Programujemy gry przygodowe

w środowisku DREAM

WGan

# 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

## Początek – 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]

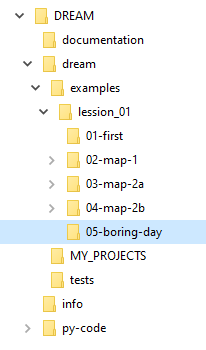
You see dark cave entrance on the east

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

??? >

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.

Spis treści

[Od autora 1](#_Toc40038716)

[Lekcja pierwsza - mapa 1](#_Toc40038717)

[Początek – potrzebujemy planu 1](#_Toc40038718)

[Pierwsza lokacja 2](#_Toc40038719)

[LOOK 3](#_Toc40038720)

[EXIT 4](#_Toc40038721)

[PRINT 4](#_Toc40038722)

[Druga lokacja i pierwszy refactoring 4](#_Toc40038723)

[GOTO 5](#_Toc40038724)

[Trening czyni mistrza 6](#_Toc40038725)

[Lekcja druga – artefakty 6](#_Toc40038726)

[Definiujemy artefakt 7](#_Toc40038727)

[Obiekt można zabrać ze sobą 7](#_Toc40038728)

[TAKE 7](#_Toc40038729)

[OBJECT 7](#_Toc40038730)

[POCKET 8](#_Toc40038731)

[EXAMINE 9](#_Toc40038732)

[DROP 9](#_Toc40038733)

[Użycie artefaktu 9](#_Toc40038734)

[EXEC 9](#_Toc40038735)

[NEW 10](#_Toc40038736)

[Kolejne komplikacje 11](#_Toc40038737)

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)