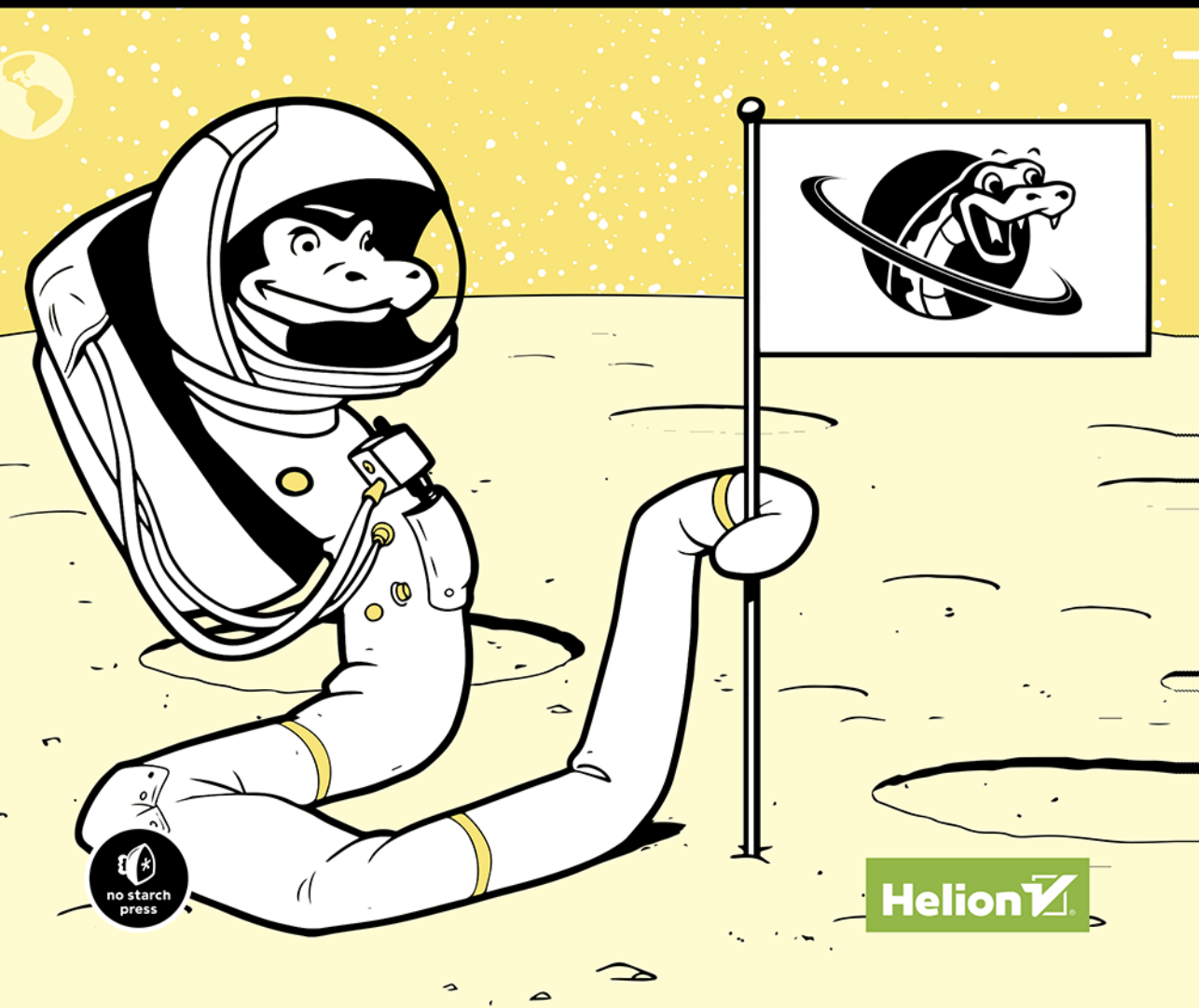
PYTHON
Z ŻYCIA WZIĘTY

ROZWIĄZYWANIE PROBLEMÓW
ZA POMOCĄ KILKU LINII KODU

LEE VAUGHAN



Tytuł oryginału: Real-World Python: A Hacker's Guide to Solving Problems with Code

Tłumaczenie: Karolina Stangel

ISBN: 978-83-283-8346-3

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/pytzyc.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/pytzyc
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

1

Uratuj rozbitków dzięki twierdzeniu Bayesa

Mniej więcej koło 1740 roku angielski prezbiteriański duchowny Thomas Bayes postanowił matematycznie udowodnić istnienie Boga. Jego pomysłowe rozwiązanie tego problemu, znane obecnie pod nazwą **twierdzenia Bayesa**, jest jedną z najbardziej przydatnych koncepcji statystycznych wszechczasów. Przez 200 lat było jednak ignorowane ze względu na to, że żmudne obliczenia matematyczne, jakich wymagało, nie nadawały się do ręcznego wykonywania. Potrzeba było wynalazku takiego jak nowoczesne komputery, aby twierdzenie Bayesa ujawniło pełnię swego potencjału. Obecnie dzięki szybkim procesorom stanowi ono podstawowy komponent uczenia maszynowego i analizy danych.

Ponieważ twierdzenie Bayesa pokazuje, jak w sposób matematyczny uwzględnić nowe dane i ponownie oszacować prawdopodobieństwo, ma zastosowanie w prawie każdej dziedzinie ludzkiego życia — od łamania szyfrów, przez typowanie zwycięzców wyborów prezydenckich, po wykazywanie, że wysoki poziom cholesterolu powoduje zawały. Lista zastosowań twierdzenia Bayesa mogłaby z powodzeniem wypełnić cały ten rozdział. Nic nie jest jednak ważniejsze od ratowania ludzkiego życia i dlatego skupimy się na wykorzystaniu sformułowanej przez Bayesa prawidłowości do ratowania rozbitków zagubionych na morzu.

W tym rozdziale stworzysz grę będącą symulatorem poszukiwań ratowników ze Straży Wybrzeża. Gracze będą korzystać z twierdzenia Bayesa do podejmowania

decyzji, mając na celu możliwie najszybsze zlokalizowanie rozbitka. W ramach projektu użyjesz popularnych narzędzi z zakresu rozpoznawania obrazów i analizy danych, takich jak biblioteki Open Source Computer Vision (OpenCV) oraz NumPy.

Twierdzenie Bayesa

Twierdzenie Bayesa pomaga określić prawdopodobieństwo tego, że coś jest prawdą, po pojawieniu się nowych danych. Jak ujął to wielki francuski matematyk Laplace: „Prawdopodobieństwo przyczyny, biorąc pod uwagę zajście zdarzenia, jest proporcjonalne do prawdopodobieństwa tego zdarzenia, biorąc pod uwagę zajście przyczyny". W swej podstawowej formie twierdzenie Bayesa wygląda następująco:

$$P(A/B) = \frac{P(B/A) \cdot P(A)}{P(B)}$$

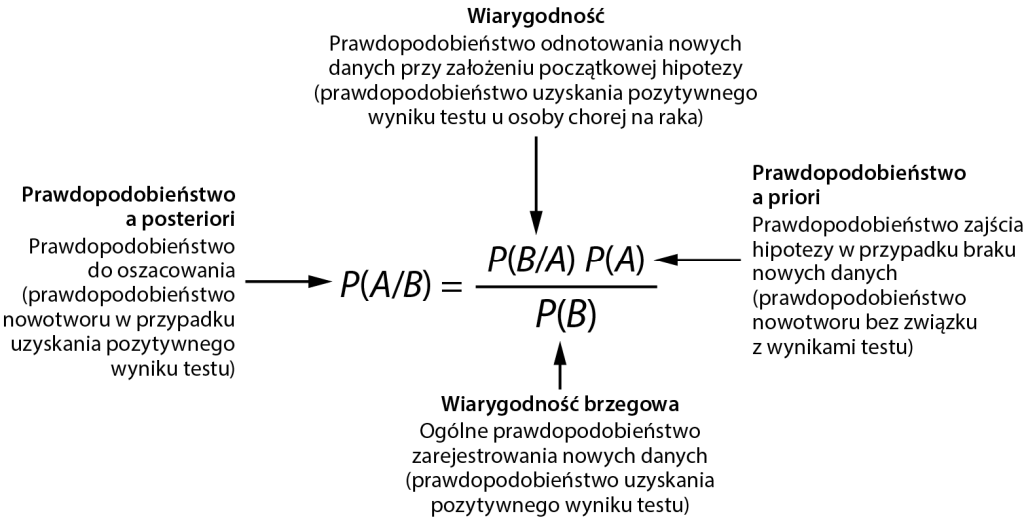
gdzie *A* to hipoteza, a *B* to dane. *P(A/B)* oznacza prawdopodobieństwo hipotezy *A* w świetle danych *B*, zaś *P(B/A)* — prawdopodobieństwo danych *B* przy założeniu hipotezy *A*. Załóżmy na przykład, że test na obecność pewnego rodzaju nowotworu nie zawsze jest dokładny i że może zwracać wyniki fałszywie pozytywne, które informują o obecności nowotworu, gdy tak naprawdę badana osoba go nie ma. Twierdzenie Bayesa przyjęłoby wówczas postać:

$$\begin{pmatrix}\text{Prawdopodobieństwo nowotworu}\\ \text{przy pozytywnym wyniku testu}\end{pmatrix} = \begin{pmatrix}\text{Prawdopodobieństwo pozytywnego wyniku}\\ \text{testu wśród pacjentów chorych na raka}\end{pmatrix} \cdot \frac{\begin{pmatrix}\text{Prawdopodobieństwo}\\ \text{nowotworu}\end{pmatrix}}{\begin{pmatrix}\text{Prawdopodobieństwo}\\ \text{pozytywnego wyniku testu}\end{pmatrix}}$$

Ustalenie podstawowych prawdopodobieństw byłoby zadaniem badań klinicznych. Na przykład 800 z 1000 osób chorych na raka uzyskuje pozytywny wynik testu, 100 na tysiąc osób zdrowych otrzymuje błędną diagnozę, a ogólne prawdopodobieństwo, że dana osoba jest chora na raka, wynosi jedynie 50 na dziesięć tysięcy. Jeżeli ogólne prawdopodobieństwo zachorowania jest niskie, a ogólne prawdopodobieństwo uzyskania pozytywnego wyniku jest względnie wysokie, prawdopodobieństwo choroby przy pozytywnym wyniku testu spada. Jeżeli badania odnotowały częstość występowania nieprawidłowych wyników testów, twierdzenie Bayesa może skorygować prawdopodobieństwo o błędy pomiaru!

Teraz, gdy znasz już przykład zastosowania, przyjrzyj się rysunkowi 1.1, który pokazuje nazwy różnych części składowych twierdzenia Bayesa wraz z informacją o tym, jak odnoszą się do przykładu z badaniami na obecność nowotworu.

Przeanalizujmy jeszcze jeden przykład. Powiedzmy, że pewna kobieta zgubiła gdzieś w domu okulary do czytania. Pamięta, że ostatni raz miała je na sobie, gdy była w swoim gabinecie. Idzie tam i się rozgląda. Nie widzi okularów, ale dostrzega filiżankę herbaty i przypomina sobie, że poszła do kuchni. W tym momencie



Rysunek 1.1. Twierdzenie Bayesa wraz z definicjami terminów oraz ich powiązaniami z przykładem dotyczącym testu na obecność nowotworu

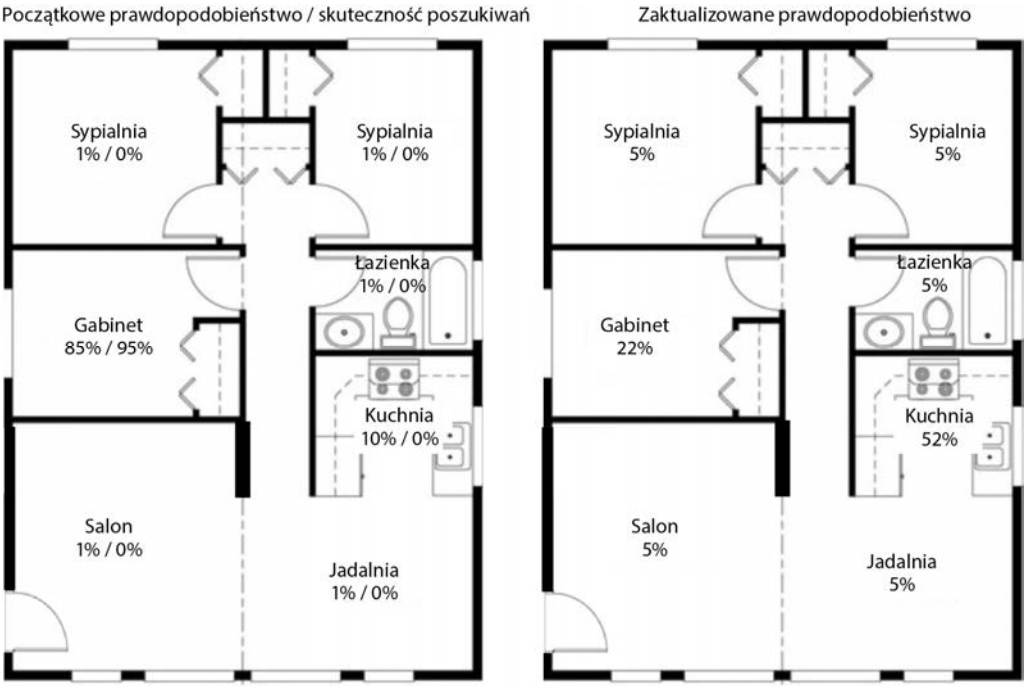
musi dokonać wyboru: albo dalej szuka w gabinecie, ale dokładniej, albo idzie sprawdzić kuchnię. Decyduje się pójść do kuchni. Nieświadomie podjęła decyzję bayesowską.

Najpierw poszła do gabinetu, ponieważ czuła, że daje jej to największe prawdopodobieństwo sukcesu. W terminach bayesowskich początkowe prawdopodobieństwo znalezienia okularów w gabinecie zwane jest **prawdopodobieństwem *a priori***. Po pobieżnym przeszukaniu pomieszczenia zmieniła decyzję na podstawie dwóch nowych informacji: po pierwsze nie znalazła tam okularów, a po drugie zobaczyła filiżankę. To jest właśnie **bayesowska aktualizacja**, w ramach której w miarę uzyskiwania dodatkowych dowodów wylicza się nowe oszacowanie prawdopodobieństwa *a posteriori* (tj. *P(A/B)* na rysunku 1.1).

Wyobraźmy sobie, że kobieta zdecydowała się zaprząc twierdzenie Bayesa do pomocy w poszukiwaniach. Oszacowałaby prawdopodobieństwo, że okulary znajdują się w gabinecie lub w kuchni (wiarygodność), a także skuteczność własnych poszukiwań w obydwu pokojach. Teraz jej decyzje nie opierają się na intuicji, lecz na podstawach matematycznych, które można stale aktualizować, jeżeli przyszłe poszukiwania skończą się na niczym.

Rysunek 1.2 ilustruje realizowane przez kobietę poszukiwania okularów z przypisanym prawdopodobieństwem.

Lewa strona diagramu przedstawia początkowe prawdopodobieństwo. Po prawej stronie mamy wartości zaktualizowane zgodnie z twierdzeniem Bayesa. Powiedzmy, że początkowo istniało 85-procentowe prawdopodobieństwo znalezienia okularów w gabinecie i 10-procentowe znalezienia ich w kuchni. Innym pomieszczeniom przypiszemy prawdopodobieństwo 1-procentowe, ponieważ w przypadku



Rysunek 1.2. Początkowe wartości prawdopodobieństwa znalezienia okularów oraz skuteczności poszukiwań (po lewej) w porównaniu ze zaktualizowanym prawdopodobieństwem (po prawej)

zastosowania wzoru Bayesa nie jest możliwe zaktualizowanie prawdopodobieństwa wynoszącego zero (a dodatkowo zawsze istnieje niewielka szansa, że kobieta zostawiła okulary w jednym z tych pomieszczeń).

Liczby po ukośniku po lewej stronie diagramu reprezentują **prawdopodobieństwo skuteczności poszukiwań** (**PSP**). Jest to szacunkowa ocena tego, jak skutecznie przeszukaliśmy dany obszar. Ponieważ na razie kobieta szukała okularów tylko w gabinecie, ta wartość wynosi zero w odniesieniu do wszystkich pozostałych pomieszczeń. Po bayesowskiej aktualizacji (zauważeniu filiżanki) nasza poszukiwaczka może ponownie obliczyć prawdopodobieństwo, opierając się na wynikach poszukiwań, tak jak pokazano po prawej. Kuchnia jest teraz najbardziej prawdopodobnym miejscem, ale prawdopodobieństwo, że okulary znajdują się w innych pokojach, również wzrosło.

Ludzka intuicja podpowiada nam, że jeżeli coś nie jest tam, gdzie się tego spodziewamy, rośnie prawdopodobieństwo, że jest gdzieś indziej. Twierdzenie Bayesa uwzględnia tę zmianę i dlatego prawdopodobieństwo znalezienia okularów w innych pokojach rośnie. Jest to jednak możliwe jedynie wtedy, gdy od samego początku istnieje choćby nikła szansa, że przedmiot znajduje się w innym pokoju.

Wzór na prawdopodobieństwo, że okulary zostały zgubione w danym pomieszczeniu, uwzględniając skuteczność poszukiwań, wygląda następująco:

$$P(G/E) = \frac{P(E/G) \cdot P_{\text{prior}}(G)}{\sum P(E/G') \cdot P_{\text{prior}}(G')}$$

gdzie G to prawdopodobieństwo, że okulary znajdują się w danym pomieszczeniu, E to skuteczność poszukiwań, a P_{prior} to oszacowanie prawdopodobieństwa *a priori* lub wstępnego, przed otrzymaniem nowych dowodów.

Zaktualizowane prawdopodobieństwo tego, że okulary znajdują się w gabinecie, możemy wyliczyć, podstawiając początkowe oszacowania prawdopodobieństwa i skuteczność poszukiwań do równania w następujący sposób:

$$\frac{(0{,}85\cdot(1-0{,}95))}{(0{,}85\cdot(1-0{,}95)+0{,}1\cdot(1-0)+0{,}01\cdot(1-0)+0{,}01\cdot(1-0)+0{,}01\cdot(1-0)+0{,}01\cdot(1-0)+0{,}01\cdot(1-0))}$$

Jak widzisz, działania matematyczne wymagane przez wzór Bayesa są proste, ale szybko może Ci się skończyć cierpliwość, jeżeli będziesz wykonywać je ręcznie. Na szczęście żyjemy we wspaniałej epoce komputerów, więc to, co nudne i żmudne, możemy zostawić Pythonowi!

Projekt #1: Symulacja misji poszukiwawczo-ratunkowej

W ramach tego projektu napiszesz program w języku Python, który wykorzystuje twierdzenie Bayesa do odnalezienia samotnego żeglarza, który zaginął na morzu w pobliżu Przylądka Python. Jesteś dowódcą akcji poszukiwawczo-ratunkowej prowadzonej przez Straż Wybrzeża na pewnym obszarze. Na podstawie rozmowy z żoną żeglarza określono jego ostatnią znaną lokalizację, ale było to ponad sześć godzin temu. Żeglarz zgłosił przez radio, że opuszcza statek, ale nikt nie wie, czy przebywa w łodzi ratunkowej, czy dryfuje na morzu. Wody wokół przylądka są ciepłe, ale jeżeli rozbitek znajduje się w wodzie, za około 12 godzin czeka go hipotermia. Jeżeli ma na sobie kamizelkę ratunkową, przy odrobinie szczęścia może przetrwać nawet trzy dni.

Prądy morskie przy Przylądku Python tworzą skomplikowany system (rysunek 1.3), a wiatr wieje obecnie z południowego zachodu. Widoczność jest dobra, ale morze jest lekko wzburzone, co sprawia, że trudno zauważyć głowę człowieka między falami.

W prawdziwej sytuacji następnym krokiem byłoby wprowadzenie wszystkich dostępnych informacji do systemu SAROPS Straży Wybrzeża, czyli systemu optymalnego planowania działań poszukiwawczo-ratunkowych. Uwzględnia on takie czynniki jak wiatry, pływy, prądy, to, czy człowiek znajduje się w wodzie, czy w łodzi, i tak dalej. Następnie generuje prostokątne obszary poszukiwań, wylicza początkowe prawdopodobieństwo znalezienia zaginionej osoby w każdym obszarze i wyznacza najbardziej wydajną ścieżkę lotu.

Rysunek 1.3. Prądy morskie w pobliżu Przylądka Python

W naszym projekcie założymy, że system SAROPS zidentyfikował trzy obszary poszukiwań. Twoim zadaniem jest napisanie programu, który wykorzystuje twierdzenie Bayesa do wspomagania procesu podejmowania decyzji. Zasoby, jakimi dysponujesz, wystarczą na przeszukanie dwóch z trzech obszarów w ciągu jednego dnia. Musisz zdecydować, w jaki sposób rozdzielić te zasoby. To duża presja, ale u swego boku masz do pomocy doskonałego doradcę — jest nim właśnie twierdzenie Bayesa.

CEL *Utwórz grę symulującą misję poszukiwawczo-ratunkową, która wykorzystuje twierdzenie Bayesa do wspierania graczy w podejmowaniu świadomych decyzji na temat tego, jak prowadzić poszukiwania.*

Strategia

Szukanie rozbitka na morzu przypomina w pewnym stopniu szukanie okularów z poprzedniego przykładu. Zaczynasz od początkowych szacunków prawdopodobieństwa, a następnie aktualizujesz je zgodnie z wynikami poszukiwań. Jeżeli dokładnie przeszukasz dany obszar, ale nic nie znajdziesz, wzrośnie prawdopodobieństwo, że zaginiony znajduje się w innym miejscu.

Tak jak w prawdziwym życiu, istnieją dwie rzeczy, które mogą pójść nie tak: możesz dokładnie przeszukać obszar, a mimo to przegapić zaginionego, albo poszukiwania mogą spalić na panewce z różnych przyczyn, a całodzienny wysiłek pójdzie na marne. Aby przełożyć to na wartość liczbową skuteczności poszukiwań, w pierwszym przypadku PSP wynosi 0,85, ale poszukiwany znajduje się na

pozostałych 15% obszaru, których nie przeczesano. W drugim przypadku PSP wynosi 0,2, czyli nie sprawdzono aż 80% obszaru!

Teraz możesz zrozumieć, przed jakimi dylematami stoją prawdziwi dowódcy. Posłuchać intuicji i zignorować twierdzenie Bayesa? Zdać się na czystą i zimną logikę wzorów matematycznych, wierząc, że to najlepsze wyjście? Czy może zastosować się do zaleceń wynikających z twierdzenia Bayesa, mimo że w głębi duszy wątpisz w ich skuteczność, tylko po to, by chronić własną karierę i reputację?

Aby wspomóc gracza, wykorzystamy bibliotekę OpenCV. Zbudujemy dzięki niej interfejs pozwalający wchodzić w interakcje z programem. Sam interfejs nie musi być dziełem sztuki. Może składać się z menu drukowanego w powłoce. Dobrze jednak by było, gdyby program wyświetlał również mapę przylądka wraz z obszarami poszukiwań. Użyjemy jej do zaznaczenia ostatniej znanej lokalizacji zaginionego oraz jego pozycji w momencie znalezienia. Biblioteka OpenCV stanowi doskonały wybór do celów tej gry, ponieważ pozwala wyświetlać obrazy oraz dodawać do nich kształty i tekst.

Instalacja bibliotek Pythona

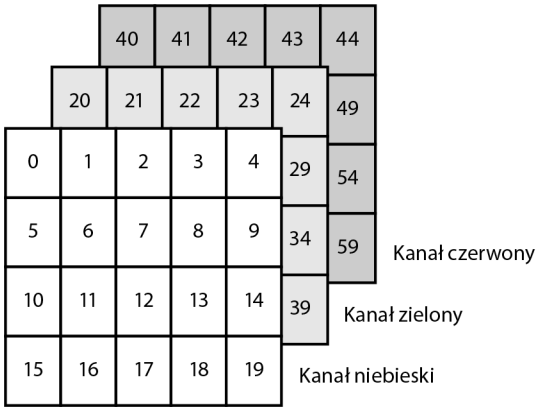
OpenCV to najpopularniejsza na świecie biblioteka z obszaru widzenia komputerowego. **Widzenie komputerowe** to podkategoria uczenia głębokiego, która umożliwia maszynom widzenie, rozpoznawanie i przetwarzanie obrazów w sposób podobny do ludzi. Projekt OpenCV rozpoczął się jako inicjatywa organizacji Intel Research w 1999 roku, a obecnie znajduje się pod pieczą OpenCV Foundation — organizacji pożytku publicznego, która udostępnia to oprogramowanie za darmo.

Biblioteka OpenCV jest napisana w języku C++, ale ma interfejsy dla innych języków, w tym dla Pythona i Javy. Chociaż została stworzona przede wszystkim z myślą o aplikacjach czasu rzeczywistego do rozpoznawania obrazów, zawiera również przydatne narzędzia do obróbki obrazów podobne do tych, jakie można znaleźć w Python Imaging Library. Gdy powstawała ta książka, najnowszą wersją biblioteki było OpenCV 4.1.

Biblioteka wymaga zainstalowania zarówno pakietu NumPy, jak i SciPy do wykonywania obliczeń numerycznych i naukowych w Pythonie. OpenCV traktuje obrazy jako trójwymiarowe tablice NumPy (rysunek 1.4). Pozwala to na łatwe współdziałanie z innymi naukowymi bibliotekami Pythona.

OpenCV przechowuje własności obrazów jako wiersze, kolumny i kanały kolorów. Wymiary tablicy stanowiącej obraz przedstawiony na rysunku 1.4 można opisać trójelementową krotką (ang. *tuple*) o postaci `(4, 5, 3)` — cztery wiersze, pięć kolumn i trzy kanały kolorów. Pojedynczy piksel obrazu jest reprezentowany przez trzy wartości zapisane w komórkach tablicy (np. 0-20-40 lub 19-39-59). Każda z tych wartości oznacza intensywność koloru w konkretnym kanale.

Ponieważ wiele projektów przedstawionych w tej książce wymaga bibliotek naukowych, takich jak NumPy czy *matplotlib*, to dobry moment, aby je zainstalować. Istnieją na to różne sposoby. Jednym z nich jest użycie SpiPy — biblioteki Pythona o ogólnodostępnym kodzie źródłowym, która jest wykorzystywana do obliczeń naukowych i technicznych (zob. *https://scipy.org/index.html*).



Rysunek 1.4. Graficzna reprezentacja obrazu jako tablicy z trzema kanałami kolorów

Jeżeli planujesz zajmować się analizą danych i ich graficznym przedstawianiem we własnym czasie, możesz zainstalować bezpłatną dystrybucję Pythona, taką jak Anaconda lub podstawowa wersja Enthought Deployment Manager (EDM), które działają na systemach operacyjnych Windows, Linux i macOS. Dzięki takim dystrybucjom nie musisz tracić czasu na znajdowanie i instalowanie odpowiednich wersji wszystkich wymaganych bibliotek naukowych, takich jak NumPy, SciPy i wiele innych. Listę dostępnych dystrybucji wraz z linkami do odpowiednich stron znajdziesz pod adresem *https://scipy.org/install.html*.

Instalacja NumPy i innych pakietów naukowych za pomocą narzędzia pip

Jeżeli chcesz zainstalować moduły bezpośrednio, użyj narzędzia `pip`, czyli *systemu zarządzania pakietami*, który ułatwia instalowanie oprogramowania w języku Python (zob. *https://docs.python.org/3/installing*). W przypadku systemu operacyjnego Windows i macOS wersje Pythona 3.4 i wyższe będą domyślnie wyposażone w ten instalator. Użytkownicy Linuksa mogą być zmuszeni do zainstalowania narzędzia `pip` oddzielnie. Aby je zainstalować lub zaktualizować, zapoznaj się z instrukcjami znajdującymi się pod adresem *https://pip.pypa.io/en/stable/installing* lub poszukaj w internecie wskazówek dotyczących instalacji narzędzia na swoim systemie operacyjnym.

Ja do instalowania pakietów naukowych użyłem narzędzia `pip` i korzystałem z instrukcji znajdujących się pod adresem *https://scipy.org/install.html*. Biblioteka *matplotlib* wymaga wielu zależności. Je również trzeba zainstalować. Na systemie operacyjnym Windows uruchom w interpreterze poleceń PowerShell następujące polecenie, specyficzne dla Pythona w wersji 3.

```
$ python -m pip install --user numpy scipy matplotlib ipython jupyter pandas
➥sympy nose
```

PowerShella uruchom z katalogu zawierającego aktualnie zainstalowaną wersję Pythona, np. klikając tam prawym przyciskiem myszy przy wciśniętym klawiszu *Shift*. Jeżeli na swoim urządzeniu masz zainstalowane wersje 2 i 3 Pythona, użyj nazwy `python3` zamiast `python`.

Aby upewnić się, że biblioteka NumPy została zainstalowana i jest dostępna do celów korzystania z biblioteki OpenCV, otwórz powłokę Pythona i wpisz następującą instrukcję:

```
>>> import numpy
```

Jeżeli nie pojawia się żaden błąd, to znaczy, że możesz przejść do instalacji OpenCV.

Instalacja OpenCV za pomocą narzędzia pip

Instrukcję instalacji biblioteki OpenCV możesz znaleźć pod adresem *https://pypi.org/project/opencv-python*. Aby zainstalować ją w standardowym środowisku używanym na komputerach osobistych (czyli na systemach Windows, macOS i prawie wszystkich dystrybucjach systemu GNU/Linux), wpisz poniższe polecenie w oknie PowerShella lub w terminalu:

```
pip install opencv-contrib-python
```

lub

```
python -m pip install opencv-contrib-python
```

Jeżeli na urządzeniu masz zainstalowane wiele wersji Pythona (np. 2.7 i 3.7), musisz określić, której z nich chcesz użyć.

```
py -3.7 -m pip install --user opencv-contrib-python
```

Jeżeli używasz Anacondy jako medium, możesz użyć poniższego polecenia:

```
conda install opencv
```

Aby sprawdzić, czy wszystkie moduły zostały poprawnie pobrane, wpisz następującą instrukcję w powłoce:

```
>>> import cv2
```

Jeżeli nie pojawiły się żadne błędy, możemy ruszać naprzód! W przeciwnym razie zapoznaj się z poradami dotyczącymi rozwiązywania najczęstszych problemów, które znajdziesz pod adresem *https://pypi.org/project/opencv-python*.

Kod programu

Program *bayes.py*, który napiszemy w tym rozdziale, to symulacja poszukiwań zaginionego żeglarza w trzech sąsiadujących ze sobą obszarach. Program będzie wyświetlał mapę, drukował menu z możliwymi wyborami użytkownika, losowo wybierał lokalizację zaginionego i albo ją pokazywał, jeżeli poszukiwania się powiodą, albo przeprowadzał bayesowską aktualizację prawdopodobieństwa w każdym obszarze. Kod wraz z mapą (*cape_python.png*) możesz pobrać pod adresem *https://ftp.helion.pl/przyklady/pytzyc.zip*.

Import modułów

Listing 1.1 przedstawia początek programu *bayes.py*. Zaczniemy od zaimportowania niezbędnych modułów i utworzenia kilku stałych. Wyjaśnianiem, co robią te moduły, zajmiemy się w momencie ich pierwszego użycia w kodzie.

Listing 1.1. Import modułów i utworzenie stałych wykorzystywanych w programie bayes.py (bayes.py, część 1)

```
import sys
import random
import itertools
import numpy as np
import cv2 as cv

MAP_FILE = 'cape_python.png'

SA1_CORNERS = (130, 265, 180, 315) # (LG X, LG Y, PD X, PD Y)
SA2_CORNERS = (80, 255, 130, 305)  # (LG X, LG Y, PD X, PD Y)
SA3_CORNERS = (105, 205, 155, 255) # (LG X, LG Y, PD X, PD Y)
```

Gdy importujemy moduły do programu, najlepiej robić to w następującej kolejności: najpierw moduły biblioteki standardowej Pythona, następnie moduły stron trzecich, a na końcu moduły zdefiniowane przez użytkownika. Moduł `sys` zawiera polecenia dla systemu operacyjnego, takie jak polecenie wyjścia z programu. Moduł `random` pozwala generować pseudolosowe liczby. Moduł `itertools` ułatwia korzystanie z pętli. Kolejne dwa importy: `numpy` i `cv2` odnoszą się do bibliotek NumPy i OpenCV. Możemy od razu przypisać im krótsze nazwy (`np` i `cv`), aby zaoszczędzić sobie później nieco pisania.

Następnie zdefiniuj stałe. Zgodnie z wytycznymi PEP8 dotyczącymi konwencji zapisu kodu w języku Python (*https://www.python.org/dev/peps/pep-0008*) nazwy stałych należy zapisywać wielkimi literami. Co prawda nie czyni to tych zmiennych prawdziwie niezmiennymi, ale informuje innych programistów, że nie powinni zmieniać ich wartości.

Mapa, której użyjemy do zobrazowania fikcyjnego Przylądka Python, znajduje się w pliku o nazwie *cape_python.png* (rysunek 1.5). Przypisz ten obraz do stałej o nazwie `MAP_FILE`.

Rysunek 1.5. Mapa Przylądka Python w skali szarości (cape_python.png)

Na mapę zostaną nałożone obszary poszukiwań w kształcie prostokątów. OpenCV definiuje prostokąty poprzez współrzędne przeciwległych rogów. W związku z tym przypiszemy cztery liczby całkowite w postaci krotki do jednej zmiennej. Kolejność podawania współrzędnych jest następująca: współrzędna *x* lewego górnego rogu, współrzędna *y* lewego górnego rogu, współrzędna *x* prawego dolnego rogu i współrzędna *y* prawego dolnego rogu. Przedrostek `SA` w nazwie zmiennej odnosi się do słów *search area* (tj. obszar poszukiwań).

Definicja klasy Search

Klasa to typ danych charakterystyczny dla programowania obiektowego, które jest podejściem alternatywnym względem programowania proceduralnego (zwanego też funkcyjnym). Programowanie obiektowe jest szczególnie przydatne w przypadku dużych i złożonych programów, gdyż wynikowy kod jest łatwiejszy do aktualizacji, utrzymania i ponownego wykorzystania, a ilość powtórzonego kodu jest mniejsza. Programowanie obiektowe koncentruje się na strukturach danych zwanych **obiektami**, które składają się z danych, metod oraz interakcji między nimi. Takie podejście sprawdza się dobrze przy tworzeniu gier, które zazwyczaj operują na interaktywnych obiektach, takich jak statki kosmiczne czy asteroidy.

Klasa jest wzorcem, na podstawie którego można stworzyć wiele obiektów. Na przykład moglibyśmy mieć klasę, która buduje pancerniki w grze rozgrywającej się w czasach II wojny światowej. Każdy statek dziedziczyłby pewne spójne cechy, takie jak tonaż, prędkość rejsową, poziom paliwa, doznawane uszkodzenia, uzbrojenie i tak dalej. Można by również nadać każdemu pancernikowi unikatowe własności, takie jak nazwa. Po utworzeniu nowego egzemplarza, zwanego też **instancją**, indywidualne cechy każdego statku mogłyby zacząć się różnicować w zależności od tego, ile paliwa spala, jakich uszkodzeń doznaje, ile amunicji zużywa i tak dalej.

W programie *bayes.py* użyjemy klasy, która będzie wzorcem do tworzenia misji poszukiwawczo-ratunkowych, w ramach których dostępne będą trzy obszary poszukiwań. Listing 1.2 pokazuje klasę `Search` (poszukiwanie), która będzie służyć za model gry.

Listing 1.2. Początek definicji klasy Search oraz metody __init__() (bayes.py, część II)

```
class Search():
    """
    Bayesowska gra do symulacji misji poszukiwawczo-ratunkowych
    z trzema obszarami poszukiwań.
    """

    def __init__(self, name):
        self.name = name
        self.img = cv.imread(MAP_FILE, cv.IMREAD_COLOR) ❶
        if self.img is None:
            print('Nie można załadować pliku z mapą {}.'.format(MAP_FILE),
                  file=sys.stderr)
            sys.exit(1)

        self.area_actual = 0 ❷
        # Lokalne współrzędne w obrębie obszaru poszukiwań
        self.sailor_actual = [0, 0]

        self.sa1 = self.img[SA1_CORNERS[1] : SA1_CORNERS[3], ❸
                            SA1_CORNERS[0] : SA1_CORNERS[2]]

        self.sa2 = self.img[SA2_CORNERS[1] : SA2_CORNERS[3],
                            SA2_CORNERS[0] : SA2_CORNERS[2]]

        self.sa3 = self.img[SA3_CORNERS[1] : SA3_CORNERS[3],
                            SA3_CORNERS[0] : SA3_CORNERS[2]]

      self.p1 = 0.2 ❹
      self.p2 = 0.5
      self.p3 = 0.3

      self.sep1 = 0
      self.sep2 = 0
      self.sep3 = 0
```

Zacznij od zdefiniowania klasy o nazwie `Search`. Zgodnie z wytycznymi PEP8 nazwa klasy powinna zaczynać się wielką literą.

Później zdefiniuj metodę, która ustala początkowe wartości atrybutów obiektu. W programowaniu obiektowym **atrybut** to nazwa wartości powiązanej z obiektem. Gdyby obiektem była osoba, atrybutami mogłyby być waga lub kolor oczu. **Metody** również są atrybutami — atrybutami w postaci funkcji. Otrzymują one referencję do powiązanej z nimi instancji. Metoda `__init__()` jest specjalną wbudowaną funkcją, którą Python wywołuje automatycznie przy tworzeniu nowego obiektu. Nadaje ona początkowe wartości atrybutom nowo utworzonego egzemplarza klasy. W tym przypadku przekazujemy do metody inicjalizacyjnej dwa argumenty: `self` oraz nazwę, jaką chcemy nadać obiektowi.

Parametr `self` to odniesienie do tej instancji klasy, która jest w trakcie tworzenia lub z której została wywołana dana metoda. W języku technicznym ten obiekt jest nazywany **instancją kontekstową**. Na przykład, jeżeli tworzymy pancernik o nazwie *Missouri*, dla tego obiektu `self` oznacza właśnie obiekt `missouri`. Z tego obiektu możemy wywołać metodę, taką jak strzał z dział okrętowych dużego kalibru, stosując notację kropkową: `missouri.fire_big_guns()`. Widoczność atrybutów każdego obiektu jest ograniczona do tego obiektu. Nie są one widoczne z innych obiektów. W ten sposób uszkodzenia, jakich doznaje jeden statek, nie są współdzielone z resztą floty.

Dobrą praktyką jest wymienienie wszystkich atrybutów obiektu i przypisanie im początkowych wartości na początku metody `__init__()`. W ten sposób użytkownicy mogą od razu zobaczyć wszystkie najważniejsze atrybuty obiektu, które będą wykorzystywane w różnych metodach. Dzięki temu kod jest bardziej czytelny i łatwiejszy w utrzymaniu. Na listingu 1.2 atrybuty obiektu możesz rozpoznać po tym, że należą do obiektu `self`. Przykładem takiego atrybutu jest `self.name`.

Atrybuty przypisane do `self` zachowują się jak zmienne globalne w programowaniu proceduralnym. Metody klasy mogą uzyskać do nich bezpośredni dostęp bez potrzeby przekazywania ich przez argument. Ponieważ widoczność tych atrybutów jest ograniczona do danej *klasy*, ich stosowanie nie jest złą praktyką, tak jak w przypadku prawdziwych zmiennych globalnych, które zostają zdefiniowane w zakresie globalnym i mogą być modyfikowane w zakresie lokalnym poszczególnych funkcji.

Przypisz do atrybutu `self.img` wynik działania funkcji `imread()` pakietu OpenCV ❶. Pierwszym argumentem niech będzie nazwa pliku mapy zapisana w zmiennej `MAP_FILE`. Sama mapa jest obrazem w skali szarości, ale w czasie poszukiwań będziemy chcieli użyć na niej kolorowych oznaczeń. Dlatego też jako drugi argument przekaż flagę `cv.IMREAD_COLOR`, aby wgrać mapę jako obraz w kolorze. Dzięki temu będziemy mogli później używać trzech kanałów kolorów (B, G, R).

Jeżeli plik z obrazem nie istnieje (lub jeżeli użytkownik wprowadził złą nazwę pliku), biblioteka OpenCV zgłosi niezbyt jasny błąd (tj. `'NoneType' object is not subscriptable`). Aby temu zapobiec, użyj instrukcji warunkowej do sprawdzenia, czy wartość atrybutu `self.img` nie jest przypadkiem równa `None`. Gdyby tak się stało, wydrukuj komunikat o błędzie, a następnie użyj modułu `sys`, aby wyjść z programu. Przekazanie do funkcji `exit` kodu wyjścia równego 1 informuje

o tym, że program zakończył się błędem. Natomiast ustawienie argumentu `file=↪stderr` spowoduje użycie standardowego, czerwonego koloru do wyświetlenia tego komunikatu o błędzie w oknie pythonowego interpretera, choć niekoniecznie w oknach innych programów, takich jak PowerShell.

Następnie utwórz dwa atrybuty, które będą określać rzeczywiste położenie rozbitka. Pierwszy będzie przechowywać numer obszaru poszukiwań ❷, a drugi — dokładną lokalizację żeglarza w formie współrzędnych *x* i *y*. Na razie przypisz im tymczasowe wartości. Później zdefiniujesz metodę, która losowo wybierze wartości docelowe. Zauważ, że do określenia położenia korzystamy z listy, ponieważ będziemy potrzebować modyfikowalnej kolekcji.

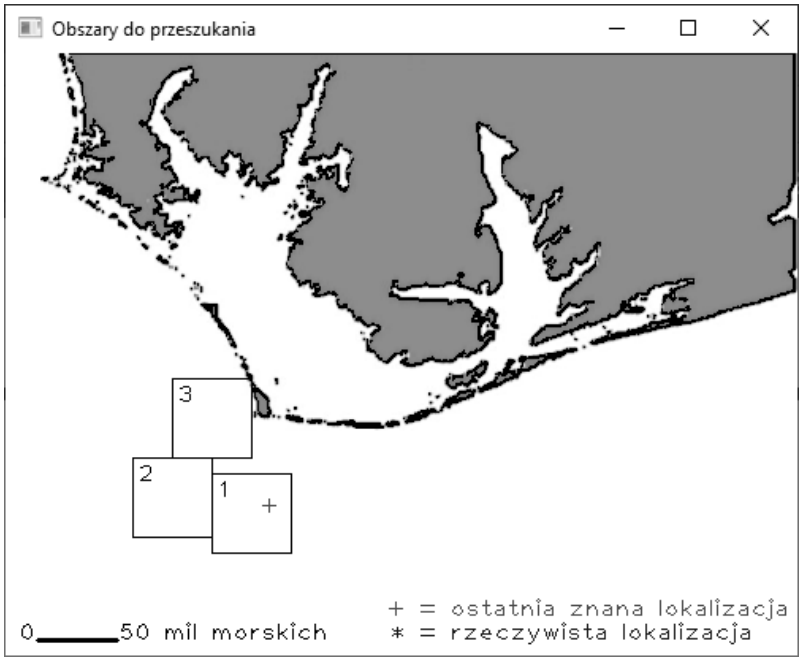
Obraz mapy jest wgrywany w postaci tablicy. **Tablica** to kolekcja o stałym rozmiarze przechowująca obiekty tego samego typu. Tablice jako kontenery na dane są wydajne z punktu widzenia zajętości pamięci oraz logiki adresowania stosowanej w komputerach, a dodatkowo obliczenia arytmetyczne na nich są szybkie. Jedną z koncepcji, które czynią bibliotekę NumPy wyjątkowo potężną, jest **wektoryzacja**, która zastępuje operacje w pętlach bardziej wydajnymi wyrażeniami tablicowymi. Zasadniczo chodzi o to, że operacje są przeprowadzane na całych tablicach jednocześnie, a nie na ich poszczególnych elementach. Biblioteka NumPy deleguje wykonanie iteracji do wydajnych funkcji języka C lub Fortran, które są szybsze niż standardowe techniki dostępne w języku Python.

Aby móc wykonywać operacje na lokalnych współrzędnych *w obrębie obszaru poszukiwań*, możemy wydzielić część tablicy do innej ❸. Zauważ, że wykorzystujemy do tego indeksy. Najpierw utwórz zakres wartości od współrzędnej *y* lewego górnego rogu do współrzędnej *y* prawego dolnego rogu, a następnie między współrzędnymi *x* lewego górnego rogu i prawego dolnego rogu. Potrzeba nieco czasu, aby przyzwyczaić się do tej kolejności w NumPy. Większość z nas przywykła do tego, że *x* znajduje się przed *y* we współrzędnych kartezjańskich.

Powtórz procedurę dla kolejnych dwóch obszarów poszukiwań, a następnie określ prawdopodobieństwo znalezienia zaginionego żeglarza w każdym obszarze przed rozpoczęciem poszukiwań ❹. W prawdziwych warunkach te dane pochodziłyby z systemu SAROPS. Oczywiście `p1` to prawdopodobieństwo dla obszaru pierwszego, `p2` — dla drugiego i tak dalej. Na koniec przypisz początkowe wartości do atrybutów reprezentujących prawdopodobieństwo skuteczności poszukiwań (`sep1` itd.).

Rysowanie mapy

W klasie `Search`, korzystając z funkcjonalności pochodzących z biblioteki OpenCV, zdefiniuj metodę, która będzie wyświetlać mapę regionu. Obraz będzie pokazywać obszary poszukiwań, pasek skali oraz ostatnią znaną pozycję zaginionego (rysunek 1.6).



Rysunek 1.6. Początkowy wygląd ekranu gry bayes.py z mapą regionu

Listing 1.3 pokazuje metodę `draw_map()`, która wyświetla początkowy ekran.

Listing 1.3. Definicja metody wyświetlającej mapę regionu (bayes.py, część III)

```
def draw_map(self, last_known):
    """
    Wyświetla mapę regionu wraz ze skalą, ostatnią znaną
    lokalizacją oraz obszarami poszukiwań.
    """
    cv.line(self.img, (20, 370), (70, 370), (0, 0, 0), 2)
    cv.putText(self.img, '0', (8, 370), cv.FONT_HERSHEY_PLAIN, 1, (0, 0, 0))
    cv.putText(self.img, '50 mil morskich', (71, 370),
               cv.FONT_HERSHEY_PLAIN, 1, (0, 0, 0))
    cv.rectangle(self.img, (SA1_CORNERS[0], SA1_CORNERS[1]), ❶
                 (SA1_CORNERS[2], SA1_CORNERS[3]), (0, 0, 0), 1)
    cv.putText(self.img, '1', (SA1_CORNERS[0] + 3, SA1_CORNERS[1] + 15),
               cv.FONT_HERSHEY_PLAIN, 1, 0)
    cv.rectangle(self.img, (SA2_CORNERS[0], SA2_CORNERS[1]),
                 (SA2_CORNERS[2], SA2_CORNERS[3]), (0, 0, 0), 1)
    cv.putText(self.img, '2', (SA2_CORNERS[0] + 3, SA2_CORNERS[1] + 15),
               cv.FONT_HERSHEY_PLAIN, 1, 0)
    cv.rectangle(self.img, (SA3_CORNERS[0], SA3_CORNERS[1]),
                 (SA3_CORNERS[2], SA3_CORNERS[3]), (0, 0, 0), 1)
    cv.putText(self.img, '3', (SA3_CORNERS[0] + 3, SA3_CORNERS[1] + 15),
               cv.FONT_HERSHEY_PLAIN, 1, 0)
    cv.putText(self.img, '+', last_known, ❷
               cv.FONT_HERSHEY_PLAIN, 1, (0, 0, 255))
```

```
        cv.putText(self.img, '+ = ostatnia znana lokalizacja', (240, 355),
                   cv.FONT_HERSHEY_PLAIN, 1, (0, 0, 255))
        cv.putText(self.img, '* = rzeczywista lokalizacja', (242, 370),
                   cv.FONT_HERSHEY_PLAIN, 1, (255, 0, 0))

        cv.imshow('Obszary do przeszukania', self.img) ❸
        cv.moveWindow('Obszary do przeszukania', 750, 10)
        cv.waitKey(500)
```

Zdefiniuj metodę `draw_map()`, podając jako parametry `self` oraz współrzędne ostatniej znanej lokalizacji zaginionego (`last_known`). Następnie użyj funkcji `line()` z biblioteki OpenCV, aby narysować pasek skali. Do funkcji przekaż następujące argumenty: obraz mapy, dwie krotki ze współrzędnymi (*x* i *y*) początku i końca, krotkę definiującą kolor linii oraz liczbę oznaczającą grubość linii.

Następnie użyj funkcji `putText()` do opisania paska skali. W tym przypadku jako argumenty przekaż: atrybut z obrazem mapy, tekst do wyświetlenia oraz krotkę ze współrzędną lewego górnego rogu tekstu. Dodatkowo musisz określić nazwę czcionki, skalę tekstu oraz krotkę definiującą kolor.

Na kolejnym etapie narysuj prostokąt pierwszego obszaru poszukiwań ❶. Jak wcześniej, do metody przekaż referencję do mapy, następnie zmienne reprezentujące cztery rogi ramki, a na koniec krotkę określającą kolor prostokąta i liczbę reprezentującą grubość linii. Użyj funkcji `putText()` jeszcze raz, aby umieścić numer obszaru w jego lewym górnym rogu. Powtórz te czynności dla obszarów 2 i 3.

Użyj funkcji `putText()`, aby umieścić znak + w miejscu ostatniej znanej pozycji zaginionego ❷. Zauważ, że znak jest czerwony, chociaż krotka definiująca kolory wygląda tak: `(0, 0, 255)`, a nie tak: `(255, 0, 0)`. Dzieje się tak dlatego, że biblioteka OpenCV używa formatu BGR: *B* — *blue* (niebieski), *G* — *green* (zielony) i *R* — *red* (czerwony), zamiast powszechnie stosowanego formatu RGB.

Później umieść tekst legendy, która opisuje symbole określające ostatnią znaną oraz rzeczywistą lokalizację zaginionego. Tę ostatnią należy wyświetlić dopiero wtedy, gdy gracz znajdzie rozbitka. Rzeczywistą lokalizację oznacz kolorem niebieskim.

Na zakończenie wyświetl mapę regionu, używając funkcji `imshow()` z biblioteki OpenCV ❸. Jako argumenty przekaż do niej tytuł okna i obraz.

Aby uniknąć zasłonięcia okna z mapą przez okno interpretera lub odwrotnie, przesuń mapę do prawego górnego rogu monitora (może istnieć potrzeba dostosowania współrzędnych do potrzeb). Użyj funkcji `moveWindow()` biblioteki OpenCV i podaj nazwę okna (tj. *Obszary do przeszukania*) wraz ze współrzędnymi jego lewego górnego rogu.

Na koniec użyj funkcji `waitKey()`, która wprowadza opóźnienie *n* milisekund przy renderowaniu obrazów w oknach. Przekaż jej wartość `500`, czyli pięciuset milisekund. Wynikiem tego powinno być pojawienie się menu gry pół sekundy po wyświetleniu mapy regionu.

Wybór końcowej lokalizacji zaginionego

Listing 1.4 pokazuje definicję metody, która losowo wybiera rzeczywistą lokalizację zaginionego żeglarza. Dla wygody współrzędne zostają najpierw określone w ramach podtablicy reprezentującej pojedynczy obszar poszukiwań, a następnie zostają przekształcone na globalne współrzędne na obrazie pokazującym całą mapę. To podejście działa, ponieważ wszystkie obszary poszukiwań są tego samego rozmiaru i kształtu, więc możemy używać tych samych współrzędnych wewnętrznych.

Listing 1.4. Definicja metody, która losowo wybiera rzeczywistą lokalizację zaginionego żeglarza (bayes.py, część IV)

```
    def sailor_final_location(self, num_search_areas):
        """Zwraca współrzędne x i y rzeczywistej lokalizacji zaginionego."""
        # Znajduje współrzędne żeglarza względem podtablicy obszaru poszukiwań.
        self.sailor_actual[0] = np.random.choice(self.sa1.shape[1], 1)
        self.sailor_actual[1] = np.random.choice(self.sa1.shape[0], 1)

        area = int(random.triangular(1, num_search_areas + 1)) ❶

        if area == 1:
            x = self.sailor_actual[0] + SA1_CORNERS[0]
            y = self.sailor_actual[1] + SA1_CORNERS[1]
            self.area_actual = 1 ❷
        elif area == 2:
            x = self.sailor_actual[0] + SA2_CORNERS[0]
            y = self.sailor_actual[1] + SA2_CORNERS[1]
            self.area_actual = 2
        elif area == 3:
            x = self.sailor_actual[0] + SA3_CORNERS[0]
            y = self.sailor_actual[1] + SA3_CORNERS[1]
            self.area_actual = 3
        return x, y
```

Zdefiniuj metodę `sailor_final_location()`, która przyjmuje dwa parametry: `self` oraz liczbę obszarów poszukiwań. Pierwszą współrzędną (x) na liście `self.sailor_actual` uzyskamy, korzystając z funkcji `random.choice()` biblioteki NumPy. Dzięki temu wybierzesz losową wartość z zakresu uzyskanego na podstawie podtablicy obszaru pierwszego. Pamiętaj, że obszary poszukiwań to tablice NumPy przekopiowane z tablicy przechowującej obraz mapy regionu. Ponieważ obszary (podtablice) są tego samego rozmiaru, uzyskane współrzędne będą pasowały do każdego z trzech prostokątów.

Współrzędne można pobrać z tablicy, odwołując się do atrybutu `shape`. Można również użyć funkcji `np.shape()`, tak jak widać poniżej:

```
>>> print(np.shape(self.sa1))
(50, 50, 3)
```

Atrybut `shape` jest krotką zawierającą tyle elementów, ile wymiarów znajduje się w tablicy. Poszczególne wartości reprezentują liczbę elementów w danym wymiarze. Pamiętajmy, że dla tablicy OpenCV kolejność elementów w krotce jest następująca: wiersze, kolumny, a potem kanały kolorów.

Każdy z istniejących obszarów poszukiwań jest trójwymiarową tablicą o wielkości 50×50 pikseli. Wartość wewnętrznych współrzędnych obszaru (*x* i *y*) będzie zatem znajdować się w przedziale od 0 do 49. Wybranie indeksu `[0]` krotki jako pierwszego argumentu funkcji `random.choice()` oznacza, że użyjemy liczby wierszy. Drugi argument o wartości 1 sprawi, że z dostępnego zakresu wybierzemy jeden element. Analogicznie indeks `[1]` oznacza wybranie kolumn.

Współrzędne generowane przez `random.choice()` będą się wahać w granicach 0 – 49. Aby przetłumaczyć te współrzędne na współrzędne mapy regionu, musimy najpierw określić, o który obszar poszukiwań chodzi ❶. Zrób to za pomocą modułu `random`, który zaimportowaliśmy na początku programu. Zgodnie z danymi z systemu SAROPS żeglarz najprawdopodobniej znajduje się w obszarze drugim. Kolejnym pod względem prawdopodobieństwa jest obszar trzeci. Ponieważ te początkowe wartości prawdopodobieństwa to tylko przypuszczenia, które nie odzwierciedlają bezpośrednio rzeczywistości, do wyboru obszaru użyjemy **rozkładu trójkątnego**. Argumentami funkcji `random.triangular()` są dolna i górna granica (`low` i `high`). Jeżeli nie podamy ostatniego argumentu (`mode`), jego domyślną wartością będzie punkt środkowy między wartościami skrajnymi. Będzie to zgodne z danymi systemu SAROPS, gdyż najczęściej będzie wybierany obszar drugi.

Zwróć uwagę na to, że w obrębie metody `sailor_final_location()` używamy zmiennej lokalnej `area`, a nie atrybutu `self.area`, ponieważ nie ma potrzeby współdzielić tej wartości z innymi metodami.

Aby określić położenie żeglarza na mapie całego regionu, do lokalnego położenia w obrębie obszaru musisz dodać współrzędne narożnika tego obszaru. Przekształca to lokalne współrzędne wewnątrz obszaru poszukiwań na globalne współrzędne na mapie regionu. Chcemy również zachować informacje na temat tego, w którym obszarze znajduje się zaginiony, dlatego zaktualizuj atrybut `self.area_actual` ❷.

Powtórz te kroki dla drugiego i trzeciego obszaru, a następnie zwróć współrzędne.

UWAGA *W prawdziwych warunkach żeglarz mógłby z wolna dryfować na wodzie, więc prawdopodobieństwo znalezienia go w obszarze trzecim rosłoby z każdym poszukiwaniem. Postanowiłem jednak zostać przy niezmiennej pozycji, aby działanie twierdzenia Bayesa było możliwie najbardziej zrozumiałe. Dlatego ten scenariusz przypomina bardziej poszukiwanie zatopionej łodzi podwodnej.*

Liczenie skuteczności i realizacja poszukiwań

W prawdziwych warunkach pogoda i problemy techniczne mogą obniżyć skuteczność poszukiwań. W związku z tym dla każdej akcji poszukiwawczej będziemy generować listę wszystkich możliwych lokalizacji w obrębie obszaru poszukiwań,

tasować tę listę, a następnie pobierać pewną liczbę próbek określoną na podstawie skuteczności poszukiwań. Ponieważ prawdopodobieństwo skuteczności poszukiwań nigdy nie osiąga wartości 1.0, gdybyśmy pobierali wartości z początku lub końca listy bez tasowania, nigdy nie byłoby możliwe dojście do współrzędnych zlokalizowanych po przeciwnej stronie.

Listing 1.5 pokazuje dalszy ciąg klasy `Search`. Teraz zdefiniujemy dwie metody: jedną, która losowo oblicza skuteczność poszukiwań, oraz drugą, która odpowiada za przeprowadzenie poszukiwania.

Listing 1.5. Definicja metod, które losowo określają skuteczność poszukiwań oraz przeprowadzają poszukiwanie (bayes.py, część V)

```
def calc_search_effectiveness(self):
    """
    Wyznacza wartość dziesiętną reprezentującą skuteczność
    poszukiwań dla każdego obszaru.
    """
    self.sep1 = random.uniform(0.2, 0.9)
    self.sep2 = random.uniform(0.2, 0.9)
    self.sep3 = random.uniform(0.2, 0.9)

def conduct_search(self, area_num, area_array, effectiveness_prob): ❶
    """Zwraca wynik poszukiwań oraz listę przeszukanych współrzędnych."""
    local_y_range = range(area_array.shape[0])
    local_x_range = range(area_array.shape[1])
    coords = list(itertools.product(local_x_range, local_y_range)) ❷
    random.shuffle(coords)
    coords = coords[:int((len(coords) * effectiveness_prob))]
    loc_actual = (self.sailor_actual[0], self.sailor_actual[1]) ❸
    if area_num == self.area_actual and loc_actual in coords:
        return 'Znaleziono w obszarze nr {}.'.format(area_num), coords
    else:
        return 'Nie znaleziono.', coords
```

Rozpocznij od zdefiniowania metody losującej wartości skuteczności poszukiwań. Jedynym parametrem, jakiego potrzebujesz, jest `self`. Dla każdego z atrybutów określających skuteczność poszukiwań (`self.sep1`, `self.sep2` i `self.sep3`) losowo wybierasz wartość z przedziału od 0,2 do 0,9. To arbitralne wartości, które oznaczają, że zawsze przeszukamy co najmniej 20% obszaru, ale nigdy nie przekroczymy 90%.

Moglibyśmy założyć, że wartości skuteczności poszukiwań dla wszystkich trzech obszarów nie są od siebie niezależne. Z jednej strony mgła mogłaby obniżyć skuteczność poszukiwań we wszystkich trzech obszarach. Z drugiej — niektóre helikoptery mogą być wyposażone w urządzenia do widzenia w podczerwieni i mogłyby się sprawdzać lepiej. W każdym razie, kiedy te wartości są od siebie niezależne, tak jak jest teraz, symulacja jest bardziej dynamiczna.

Następnie zdefiniuj metodę odzwierciedlającą samo poszukiwanie ❶. Niezbędnymi parametrami będą: sam obiekt (`self`), numer obszaru (wybrany przez użytkownika), podtablica danego obszaru oraz losowo wybrana skuteczność poszukiwań.

Najpierw musimy utworzyć listę wszystkich współrzędnych w obrębie danego obszaru. Wygeneruj możliwe wartości współrzędnej *y* i zapisz je do zmiennej `local_y_range`. Zakres możesz utworzyć na podstawie pierwszego indeksu z krotki zwracanej przez atrybut `shape` tablicy. Indeks ten reprezentuje wiersze. Powtórzmy to samo dla współrzędnych *x* i zapiszmy zakres w zmiennej `local_x_range`.

Aby wygenerować listę wszystkich współrzędnych w obszarze poszukiwań, użyjemy modułu `itertools` ❷. Udostępnia on grupę funkcji ze standardowej biblioteki Pythona, które tworzą iteratory pozwalające na wydajniejsze korzystanie z pętli. Funkcja `product()` zwraca krotki wszystkich permutacji z powtórzeniami dla danej sekwencji. W tym przypadku znajdziemy wszystkie możliwe sposoby połączenia iksów i igreków w obszarze poszukiwań. Aby zobaczyć tę funkcję w akcji, wpisz następujący kod w powłoce:

```
>>> import itertools
>>> x_range = [1, 2, 3]
>>> y_range = [4, 5, 6]
>>> coords = list(itertools.product(x_range, y_range))
>>> coords
[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]
```

Jak widać, lista współrzędnych zawiera każdą możliwą parę elementów z list `x_range` i `y_range`.

Wracając do głównego programu, po uzyskaniu listy współrzędnych musisz je przetasować. Chodzi o to, żebyśmy nie szukali za każdym razem od jednego krańca listy. W następnej linii kodu użyj mechanizmu odwoływania się do podzbioru elementów listy (ang. *index slicing*), aby ograniczyć ich liczbę, biorąc pod uwagę prawdopodobieństwo skuteczności poszukiwań. Na przykład niska skuteczność poszukiwań o wartości 0,3 oznacza, że jedynie jedna trzecia możliwych lokalizacji w obszarze znajdzie się na liście. Ponieważ potem sprawdzamy, czy rzeczywista lokalizacja zaginionego znajduje się na liście, dwie trzecie obszaru nie zostaje przeszukane.

Utwórz zmienną lokalną `loc_actual`, aby przechowywała rzeczywistą lokalizację rozbitka ❸. Następnie użyj instrukcji warunkowej, aby sprawdzić, czy odnaleziono żeglarza. Jeżeli użytkownik wybrał właściwy obszar poszukiwań, a znajdujące się na przetasowanej i skróconej liście współrzędne zawierają lokalizację żeglarza (*x* i *y*), zwracamy ciąg znaków stwierdzający, że żeglarz został odnaleziony, wraz z listą `coords`. W przeciwnym wypadku zwracamy komunikat o tym, że poszukiwania się nie powiodły, również z listą współrzędnych.

Zastosowanie twierdzenia Bayesa i utworzenie menu

Listing 1.6 nadal dotyczy klasy `Search`. Definiujemy w nim jedną metodę i jedną funkcję. Metoda `revise_target_probs()` korzysta z twierdzenia Bayesa do aktualizacji prawdopodobieństwa znalezienia żeglarza w danym obszarze. Funkcja `draw_menu()`, zdefiniowana poza klasą `Search`, wyświetla menu, które będzie służyć za

graficzny interfejs użytkownika (ang. *graphical user interface* — GUI) umożliwiający udział w grze.

Listing 1.6. Zastosowanie twierdzenia Bayesa oraz drukowanie menu w powłoce Pythona (bayes.py, część VI)

```
    def revise_target_probs(self):
        """
        Aktualizuje prawdopodobieństwo dla każdego obszaru
        na podstawie skuteczności poszukiwań.
        """
        denom = self.p1 * (1 - self.sep1) + self.p2 * (1 - self.sep2) \
                + self.p3 * (1 - self.sep3)
        self.p1 = self.p1 * (1 - self.sep1) / denom
        self.p2 = self.p2 * (1 - self.sep2) / denom
        self.p3 = self.p3 * (1 - self.sep3) / denom

def draw_menu(search_num):
    """Drukuje menu z wyborem obszaru do przeszukania."""
    print('\nPodejście nr {}'.format(search_num))
    print(
        """
        Wybierz następne obszary do przeszukania:

        0 - Wyjdź z programu
        1 - Przeszukaj dwukrotnie obszar pierwszy
        2 - Przeszukaj dwukrotnie obszar drugi
        3 - Przeszukaj dwukrotnie obszar trzeci
        4 - Przeszukaj obszary pierwszy i drugi
        5 - Przeszukaj obszary pierwszy i trzeci
        6 - Przeszukaj obszary drugi i trzeci
        7 - Zacznij od początku
        """
    )
```

Zdefiniuj metodę `revise_target_probs()`. Będzie ona służyć do aktualizowania prawdopodobieństwa, że poszukiwany znajduje się w danym obszarze. Jedynym parametrem metody jest `self`.

Dla wygody podziel wzór Bayesa na dwie części. Zacznij od mianownika. Musisz pomnożyć uprzednio założone prawdopodobieństwo przez aktualną wartość skuteczności poszukiwań. Jeżeli chcesz sobie przypomnieć, jak to działa, wróć na chwilę do podrozdziału „Twierdzenie Bayesa".

Gdy mamy obliczony mianownik, możemy dokończyć obliczanie równania Bayesa. W programowaniu obiektowym nie musimy nic zwracać. Po prostu aktualizujemy atrybut bezpośrednio w metodzie, tak jak gdyby była to globalna zmienna w programowaniu proceduralnym.

Następnie w przestrzeni globalnej zdefiniuj funkcję `draw_menu()`, która wyświetla menu. Jedynym parametrem, jaki przyjmuje, jest numer poszukiwania do przeprowadzenia. Ponieważ w ramach tej funkcji nie mamy potrzeby odnosić się do obiektu `self`, nie musimy dołączać tej funkcji do klasy, chociaż nie byłoby to nieprawidłowe.

Zacznij od wydrukowania numeru poszukiwania. Będzie nam to potrzebne do kontrolowania, czy znaleźliśmy żeglarza w co najwyżej trzy dni (podejścia).

Użyjmy potrójnego cudzysłowu wokół argumentu metody `print()`, aby wyświetlić całe menu. Zauważ, że użytkownik ma możliwość przydzielić oba zespoły ratunkowe do jednego obszaru lub podzielić je między dwa obszary.

Definicja funkcji main()

Teraz, gdy zakończyliśmy programowanie klasy `Search`, jesteśmy gotowi, aby zaprząc wszystkie te atrybuty i metody do pracy! Listing 1.7 pokazuje początek definicji funkcji `main()`, używanej do uruchomienia programu.

Listing 1.7. Definicja początku funkcji main(), która służy do uruchomienia programu (bayes.py, część VII)

```
def main():
    app = Search('Cape_Python')
    app.draw_map(last_known=(160, 290))
    sailor_x, sailor_y = app.sailor_final_location(num_search_areas=3)
    print("-" * 65)
    print("\nPoczątkowe oszacowanie prawdopodobieństwa (P):")
    print("P1 = {:.3f}, P2 = {:.3f}, P3 = {:.3f}".format(app.p1, app.p2, app.p3))
    search_num = 1
```

Funkcja `main()` nie wymaga argumentów. Zacznij od utworzenia aplikacji z grą. W zmiennej o nazwie `app` zapiszemy nową instancję klasy `Search`. Jej nazwą (atrybut `name`) będzie `Cape_Python`.

Następnie wywołaj metodę, która wyświetla mapę. Przekaż jej jako argument ostatnią znaną lokalizację żeglarza w formie krotki z dwoma współrzędnymi (x i y). Zwróć uwagę na użycie nazwanego argumentu: `last_known=(160, 290)`. Dzięki temu kod jest bardziej czytelny.

Teraz pobierz rzeczywistą lokalizację żeglarza (x i y), wywołując metodę do tego przeznaczoną z liczbą obszarów poszukiwań jako argumentem. Następnie wydrukuj początkowe oszacowanie prawdopodobieństwa (prawdopodobieństwo wstępne lub *a priori*), które zostało wyliczone przez oficerów Straży Wybrzeża z wykorzystaniem metody Monte Carlo, a nie twierdzenia Bayesa. Na koniec utwórz zmienną `search_num` i przypisz jej wartość 1. Dzięki temu będziemy mogli kontrolować, ile poszukiwań mamy już za sobą.

Interpretacja wyboru opcji z menu

Listing 1.8 pokazuje pętlę `while` w funkcji `main()`. W ramach tej pętli gracz ocenia sytuację i wybiera opcje z menu. Wybory obejmują przeszukanie jednego obszaru dwukrotnie, podział wysiłku między dwa obszary, zrestartowanie gry oraz wyjście z programu. Zauważ, że gracz może przeprowadzić dowolną liczbę poszukiwań aż do znalezienia żeglarza. Ograniczenie do trzech dni nie zostało wpisane w logikę gry na sztywno.

Listing 1.8. Użycie pętli do prowadzenia poszukiwań i interpretacji wyborów gracza (bayes.py, część VIII)

```
while True:
    app.calc_search_effectiveness()
    draw_menu(search_num)
    choice = input("Wybierz opcję: ")

    if choice == "0":
        sys.exit()

    elif choice == "1": ❶
        results_1, coords_1 = app.conduct_search(1, app.sa1, app.sep1)
        results_2, coords_2 = app.conduct_search(1, app.sa1, app.sep1)
        app.sep1 = (len(set(coords_1 + coords_2))) / (len(app.sa1)**2) ❷
        app.sep2 = 0
        app.sep3 = 0

    elif choice == "2":
        results_1, coords_1 = app.conduct_search(2, app.sa2, app.sep2)
        results_2, coords_2 = app.conduct_search(2, app.sa2, app.sep2)
        app.sep1 = 0
        app.sep2 = (len(set(coords_1 + coords_2))) / (len(app.sa2)**2)
        app.sep3 = 0

    elif choice == "3":
        results_1, coords_1 = app.conduct_search(3, app.sa3, app.sep3)
        results_2, coords_2 = app.conduct_search(3, app.sa3, app.sep3)
        app.sep1 = 0
        app.sep2 = 0
        app.sep3 = (len(set(coords_1 + coords_2))) / (len(app.sa3)**2)

    elif choice == "4": ❸
        results_1, coords_1 = app.conduct_search(1, app.sa1, app.sep1)
        results_2, coords_2 = app.conduct_search(2, app.sa2, app.sep2)
        app.sep3 = 0

    elif choice == "5":
        results_1, coords_1 = app.conduct_search(1, app.sa1, app.sep1)
        results_2, coords_2 = app.conduct_search(3, app.sa3, app.sep3)
        app.sep2 = 0

    elif choice == "6":
        results_1, coords_1 = app.conduct_search(2, app.sa2, app.sep2)
        results_2, coords_2 = app.conduct_search(3, app.sa3, app.sep3)
        app.sep1 = 0

    elif choice == "7": ❹
        main()

    else:
        print("\nTo nie jest poprawny wybór.", file=sys.stderr)
        continue
```

Rozpocznij pętlę `while`, która będzie działać, dopóki gracz nie zdecyduje się wyjść z programu. Użyj notacji kropkowej, aby wywołać na utworzonej grze metodę, która oblicza skuteczność poszukiwań. Następnie wywołaj funkcję, która wyświetla menu gry, przekazując jako argument numer poszukiwania. Po zakończeniu przygotowywań poproś użytkownika o dokonanie wyboru za pomocą funkcji `input()`.

Wybór gracza zostanie zinterpretowany przez serię instrukcji warunkowych. Jeżeli gracz wybrał `0`, nastąpi wyjście z gry. Ta operacja wykorzystuje moduł `sys`, zaimportowany na początku programu.

Jeżeli gracz wybierze opcję 1, 2 lub 3, wyznaczy oba zespoły do poszukiwań w obszarze o odpowiednim numerze. W takim przypadku musisz dwukrotnie wywołać metodę `conduct_search()`, aby wygenerować dwa zestawy wyników i współrzędnych ❶. Trudną częścią jest tutaj określenie całkowitego prawdopodobieństwa skuteczności poszukiwań (PSP), ponieważ każde poszukiwanie ma swoją wartość tego parametru. Aby to zrobić, musisz dodać do siebie dwie listy współrzędnych i przekształcić wynik w zbiór (`set`), aby usunąć duplikaty ❷. Pobierz liczbę elementów zbioru, a następnie podziel ją przez liczbę pikseli w obszarze o wielkości 50×50 pikseli. Ponieważ nie szukaliśmy w innych obszarach, ich wartości PSP pozostają na poziomie `0`.

Kod dla drugiego i trzeciego obszaru wygląda analogicznie. Użyj instrukcji `elif`, ponieważ w każdym obrocie pętli można wybrać tylko jedną opcję. To wydajniejsze niż zastosowanie dodatkowych instrukcji `if`, gdyż wszystkie instrukcje `elif`, które znajdują się poniżej prawdziwego warunku, zostaną pominięte.

Jeżeli gracz wybierze opcję 4, 5 lub 6, oznacza to, że chce podzielić zespoły na dwa obszary. W tym przypadku nie ma potrzeby wyliczać całkowitej wartości PSP ❸.

Jeżeli gracz znajdzie żeglarza i zechce zagrać od początku lub po prostu będzie chciał zrestartować grę, wywołamy funkcję `main()` ❹. Spowoduje to rozpoczęcie gry od początku i wyczyszczenie mapy.

Jeżeli gracz dokona nieprawidłowego wyboru, np. wpisze „Bob”, poinformuj go o tym odpowiednim komunikatem, a potem powróć do początku pętli, używając instrukcji `continue`.

Zakończenie pętli i wywołanie funkcji main()

Listing 1.9 pokazuje pozostały kod z wnętrza pętli `while`, koniec definicji funkcji `main()` oraz wywołanie funkcji `main()` w celu uruchomienia programu.

Listing 1.9. Końcowa część funkcji main() i jej wywołanie (bayes.py, część IX)

```
    # Używa twierdzenia Bayesa do zaktualizowania prawdopodobieństwa.
    app.revise_target_probs()

    print("\nPodejście nr {} - wynik 1: {}"
          .format(search_num, results_1), file=sys.stderr)
    print("Podejście nr {} - wynik 2: {}\n"
          .format(search_num, results_2), file=sys.stderr)
```

```
        print("Skuteczność poszukiwań (E) dla podejścia nr {}:"
              .format(search_num))
        print("E1 = {:.3f}, E2 = {:.3f}, E3 = {:.3f}"
              .format(app.sep1, app.sep2, app.sep3))

        if results_1 == 'Nie znaleziono.' and results_2 == 'Nie znaleziono.': ❶
            print("\nNowe oszacowanie prawdopodobieństwa (P) "
                  "dla podejścia nr {}:".format(search_num + 1))
            print("P1 = {:.3f}, P2 = {:.3f}, P3 = {:.3f}"
                  .format(app.p1, app.p2, app.p3))
        else:
            cv.circle(app.img, (sailor_x[0], sailor_y[0]), 3, (255, 0, 0), -1)
            cv.imshow('Obszary do przeszukania', app.img) ❷
            cv.waitKey(1500)
            main()
        search_num += 1

if __name__ == '__main__':
    main()
```

Wywołaj metodę `revise_target_probs()`, aby zastosować twierdzenie Bayesa i ponownie obliczyć prawdopodobieństwo, że żeglarz znajduje się w danym obszarze, uwzględniając wyniki poszukiwań. Następnie wyświetl wyniki oraz wartości PSP w powłoce.

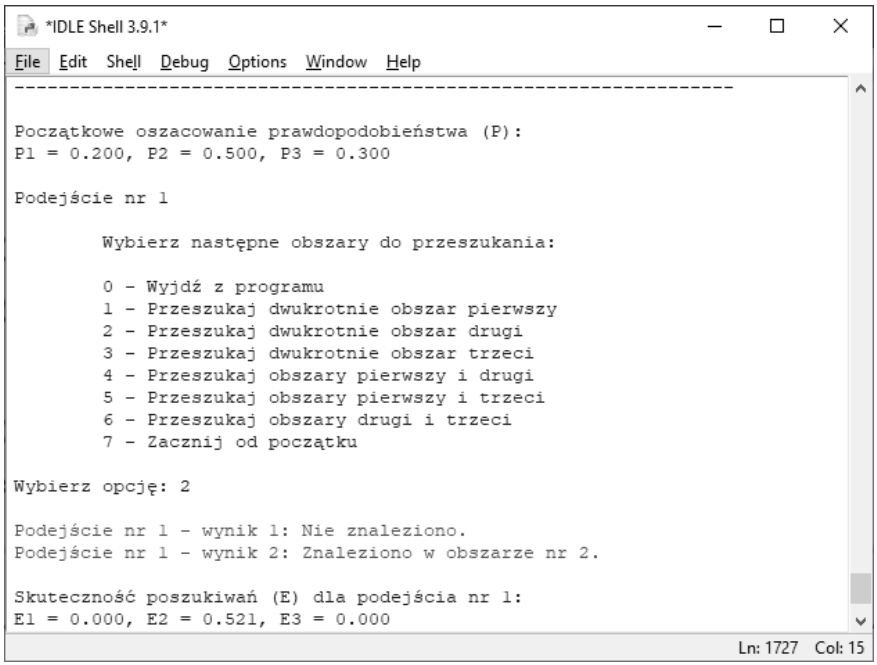
Jeżeli oba poszukiwania były nieskuteczne, wyświetl zaktualizowane oszacowania prawdopodobieństwa, tak aby gracz mógł wziąć je pod uwagę przy wyborze następnego obszaru do przeszukania ❶. W przeciwnym wypadku wyświetl lokalizację żeglarza na mapie. Użyj biblioteki OpenCV, aby wyświetlić niewielkie koło. Jako argumenty przekaż funkcji obraz mapy, lokalizację żeglarza w formie krotki (x, y) jako punkt środkowy, promień w pikselach, kolor i grubość konturu ustawioną na `-1`. Ujemna wartość grubości sprawi, że okrąg zostanie wypełniony wybranym kolorem.

Na koniec funkcji `main()` wyświetl obraz mapy za pomocą kodu podobnego do tego z listingu 1.3 ❷. Do funkcji `cv.waitKey()` przekaż jako argument wartość 1500, aby wyświetlić rzeczywistą lokalizację żeglarza przez 1,5 sekundy, zanim gra wywoła funkcję `main()` i automatycznie się zresetuje. Później zwiększ o 1 liczbę przeprowadzonych poszukiwań ze zmiennej `search_num`. Chcemy dokonać tego na samym końcu pętli, aby niepoprawny wybór opcji nie liczył się jako poszukiwanie.

Po zakończeniu funkcji `main()` powracamy do przestrzeni globalnej. Dopisz tutaj fragment kodu, który pozwoli zaimportować program jako moduł lub uruchomić go jako niezależną aplikację. Zmienna `__name__` jest wbudowaną zmienną wykorzystywaną do tego, aby ocenić, czy program jest autonomiczny, czy zaimportowany do innego programu. Jeżeli uruchomimy program bezpośrednio, zmienna `__name__` przyjmuje wartość `__main__`, a warunek instrukcji warunkowej zostaje spełniony i funkcja `main()` zostaje wywołana automatycznie. Jeżeli program zaimportowano, nie zostanie uruchomiony automatycznie, a funkcja `main()` musi zostać wywołana celowo.

Uruchomienie gry

Aby zagrać w grę, wybierz opcję *Run/Run module* (uruchom/uruchom moduł) w edytorze tekstów lub po prostu wciśnij klawisz *F5*. Rysunki 1.7 i 1.8 pokazują zrzuty ekranu z uruchomionej gry oraz wyniki pierwszego udanego poszukiwania.

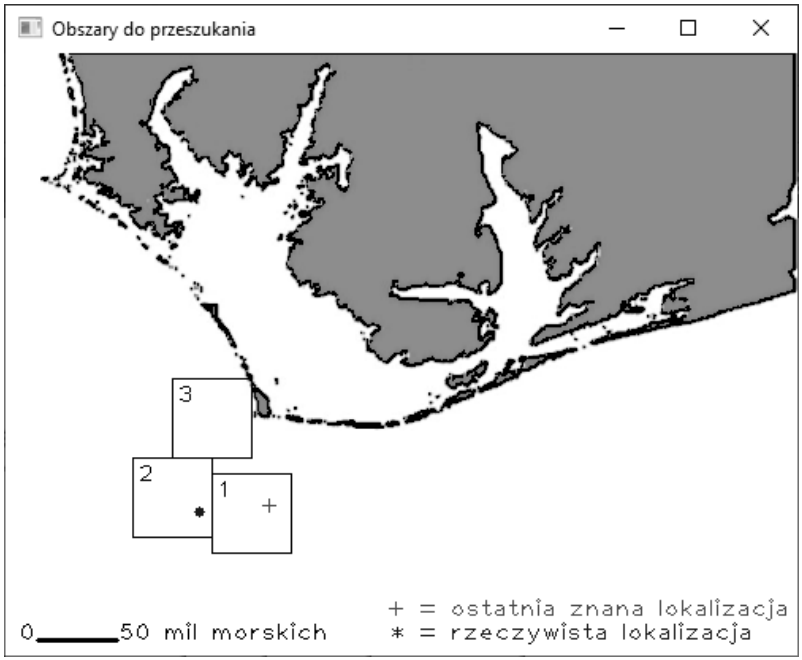


Rysunek 1.7. Okno programu IDLE z wynikiem udanego poszukiwania

W tym przykładzie gracz decyduje się zrealizować oba poszukiwania w obszarze drugim, który początkowo ma 50% szans na bycie miejscem pobytu żeglarza. Pierwsze poszukiwanie się nie udało, ale za drugim razem znaleziono zaginionego. Zauważ, że efektywność poszukiwań wyniosła jedynie nieco ponad 50%. Oznacza to, że szanse na znalezienie żeglarza wynosiły jedynie jeden do czterech (0,5·0,521 = 0,260). Pomimo mądrego wyboru gracz nadal musiał polegać na odrobinie szczęścia!

Gdy grasz w grę, postaraj się wczuć w swoją rolę. Decydujesz o ludzkim życiu lub śmierci. Nie masz też dużo czasu. Jeżeli żeglarz dryfuje na wodzie, masz tylko trzy próby. Wykorzystaj je mądrze!

Na podstawie wartości prawdopodobieństwa założonych na początku gry można założyć, że żeglarz znajduje się na obszarze drugim. Szanse na znalezienie go w obszarze trzecim są nieco mniejsze. W związku z tym dobrą strategią na początek jest albo dwukrotne przeszukanie obszaru drugiego (opcja 2 w menu), albo przeszukanie obszarów drugiego i trzeciego jednocześnie (opcja 6 w menu). Kontroluj



Rysunek 1.8. Mapa regionu z wynikiem udanego poszukiwania

wyniki dotyczące skuteczności poszukiwań. Jeżeli jakiś obszar został dość dokładnie przeszukany (czyli ma wysoki wynik skuteczności poszukiwań), możesz chcieć skoncentrować swoje działania w innym miejscu.

Poniższy scenariusz pokazuje jedną z najgorszych sytuacji, w jakich możesz się znaleźć jako osoba podejmująca decyzje:

```
Podejście nr 2 - wynik 1: Nie znaleziono.
Podejście nr 2 - wynik 2: Nie znaleziono.

Skuteczność poszukiwań (E) dla podejścia nr 2:
E1 = 0.000, E2 = 0.234, E3 = 0.610

Nowe oszacowanie prawdopodobieństwa (P) dla podejścia nr 3:
P1 = 0.382, P2 = 0.395, P3 = 0.223
```

Po podejściu nr 2 zostajesz z jedną próbą do wykorzystania. Oszacowania prawdopodobieństwa są tak podobne, że w niewielkim stopniu pomagają w podjęciu decyzji. W takim przypadku najlepiej przeznaczyć ostatnie podejście na przeszukanie dwóch obszarów i nie tracić nadziei.

Zagraj w grę kilka razy, na ślepo przeszukując obszary w kolejności początkowego prawdopodobieństwa. Wykonaj dwa przeszukiwania w obszarze drugim, kolejne dwa w obszarze trzecim, a resztę w obszarze pierwszym. Następnie staraj się z żelazną konsekwencją trzymać wyników twierdzenia Bayesa. Zawsze podwajaj

poszukiwania w obszarze o najwyższym prawdopodobieństwie. Potem staraj się podzielić poszukiwania na obszary z dwoma największymi prawdopodobieństwami. Później pozwól dojść do głosu swojej intuicji i ignoruj Bayesa, gdy czujesz, że to wskazane. Jak możesz sobie wyobrazić, przy większej liczbie obszarów oraz dni ludzka intuicja wkrótce by się ugięłaby pod ciężarem możliwych wyborów.

Podsumowanie

W tym rozdziale poznaliśmy twierdzenie Bayesa — prostą statystyczną koncepcję o szerokim zastosowaniu we współczesnym świecie. Napisaliśmy program, który wykorzystuje to twierdzenie, aby uwzględniać nowe informacje w formie szacunków dotyczących efektywności poszukiwań, i aktualizuje prawdopodobieństwo znalezienia rozbitka w przeszukiwanych obszarach.

Pobraliśmy też i użyliśmy wiele pakietów naukowych, takich jak NumPy czy OpenCV, które będziemy wykorzystywać również w dalszej części książki. Ponadto zastosowaliśmy kilka użytecznych narzędzi, w tym moduły `itertools`, `sys` i `random` ze standardowej biblioteki Pythona.

Dalsza lektura

Książka *The Theory That Would Not Die: How Bayes' Rule Cracked the Enigma Code, Hunted Down Russian Submarines, and Emerged Triumphant from Two Centuries of Controversy* (Yale University Press, 2011) autorstwa Sharon Bertsch McGrayne opisuje okoliczności odkrycia twierdzenia Bayesa oraz powiązaną z nim historię kontrowersji. Dodatek zawiera kilka przykładów zastosowania twierdzenia Bayesa. Jeden z nich był inspiracją dla scenariusza z zaginionym żeglarzem, którego użyłem w tym rozdziale.

Głównym źródłem dokumentacji na temat biblioteki NumPy jest strona internetowa *https://docs.scipy.org/doc*.

Samodzielny projekt: Inteligentniejsze poszukiwania

Obecnie program *bayes.py* umieszcza wszystkie współrzędne z obszaru poszukiwań na liście i dokonuje losowego tasowania. Kolejne poszukiwania w tym samym obszarze mogą dotyczyć tych samych lokalizacji. Niekoniecznie jest to złe, biorąc pod uwagę rzeczywiste uwarunkowania. Żeglarz cały czas może dryfować w różnych kierunkach. Co do zasady dobrze byłoby jednak objąć poszukiwaniami możliwie największy obszar bez powtórzeń.

Skopiuj program i zmodyfikuj go w taki sposób, aby kontrolował, które współrzędne zostały już przeszukane w danym obszarze. Wykluczaj je z dalszych

poszukiwań do czasu ponownego wywołania funkcji `main()` — niezależnie od tego, czy będzie ono wynikało ze znalezienia żeglarza, czy wyboru opcji 7 oznaczającej restart. Przetestuj dwie wersje gry, aby zobaczyć, czy zmiana znacząco wpływa na wynik.

Samodzielny projekt: Znajdź najlepszą strategię dzięki metodzie Monte Carlo

Symulacja metodą Monte Carlo wykorzystuje wielokrotne próbkowanie do przewidywania różnych wyników w określonym zakresie warunków. Stwórz wersję programu *bayes.py*, która automatycznie wybiera opcje z menu i przechowuje tysiące wyników, aby móc określić najskuteczniejszą strategię poszukiwań. Na przykład niech program wybiera opcje 1, 2 i 3 zależnie od tego, który obszar ma największe bayesowskie prawdopodobieństwo, a następnie zapisz numer poszukiwania, w którym żeglarz zostaje znaleziony. Powtórz procedurę 10 tysięcy razy i wyciągnij średnią ze wszystkich wyników. Następnie ponownie uruchom pętlę, ale tym razem wybieraj opcje 4, 5 lub 6, opierając się na najwyższym połączonym prawdopodobieństwie. Porównaj końcowe średnie. Czy lepiej dwukrotnie przeczesać ten sam obszar, czy podzielić poszukiwania między dwa obszary?

Samodzielny projekt: Obliczanie prawdopodobieństwa wykrycia

W prawdziwych warunkach w ramach akcji poszukiwawczo-ratunkowej szacowalibyśmy *oczekiwane* prawdopodobieństwo skuteczności poszukiwań dla obszaru przed przeprowadzeniem poszukiwania. Oczekiwane (*planowane*) prawdopodobieństwo zależałoby głównie od warunków pogodowych. Na przykład nad jednym z obszarów mogłaby zalegać mgła, a niebo nad pozostałymi dwoma mogłoby być bezchmurne.

Mnożąc założone prawdopodobieństwo przez planowane prawdopodobieństwo skuteczności poszukiwań, otrzymujemy prawdopodobieństwo wykrycia dla danego obszaru. **Prawdopodobieństwo wykrycia** to prawdopodobieństwo, że dany obiekt zostanie wykryty z uwzględnieniem wszystkich znanych błędów i źródeł szumu.

Napisz wersję programu *bayes.py,* która losowo generuje planowane prawdopodobieństwo skuteczności poszukiwań w każdym obszarze. Pomnóż oszacowane prawdopodobieństwo dla każdego obszaru (tj. `self.p1`, `self.p2` i `self.p3`) przez nowe zmienne, aby uzyskać prawdopodobieństwo wykrycia dla danego obszaru. Na przykład: jeżeli bayesowskie oszacowanie prawdopodobieństwa dla obszaru trzeciego wynosi 0,90, ale planowane PSP wynosi tylko 0,1, prawdopodobieństwo wykrycia wynosi 0,09.

W konsoli pokaż graczowi oszacowane prawdopodobieństwo, planowane PSP i prawdopodobieństwo wykrycia dla każdego obszaru, tak jak pokazano poniżej. Gracze mogą użyć tych informacji, aby dokonywać lepszych wyborów.

```
Rzeczywista skuteczność poszukiwań (E) dla podejścia nr 1:
E1 = 0.190, E2 = 0.000, E3 = 0.000

Nowa planowana skuteczność poszukiwań (E) oraz oszacowanie
prawdopodobieństwa (P) dla podejścia nr 2:
E1 = 0.509, E2 = 0.826, E3 = 0.686
P1 = 0.168, P2 = 0.520, P3 = 0.312

Podejście nr 2

        Wybierz następne obszary do przeszukania:

        0 - Wyjdź z programu

        1 - Przeszukaj dwukrotnie obszar pierwszy
          Prawdopodobieństwo wykrycia: 0.164

        2 - Przeszukaj dwukrotnie obszar drugi
          Prawdopodobieństwo wykrycia: 0.674

        3 - Przeszukaj dwukrotnie obszar trzeci
          Prawdopodobieństwo wykrycia: 0.382

        4 - Przeszukaj obszary pierwszy i drugi
          Prawdopodobieństwo wykrycia: 0.515

        5 - Przeszukaj obszary pierwszy i trzeci
          Prawdopodobieństwo wykrycia: 0.3

        6 - Przeszukaj obszary drugi i trzeci
          Prawdopodobieństwo wykrycia: 0.643

        7 - Zacznij od początku

Wybierz opcję:
```

Aby obliczyć połączone prawdopodobieństwo wykrycia (*PW*) przy dwukrotnym przeszukiwaniu tego samego obszaru, użyj następującego wzoru:

$$1 - (1 - PW)^2$$

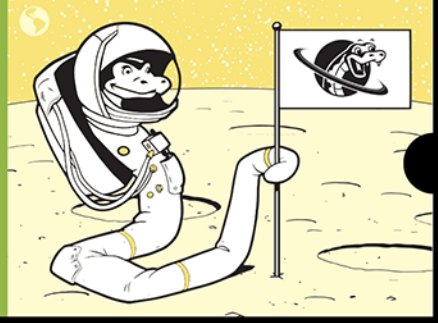
W innych przypadkach po prostu zsumuj prawdopodobieństwo.

Gdy obliczasz rzeczywistą wartość prawdopodobieństwa skuteczności poszukiwań dla danego obszaru, ogranicz je w jakiś sposób do oczekiwanej wartości. Odzwierciedla to ogólną dokładność prognoz pogody na dzień naprzód. Zastąp funkcję `random.uniform()` funkcją z jakimś rodzajem rozkładu (np. rozkładem trójkątnym), uwzględniającą planowaną wartość PSP. Listę dostępnych rodzajów rozkładów możesz znaleźć pod adresem *https://docs.python.org/3/library/random*.

html#real-valued-distributions. Oczywiście rzeczywiste prawdopodobieństwo skuteczności poszukiwań dla obszaru nieprzeszukanego zawsze będzie wynosić zero.

W jaki sposób widoczność planowanego prawdopodobieństwa skuteczności poszukiwań wpływa na sposób gry? Czy jest łatwiej, czy trudniej wygrać? Czy trudniej jest zrozumieć, w jaki sposób działa twierdzenie Bayesa? Czy w przypadku prawdziwych poszukiwań wchodziłoby w grę zignorowanie obszaru o wysokim prawdopodobieństwie, ale niskiej planowanej skuteczności ze względu na niespokojne morze? Jaka byłaby Twoja decyzja: szukać dalej, odwołać poszukiwania czy przenieść poszukiwania do obszaru o niższym prawdopodobieństwie sukcesu, ale lepszej pogodzie?


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion



PYTHON. BYĆ MOŻE POLEGNIESZ, A MOŻE CI SIĘ UDA!

Po nauczeniu się podstaw Pythona przychodzi czas na coś poważniejszego. Umiesz już napisać kilkanaście linii kodu, który działa zgodnie z oczekiwaniami, znasz składnię języka i wiesz, jakie możliwości oferuje. Być może nawet traktujesz tworzenie kodu w Pythonie jako świetną zabawę. W każdym razie, skoro znasz już podstawy, możesz się zająć prawdziwymi projektami. Dzięki nim nie tylko rozwiążesz palące problemy codziennego życia, ale również nauczysz się tworzyć kompletne, funkcjonalne programy. Programy, które będą działać.

Ta książka jest sposobem na dalszą naukę programowania poprzez realizację projektów. Każdy z nich został wyjaśniony krok po kroku, opisano też sposoby korzystania z licznych bibliotek i pakietów Pythona. Dzięki projektom dowiesz się, jak wykorzystywać programowanie do realizacji eksperymentów, testowania teorii, naśladowania natury lub po prostu do zabawy. Nabierzesz wprawy w pracy z bibliotekami i modułami Pythona, nauczysz się także pożytecznych skrótów, przydatnych funkcji i innych pomocnych technik. Po lekturze w łatwy sposób zrealizujesz zadania, które kiedyś spędzały sen z powiek geniuszom — a to dopiero początek przygody z Pythonem!

Dzięki książce dowiesz się, jak:

- używać bibliotek: matplotlib, NumPy, Bokeh, pandas, Requests, Beautiful Soup i turtle
- tworzyć szyfry, szyfrować i odszyfrowywać wiadomości
- przetwarzać język naturalny i pisać kod do rozpoznawania obrazów
- pisać programy wykrywające i śledzące obiekty
- korzystać z narzędzi do analizy i wizualizacji danych

Lee Vaughan jest emerytowanym geologiem, programistą, wielbicielem kultury popularnej, nauczycielem i autorem książek. Wcześniej pracował w ExxonMobil, gdzie tworzył i recenzował skomplikowane modele komputerowe, rozwijał i testował oprogramowanie, a także szkolił geologów i inżynierów. Mieszka w The Woodlands w Teksasie.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej! ▸
ISBN 978-83-283-8346-3
9 788328 383463
Cena: 79,00 zł
no starch press
INFORMATYKA W NAJLEPSZYM WYDANIU