Programujemy gry przygodowe

w środowisku DREAM

WGan

Spis treści

[Od autora 2](#_Toc42789724)

[Lekcja pierwsza – mapa 2](#_Toc42789725)

[Potrzebujemy planu 2](#_Toc42789726)

[Pierwsza lokacja 3](#_Toc42789727)

[THEN 4](#_Toc42789728)

[LOOK 4](#_Toc42789729)

[EXIT 5](#_Toc42789730)

[PRINT 5](#_Toc42789731)

[Druga lokacja i pierwszy refactoring 5](#_Toc42789732)

[GOTO 6](#_Toc42789733)

[Trening czyni mistrza 7](#_Toc42789734)

[Lekcja druga – artefakty 7](#_Toc42789735)

[Definiujemy artefakt 7](#_Toc42789736)

[Obiekt można zabrać ze sobą 8](#_Toc42789737)

[TAKE 8](#_Toc42789738)

[OBJECT 8](#_Toc42789739)

[POCKET 9](#_Toc42789740)

[EXAMINE 9](#_Toc42789741)

[DROP 10](#_Toc42789742)

[Użycie artefaktu 10](#_Toc42789743)

[EXEC 10](#_Toc42789744)

[NEW 10](#_Toc42789745)

[Kolejne komplikacje 11](#_Toc42789746)

[DEL 11](#_Toc42789747)

[Lekcja trzecia – właściwości 12](#_Toc42789748)

[Kopalnia złota 12](#_Toc42789749)

[MODIFY … BY … 14](#_Toc42789750)

[MODIFY … TO … 14](#_Toc42789751)

[WHEN 14](#_Toc42789752)

[DICE 14](#_Toc42789753)

[ITEM 16](#_Toc42789754)

[Lekcja czwarta – inne zastosowanie właściwości 17](#_Toc42789755)

[PLAYER 19](#_Toc42789756)

[LOCATION 19](#_Toc42789757)

[BECOME 19](#_Toc42789758)

[TRACE 19](#_Toc42789759)

[DUMP 20](#_Toc42789760)

[Uwagi na koniec 21](#_Toc42789761)

# Od autora

Niniejszy dokument powstał jako podręcznik programowania dla osób które chciałyby rozpocząć swoją przygodę z komputerami także „od kuchni”. Nie jest to jednak podręcznik związany z jakimś obecnym na rynku językiem, lecz jest to rodzaj eksperymentu w którym mam nadzieję nauczyć pewnych nawyków myślenia które przydadzą się bez względu na to w jakim języku czy środowisku czytelnik będzie się dalej kształcił. Eksperymentu w którym uczeń, zamiast poznawać sposoby implementacji coraz to innych książkowych algorytmów, stworzy grę przygodową ucząc się przy okazji planowania, abstrahowania, czy delegowania oraz pracy na wielu poziomach abstrakcji.

# Lekcja pierwsza – mapa

Pisząc grę powinniśmy mieć wyobrażenie jak mniej więcej będzie wyglądał świat naszej gry. I nie chodzi to o grafikę, ale o to jaki będzie świat do którego trafi gracz.

## Potrzebujemy planu

Jeśli chcemy zacząć programować – musimy rozpocząć od określenia planu: co chcemy zrobić. Plan powinien być:

* Realistyczny
* Zrozumiały

Nie jest na przykład dobrym planem „chciałbym napisać taki program, żebym zarobił na nim dużo kasy”, choć może być to naszym celem. Nie mając doświadczenia w planowaniu realizacji projektów inżynierskich, zamiast tego, spojrzymy na tworzoną przez nas grę na najbardziej ogólnym poziomie – czyli na określeniu świata w którym znajdzie się gracz.

Gra musi się przecież gdzieś rozgrywać. Gracz powinien mieć możliwość poruszania się, dlatego pierwszym krokiem będzie przygotowanie mapy składającej się z 5-10 lokacji. Nawet jeśli przewidujemy stworzenie bardzo złożonej gry, w wieloma niezależnymi przygodami i dużą liczbą artefaktów mających wpływ na przebieg gry, prace powinniśmy rozpocząć od kilku lokacji pomiędzy którym gracz będzie się mógł poruszać.

Mapą może być mapa krainy w której gracz będzie przeżywał przygody, zawierającej lokacje ogólne które później będą obudowywane dodatkowymi lokacjami. Może to być konkretny obszar podzielony na konkretne lokacje – na przykład dom podzielony na pokoje albo jaskinia w wieloma salami i korytarzami.

Taką mapę dobrze sobie przygotować na kartce. To znacznie wygodniejsze niż używanie programów do tworzenia map. Kartka i ołówek niosą znacznie mniej ograniczeń, i pozwolą na dodawanie notatek i pomysłów, a regularnie robione fotografie albo skany pozwolą nie tylko na prowadzenie dokumentacji, ale także do wrócenia do poprzedniej wersji rysunku – w takim wypadku trzeba go tylko wydrukować. Ważne jest także to, że kartka i ołówek nie dają ograniczeń. A jeśli na jakieś się trafi – to zazwyczaj pomaga zakreślacz, doklejony arkusz lub kredki.

Mapa powinna być w miarę prosta ale nietrywialna. Ważne, by nie łączyć każdej lokacji z każdą inną, bo taka mapa jest nudna i nadaje się tylko na najbardziej ogólnym poziomie gry. Na przykład – w grze osadzonej w świecie szpiegowskich intryg – połączenia lotnicze między poszczególnymi państwami można by tak zrealizować. Jednak i tu lepiej gdy w niektóre miejsca trzeba lecieć z przesiadkami, czasami zmuszając gracza do wykonania kilku ruchów w mieście tranzytowym. Mając mapę, wybieramy punkt startowy i rozpoczynamy programowanie gry.

No może nie od razu programowanie. Na początek musimy ogarnąć środowisko i oswoić się ze sposobem zapisu gry.

## Pierwsza lokacja

Każdy element naszej gry – jest zapisany w oddzielnym pliku. Czy to będzie lokacja[[1]](#footnote-1), przedmiot[[2]](#footnote-2) inny element gry, zawsze będzie to plik tekstowy z rozszerzeniem .def (ang. *definition* – definicja). W takim pliku informacje podzielone są na sekcje – każda sekcja – to linia z nazwą sekcji umieszczonej w nawiasach kwadratowych. Wewnątrz każdej sekcji występuje kilka linii tekstu zawierających informacje. Popatrzmy na zawartość przykładowego pliku:

[type]

location

[name]

entrance

[description]

You see dark cave entrance on the east

You are in small valley covered with bushes. In the bushes on the east, you see a hole looking like entrance to the cave which looks pretty interesting.

W tym pliku mamy trzy sekcje:

* type – typ obiektu którego opis znajduje się w pliku. W następnej linii powinna znajdować się nazwa typu i powinien to być typ znany środowisku. Na początek wystarczy nam, że dobrym typem jest location.
* name – nazwa obiektu. W następnej linii powinna znajdować się podstawowa nazwa obiektu i powinna być to taka sama nazwa jak nazwa pliku – tyle że bez rozszerzenia .def. Jedyną różnicą jest to, że w nazwie pliku – spacje powinny być zastąpione znakami podkreślenia. Takie dziwactwo, ale warto tego przestrzegać. W kolejnych liniach można podać alternatywne nazwy, ale na razie nie będziemy na to zwracać uwagi.
* description – opis obiektu. W tej sekcji powinny pojawić się dwie linie. Pierwsza – to krótki opis, który pojawiać się będzie gdy gra tylko wspomina o obiekcie – w przypadku lokacji – gdy do niej wchodzimy. Druga linia zawiera dokładny opis który pojawia się gdy obiektowi się przyglądamy dokładniej.

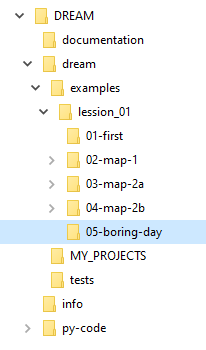
Jeśli zapiszemy nasz plik, możemy go uruchomić jako naszą pierwszą grę. Uruchamianie gry jest nieco skomplikowane: musimy wywołać program DREAM.py umieszczony w katalogu py-code, przekazując mu dwa parametry:

* Nazwę katalogu w którym są umieszczone pliki gry – w tym wypadku stworzony przez nasz plik z opisem wejścia do jaskini.
* Nazwę lokacji w której rozpoczyna się gra. Nazwa lokacji nie powinna zawierać rozszerzenia .def.

Ponieważ może to być pewnym problemem, w przykładach znajduje się plik RUN.cmd, który uruchamiamy już bez żadnych parametrów. Plik ten powinien znajdować się w katalogu w którym budujemy grę, a katalog py-code, powinien znajdować się w którymś z katalogów nadrzędnych. Przykładem organizacji może być struktura katalogów zawierających przykłady używane w opisie lekcji.

Po uruchomieniu gry, powinniśmy zobaczyć krótki opis naszej lokacji:

You are in entrance

You see dark cave entrance on the east

??? >

Teraz możemy napisać co chcemy zrobić. Niestety wszelkie próby wykonania jakiegokolwiek polecenia – kończy się informacją o błędzie:

??? > hello

I do not know how to hello

??? >

### THEN

Aby nauczyć nasz grę na reagowanie na polecenia użytkownika, musimy te polecenia zdefiniować. Możemy to zrobić dodając do definicji naszej lokacji sekcję actions, w której każda linia określa jedną akcję:

[actions]

look THEN LOOK

wake up; exit THEN PRINT You woke up from the dream.; EXIT

W tym przykładzie dodajemy dwie akcje. Każda akcja zawiera nazwy, czyli teksy z którymi porównujemy to co wpisać grający jako polecenie. Poszczególne nazwy oddzielone są średnikiem. Po nazwach występuje słowo kluczowe THEN (ang. *then* -> wtedy), po którym następuje, także lista, poleceń języka *dL* pozwalających na manipulowanie danymi gry. Każde polecenie zawiera jedno słowo kluczowe i może zawierać dodatkowe elementy.

### LOOK

Spójrzmy na pierwszą akcję. Jeśli użytkownik wpisze słowo *look*, program wywoła wbudowaną instrukcję LOOK, która wyświetla rozszerzony opis lokacji w której w danej chwili znajduje się gracz. Opis ten będzie uzupełniony także o inne elementy związane lokacja – na przykład listę elementów które się w niej zładują.

Instrukcja LOOK nie wymaga podawania żadnych parametrów.

Jeśli uruchomimy naszą grę, i spróbujemy teraz wydać polecenie, zobaczymy:

You are in entrance

You see dark cave entrance on the east

??? > look

You are in small valley covered with bushes. In the bushes on the east, you see a hole looking like entrance to the cave which looks pretty interesting.

??? >

### EXIT

W drugiej instrukcji, która ma dwie alternatywne nazwy, wywołujemy dwie instrukcje. Ostatnią instrukcją jest EXIT powodująca natychmiastowe wyjście z programu. Także i ta instrukcja ta nie wymaga żadnych parametrów.

### PRINT

Znacznie ciekawsza jest pierwsza instrukcja zawierająca słowo kluczowe PRINT, po którym następuje tekst który zostanie napisany na ekranie. W tekście tym mogą wystąpić specjalne słowa SUBJECT i OBJECT. Ich znaczenie będzie omówione w drugiej lekcji, w której skupimy się na obiektach które gracz może znajdować i używać w trakcie przebiegu gry.

W naszym przykładzie, instrukcja PRINT służy do wyświetlenia pożegnalnego komunikatu:

??? > wake up

You woke up from the dream.

C:\Users\wgan\Desktop\DREAM\dream\lession\_01\first>\_

Warto tu zauważyć, że wszystkie słowa kluczowe zapisywane są wielkimi literami. Zostało to zrobione by je wyróżnić i jednocześnie nie mylić z parametrami. Dzięki temu nie trzeba dodawać specjalnych znaków – nawiasów, apostrofów, przecinków co przybliża zapis programu do języka naturalnego.

Teraz możemy sprawdzić czy opisy i komunikaty które przygotowaliśmy są poprawne i czas na ciąg dalszy:

## Druga lokacja i pierwszy refactoring

Kolejna lokacja – to kolejny plik tekstowy w folderze z naszą grą. Wybierzmy taki do którego można przejść z lokacji startowej. Zawartość pliku będzie podobna jak w przypadku pierwszej lokacji, zmienimy tylko nazwę oraz opis. Jeśli pierwszą lokacją w naszym przykładzie było wejście do jaskini, to drugą – będzie sama jaskinia:

[type]

location

[name]

cave

[description]

You see a cave walls around

You are in a cave which look like ordinary cave. Somebody uses it as a shelter long time ago. You can find a fireplace in the centre. You see a window of a sky on west. After a while you can examine the cave and you see few holes and cracks around you can try squeeze through.

Do akcji jakie można wykonać w pierwszej lokacji dodajmy jedną linię pozwalająca na przejście do właśnie przygotowanej jaskini:

e; east; go east THEN GOTO cave

Tu polecenia ma aż trzy warianty – nazwane od kierunków geograficznych. Możemy oczywiście użyć dowolnych innych słów na określenie polecenia wejścia do jaskini, ale kierunki geograficzne wydają się całkiem dobrym wyborem.

Ponowne uruchomienie gry powoli na sprawdzenie połączenia:

You are in entrance

You see dark cave entrance on the east

??? > east

You are in cave

You see a cave walls around

??? >

Niestety teraz nie możemy już nic zrobić. Trafiliśmy do lokacji w której nie ma określonych akcji. Aby to naprawić - musimy do jaskini dodać odpowiednie akcje – pozwalające na powrót do wejścia oraz te które pozwolą rozejrzeć się i wyjść z gry.

[actions]

w; west; go west THEN PRINT You exit the cave; GOTO entrance

look THEN LOOK

wake up; exit THEN PRINT You woke up from the dream.; EXIT

Przy okazji dodaliśmy komunikat o tym że wychodzimy z jaskini.

### GOTO

Instrukcja GOTO którą użyliśmy, pozwala na przejście do nowej lokacji. Instrukcja ta wymaga podania nazwy miejsca do którego przechodzimy. Może to być nazwa dowolnej lokacji która znajduje się w naszej grze. Lokacja musi być zdefiniowana – to znaczy musi istnieć plik o takiej nazwie w katalogu z tworzoną przez nas grą. Jeśli odpowiedna definicja nie istnieje – program zgłosi błąd.

Jeśli dodany polecenie odwołujące się do nieistniejącej lokacji:

s THEN GOTO anywhere

to nasza gra uruchomi się normalnie i będzie normalnie działać, jednak przy próbie wykonania tego polecenie – otrzymamy błąd:

Processing command 's' raised error: 'Cannot create object named 'anywhere''

**UWAGA: Poprawienie błędu nie oznacza, że gra zacznie działać poprawnie. Definicje obiektów są ładowane gdy używamy ich po raz pierwszy, i jeśli są one błędne, to do ponownego uruchomienia gry takie pozostaną nawet jeśli te błędy już poprawimy.**

Czytelnik pewnie już zauważył, że w obu lokacjach musieliśmy powtórzyć te same akcje które wyglądają na ogólne – wyjście z programu i rozglądanie się w lokacji w której znajduje się gracz. Powtarzanie tego samego kodu jest nudne i trudne do poprawienie jeśli zmienimy zdanie i będziemy chcieli je zmodyfikować. Dlatego definicja lokacji może zawierać jeszcze jedną sekcję: realm, w której możemy określić do jakiej rzeczywistości należy dana lokacja. Rzeczywistość jest także obiektem gry w którym również możemy określić akcje.

Przygotujmy taki obiekt dla naszej gry:

[type]

realm

[name]

reality

[actions]

wake up; exit THEN PRINT You woke up from the dream.; EXIT

look THEN PRINT You look around; LOOK

Jeśli teraz zarówno do wejścia jak i do jaskinie dodamy sekcję:

[realm]

reality

To akcje zdefiniowane na poziomie rzeczywistości – będą również tam dostępne.

A co jeśli akcję o takiej samej nazwie zdefiniujemy na poziomie rzeczywistości i lokacji? Wtedy wykonana będzie tak która jest typowa dla lokacji – jako bardziej precyzyjna. Dzięki temu możemy określać domyślne znaczenia akcji na poziomie całej rzeczywistości, a konkretne realizacje – w lokacjach. Przykładem może być ogólne określenie możliwych kierunków ruchu. Jeśli na poziomie rzeczywistości dodamy akcje:

n; north; go north THEN PRINT It is not possible to go north

ne; northeast; go northeast THEN PRINT It is not possible to go northeast

e; east; go east THEN PRINT It is not possible to go east

se; southeast; go southeast THEN PRINT It is not possible to go southeast

s; south; go south THEN PRINT It is not possible to go south

sw; southwest; go southwest THEN PRINT It is not possible to go southwest

w; west; go west THEN PRINT It is not possible to go west

nw; northwest; go northwest THEN PRINT It is not possible to go northwest

Wtedy przy próbie poruszania się w kierunku dla którego nie określono wyjścia zamiast komunikatu o błędzie:

??? > south

I do not know how to south

Dostaniemy nieco bardziej odpowiedni:

??? > south

It is not possible to go south

## Trening czyni mistrza

Gdy już wiemy jak budować lokacje, możemy przygotować sobie więcej lokacji implementując przygotowaną wcześniej mapę. Możemy się także zastanowić nad nieco innym spojrzeniem na poszczególne lokacje. Nie muszą one odpowiadać ściśle określonym miejscom. Mogą na przykład odpowiadać określonym stanom w jakim znajduje się gracz. Przykładem może być nudny dzień – ciąg lokacji które trzeba po kolei przechodzić tylko po to by dotrzeć do takiej w której możemy ponownie rozpocząć grę – na przykład w ramach porażki w trakcie przeżywania przygody.

# Lekcja druga – artefakty

W każdej grze przygodowej możemy odnaleźć różne artefakty które są potrzebne do skończenia gry. Może to być skarb który trzeba odnaleźć i przynieść, a mogą to być obiekty które są potrzebne by wykonać jakąś czynność. Na przykład – by wykopać skarb, możemy potrzebować szpadla, który musimy przynieść w miejsce w którym skarb jest ukryty.

W naszej grze – rolę takich obiektów/artefaktów tworzy item.

## Definiujemy artefakt

Definiujemy go tak samo jak lokację – w pliku tekstowym podzielonym na sekcje. Przygotujmy sobie taki obiekt przydatny w naszej grze:

[type]

item

[name]

shovel

[description]

Ordinary shovel.

This is ordinary shovel. Maybe it is not so big, but handy.

Teraz nasz obiekt możemy gdzieś umieścić – na przykład w jaskini. Aby to zrobić, musimy dodać do pliku cave.def – dodatkową sekcję zawierającą elementy które znajdują się w tej lokacji:

[items]

shovel

Jeśli teraz uruchomimy grę, i wejdziemy do jaskini w której umieściliśmy szpadel – możemy go zobaczyć jeśli się rozejrzymy:

You are in cave

You see a cave walls around

??? > look

You look around

You are in a cave which look like ordinary cave. Somebody uses it as a shelter long time ago. You can find a fireplace in the centre. You see a window of a sky on west. After a while you can examine the cave and you see few holes and cracks around you can try squeeze through.

You notice:

- Ordinary shovel.

## Obiekt można zabrać ze sobą

Obiekt nie byłyby użyteczne, gdyby nie to, że możemy je przenosić i używać, a także – to że mogą definiować własne akcje, które mają pierwszeństwo w stosunku do akcji określonych w lokacji i rzeczywistości. Dlatego posiadając określony przedmiot możemy wykonywać pewne akcje z innym skutkiem.

### TAKE

Na początek jednak musimy taki obiekt ze sobą zabrać. Do tego służy polecenie TAKE które jako parametru wymaga nazwy obiektu który należy zabrać. Obiekt ten zostaje przeniesiony do listy przedmiotów jakie postać gracza nosi ze sobą.

Możemy odpowiednią akcję dodać do akcji danej lokacji:

take THEN PRINT You grab shovel.; TAKE shovel

### OBJECT

To będzie działać, niestety także wtedy gdy już zabierzemy łopatę i przeniesiemy ją do innego pomieszczenia. Lepiej byłoby napisać ogólną akcję zabierania różnych rzeczy. Aby to osiągnąć musimy użyć słowa kluczowego OBJECT, które odpowiada nazwie obiektu który był wymieniony w poleceniu jakie wydał gracz. Taką akcję możemy zdefiniować jako:

take THEN PRINT You grab OBJECT; TAKE OBJECT

a następnie wywołać:

??? > take shovel

You grab shovel

??? >

Przy takim wywołaniu, zostanie zawołana akcja take, natomiast jako OBJECT, będzie użyty dostępny obiekt z listy obiektów użytkownika lub lokacji, którego nazwa pasuje do tekstu który użytkownik umieścił po take. Jeśli taki element nie istnieje – gra wyświetli błąd:

??? > take gizmo

I see no gizmo

??? >

Błąd powstanie przy pierwszym użyciu parametru OBJECT i dalsze wykonywanie akcji będzie przerwane.

Wspominaliśmy, że sekcja name w definicji elementu może zawierać wiele nazw. Przy poszukiwaniu obiektu w czasie dopasowywania do wartości sprawdzane są wszystkie nazwy. To pozwala nieco przybliżyć interfejs gry do języka naturalnego. Nasza łopata mogłaby posiadać kilka nazw na przykład dodatkową nazwą może być spade. Wtedy również będziemy mogli wydać polecenie:

??? > take spade

You grab the shovel

??? >

Zauważmy, że w komunikacie użyta zostanie nazwa podstawowa tego obiektu.

Co ciekawe podobnie zadziała polecenie

??? > take small rusty spade

Dzieje się tak dlatego, że do porównania użyty jest nie tylko cały tekst polecenia po odcięciu nazwy akcji, ale także ten tekst po odcięciu pierwszego słowa które może oznaczać dodatkowe określenie przedmiotu. Takie odcinanie wykonywane jest do odnalezienia przedmiotu albo do momentu gdy zostanie ostatnie słowo. W naszym przypadku poszukiwane będą kolejno:

* small rusty spade
* rusty spade
* spade

OBJECT może być także użyty w instrukcji PRINT – jest wtedy zamieniany na podstawową nazwę wskazanego obiektu.

### POCKET

Obiekty które nosi ze sobą postać – możemy wyświetlić instrukcją POCKET. Jeśli przewidujemy użycie przedmiotów (a trudno ich nie używać) – warto dodać do naszej rzeczywistości polecenie:

i; inventory; pocket THEN POCKET

wtedy możemy użyć polecenia:

??? > i

You have:

- Ordinary shovel.

??? >

Polecenie to wyświetla krótkie opisy poszczególnych obiektów które postać ma przy sobie.

### EXAMINE

Jeśli chcemy wyświetlić długi opis przedmiotu, na przykład poszukując tam wskazówek czy sugestii co do przebiegu dalszej gry, musimy użyć polecenia EXAMINE. Polecenie to przyjmuje jedne parametr – obiekt którego opis ma być wyświetlony. Polecenie to warto dodać do listy poleceń zdefiniowanych dla całej rzeczywistości:

examine; describe; look at THEN EXAMINE OBJECT

Zauważmy, że może tu nastąpić kolizja z akcją *look*. Możemy ją ograniczyć definiując akcję *look at* wcześniej – tak by przy porównaniu polecenia użytkownika najpierw sprawdzić dłuższą wersję polecenia.

Polecenie działa na obiekty dostępne w danym momencie gry – czyli obiekty umieszczone w lokacji i w „kieszeni” gracza.

??? > examine small spade

This is ordinary shovel. Maybe it is not so big, but handy.

??? >

### DROP

Odwrotne działanie ma polecenie DROP, które pozostawia przedmiot w bieżącej lokacji. Także i tu możemy użyć parametru OBJECT.

drop THEN PRINT You drop OBJECT; DROP OBJECT

Obiekty które są umieszczone w lokacji – zobaczymy gdy przyjrzymy się lokacji.

## Użycie artefaktu

Przedmioty byłyby w grze bezużyteczne, gdyby nie możliwość dodawanie do nich akcji. Sęk w tym, że niektóre akcje mogą być charakterystyczne nie tylko dla przedmiotów, ale i dla lokacji.

Na przykład, skarb może być wykopany tylko w piaszczystej jaskini i tylko wtedy gdy mamy przy sobie mapę oraz łopatę. Czy środowisko DREAM pozwala na stawianie takich warunków? Pozwala, ale zanim dojdziemy do warunków wykonywania poleceń, sięgniemy po nieco prostsze metody i użyjemy polecenia EXEC.

### EXEC

Polecenie EXEC pozwala na wywołanie innej akcji. Cała treść która znajduje się po słowie kluczowym EXEC, jest przekazywana do interpretacji tak, jakby wydał to polecenie gracz. Są jednak pewne drobne różnice:

1. Jeśli polecenie nie może być wykonane, bo nie jest w danej chwili dostępne – informacja o tym nie pojawia się na konsoli. Zamiast tego zgłaszany jest błąd informujący, że cała akcja nie może zostać wykonana.
2. Możemy wywoływać także specjalne akcje których nazwa zaczyna się od znaku podkreślenia. Takie akcje nie są dostępne dla normalnego użytkownika.

Aby osiągnąć cel – wykopanie skarbu w określonym miejscu i przy użyciu określonego narzędzia, musimy przygotować kilka akcji związanych z wszystkimi zaangażowanymi elementami gry.

Na początek – nasza łopata pozwala na kopanie. Dlatego dodamy do obiektu shovel sekcję zawierającą akcję:

dig THEN PRINT You dig in the ground.; EXEC \_ dig ground

Akcja ta deleguje samą akcję kopania pozwalając uzależnić ją od miejsca w którym kopiemy. Na początku wyświetlamy komunikat o tym co robimy, a potem, konkretną akcję delegujemy do innej akcji którą może być różnie określana w zależności od miejsca.

Jeśli nie zdefiniujemy nigdzie polecenia *„\_ dig ground*”, nasza akcja zakończy w tym miejscu się zakończy. Jeśli nie da się wykonać EXEC \_ dig ground, to nic się nie stanie. Po prostu akcja na tym się kończy.

??? > dig

You dig in the ground.

??? >

Odpowiednią akcję *„\_ dig ground*” możemy zdefiniować w lokacji w której skarb powinniśmy znaleźć, czyli w piaszczystej jaskini znajdującej się obok podziemnego jeziora. Do tamtejszych akcji dodamy:

\_ dig ground THEN EXEC \_ check treasure map; TAKE NEW treasure; PRINT You found a treasure

### NEW

To co robi ta akcja wygląda w miarę czytelnie. Na początku uruchamiamy akcję sprawdzającą położenie skarbu. Jeśli taka akcja nie jest nigdzie zdefiniowana. Na tym kończymy. Kolejne instrukcje tworzące i dodające obiekt do kieszeni gracza po prostu się nie wykonają. Jeśli jednak sprawdzenie się powiedzie – kolejnym poleceniem jest plecenie TAKE – znane już, ale tu użyte z tą różnicą, że nie odwołujemy się do obiektu który jest dostępny, ale do nowo utworzonego obiektu o zadanej nazwie. To właśnie oznacza słowo NEW – utworzenie nowego obiektu.

Teraz wystarczy przygotować mapę i umieścić ją w którejś lokacji w jaskini:

[type]

item

[name]

treasure map

map

[description]

A piece of map with red 'x' on it.

A piece of old paper with map of cave labirynth. It looks like something is hidden in cave with sand.

[actions]

\_ check treasure map THEN PRINT (because you looked at the treasure map and found where the treasure is hidden)

Zwróćmy uwagę na dodają akcję – jeśli będziemy mieli mapę – to będzie zdefiniowana akcja sprawdzenia położenia skarbu – sprawdzana podczas kopania we właściwej lokacji.

Potrzebujemy także samego skarbu:

[type]

item

[name]

treasure

[description]

A valuable treasure

It does not look like something you already seen, but it looks really valuable.

Jako samodzielne ćwiczenie proponuję dodanie domyślnej akcji kopania – takiej gdy nie mamy szpadla.

## Kolejne komplikacje

Przy przygotowywaniu gry możemy pomyśleć o kolejnych komplikacjach. Na przykład żeby odczytać mapę, musimy zabrać ją do kogoś kto zna starożytny język, albo samą mapę musimy ogrzać nad ogniskiem (które trzeba rozpalić). Pamiętajmy także, by w grze umieścić odpowiednie podpowiedzi, tak by gracz mógł ukończyć grę bez zaglądania do kodu.

### DEL

Przydatna będzie jeszcze jedna komenda: DEL pozwalająca na usuwanie elementu. Wtedy obiekty będziemy mogli podmieniać – jak w przykładzie ogniska – przeznaczonego do umieszczenia w dowolnej lokacji:

[type]

item

[name]

fireplace

[description]

A fireplace on the floor

It is a fireplace with some wood covered with sand. It looks like it was extinguished by someone and you can burn it.

[actions]

\_ burn THEN PRINT You burn a fire; DEL fireplace; DROP NEW fire

take fireplace THEN PRINT You cannot take the fireplace.

[type]

item

[name]

fire

[description]

A fire burning in a fireplace on the floor

The fire is warm and gives some light around.

[actions]

take fire THEN PRINT You cannot take the fire.

extinguish THEN PRINT You extinguish the fire with some sand.; DEL fire; DROP NEW fireplace

# Lekcja trzecia – właściwości

Jeśli budujemy zaawansowane gry o wielu możliwościach, może się zdarzyć, że do pewnych działań, musimy użyć konkretnego przedmiotu, który może być gdzie indziej używany zamiennie z innym. Może też się zdarzyć, że ta sama akcja powinna być różnie wykonana w zależności od rodzaju użytego narzędzia. Możemy w końcu potrzebować zdobycia pewnego doświadczenia by móc przeprowadzić jakąś akcję.

Do wszystkich tych celów użyjemy właściwości.

Właściwości – to wartości posiadające nazwę. Możemy je przypisać do przedmiotu, lokacji i gracza i możemy w nich przechowywać wartości które w trakcie gry mogą się zmieniać. W naszym programie wartości mogą być liczbą z zakresu od 0 do 9999. Nazwy mogą być dowolne.

Po tych ogólnikach – czas na konkretny przykład. Dla przedmiotu możemy określić jego wagę. Wystarczy w sekcji [properties] dopisać:

[type]

Item

[name]

Hammer

**[properties]**

**Weight = 20**

[description]

A hammer

An iron, pretty heavy hammer

Jak już wiemy jak definiować właściwości – pora na konkretne przykłady ich użycia.

## Kopalnia złota

Zbudujmy kopalnię złota w której można wydobywać złoto, ale ilość wydobytego złota zależy od rodzaju narzędzia którego użyjemy. Potrzebujemy lokacji „kopalnia złota”:

[type]

location

[realm]

realm

[name]

gold mine

[description]

You see a tunnel of old gold mine

You are in gold mine. It looks like abandoned long time ago, but when you look closely at the walls you see a stain of glitter. On the west you see the exit.

Jeśli kopalnia – to oczywiście potrzebujemy także złoto:

[type]

item

[name]

gold

[description]

Some freshly dig gold

Just a gold. It need cleaning and should be smelt before use for crafting, but it have a value, however it is not easy to pay with it in the shop.

Teraz zdefiniujmy dwa narzędzia o różnych „możliwościach wydobywczych” i dodajmy złoto które będziemy wydobywać. Możliwości wydobywcze określimy właśnie przy pomocy właściwości.

Pierwszym narzędziem którym możemy „drapać ściany” – jest nóż. Przedmiot przydatny chyba w każdej grze.

[type]

item

[name]

knife

simple knife

[properties]

power = 15

[description]

Simple knife

A simple piece of poor steel formed into primitive knife with wooden handle.

No i coś mocniejszego - kilof:

[type]

item

[name]

pickaxe

[properties]

power = 110

[description]

A strong and pretty new pickaxe

A pickaxe good for digging in hard ground and even in rock. Good for mining.

Pozostaje dodanie do kopalni złota – akcji które pozwolą na wydobycie. Akcje te uzależnimy od tego jakie przedmiot będzie użyty w poleceniu kopania. Możemy to tego podejść na dwa sposoby – uzależnić ilość wykopanego złota bezpośrednio od siły narzędzia.

[action]

dig THEN TAKE NEW gold; MODIFY gold value BY OBJECT power

Właściwość value ma specjalne znaczenie – jeśli wyświetlimy listę obiektów które posiada gracz, lub które znajdują się w lokacji i któryś z przedmiotów ma właściwość value ustawioną na wartość różną od zera – wartość ta wyświetla się obok tego przedmiotu.

### MODIFY … BY …

W naszej akcji użyliśmy polecenia modyfikującego tą właściwość. Polecenie MODIFY … BY zmienia wskazaną właściwość o zadaną wartość. Po słowie MODIFY podajemy nazwę właściwości dowolnego obiektu który znajduje się w naszym zasięgu. W tym wypadku modyfikujemy przedmiot gold, który właśnie stworzyliśmy, a następnie dodajemy do niego wartość właściwości power elementu który został przekazany jako parametr. Obiekt możemy określić, podobnie jak w poleceniach TAKE czy DROP podając po prostu jego nazwę, lub możemy użyć obiektu który został wspomniany w poleceniu wydanym przez użytkownika.

Zauważmy, że zarówno nazwa obiekty jak i nazwa właściwości może składać się z wielu słów. W takim wypadku program będzie próbował znaleźć obiekt którego nazwa będzie złożona z pierwszych słów, pozostałe interpretując jako nazwę właściwości.

Uwaga: jeśli obiekt nie istnieje, lub nie ma zdefiniowanej właściwości która sprawdzamy – wartość wynosi zero. Jeśli popełnimy literówkę w nazwie obiektu albo w nazwie właściwości - nie będą zgłaszane błędy, a jedynie wynik może nie do końca odpowiadać naszym oczekiwaniom.

Jeśli próbujemy modyfikować właściwość która nie istnieje, również nie będzie zgłoszony błąd. Program założy, że poprzednia wartość tej właściwości wynosi 0.

### MODIFY … TO …

Jeśli nie chcemy zmieniać właściwości a jedynie ją ustawić – możemy użyć innego wariantu polecenia modyfikującego właściwości – MODIFY … TO … - wariantu który ustawia właściwość na zadaną wartość.

Zaproponowane rozwiązanie jest proste Jest jednej problem. Za każdym razem będzie tworzony nowy obiekt gold, a właściwość value będzie modyfikowana w pierwszym znalezionym a nie w tym który właśnie stworzyliśmy. Jeśli więc mamy już złoto nie ma sensu go tworzyć.

### WHEN

W języku gry, mamy możliwość uzależnienia wykonania polecenia w zależności od właściwości. Służy do tego słowo kluczowe WHEN umieszczone wraz z warunkiem przed THEN:

[action]

dig WHEN **gold value > 0** THEN MODIFY gold value BY OBJECT power

dig THEN TAKE NEW gold; MODIFY gold value BY OBJECT power

Aby uzyskać różne działania tego samego polecenia – musimy je niestety powtórzyć.

### DICE

Gra bez elementów losowych mogłaby być nudna. Czasami chcielibyśmy, by rezultat jakiejś akcji był zależny od wartości losowej. W takim wypadku zamiast wartości właściwości możemy użyć słowa kluczowego DICE, które wylosuje wartość z zakresu od 1 do 6 – tak jakbyśmy rzucili kostką, i wartość tą zwróci. Jeśli wiec dodamy akcję:

dig WHEN DICE <= 4 THEN PRINT You attempt to dig but found nothing.

Polecenia kopania może się nie powieść – zamiast dodatkowego złota – otrzymamy komunikat o niepowodzeniu.

Tu istotna uwaga – polecenia są przeszukiwane w określonej kolejności i jeśli któreś polecenie będzie mogło się wykonać – to na jego wykonaniu gra zakończy próby. Jeśli wiec umieścimy tą akcję za wcześniej zdefiniowanymi – to nigdy się nie wykona, bo wcześniej mamy polecenie dig bez żadnych warunków. Jeśli jednak umieścimy je na początku – to jeśli wylosowana liczba będzie zmniejsza lub równa 4 – wyświetlony zostanie komunikat. Jeśli wylosujemy większą wartość – warunek nie będzie spełniony i program będzie próbował dopasować kolejne akcje.

Po słowie kluczowym WHEN możemy podać wiele warunków oddzielając je średnikami. Aby polecenie się wykonało – wszystkie warunki muszą być spełnione.

Po DICE można podać wartość która pozwoli na rozszerzenie losowanych wartości. Na przykład DICE 20 będzie odpowiadało rzutowi kostką dwudziestościenną a zwrócona wartość będzie z przedziału od 1 do 20. Jako wartość można podać również wartość jakiejś właściwości a nawet wyrażenie.

Podobnie wyrażenie można podać jako wartość po BY lub TO. DREAM dopuszcza tylko proste wyrażenia – zawierające właściwości, liczby oraz proste operacje matematyczne – dodawanie, odejmowanie, mnożenie i dzielenie. W obecnej wersji nie dopuszcza się użycia nawiasów.

Wyrażenia mogą występować także w warunkach, występując nawet po obu stronach porównań. Do dyspozycji mamy porównania <, <=, =, != (nie równe), >= i >.

Przeróbmy teraz nasze akcje tak, by kopać losową ilość złota, ale zakres ilości oraz prawdopodobieństwo powodzenia uzależnić od siły narzędzia. Przy okazji trochę rozbudujemy akcje tak, by można ich było używać także w innych kopalniach.

Po pierwsze: Określmy możliwe warianty akcji na poziomie ogólnym – w ramach testów możemy je zdefiniować w kopalnie złota, ale warto byłoby przenieść je na poziom całej rzeczywistości i definiować je jako akcje rzeczywistości do której należy kopalnia:

dig WHEN OBJECT power > 100 THEN EXEC \_ dig with pickaxe

dig WHEN OBJECT power > 10 THEN EXEC \_ dig with knife

dig THEN PRINT You cannot dig with bare hands

\_ dig with pickaxe; \_ dig with knife THEN PRINT You dig in the ground but you found nothing

Uzależniamy tu możliwości kopania od siły naszych narzędzi, ale robimy to tylko w trzech stopniach – jeśli narzędzie jest słabe – efekt jest taki jak przy kopaniu gołymi rękami – nic się nie uda wykopać (chyba że na poziomie lokacji określimy inaczej). Przy sile powyżej 100 oraz między 10 a 99 – wołane są akcje ukryte, dla których domyślne rezultaty – to także niemożliwość wykopania czegokolwiek. Taka definicja na poziomie rzeczywistości pozwoli nam kopać w zasadzie wszędzie, tyle że bez rezultatów.

W kopalnie kopanie powinno coś powodować. Dlatego w kopalni złota określamy akcje:

\_ dig with pickaxe WHEN DICE < 4 THEN PRINT You attempt to dig gold, but you found nothing.

\_ dig with pickaxe THEN PRINT You dig some gold; EXEC \_ create gold when needed; MODIFY gold value BY 10 + DICE 50

\_ dig with knife WHEN DICE < 5 THEN PRINT You attempt to dig gold by scratching the wall, but you found nothing.

\_ dig with knife THEN PRINT You dig some gold; EXEC \_ create gold when needed; MODIFY gold value BY DICE

\_ create gold when needed WHEN gold value = 0 THEN TAKE NEW gold

\_ create gold when needed THEN PRINT You add the new gold to the one you already found.

Tu dla każdej akcji przygotowujemy dwa warianty – zależne od rzutu kostką. Zauważmy, że warunek rzutu kostką sprawdzany jest raz. Gdybyśmy sprawdzili drugi raz rzut kostką – oznaczało by to ponowne losowanie.

Przy okazji dodaliśmy akcje dodawania złota jako obiektu do stanu posiadania gracza – tylko wtedy gdy jest to koniczne – przy użyciu dwóch funkcji. Pamiętamy, że polecenie EXEC musi wywołać jakąś akcję, inaczej wykonywanie polecenia zostanie przerwane.

Przykładowe użycie naszej kopalnie wygląda tak:

### ITEM

I na koniec jedna uwaga – jeśli chcemy uprościć grę i nie zawracać głowy użytkownikowi określeniem jakich przedmiotów używa, zamiast słowa kluczowego OBJECT, można w warunkach i akcjach użyć słowa kluczowego ITEM. Musi ono jednak wystąpić najpierw w kontekście właściwości – na przykład w warunku. ITEM oznacza ten przedmiot będący w posiadaniu gracza lub należący do lokacji, który ma największą wartość właściwości która była sprawdzana (lub zmieniana) jaki pierwsza.

Jeśli zmienimy nasze akcje:

dig WHEN ~~OBJECT~~ ITEM power > 100 THEN EXEC \_ dig with pickaxe

dig WHEN ~~OBJECT~~ ITEM power > 10 THEN EXEC \_ dig with knife

Nie będziemy musieli określać jakiego przedmiotu chcemy użyć do kopania. Użyjemy najsilniejszego jaki jest w naszym zasięgu.

You are in gold mine

You see a tunnel of old gold mine

??? > i

You have:

- Simple knife

??? > dig

You cannot dig with bare hands

??? > dig with knife

You attempt to dig gold by scratching the wall, but you found nothing.

??? > dig with knife

You dig some gold

??? > dig with knife

You dig some gold

You add the new gold to the one you already found.

??? > dig with knife

You dig some gold

You add the new gold to the one you already found.

??? > dig with knife

You dig some gold

You add the new gold to the one you already found.

??? > i

You have:

- Simple knife

- Some freshly dig gold (18)

??? >

Teraz pozostaje przygotowanie reszty gry – banku w którym możemy zamienić złoto na pieniądze i sklepu w którym możemy kupić narzędzia.

# Lekcja czwarta – inne zastosowanie właściwości

Właściwości mogą posiadać nie tylko artefakty, ale także lokacje oraz gracz. Pozwala to na przechowywanie informacji. Pozwala to na zmianę reakcji na polecenia w zależności od tego co zrobiliśmy wcześniej. Jako przykład zaproponujemy grę w rozwiązywanie zagadki detektywistycznej. W takiej zagadce musimy zdobyć informacje po to by móc zdobyć kolejne. Grę rozpoczynamy w lokacji w której znajduje się ciało i lokalny włóczęga od którego możemy uzyskać pewne informacje. Zwłoki możemy przeszukać znajdując przedmioty – podpowiedzi, a z włóczęgą możemy porozmawiać uzyskując istotne informacje.

Pierwsze zadanie jakie musimy wypełnić – to zbudować tak akcję oglądania zabitego, by przedmioty znaleźć tylko za pierwszym razem. Przygotujmy ofiarę jako obiekt który umieścimy w grze:

[type]

item

[name]

corpse

[properties]

locked = 1

have wallet = 1

[description]

The human corpse

It is 40-45 years old white male lying on the path. It looks dead but looks like sleeping. You see no cause of its current state.

[actions]

examine WHEN OBJECT have wallet > 0 THEN EXAMINE OBJECT; PRINT You search him and found wallet containing pawn shop ticket.; DROP NEW wallet; DROP NEW pawn shop ticket; MODIFY PLAYER know corpse BY 1; MODIFY OBJECT have wallet TO 0

Nasz obiekt posiada dwie właściwości. Pierwsza – locked – służy to poinstruowania gry, że obiekt o takiej właściwości nie może zostać zabrany. Nie zapewnia nam tego sam silnik gry, ale możemy samodzielnie to oprogramować na poziomie rzeczywistości. Wystarczy zamiast typowej implementacji polecenia take, napisać:

take WHEN OBJECT locked = 0 THEN TAKE OBJECT; PRINT You take OBJECT

take THEN PRINT You cannot take OBJECT

Nieco ciekawszym zastosowaniem właściwości jest użycie właściwości have wallet. Tu służy do włączenia specjalnej wersji akcji examine, zdefiniowanej w obiekcie którego dotyczy, niezależnie od wersji która jest określona dla całej rzeczywistości:

examine THEN EXAMINE OBJECT

W tej specjalnej wersji sprawdzamy czy obiekt już być oglądany, i jeśli nie – podejmowane są dodatkowe akcje tworzące kolejne obiekty które mogą być zabrane. Rozgrywka może wyglądać na przykład tak:

You are in Central Park

This is central park path.

??? > look

This part of Central Park is separated from the more crowded places of NY. Maybe this is why this place was selected for assasination.

You notice:

- The human corpse

- A typical NY hobo

??? > examine corpse

It is 40-45 years old white male lying on the path. It looks dead but looks like sleeping. You see no cause of its current state.

You search him and found wallet containing pawn shop ticket.

??? > look

This part of Central Park is separated from the more crowded places of NY. Maybe this is why this place was selected for assasination.

You notice:

- The human corpse

- A typical NY hobo

- An empty wallet

- A pawn shop ticket

??? > take corpse

You cannot take corpse

??? > take ticket

You take pawn shop ticket

??? > i

You have:

- A pawn shop ticket

??? >

Podobnie możemy rozegrać rozmowy. Aby uzyskać informacje potrzebujemy informacji na których możemy bazować. Jako przykład – napiszmy dialog z aktorką którą możemy spotkać na południe od central parku. Uzyskamy od niej informacje że podejrzana osoba poszła na południe. Ale aby taką informację uzyskać – musimy najpierw dowiedzieć się, że aktorka obserwuje przechodniów i że ktoś podejrzany mógł tędy przechodzić. Pierwszą informację powinniśmy uzyskać obserwując aktorkę a następnie zagajając rozmowę. Drugą – rozmawiając z włóczęgą w central parku.

[type]

item

[name]

actress

[properties]

locked = 1

is actress = 1

[description]

Actress in clown disguise

A young actress who attempt to play street clown. It is interesting - is it for fun or it prepaares for the role. Or, it is just student task.

[actions]

examine WHEN OBJECT is actress = 1 THEN EXAMINE OBJECT; MODIFY PLAYER know actress BY 1

talk WHEN PLAYER know hobo observation > 0; PLAYER know about observation > 0 THEN PRINT You say:; PRINT - Did you notice something strange?; PRINT - I think... Yes. One guy was emotionally totally unstable. He went south - to wall street.

talk WHEN PLAYER know actress > 0 THEN PRINT You say:; PRINT - Hello, what are you doing? Are you preparing yourself to the role?; PRINT - Yes, and I am studying peoples faces, their emotions.; MODIFY PLAYER know about observation BY 1

talk THEN PRINT You have no subject to talk about.

### PLAYER

Mamy tu trzy właściwości określające stan, wszystkie zdefiniowane dla gracza – co opisuje słowo kluczowe PLAYER. Po nim znajdują się nazwy właściwości:

* know hobo obserwation – różna od zera wtedy gry porozmawialiśmy z włóczęgą i wiemy, że potencjalny sprawca był w okolicy
* know actress – rózna od zera gdy gracz przyjrzał się aktorce
* know about observation – różna od zera gdy gracz dowiedział się, że aktorka przygląda się przechodniom.

Wygląda to w miarę prosto, ale budowa takich kaskad poleceń może okazać się skomplikowana zwłaszcza, że definicje mogą być porozrzucane po wielu plikach definiujących grę.

### LOCATION

Podobnie jak właściwości gracza, możemy modyfikować i sprawdzać właściwości lokacji – i nie wyświetlać pełnego opisu jeśli lokację już odwiedzaliśmy, albo inaczej interpretować akcje gdy jesteśmy w lokacji po raz pierwszy.

### BECOME

Lokację można zmieniać poleceniem GOTO. Istnieje też polecenie pozwalające zmienić gracza – to znaczy wszystkie jego właściwości oraz listę przedmiotów które posiada. Służy do tego polecenie BECOME które pozwala na załadowanie tych części stanu gry z pliku o podobnym formacie jak opis lokacji czy przedmiotu, z tą różnicą, że gracz jest typu player:

[type]

player

[items]

matches

[properties]

force = 1000

mana = 15

### TRACE

Definiując akcje na poziomie elementów, lokacji i rzeczywistości oraz manipulując właściwościami przedmiotów, gracza i lokacji łatwo się pogubić i popełnić trudne do znalezienia błędy które zmienią reakcję gry na polecenia. Aby łatwiej odnaleźć takie miejsca i sprawdzić co robi program sterujący grą, istnieje możliwość włączenia śledzenia. Po wykonaniu polecenia TRACE 1, na ekranie będą pojawiały się po kolei komunikaty gdzie poszukiwane są akcje pasujące do polecenia wydanego przez gracza. Jeśli akcja ma określone warunki wykonania – to także informacje który warunek jako pierwszy nie był spełniony jeśli dana akcja została odrzucona. Gdy warunki są spełnione – wyświetlane są po kolei polecenia jakie wykonuje gra. Śledzenie można w każdej chwili wyłączyć poleceniem TRACE 0.

Polecenia te nie mogą być wydane bezpośrednio przez gracza. Na czas przygotowywania gry warto je dodać do puli poleceń globalnych – zdefiniowanych na poziomie rzeczywistości:

trace THEN TRACE 1

ntrace THEN TRACE 0

Jako przykład działania – obaczymy jak wygląda ślad wykonania polecenia talk w lokacji z aktorką, jeśli nie mamy żadnych dodatkowych informacji:

??? > talk

You have no subject to talk about.

??? > trace

# ----- trace started

??? > talk

# attempt to execute player items actions

# attempt to execute location items actions

# - action 'talk WHEN PLAYER know hobo observation > 0; PLAYER know about observation > 0 THEN PRINT You say:; PRINT - Did you notice something strange?; PRINT - I think... Yes. One guy was emotionally totally unstable. He went south - to wall street.' mach user command

# - condition 'PLAYER know about observation > 0' faliled; command not executed

# - action 'talk WHEN PLAYER know actress > 0 THEN PRINT You say:; PRINT - Hello, what are you doing? Are you preparing yourself to the role?; PRINT - Yes, and I am studying peoples faces, their emotions.; MODIFY PLAYER know about observation BY 1' mach user command

# - condition 'PLAYER know actress > 0' faliled; command not executed

# - action 'talk THEN PRINT You have no subject to talk about.' mach user command

# - all conditions satisfied;

# - executing: PRINT You have no subject to talk about.

You have no subject to talk about.

### DUMP

Podobnie pomocnym poleceniem jest polecenie DUMP, które pozwala na wyświetlenie zawartości wskazanego obiektu. Musi to być obiekt który jest dostępny czyli jest obiektem w lokacji lub z przedmiotach zebranych przez gracza. Dodatkowo można taż się dostać do obiektów specjalnych:

LOCATION

DUMP LOCATION

Wyświetla komplet informacji o bieżącej lokacji wyświetlając ją podobnie jak obiekty w takiej postaci w jakiej definiujemy lokacje. W odróżnieniu od definicji obiektu w pliku – tu wyświetlona będzie rzeczywisty stan lokacji w której znajduje się gracz

REALM

DUMP REALM

Wyświetla informacje na temat stanu rzeczywistości. Tu obraz danych powinien być taki sam jak w pliku który definiuje rzeczywistość, bo żadne właściwości nie są tu przechowywane.

PLAYER

DUMP PLAYER

Wyświetla wszystkie informacje o bieżącym stanie gracza.

Dopóki prowadzimy prace nad grą, warto dodać do akcji na poziomie całej rzeczywistości kilka poleceń które pozwolą nam sprawdzić stan gry:

stat realm; dump realm; realm THEN DUMP REALM

stat player; dump player; player THEN DUMP PLAYER

stat location; dump location; location THEN DUMP LOCATION

stat; dump THEN DUMP OBJECT

trace THEN TRACE 1

ntrace THEN TRACE 0

# Uwagi na koniec

Przygotowując grę, warto myśleć o wielu zastosowaniach tych samych przedmiotów. Z jednej strony – nie potrzebujemy tak wielu przedmiotów tworzyć specjalizując dla nich akcje, z drugiej strony – uczymy gracza skojarzeń i nieco niestandardowego myślenia. Pomaga w tym myślenie o przedmiotach jako o elementach bibliotecznych – takich które będziemy wykorzystywać także w innych grach.

1. Lokacja – miejsce w którym może znajdować się gracz. W lokacji mogą znajdować się przedmioty i mogą być w niej określone akcje. [↑](#footnote-ref-1)
2. Przedmiot – obiekt który można umieścić w lokacji a także zabrać ze sobą czy przetworzyć. Z przedmiotem mogą być związane akcje [↑](#footnote-ref-2)