoznaczna w ramach struktury SHEAF.
- Łańcucha złożonego z identyfikatora albo unikatowej etykiety, znaku „/” i łańcucha `Characters`, `Items` albo `Narration`. Ta forma referencji pozwala wskazywać tablice.

Dwa pierwsze typy referencji będziemy dalej nazywać NODE-REF, a ostatnią: ARRAY-REF. Obecna wersja specyfikacji standardu nie obejmuje referencji krawędziowych. Jeżeli w przyszłości będziemy odwoływać się do krawędzi, to wprowadzimy także pojęcie CONNECTION-REF.

Odwoływanie się do węzłów ze snopków

Wiele produkcji opisujących akcje wykonywane w świecie gry będzie musiało wykonywać operacje nie tylko na słownikach jawnie wskazanych, lecz również na słownikach będących z nimi w relacji zależności, czyli będących ich potomkami w grafie stanu świata gry, czyli wewnątrz snopka.

Operacje takie pojawiają się np. wtedy, gdy mamy po lewej stronie jakąś postać, a operacje chcemy wykonać na wszystkich przedmiotach przez nią posiadanych. Do wskazywania takich, być może pustych, zbiorów wierzchołków zagnieżdżonych hierarchicznie wewnątrz jawnie wskazanych wierzchołków służą multireferencje.

Multireferencja (ang. *multireference*) to łańcuch złożony z rozdzielonych znakiem „/” segmentów. Pierwszym segmentem, tak samo jak w przypadku zwykłych referencji, jest identyfikator słownika albo jego unikatowa etykieta. Następne segmenty to na przemian:

- jeden z trzech łańcuchów `Characters`, `Items` albo `Narration`, wskazujący jedną z tablic zagnieżdżonych w rozważanym słowniku,
- nazwa, wskazująca wszystkie słowniki o tej nazwie zagnieżdżone w rozważanej tablicy (może ich być zero i to nie będzie błędem).

Multireferencje jako typ danych będziemy dalej nazywać: NODE-MULTIREF.

Możliwa jest multireferencja jednosegmentowa, która sprowadza się do referencji NODE-REF.

Multireferencje służą tylko do wskazywania słowników, nie tablic. Co za tym idzie, w multireferencji skonstruowanej zgodnie z powyższymi zasadami zawsze będzie nieparzysta liczba segmentów.

Zamiast nazwy można podać znak „*”, co oznacza, że w tym miejscu może wystąpić słownik o dowolnej etykiecie (wyrażonej przez klucz `Name`). Dwuznak „**” oznacza zaś, że w tym miejscu może wystąpić dowolna liczba dowolnych segmentów (w tym zero). Pojedynczej gwiazdki można użyć w środku lub na końcu multireferencji, ale nie na jej początku.

Znajdowanie rozwinięć podwójnej gwiazdki działa na zasadzie przeszukiwania drzewa wszerz (BFS). Dla multireferencji `"Hero/**/Items/Sword"` w pierwszej kolejności musi być znaleziony `"Hero/Items/Sword"`, w drugiej `"Hero/Items/Backpack/Items/Sword"` oraz `"Hero/Characters/Princess/Items/Sword"` (kolejność w ramach referencji o tej samej liczbie segmentów nie jest ustalona), a dopiero w trzeciej kolejności `"Hero/Items/Backpack/Items/Scabbard/Items/Sword"`.

Przy wykonywaniu operacji najpierw rozwijamy wszystkie multireferencje w kolejności wynikającej z przeszukiwania drzewa wszerz znajdując konkretne obiekty w świecie a później wykonujemy operacje po kolei.

Jeśli w multireferencji jest więcej niż jeden dwuznak **„**”**, to są one rozwijane tak, aby rozwinięcia tych bliżej początku łańcucha miały możliwie jak najmniej segmentów.

Instrukcje modyfikujące graf

Modyfikacje grafu mogą oznaczać zmiany węzłów (tworzenie, usuwanie, zmiany własności) lub krawędzi. W systemie StoryGraph możemy modyfikować dowolne węzły, ale krawędzie tylko te, które reprezentują relacje podrzędności (por rozdział xx). Ponieważ relacje te wyznaczane są przez zagnieżdżanie węzłów podrzędnych w nadrzędnych w strukturze JSON, modyfikacja krawędzi może być wyrażona przez przesuwanie węzłów w hierarchii. Oznacza to, że każdą modyfikację grafu możemy sprowadzić do operacji na węzłach (tworzenie, usuwanie, zmiany własności oraz zmiana położenia)

Każda instrukcja zawiera w sobie wskazanie wybranego węzła i inne potrzebnie do jej wykonania parametry.

W momencie wystąpienia potrzeby modyfikacji krawędzi reprezentujących relacje inne niż podrzędności w kolejnych wersjach specyfikacji zostaną zdefiniowane instrukcje modyfikacji krawędzi.

Warunki dodatkowe stosowania produkcji

Podstawowym warunkiem wykonania produkcji jest dopasowanie jej lewej strony do grafu, który chcemy modyfikować. Modelowane przy pomocy produkcji zmiany stanu świata gry często w praktyce uzależnione są jednak od czegoś więcej niż tylko samo istnienie wierzchołków mających określone etykiety i połączonych w określony sposób krawędziami.

Takie dodatkowe ograniczenia znane są w literaturze jako warunki stosowalności albo predykaty stosowalności (ang. *preconditions*). Są one zapisywane jako wyrażenia logiczne odwołujące się do wartości atrybutów w dopasowanych do lewej strony produkcji wierzchołkach transformowanego grafu. Na poziomie specyfikacji nie jest narzucony język wyrażeń logicznych, zakłada się, że jest to język właściwy dla implementacji, tzn. określając wartość logiczną predykatu korzystamy z funkcji *eval* wbudowanej w używany język programowania.

Oprócz warunków odwołujących się do wartości atrybutów prawie na pewno potrzebne będą warunki uzależnione od innych własności świata gry. Dlatego właśnie warunki są zapisywane jako słowniki — taki format, tak samo jak w przypadku instrukcji, pozwala łatwo wyrazić nowe rodzaje warunków.

Na przykład wdarcie się przemocą do komnaty może być fabularnie możliwe tylko wtedy, gdy wyważająca drzwi postać ma atrybut siły o wartości większej niż zadana. Innym przykładem będzie akcja skrytobójczego, potajemnego zabicia ofiary, którą można wykonać tylko wtedy, gdy w danej lokacji nie ma żadnej trzeciej postaci. Aby móc ją zapisać potrzebny jest taki warunek stosowalności, który potrafi odwołać się do liczby elementów tablicy „Characters” zagnieżdżonej w słowniku reprezentującym daną lokację.

Procedura modyfikacji grafu będzie więc wyglądała następująco:

1) Znalezienie dopasowania lewej strony produkcji → 2) sprawdzenie warunków stosowalności → 3) wykonanie instrukcji na wskazanych węzłach

5. Struktury danych w produkcji grafowej

Produkcja jest słownikiem z trzema kluczami głównymi, wynikającymi z definicji: `LSide`, `Preconditions` i `Instructions` i czterema kluczami dodatkowymi: `Title`, `TitleGeneric`, `Description` oraz `Override`. Pierwszy definiuje lewą stronę produkcji, drugi warunki stosowalności, a trzeci procedurę modyfikującą przekształcany graf. Rola kolejnych kluczy nie wynika bezpośrednio z definicji grafu *n*-warstwowego, ale jest związana z rozróżnianiem produkcji, ustalaniem częściowego porządku i wprowadzeniem mechanizmu przysłaniania produkcji.

Tylko klucz `LSide` jest obowiązkowy, pozostałe są opcjonalne. Dopuszcza to produkcje dodatkowych warunków, które wykonają się, jeśli tylko znajdziemy w świecie dopasowanie lewej strony. Dopuszcza to także produkcje bez instrukcji, czyli takie, które nie zmieniają stanu świata gry. Na pierwszy rzut oka wydaje się to nie mieć praktycznego sensu, ale być może przydadzą się one do reprezentowania stanów kończących grę (np. bohater musi odzyskać i odnieść do świątyni trzy talizmany, nie jest ważne w jakiej kolejności to zrobi, gra się kończy, gdy produkcja z lewą stroną pokazującą wszystkie talizmany leżące na ołtarzu zostanie dopasowana do grafu stanu świata).

W implementacji referencyjnej (JSON schema) wymagamy istnienia wszystkich kluczy, ale mogą mieć jako wartość pustą tablice.

Kod 5. Struktura produkcji w zapisie JSON.

```
{
   "Title": "Nazwa produkcji",
   "TitleGeneric": "Nazwa produkcji bazowej dla danej produkcji",
   "Description": "Opis produkcji wyjaśniający ją i wykorzystywany przy tworzeniu gier
    typu tekstowego.",
   "Override": liczba z zakresu 0-2,
   "LSide": {},
   "Preconditions": [],
   "Instructions": []
 }
```

Lewa strona produkcji

Lewa strona produkcji reprezentowana jest przez strukturę SHEAF. Od struktur wykorzystanych do opisu świata różni się tym, że:

- Klucze `Name` nie są obowiązkowe (wierzchołek nie musi mieć etykiety).
- Możemy także nie definiować dokładnej wartości dla kluczy w tablicy `Attributes`, tylko przypisać danemu kluczowi wartość *null*. Zakresy konkretnych wartości definiowane są wtedy w słowniku `Preconditions` lub dopasowany jest podgraf ze zdefiniowanym atrybutem o danej nazwie niezależnie od jego wartości.
- Klucze `Id` w odniesieniu do obiektów lewej strony są używane w definiowaniu nie tylko połączeń, ale także warunków stosowalności i instrukcji, nie są więc usuwane podczas deserializacji po tym, jak odniesienia do `Id` w słowniku `Destination` zamienione zostaną na wskaźniki do obiektów. Serializacji danych lewej strony produkcji nie wykonujemy, ponieważ nie jest modyfikowana w trakcie działania aplikacji, więc nie jest zapisywania do pliku.

Zauważmy, że ponieważ ograniczenia w wykorzystaniu możliwości struktury SHEAF dotyczą właśnie światów, to każdy świat będzie poprawną lewą stroną produkcji a na odwrót niekoniecznie. Co to oznacza w praktyce? Możemy szukać w świecie dopasowania konkretnego jego fragmentu, ale możemy lewą stronę produkcji zdefiniować bardziej ogólnie, żeby pasowała do różniących się fragmentów światów. Na przykład, jeżeli nie zdefiniujemy w lewej stronie klucza `Name` dla postaci, to możemy szukać dowolnej postaci lub dowolnej postaci o jakimś atrybucie, jeśli taki został w lewej stronie wskazany. Te możliwości często wykorzystujemy w projektowaniu produkcji.

Predykaty stosowalności

Predykaty stosowalności umieszczone są w liście w kluczu `Preconditions`. Każdy predykat jest słownikiem ze ściśle określoną liczbą i zakresem kluczy. Standard definiuje dwa typy takich obiektów. Dopuszczalne jest w ramach konkretnych implementacji rozszerzanie funkcjonalności systemu o inne obiekty.

Kolejność sprawdzania predykatów nie ma znaczenia. Wartością zwracaną po sprawdzeniu każdego warunku jest *true* lub *false*. Procedura sprawdzania predykatów zwraca sumę logiczną wartości poszczególnych predykatów. Zwrócenie wartości *false* spowoduje odrzucenie produkcji mimo dopasowania lewej strony produkcji do instancji świata.

Obiekty określone w standardzie StoryGraph 1.2 to:

```
{ "Cond": "BOOLEAN-EXPRESSION" }
{ "Count": "NODE-MULTIREF", "Min": NUMBER, "Max": NUMBER }
```

- `Cond` pozwala określić zależność arytmetyczną lub logiczną pomiędzy różnymi wartościami atrybutów obiektów wskazanych w lewej stronie produkcji, gdzie parametr `BOOLEAN-EXPRESSION` jest wyrażeniem logicznym języka programowania (np. Python, GDScript),
- `Count` pozwala ograniczyć od dołu lub od góry (jedno z ograniczeń można pominąć) liczność obiektów spełniających podaną multireferencję. Liczba słowników, które się do niej w grafie stanu świata dopasują, jest porównywana z podanymi w warunku ograniczeniami i jeśli się w nich mieści, to warunek jest spełniony. Jeden z parametrów `Min`, `Max` można pominąć.

Przykłady użycia:

Na przykład wdarcie się przemocą do komnaty może być fabularnie możliwe tylko wtedy, gdy wyważająca drzwi postać ma atrybut siły wynoszący co najmniej 8. Jeśli po lewej stronie jest wierzchołek z unikalną nazwą „Hero” i to on reprezentuje postać wykonującą tę akcję, to predykat dla tej produkcji można by zapisać jako:

```
{ "Cond": "Hero.Str >= 8" }
```

Przykładowe zliczania obiektów: liczby bohaterów w danej lokacji (oczekujemy co najwyżej dwóch postaci), liczby mieczy dostępnych bezpośrednio w ekwipunku bohatera (mamy nadzieję na co najmniej dwa) oraz liczby mieczy dostępnych dla bohatera zarówno bezpośrednio w ekwipunku jak i zagnieżdżonych w dowolnym z jego węzłów potomnych (akceptujemy nie mniej niż dwa, ale nie więcej niż cztery miecze).

```
{ "Count": "Somewhere/Characters/*", "Max": 2 }
{ "Count": "Hero/Items/Sword", "Min": 2 }
{ "Count": "Hero/**/Items/Sword", "Min": 2, "Max": 4 }
```

Instrukcje

Instrukcje są zebranymi w tablicę słownikami ze ściśle określoną liczbą i zakresem kluczy. Standard StoryGraph 1.2 definiuje pięć typów takich obiektów. Dopuszczalne jest w ramach konkretnych implementacji rozszerzanie funkcjonalności systemu o inne typy instrukcji.

Instrukcje zdefiniowane są poprzez klucz `Op` przyjmujący jako wartość nazwę instrukcji.

```
{ "Op": "OPERATION-NAME", ... }
```

Pozostałe parametry będą przyjmowały różne wartości, w szczególności w dalszej części wyróżnimy wartości typu EXPRESSION. Będziemy też definiować wartości odnoszące się do węzłów: NODE-REF oraz parametry związane z atrybutami: NODE-REF.ATTR-NAME oraz CONNECTION-REF.ATTR-NAME.

Typy instrukcji

Zbiór instrukcji możemy w obrębie specyfikacji dowolnie rozszerzać, opisując dodaną instrukcją nowy sposób modyfikacji dowolnego węzła lub krawędzi w GWSG. Można wszakże zauważyć, że jest kilka poziomów zależności węzłów modyfikowanych od węzłów dopasowanych do lewej strony produkcji.

Pierwszy poziom obejmuje modyfikacje ograniczone do węzłów dopasowanych do lewej strony. W takim wypadku produkcja grafowa daje się wyrazić w modelu lewa strona-prawa strona. W obecnej specyfikacji takimi operacjami są instrukcja `create` i wszystkie operacje na atrybutach oraz pozostałe operacje, jeśli parametr `Nodes` wyrażony jest multireferencją jednosegmentową.

Drugi poziom pozwala na modyfikacje węzłów z całego snopka węzła dopasowanego do lewej strony produkcji. Obejmuje operacje `move`, `copy` oraz `delete` jeśli parametr `Nodes` wymagany w tych operacjach wyraża się multireferencją wielosegmentową.

Trzeci poziom pozwalający na największą dowolność w kształtowaniu świata, ale wymagający wskazywania modyfikowanych węzłów bezpośrednio w instrukcji, zakłada odwołanie się do każdego węzła GWSG w sposób zdefiniowany bezpośrednio w implementacji. Operacje takie nie podlegają specyfikacji.

Poniżej opisanych jest osiem operacji z pierwszego i drugiego poziomu. Wystarczają one w zupełności do przedstawienia typowych sytuacji fabularnych, ale mają swoje ograniczenia, o których wspomniano na końcu rozdziału.

Żadna z opisanych operacji nie pozwala na modyfikowanie relacji między lokacjami (połączeń). Jest to bowiem niedozwolone w działaniach w trakcie rozgrywki. Inne relacje mogą być swobodnie zmieniane, ale ponieważ grafy mają ściśle hierarchiczną strukturę snopkową, to wszystkie modyfikacje struktury grafu to operacje na wierzchołkach. Modyfikacje stanu świata przez projektanta *in statu nascendi* nie jest objęte niniejszą specyfikacją i w szczególności wymaga tworzenia połączeń między lokacjami.

Przenoszenie słownika

Operacja `move` powoduje, że wskazany słownik, razem z zagnieżdżonymi w nim tablicami i słownikami, jest usuwany z tablicy, w której do tej pory się znajdował, i wstawiany do innej.

Operacja przenoszenia wymaga podania pary argumentów `Nodes` i `To`.

Jeśli multireferencja w parametrze `Nodes` będzie wielosegmentowa, pozwoli to przenieść wiele słowników jedną instrukcją.

Składnia:

```
{ "Op": "move", "Nodes": "NODE-MULTIREF", "To": "ARRAY-REF" }
```

Opcjonalny parametr `"Limit": NUMBER` pozwala ograniczyć przenoszenie do co najwyżej wskazanej liczby elementów. W pierwszej kolejności przenoszone będą elementy o najkrótszym rozwinięciu multireferencji w ścieżkę. Kolejności elementów o ścieżkach identycznej długości specyfikacja nie precyzuje.

Przykłady użycia:

```
{ "Op": "move", "Nodes": "Inn/Items/Coin", "To": "Hero/Items" }
```

```
{ "Op": "move", "Nodes": "Hero/**/Items/Carrot", "To": "Cooking_pot/Items",
    "Limit": 3 }
```

```
{ "Op": "move", "Nodes": "Inn/Items/Coin", "To": "Hero/Items"}
```

```
{ "Op": "move", "Nodes": "Hero/**/Items/Carrot", "To": "Cooking_pot/Items",
    "Limit": 3 }
```

Pierwsza instrukcja to podnoszenie przez bohatera z podłogi karczmy wszystkich leżących na niej monet, druga to wrzucanie posiadanych przez niego marchewek do garnka z zupą, ale dzięki użyciu parametru `Limit` nie więcej niż trzech. Ten opcjonalny parametr pozwala określić maksymalną liczbę słowników, którą instrukcji wolno przenieść. W trzecie instrukcji `"Hero/**/Items/Carrot"` może się rozwinąć i do `"Hero/Items/Carrot"` (marchewki trzymane bezpośrednio w ręku), i do `"Hero/Items/Bag/Items/Carrot"` (marchewki z worka), i do...

Przykład użycia kombinacji * i ** w multireferencji:

```
{ "Op": "move", "Nodes": "Hero/**/Items/*", "To": "Hero/Items"}
```

Jest to konsolidacja zasobów bohatera: przenosimy wszystkie przedmioty zagnieżdżone w snopku bohatera bezpośrednio do jego ekwipunku, czyli np. jeśli miał skrzynię, w której miał

plecak, w której miał małpkę, która trzyma mydło, to po wykonaniu produkcji powinien mieć skrzynię, plecak z małpką i mydło (małpka nie jest przedmiotem).

Tworzenie słownika

Operacja `create` wstawia do wskazanej tablicy słownik o podanej zawartości, reprezentującej wierzchołek grafu z ewentualnymi atrybutami i zagnieżdżonymi wierzchołkami potomnymi.

Składnia:

```
{ "Op": "create", "In": "ARRAY-REF", "Sheaf": CHAR-DICT }
{ "Op": "create", "In": "ARRAY-REF", "Sheaf": ITEM-DICT }
{ "Op": "create", "In": "ARRAY-REF", "Sheaf": NARR-DICT }
```

Opcjonalny parametr `"Count": NUMBER` pozwala stworzyć w tablicy docelowej zadaną liczbę identycznych elementów.

Przykład użycia:

```
{ "Op": "create", "In": "Hero/Items", "Sheaf": {
    "Name": "Bag", "Items": [ { "Name": "Floo_powder" } ] } }
```

Instrukcja umieści w torbie bohatera proszek Fiuu. Możemy domniemywać, że rzecz dzieje się w Hogwarcie ;--)

Po wykonaniu instrukcji `create` nie możemy odwoływać się do tego węzła (np., żeby coś do niego przenieść) w ramach tej samej produkcji.

Kopiowanie słownika

Operacja `copy` powoduje wstawienie do wskazanej tablicy kopii podanego węzła.

Składnia:

```
{ "Op": "copy", "Nodes": NODE-MULTIREF, "To": ARRAY-REF }
```

Opcjonalny parametr `"Count": NUMBER` pozwala stworzyć w tablicy docelowej zadaną liczbę kopii.

Przykład użycia:

```
{ "Op": "copy", "Nodes": Treasure_info, "To": Hero/Narration }
```

Bohater pozyskuje wiedzę o skarbie, ale nie znika ona z poprzedniej lokalizacji, czyli w szczególności nasz bohater nie zawłaszcza wiedzy, każdy inny bohater będzie mógł dowiedzieć się tego samego.

Usuwanie słownika

Operacja `delete` usuwa wskazany słownik.

Składnia:

```
{ "Op": "delete", "Nodes": "NODE-MULTIREF" }
```

Opcjonalny parametr `"Limit": NUMBER` pozwala ograniczyć usuwanie do co najwyżej wskazanej liczby elementów. W pierwszej kolejności usuwane będą elementy o najkrótszym rozwinięciu multireferencji w ścieżkę. Kolejności elementów o ścieżkach identycznej długości specyfikacja nie precyzuje.

W przypadku grafów hierarchicznych zawsze pojawia się pytanie, co zrobić z dziećmi usuwanego wierzchołka. Dwie typowe odpowiedzi to: rekursywnie usuwać dzieci razem z rodzicem; nie zezwalać na wykonywanie operacji usuwania w odniesieniu do wierzchołków mających dzieci.

W grafach snopkowych możliwy jest też trzeci sposób postępowania: przenoszenie osieroconych dzieci o jeden poziom hierarchii w górę. Po przetłumaczeniu na operacje na strukturze SHEAF oznacza to przeniesienie wszystkich słowników-dzieci z tablic `"Characters"`, `"Items"` i `"Narration"` do tak samo nazywających się tablic w słowniku-dziadku.

Opcjonalne parametry `ChildrenLimiter`, `CharactersLimiter`, `ItemsLimiter` oraz `NarrationLimiter` przyjmują jedną z trzech wartości: `delete` (zachowanie domyślne), `move` lub `prohibit`. Domyśla wartość oznacza, że wraz z węzłem usuwamy wszystkie jego węzły potomne, a pozostałe wartości pozwalają węzły dzieci przenieść lub zablokować usunięcie węzła niepustego. Dodajmy, że parametr `prohibit` powoduje nieuwzględnianie danego węzła w limicie określonym przez parametr `Limit`.

Parametry te pozwalają określić podejmowane czynności dla wszystkich typów dzieci na raz lub odrębnie dla każdej kategorii wierzchołków. Parametr `LocationsLimiter` nie jest potrzebny, bo lokacji nie można zagnieżdżać w innych wierzchołkach.

Nie jest dozwolone stosowanie jednocześnie parametru `ChildrenLimiter` i któregoś z pozostałych.

Przykład użycia:

```
{ "Op": "delete", "Nodes": "Squad", "CharactersLimiter": "prohibit", "Items
Limiter": "move" }
```

Powyższa instrukcja usunie słownik reprezentujący oddział wojska tylko jeśli w tym oddziale nie pozostali już żadni żołnierze (bo np. wszyscy po kolei zginęli w walce z bohaterem).

Przedmioty będące na wyposażeniu oddziału zostaną przeniesione o jeden poziom wyżej (czyli pojawią się na ziemi w lokacji, w której oddział do tej pory był), a węzły narracyjne (np. pochwały udzielone oddziałowi przez króla) zostaną usunięte.

Operacje na atrybutach

Są to operacje: `set`, `unset`, `mul` i `add`.

Atrybuty można tworzyć (nadając im przy tym jakąś wartość), modyfikować (czyli zmieniać wartość atrybutu) oraz usuwać. Tworzymy atrybut operacją `set`, `add` lub `mul`, modyfikujemy istniejący atrybut operacją `set` poprzez nadanie nowej wartości lub (dla atrybutów liczbowych) operacją `add`, dodając do aktualnej wartości podany składnik albo `mul`, mnożąc aktualną wartość przez podany czynnik. Usuwamy operacją `unset`. Wartości wstawiane do atrybutów mogą być stałymi podawanymi wprost w instrukcji albo mogą być obliczane na podstawie wcześniejszej wartości danego atrybutu i wartości innych atrybutów.

Często używanymi operacjami są dodawanie stałej do aktualnej wartości atrybutu lub mnożenie jej przez stałą. Dla nich przewidziano skrócony zapis, który ma tę dodatkową zaletę, że jest niezależny od używanego języka programowania i od dostępności `eval()`.

Składnia:

```
{ "Op": "set", "Attribute": "NODE-REF.ATTR-NAME", "Value": STRING }
{ "Op": "set", "Attribute": "NODE-REF.ATTR-NAME", "Value": NUMBER }
{ "Op": "set", "Attribute": "NODE-REF.ATTR-NAME", "Value": BOOLEAN }
{ "Op": "set", "Attribute": "NODE-REF.ATTR-NAME", "Expr": EXPRESSION }

{ "Op": "add", "Attribute": "NODE-REF.ATTR-NAME", "Value": NUMBER }
{ "Op": "add", "Attribute": "NODE-REF.ATTR-NAME", "Expr": EXPRESSION }

{ "Op": "mul", "Attribute": "NODE-REF.ATTR-NAME", "Value": NUMBER }
{ "Op": "mul", "Attribute": "NODE-REF.ATTR-NAME", "Expr": EXPRESSION }

{ "Op": "unset", "Attribute": "NODE-REF.ATTR-NAME" }
```

Operacja `set` jest używana i do tworzenia, i do modyfikowania atrybutów.

Jeśli atrybut nie istniał lub jego wartość nie była liczbą, to `add` i `mul` przyjmują odpowiednio 0 i 1 jako poprzednią wartość atrybutu.

Przykłady użycia:

```
{ "Op": "set", "Attribute": "Dragon.IsDead", "Value": true }
{ "Op": "add", "Attribute": "Bear.HP", "Value": -8 }
{ "Op": "mul", "Attribute": "Hero.Money", "Value": 0.9 }   // 10% city tax
```

Wymiana liczby punktów życia pomiędzy czarodziejem a niedźwiedziem:

```
{ "Op": "set", "Attribute": "Wizard.HP", "Expr": "Wizard.HP + Bear.HP" }
{ "Op": "set", "Attribute": "Bear.HP",   "Expr": "Wizard.HP - Bear.HP" }
{ "Op": "set", "Attribute": "Wizard.HP", "Expr": "Wizard.HP - Bear.HP" }
```

Operacje na krawędziach

Operacji pozwalających tworzyć i usuwać połączenia pomiędzy lokacjami na razie nie ma potrzeby definiować, bo w najbliższej przyszłości nie przewidujemy projektowania gier z dynamicznie zmieniającą się mapą świata.

Bardziej prawdopodobne jest, że pojawi się potrzeba korzystania z atrybutów połączeń, aby przy ich pomocy wyrażać czas podróży, cenę przejazdu dyliżansem itp. W takim przypadku wprowadzona zostanie możliwość dodawania do słowników reprezentujących połączenia kluczy `"Id"` oraz `"Attributes"`, a operacje na atrybutach zostaną przedefiniowane tak, aby oprócz `"NODE-REF.ATTR-NAME"` akceptowały jako argument również `"CONNECTION-REF.ATTR-NAME"`.

6. Przykładowe produkcje

Poniżej znajduje się przykłady produkcji i sposobów zapisu w produkcji pożądanych zależności fabularnych. Pierwszy przykład to bardzo ogólna produkcja pozwalająca dowolnym dwóm postaciom w dowolnej lokalizacji dokonać transakcji kupna-sprzedaży dowolnego przedmiotu, pod warunkiem, że kupujący ma wystarczająco dużo pieniędzy (co najmniej tyle, ile wynosi wartość przedmiotu).

Te przykłady to przykłady dydaktyczne mające pokazać, jak się definiuje produkcje. Nie układają się w żadną spójną fabułę. Każdy z nich należy rozważać odrębnie, tak jakby był jedyną produkcją w całej gramatyce.

Transakcja kupna-sprzedaży

Przyjmujemy, że finanse bohatera są reprezentowane pojedynczym liczbowym atrybutem „Money” odzwierciedlającym bieżący stan gotówki. Wszystkie przedmioty sprzedawane przez handlarza mają liczbowy atrybut reprezentujący ich cenę. Handlarz może posiadać również inne przedmioty, np. sztylet do obrony przed złodziejami, ale jeśli one nie mają atrybutu „Value”, to tym samym nie są na sprzedaż.

Kod 6. Transakcja kupna-sprzedaży.

```
{
    "Title": "Making a deal / Transakcja kupna-sprzedaży",
    "TitleGeneric": "Item acquisition from another character / Przejęcie przedmiotu od
     innej postaci",
    "Description": "«Seller» sprzedaje kupującemu («Buyer») przedmiot («Something»).",
    "Override": 0,
    "LSide": {
      "Locations": [
        {
          "Id": "Somewhere",
          "Attributes": {},
          "Characters": [
            {
              "Id": "Buyer",
              "IsObject": true,
              "Attributes": {
                "Money": null
              }
            },
            {
              "Id": "Seller",
              "IsObject": true,
              "Attributes": {
                "Money": null
              },
```

```
            "Items": [
              {
                "Id": "Something",
                "Attributes": {
                  "Value": null
                }
              }
            ]
          }
        ]
      }
    ]
  },
  "Preconditions": [
    {
      "Cond": "Buyer.Money >= Something.Value"
    }
  ],
  "Instructions": [
    {
      "Op": "move",
      "Nodes": "Something",
      "To": "Buyer/Items"
    },
    {
      "Op": "set",
      "Attribute": "Buyer.Money",
      "Expr": "Buyer.Money - Something.Value"
    },
    {
      "Op": "set",
      "Attribute": "Seller.Money",
      "Expr": "Seller.Money + Something.Value"
    }
  ]
}
```

Zabicie smoka, wersja pierwsza

Jeśli w smoczej jaskini jest uzbrojony w miecz bohater i smok, to bohater może smoka mieczem dźgnąć i go zabić (czyli ustawić mu atrybut "IsDead"). Przykładem takiej produkcji zaprezentowany jest poniżej. Wadą tej produkcji jest to, że można ją dopasować do zabitego już wcześniej smoka. Można jednak tę wadę zignorować, bo dźganie truchła niczego w grafie stanu świata nie zmienia (ponowne ustawianie atrybutu na tę samą wartość jest operacją pustą).

Kod 7. Zabicie smoka, wersja pierwsza

```
[
    {
        "Title": "Killing dragon using sword / Zabicie smoka z użyciem miecza",
        "TitleGeneric": "",
        "Description": "Bohater w smoczej jaskini zabija smoka używając miecza.",
        "Override": 0,
        "LSide": {
            "Locations": [ {
                "Name": "Dragons_lair",
                "Characters": [ {
                    "Name": "Hero",
                    "IsObject": true,
                    "Items": [ {
                        "Name": "Sword"
                    } ]
                }, {
                    "Name": "Dragon"
                } ]
            } ]
        },
        "Instructions": [
            { "Op": "set", "Attribute": "Dragon.IsDead", "Value": true }
        ]
    }
]
```

Zabicie smoka, wersja druga

Modyfikacja poprzedniego przykładu poprzez dodanie warunku stosowalności, który sprawdza, czy smok jest żywy. Aby móc ten warunek zapisać przyjęto założenie, że smok zawsze posiada atrybut o nazwie „IsDead”, tyle tylko, że za życia ma on wartość *false*, a po śmierci *true*.

Kod 8. Zabicie smoka, wersja druga

```
[
    {
        "Title": "Killing dragon using sword 2 / Zabicie smoka z użyciem miecza 2",
        "TitleGeneric": "",
        "Description": "Bohater w smoczej jaskini zabija smoka używając miecza.",
        "Override": 0,
        "LSide": {
        "Locations": [ {
            "Name": "Dragons_lair",
            "Characters": [ {
                "Name": "Hero",
                "Items": [ {
                    "Name": "Sword"
                } ]
            }, {
                "Name": "Dragon"
            } ]
        } ]
    },
    "Preconditions": [
        { "Cond": "Dragon.IsDead == false" }
    ],
    "Instructions": [
        { "Op": "set", "Attribute": "Dragon.IsDead", "Value": true }
    ]
    }
]
```

Wadą jest tu oczywiście konieczność przyjęcia dodatkowego założenia i, co za tym idzie, pamiętania, aby podczas dodawania do świata nowych smoków ustawiać im atrybut „IsDead” z wartością *false*. Bardzo mało intuicyjne, wysokie ryzyko, że ktoś o tym zapomni.

Zabicie smoka, wersja trzecia

Bohater zabija smoka mieczem, po czym smok-postać jest usuwany z grafu stanu świata i zamiast niego pojawia się martwe cielsko-przedmiot.

Problem związany z tym podejściem staje się widoczny, jeśli przyjąć, że smok mógł posiadać jakieś przedmioty i postacie (może były one przez niego połknięte?). Te potomne wierzchołki należałoby przepiąć do nowego rodzica, czyli do świeżo stworzonego wierzchołka „Dragon_body”. Niestety, nie da się tego zrobić używając operacji `move`, bo ona wymaga wskazania jednego z wierzchołków lewej strony jako nowego rodzica.

Kod 9. Zabicie smoka, wersja trzecia

```
    {
        "Title": "Killing dragon using sword 3 / Zabicie smoka z użyciem miecza 3",
        "TitleGeneric": "",
        "Description": "Bohater w smoczej jaskini zabija smoka używając miecza.",
        "Override": 0,
        "LSide": {
            "Locations": [ {
                "Name": "Dragons_lair",
                "Characters": [ {
                    "Name": "Hero",
                    "Items": [ {
                        "Name": "Sword"
                    } ]
                },
                {
                    "Name": "Dragon"
                } ]
            } ]
        },
        "Instructions": [
            { "Op": "create", "In": "Dragons_lair/Items", "Sheaf": { "Name": "Dragon_body"
} },
            { "Op": "delete", "Nodes": "Dragon" }
        ]
    }
```

Przechodzenie pomiędzy połączonymi lokacjami

Produkcja w 100% generyczna, pozwalająca (dowolnym) postaciom przemieszczać się wzdłuż krawędzi łączących (dowolne) lokacje.

Kod 10. Przechodzenie pomiędzy połączonymi lokacjami

```
{
    "Title": "Location change WP / Zmiana lokacji WP",
    "TitleGeneric": "",
    "Description": "Bohater przechodzi z lokacji do lokacji.",
    "Override": 0,
    "LSide": {
        "Locations": [ {
            "Id": "Loc",
            "Characters": [ {
                "Id": "ch"
            } ],
            "Connections": [
                { "Destination": "dest" }
            ]
        }, {
            "Id": "dest"
        } ]
    },
    "Instructions": [
        { "Op": "move", "Nodes": "ch", "To": "dest/Characters" }
    ]
}
```

Zabijanie komarów

Bohater może rozgnieść komara tak, że śladu po nim nie zostanie bez względu na to, czy są w karczmie czy w lesie (ale muszą być obaj w tej samej lokacji, bo zdalne zabijanie komarów nie jest możliwe). Jeśli w lokacji jest kilka komarów, to zabity zostanie jeden przypadkowo wybrany.

Kod 11. Zabijanie komarów

```
{
    "Title": "Killing mosquitos / Zabijanie komarów",
    "TitleGeneric": "",
    "Description": "Bohater przechodzi z lokacji do lokacji.",
    "Override": 0,
    "LSide": {
        "Locations": [ {
            "Id": "Loc",
            "Characters": [ {
                "Name": "Hero"
              }, {
                "Name": "Mosquito"
              } ]
        } ]
    },
    "Instructions": [
        { "Op": "delete", "Nodes": "Mosquito" }
    ]
}
```

Zamach skrytobójczy

Rozważany już wcześniej przykład skrytobójcy potajemnie zabijającego swoją ofiarę, rozpisany tutaj w formie pełnej produkcji. Oprócz warunku sprawdzającego, czy w pomieszczeniu nie ma osób postronnych, w produkcji jest dodatkowy warunek, sprawdzający czy potencjalna ofiara przypadkiem nie niesie innej postaci (z fabularnego punktu widzenia ta postać byłaby świadkiem zamachu, więc skrytobójca nie może w takiej sytuacji zaatakować).

Kod 12. Zamach skrytobójczy

```
{
    "Title": "Assassin attack / Zamach skrytobójczy",
    "TitleGeneric": "",
    "Description": "Bohater przechodzi z lokacji do lokacji.",
    "Override": 0,
    "LSide": {
        "Locations": [ {
            "Id": "loc",
            "Characters": [ {
              "Name": "Assassin"
            }, {
              "Id": "ch02"
            } ]
        } ]
    },
    "Preconditions": [
        { "Count": "loc/Characters/*", "Max": 2 },
        { "Count": "ch02/Characters/*", "Max": 0 }
    ],
    "Instructions": [
        { "Op": "delete", "Nodes": "ch02", "ItemsLimiter": "move" }
    ]
}
```

Walka z opcjonalnymi konsekwencjami

Multireferencje pozwalają przenosić i usuwać przedmioty opcjonalne. Tych przedmiotów nie można podać po lewej stronie produkcji, bo wszystko, co po lewej, musi być w grafie stanu gry. Przykład: bohater może, ale nie musi, mieć na palcu rodowy pierścień odziedziczony po ojcu. Jeśli go ma, to podczas walki ze śluzowatym potworem ten pierścień powinien ześlizgnąć się z palca na ziemię (a jeśli go nie miał, to nic się nie dzieje).

Kod 13. Walka z utratą pierścienia

```
{
    "Title": "Fight with the ring loss / Walka z utratą pierścienia",
    "TitleGeneric": "",
    "Description": "Bohater walczy i może zgubić pierścień, jeśli posiada.",
    "Override": 0,
    "LSide": {
        "Locations": [ {
            "Id": "loc",
            "Characters": [ {
                "Name": "Hero",
                "Items": [ {
                    "Name": "Mace"
                } ]
            }, {
              "Name": "Ooze_monster"
            } ]
        } ]
    },
    "Instructions": [
        { "Op": "delete", "Nodes": "Ooze_monster" },
        { "Op": "set", "Attribute": "Hero.IsSlimed", "Value": true },
        { "Op": "move", "Nodes": "Hero/Items/Heirloom_ring", "To": "loc/Items" }
    ]
}
```

W tym przykładzie przyjęto założenie, że pierścień zawsze jest noszony na palcu. Jeśli fabuła gry dopuszczałaby również noszenie pierścienia w kieszeni, to oczywiście trzeba by produkcję odpowiednio zmodyfikować.

7. Wczytywanie i zapisywanie świata z i do pliku JSON

Poprawność zapisu JSON

Aby świat lub produkcja zostały uznane za poprawne, muszą spełniać trzy rodzaje warunków:

- Po pierwsze muszą być zgodne ze specyfikacją formatu JSON.
- Po drugie muszą spełniać warunki zawarte w schemacie JSON (ang. *JSON schema*), dołączonej do tej specyfikacji jako załącznik xx a omówionej w rozdziale yy. Schemat

sprawdza formalną poprawność struktur SHEAF, LOC-DICT, CHAR-DICT, ITEM-DICT, NARR-DICT, CONN-DICT oraz ATTR-DICT.

- Po trzecie muszą spełniać dodatkowe warunki, które wynikają ze specyfikacji a nie da się ich wyrazić poprzez schemat z powodu jego ograniczeń formalnych, w szczególności ograniczeń narzuconych na strukturę SHEAF reprezentującą świat oraz poprawność łańcuchów NODE-REF, ARRAY-REF oraz CONNECTION-REF (omówione w rozdziale 5, nie dotyczy zapisu statycznego świata).
  Te warunki muszą zostać sprawdzone w implementacji. W implementacji referencyjnej służy do tego funkcja get_json_validated().
- Możliwe jest dodanie na tym ostatnim poziomie kolejnych obostrzeń dotyczących danych, ale nie jest dozwolone luzowanie obostrzeń wynikających z definicji formatu json, schematu i specyfikacji.

Ponieważ lewa strona produkcji jest strukturą SHEAF i świat gry jest strukturą SHEAF a na poziomie schematu te struktury się nie różną, to w implementacji referencyjnej narzucono opakowanie świata w dodatkowy słownik, tak aby format danych json dla produkcji i świata był identyczny. Umożliwiło to zastosowanie jednego schematu dla obu struktur danych.

Świat w zapisie tekstowym będzie więc wyglądał tak:

Kod 14. Zapis JSON świata wymagany do zgodności ze schematem w implementacji referencyjnej.

```
{
   "Title": "Dowolna nazwa, np. Świat DragonStory / World DragonStory",
   "TitleGeneric": "",
   "Description": " Dowolny opis, np. świat, w którym może wydarzyć się misja
    DragonStory.",
   "Override": 0,
   "LSide": { obiekt typu SHEAF ograniczony zgodnie ze specyfikacją świata },
   "Preconditions": [],
   "Instructions": []
 }
```

Cechą pozwalającą rozróżnić świat i produkcję jest pusta lista instrukcji świata.

Serializacja i deserializacja danych

Jeśli w grafie jest kilka lokacji o tej samej nazwie, to przy zapisywaniu grafu do JSON-owego pliku trzeba użyć kluczy `Id`, aby móc jednoznacznie określić, co z czym jest połączone. W przypadku, gdy klucz `Name` jest unikatowy, można wykorzystać go zamiast tworzyć kluczy `Id`, ale w praktyce bezpieczniej jest wygenerować unikatowe kluczy `Id` niż sprawdzać, czy dana nazwa się nie powtarza. Np. wioska, zamek i długa droga pomiędzy nimi:

```
{
    "Locations": [ {
        "Name": "Village",
        "Connections": [
            { "Destination": "rd01" }
        ]
    }, {
        "Name": "Road",
        "Id": "rd01",
        "Connections": [
            { "Destination": "Village" },
            { "Destination": "rd02" }
        ]
    }, {
        "Name": "Road",
        "Id": "rd02",
        "Connections": [
            { "Destination": "rd01" },
            { "Destination": "Castle" }
        ]
    }, {
        "Name": "Castle",
        "Connections": [
            { "Destination": "rd02" }
        ]
    } ]
}
```

Jeśli użyty język programowania traktuje słowniki jako pełnoprawne obiekty, to przy ładowaniu grafu do pamięci można zamienić tekstowe referencje na referencje natywne. Innymi

słowy, natychmiast po wczytaniu z pliku powyższej JSON-owej struktury danych modyfikuje się ją w następujący sposób:

```
world.Locations[0].Connections[0].Destination = world.Locations[1];
world.Locations[1].Connections[0].Destination = world.Locations[0];
world.Locations[1].Connections[1].Destination = world.Locations[2];
world.Locations[2].Connections[0].Destination = world.Locations[1];
world.Locations[2].Connections[1].Destination = world.Locations[3];
world.Locations[3].Connections[0].Destination = world.Locations[2];
```

Potem można usunąć ze słowników wszystkie klucze `Id`. W grafie stanu świata nie są one potrzebne i mogłyby tylko zawadzać przy dalszych operacjach wykonywanych na tej strukturze danych.

```
delete world.Locations[1].Id;
delete world.Locations[2].Id;
```

8. Stosowanie produkcji

Aby zastosować produkcję, musimy znaleźć dopasowanie generyczne między lewą stroną produkcji a grafem stanu świata, sprawdzić, czy spełnione są predykaty stosowalności (niespełnienie ich pozwala odrzucić produkcję mimo dopasowania lewej strony produkcji do grafu) i wykonać ciąg instrukcji operujących na wierzchołkach wybranego podgrafu i strukturach snopkowych zbudowanych w oparciu o te wierzchołki.

Czytelnicy planujący korzystanie z implementacji referencyjnej jako silnika fabularnego poprzez API, proszeni są o przejście do rozdziału xx. Czytelnicy pragnący zaimplementować algorytmy pozwalające na aplikację produkcji nie mogą ominąć rozdziału yy, zz.

Wykorzystanie implementacji referencyjnej jako silnika fabularnego gry

Poprzez API można korzystać z implementacji referencyjnej jako zewnętrznego silnika narracyjnego.

Implementacja algorytmu dopasowania lewej strony produkcji do świata gry

Pierwszym krokiem jest znalezienie dopasowania lewej strony produkcji do wybranego fragmentu grafu świata. Dopasowanie musi zachowywać etykiety wierzchołków, relacje między nimi, oraz wartości atrybutów (chyba że po lewej stronie jako wartość jest podane `null`, wtedy wystarczy samo istnienie tak się nazywającego atrybutu w dopasowanym w grafie świata wierzchołku).

Dopasowanie musi być iniektywne. Jeśli po lewej stronie produkcji generycznej jest karczma, a w niej dwie nieokreślone postacie, to nie można obu dopasować do karczmarza. Innymi słowy, jeśli dopasowanie jest pamiętane w mapie `match`, to wartości tej mapy muszą być różne od siebie.

Drugi krok to ewaluacja warunków stosowalności. Podczas ewaluacji tych, które odwołują się do wartości atrybutów, mogą wystąpić sytuacje wyjątkowe.

W przykładach używamy odwołań postaci `"N.A"`, gdzie „N” jest identyfikatorem lub unikalną nazwą słownika z lewej strony produkcji. Można więc odszukać w strukturze `lhs` odpowiadający mu słownik i wynik tego wyszukiwania zapamiętać jako natywną referencję `n`. Ewaluacja predykatów jest robiona w kontekście jakiegoś ustalonego dopasowania lewej strony. Co za tym idzie, `match[n]` będzie tym słownikiem ze struktury `world`, do którego dopasował się słownik „N”. W jego podsłowniku `Attributes` szukamy klucza „A”. Wartość związana z tym kluczem jest wstawiana do wyrażenia w miejsce referencji `"N.A"`.

Problemy pojawiają się wtedy, gdy w słowniku atrybutów brak klucza `"A"`, albo gdy wartość z nim związana jest nieodpowiedniego typu (powiedzmy, że mamy predykat `"Monster.HP > 10"`, a wartością tego atrybutu okazała się być nie liczba, lecz łańcuch `"lots"`).

O ile przed pierwszą sytuacją można się próbować zabezpieczyć specyfikując po lewej stronie wykorzystywane w warunkach stosowalności atrybuty (z wartościami `null`, co zagwarantuje ich obecność i nic więcej), to przed tą drugą zabezpieczyć się nie ma jak.

Aby nie utrudniać życia osobom implementującym niniejszą specyfikację jako rozwiązanie generalne przyjmujemy, że silnik gry w razie wystąpienia sytuacji wyjątkowej może zachować się w dowolny sposób („undefined behavior” znany z języka C). Obejmuje to nie tylko dwie powyższe, lecz wszelkie wyjątkowe sytuacje (np. wyrażenie nie będące poprawnym wyrażeniem logicznym języka GDScript, pominięcie obu ograniczeń w warunku typu `"Count"`, itd.).

Trzeci krok, wykonywany jeśli wszystkie warunki stosowalności są spełnione, to wykonanie instrukcji w takiej kolejności, w jakiej są one stablicowane. Wymaga to zamiany podanych jako

ich argumenty tekstowych referencji i multireferencji na referencje natywne, wskazujące obiekty w strukturze `world`.

Pierwszym segmentem i w referencjach, i w multireferencjach jest identyfikator lub unikalna nazwa słownika ze struktury `lhs`. Zamieniamy go na referencję natywną, z `match` bierzemy odpowiadającą mu referencję natywną do słownika w strukturze `world`. Jeśli po pierwszym segmencie były jakieś następne, to próbujemy je rozwinąć w ramach struktury `world`.

Ze względów praktycznych warto przyjąć zasadę, że każdy słownik odpowiadający wierzchołkowi grafu zawsze ma podtablice `"Characters"`, `"Items"` i `"Narration"`, nawet jeśli miałyby być one puste. W pliku JSON może ich nie być (i nawet nie powinno, bo zmniejszają czytelność), ale przy wczytywaniu grafów do pamięci są one automatycznie dodawane.

Ta zasada gwarantuje nam, że referencje typu `"Inn/Items"`, `"Hero/Characters"` czy `"Forest/Narration"` zawsze będą poprawnie wskazywać na istniejące w strukturze `world` tablice.

9. Dobre praktyki nazewnicze

Nazwy kluczy piszemy w PascalCase literami z zakresu ASCII
Wartości zasadniczo piszemy jak chcemy, ale:

- Wartości atrybutów `"Name"` i `"Id"` zaczynamy wielkimi literami, zamiast spacji dajemy podkreślenia i pozbawiamy znaków innych niż litery, podkreślenia i cyfry.
- W konkretnym świecie gry dodatkowo nazwy ograniczamy do czterech określonych rozłącznych zbiorów.
- Zbiór wartości atrybutów `"Id"` jest rozłączny ze zbiorem nazw.
- Nazwy kluczy atrybutów piszemy w PascalCase (jak każdy klucz) dodatkowo te klucze, które przyjmują wartości true/false zaczynać od prefiksu „Is”, np.: `"IsTroublemaker"`, `"IsDead"` itp.

Tworzenie opisu produkcji

Opis tworzymy tak jak dla gry tekstowej, tzn. wyjaśniający co się dzieje w produkcji. Id piszemy w nawiasach francuskich, w potencjalnej grze zostaną wymienione na nazwy dopasowanych węzłów.

Wytyczne dotyczące cudzysłowów francuskich:

- Jeśli nie ma żadnego Id, nie używamy ich. Ale, uwaga, nazwa bohatera powinna się wtedy znaleźć w oryginalnym brzmieniu w opisie, np.: „Main_hero przechodzi z lokacji Road do lokacji Inn.”
- Jeżeli jest Id tylko w lokacji, lokacja jest jedna i nie ma wpływu na fabułę nie musimy umieszczać jej w opisie, np.: „Main_hero upuścił «Something».” a nie „Main_hero upuścił «Something» w lokacji «Anywhere».”
- jeżeli Id dotyczy bohaterów lub przedmiotów lub lokacji jest więcej niż jedna i są istotne dla fabuły, to umieszczamy je w opisie, np.: „«BohaterA» pokonuje inną postać («BohaterB»), która ucieka do sąsiedniej lokacji («LokacjaB»).” Id w mianowniku umieszczamy bezpośrednio, w przypadkach zależnych po określeniu ogólnym, najczęściej w nawiasach.