

# Duszek znajduje wyjście z labiryntu

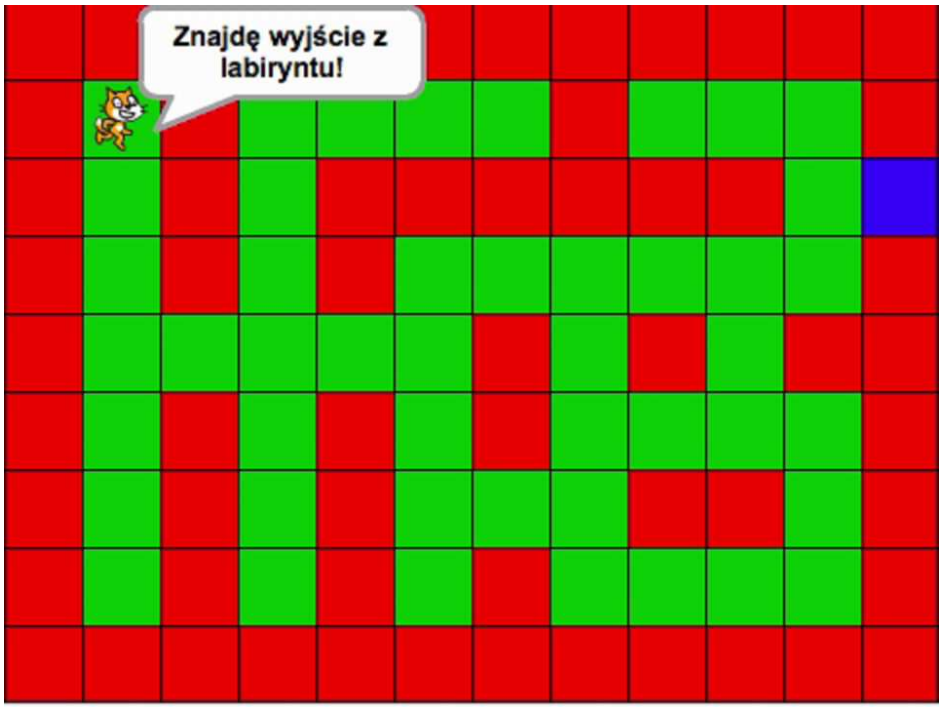
Materiał opisuje projekt "Duszek znajduje wyjście z labiryntu" i składa się z sekcji: "Scenariusz projektu", "Algorytm znajdowania wyjścia z labiryntu", "Tworzymy skrypty dla duszka", "Podsumowanie", "Zadania uzupełniające".

Materiał zawiera 11 zrzutów ekranu, 3 filmy, 9 ćwiczeń, w tym 2 interaktywne.

# Duszek znajduje wyjście z labiryntu

## Scenariusz projektu

Pewnie nie raz bawiliście się grą, w której bohater musiał przejść przez jakiś labirynt. Być może na zajęciach komputerowych w szkole podstawowej był wykorzystywany e-podręcznik i projektowaliście grę, w której gracz sterował bohaterem przeprowadzając go przez labirynt.
Teraz zadanie dla duszka będzie trudniejsze. Po kliknięciu w zieloną flagę będzie musiał samodzielnie znaleźć drogę wyjścia z labiryntu. Będzie realizował algorytm, który przeprowadzi go do wyjścia przez labirynt. Przykładowe działanie aplikacji możesz obejrzeć na poniższym filmie.




Film dostępny na portalu epodreczniki.pl

Źródło: Janusz Wierzbicki, Maciej Borowiecki, licencja: CC BY 3.0.

G4_film1

---

Na planszy w powyższej aplikacji są pola w trzech kolorach – zielonym, czerwonym i jedno niebieskie. Zielony oznacza korytarz, czyli pola po których duszek może się poruszać, a czerwony - ściany, czyli pola, niedostępne dla duszka. Wyjście oznaczone jest kolorem niebieskim. Plansza składa się z 12x9 pól, czyli długość boku pojedynczego pola wynosi 40. Przykładowy plik z planszą możesz pobrać poniżej.

Źródło: Janusz Wierzbicki, Maciej Borowiecki, licencja: CC BY 3.0.

> **Ćwiczenie 1**
>
> Wczytaj planszę z labiryntem jako nowe tło sceny. Pamiętaj o usunięciu domyślnego, białego tła. Zmień rozmiary duszka, tak aby mieścił się w pojedynczym polu.

# Algorytm znajdowania wyjścia z labiryntu

> **Ćwiczenie 2**
>
> Zastanów się nad strategią duszka, jak ma postępować, poszukując drogi. Przyjmij założenie, że duszek „patrzy" tylko o jedno pole wprzód oraz pamięta (oznacza) pola, które już odwiedził. Może więc stwierdzić, czy sąsiednie pole jest ścianą, wolnym polem albo polem, przez które już przeszedł. Może też cofać się, jak trafi w ślepą uliczkę. Przedyskutuj swoje propozycje z kolegami i koleżankami oraz nauczycielem, następnie porównaj z odpowiedzią.

Spróbujmy zapisać ściślej propozycję z odpowiedzi do poprzedniego zadania.

Szukaj drogi:
1.Oznacz pole, na którym stoisz.
2.Powtórz 4 razy:
2.1.Jeżeli pole na wprost jest wyjściem to:
2.1.1.Przejdź na pole na wprost (Przesuń o 40 kroków).
2.1.2.Powiedz „Znalazłem wyjście!".
2.1.3.Zatrzymaj wszystko.
2.2.Jeżeli pole na wprost jest wolne i nie było odwiedzane to:
2.2.1.Przejdź na pole na wprost (Przesuń o 40 kroków).
2.2.2.Szukaj drogi.
2.2.3.Wycofaj się z ostatniego ruchu (Przesuń o -40 kroków).
2.3.Jeżeli pole na wprost jest ścianą lub było odwiedzone to:
2.3.1.Obróć się w prawo o 90 stopni.
3.Zatrzymaj ten skrypt.

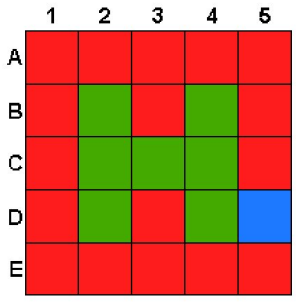
Zwróć uwagę na bardzo ważny punkt 2.2.2. Duszek rozpoczyna ponownie wykonywanie tego samego algorytmu od początku, ale z innego pola. Jeżeli z tego pola jest wyjście, nie wróci już do punktu 2.2.3 (ponieważ zakończy wykonywanie algorytmu w punkcie 2.1.2). Natomiast jeżeli poszukiwanie drogi z tego pola zakończy się niepowodzeniem, wróci do punktu 2.2.3 i wycofa się z ostatniego ruchu. Zwróć też uwagę, że sprawdzenie warunku

w punkcie 2.3 jest niepotrzebne. Jeżeli sąsiednie pole nie było wyjściem (punkt 2.1) i nie dało się przez nie dojść do wyjścia (punkt 2.2), to jest ścianą lub zostało oznaczone jako odwiedzone. Powyższy przepis można więc nieznacznie uprościć.

Szukaj drogi:
1.Oznacz pole, na którym stoisz.
2.Powtórz 4 razy:
2.1.Jeżeli pole na wprost jest wyjściem to:
2.1.1.Przejdź na pole na wprost (Przesuń o 40 kroków).
2.1.2.Powiedz „Znalazłem wyjście!”.
2.1.3.Zatrzymaj wszystko.
2.2.Jeżeli pole na wprost jest wolne i nie było odwiedzane to:
2.2.1.Przejdź na pole na wprost (Przesuń o 40 kroków).
2.2.2.Szukaj drogi.
2.2.3.Wycofaj się z ostatniego ruchu (Przesuń o -40 kroków).
2.3.Obróć się w prawo o 90 stopni.
3.Zatrzymaj ten skrypt.

### Ćwiczenie 3

Wykonaj ręcznie algorytm dla poniższej planszy:
Zapisuj na kartce papieru kolejne pola, z których szukasz wyjścia.
Wypełniaj tabelkę opisującą aktualną pozycję (pole) i kierunek duszka.
Wydrukuj planszę i oznaczaj na niej pola odwiedzone np. przez przekreślenie.
Przyjmij założenie, że duszek na starcie stoi na polu B2 i patrzy w dół.




| Pozycja (pole) duszka | Kierunek |
|---|---|
| B2 | dół |
| ... | ... |
| | |
| | |
| | |
| | |

Źródło: Janusz Wierzbicki, Maciej Borowiecki, licencja: CC BY 3.0.
Źródło: Janusz Wierzbicki, Maciej Borowiecki, licencja: CC BY 3.0.
Źródło: Janusz Wierzbicki, Maciej Borowiecki, licencja: CC BY 3.0.

W rozwiązaniu problemu odwołaliśmy się do tego samego problemu, ale dla innych danych (w tym przypadku innego pola). Takie podejście nazywamy **rekurencją**, a algorytmy je wykorzystujące **algorytmami rekurencyjnymi**.

# Tworzymy skrypty dla duszka

Przed przystąpieniem do tworzenia skryptów należy jeszcze rozstrzygnąć dwa problemy:
•Jak duszek ma oznaczać pole.
•W jaki sposób duszek ma rozpoznać kolor sąsiedniego pola.

Zwróć uwagę, że nie ma znaczenia, czy pole jest ścianą, czy było już odwiedzone. Najwygodniej będzie oznaczać pola odwiedzone tym samym kolorem co ściana. Jak można zauważyć na filmach, duszek stawia na nowo odwiedzanym polu kropkę w kolorze ściany.

Żeby sprawdzić w jakim kolorze jest sąsiednie pole, duszek musi niestety na nim stanąć. Przejdzie więc najpierw na sąsiednie pole. Jeżeli pod duszkiem nie ma koloru czerwonego, kontynuuje działania. Jeżeli jest kolor czerwony, wycofuje się. W związku z tym dostosujemy nasz algorytm w postaci listy kroków do możliwości duszka w środowisku Scratch.

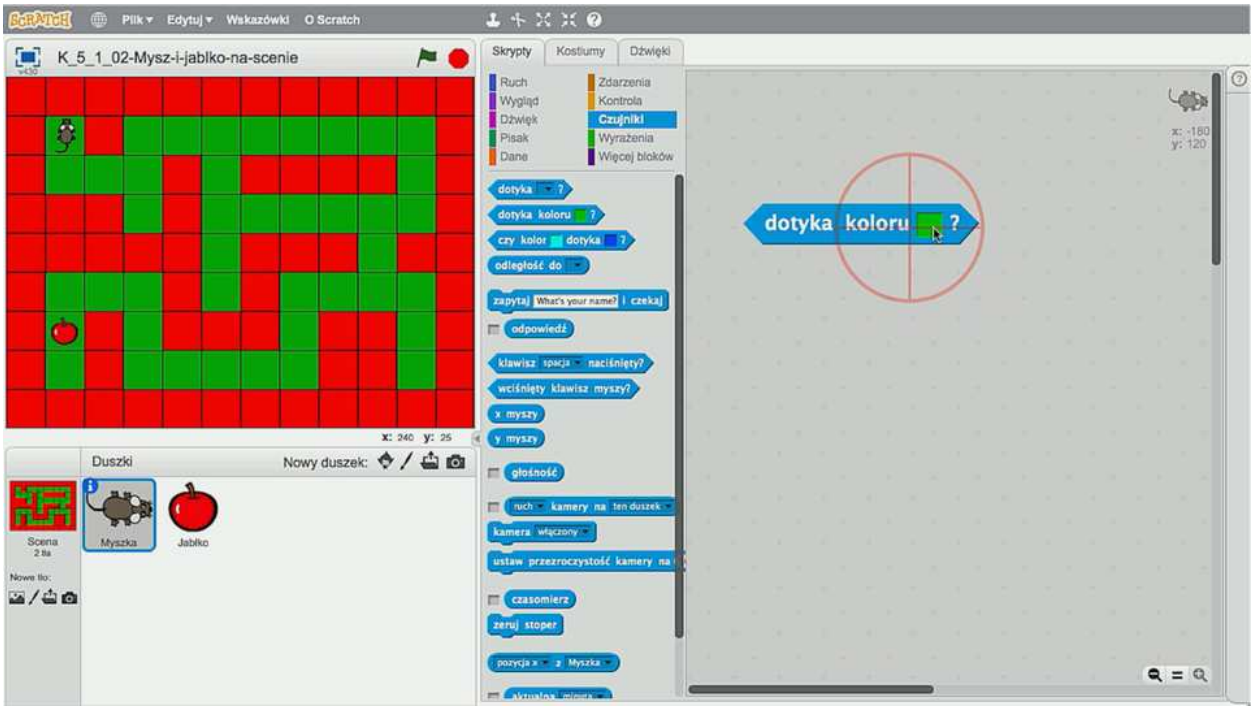
Szukaj drogi:
1.Oznacz pole, na którym stoisz.
2.Powtórz 4 razy:
2.1.Przesuń o 40 kroków.
2.2.Jeżeli dotyka koloru niebieskiego to:
2.2.1.Powiedz „Znalazłem wyjście!” przez np. 2 s.
2.2.2.Zatrzymaj wszystko.
2.3.Jeżeli nie dotyka koloru czerwonego to:
2.3.1.Szukaj drogi.
2.4.Przesuń o -40 kroków.
2.5.Obróć się w prawo o 90 stopni.
3.Zatrzymaj ten skrypt.

### Ćwiczenie 4

Zbuduj nowy własny skrypt realizujący powyższy algorytm. Wcześniej utwórz nowy blok *OznaczPole*. Pozostaw go na razie pustym, potem wypełnisz go treścią.

#### Wskazówka

Po utworzeniu nowego bloku jest on od razu dostępny w kategorii Więcej bloków. Możesz go wykorzystać w definicji tego bloku. Zrealizujesz w ten sposób wywołanie rekurencyjne. Jak wybrać kolor w klocku dotyka koloru ... ? możesz obejrzeć na poniższym filmie.




Film dostępny na portalu epodreczniki.pl

Wybór koloru

Film: Ustawienie właściwego koloru

---

### Ćwiczenie 5

Uzupełnij blok *OznaczPole*. Zastanów się, jak narysować czerwony ślad. Pamiętaj, że duszek po narysowaniu musi być na tej samej pozycji, co przed rysowaniem.

> **Ćwiczenie 6**
>
> Zastanów się, jakie czynności początkowe, oprócz ustawienia atrybutów pisaka (rozmiaru, koloru), należy wykonać po kliknięciu zielonej flagi. Przygotuj odpowiedni skrypt. Pamiętaj o wywołaniu skryptu *SzukajDrogi* po wykonaniu ustawień początkowych.

Uruchom i przetestuj działanie aplikacji. Prawdopodobnie duszek bardzo szybko biega po labiryncie. Możesz go spowolnić, wstawiając klocek czekaj … s z odpowiednio dobranym czasem opóźnienia przed wywołaniem bloku *SzukajDrogi*. Możesz także przygotować drugi skrypt uruchamiany po kliknięciu zielonej flagi animujący duszka podczas chodzenia po labiryncie.

# Podsumowanie

W tym rozdziale została przedstawiona bardzo silna technika projektowania algorytmów – ***rekurencja***. Budując algorytm rekurencyjny należy pamiętać o podstawowych zasadach: wywołanie rekurencyjne powinno nastąpić dla zmienionych danych przy spełnieniu pewnego warunku. Ciąg wywołań rekurencyjnych powinien się kiedyś zakończyć. Najczęściej budujemy blok rekurencyjny z parametrem, a warunek, kiedy następuje wywołanie rekurencyjne zależy od tego parametru.
W omawianym algorytmie rekurencyjnym (wychodzenia z labiryntu) rolę parametru pełni pole, na którym stoi duszek, choć sam blok (w skryptach) jest zdefiniowany bez parametrów. Wywołanie rekurencyjne zależy od koloru pola, na które próbuje przejść duszek.
Porównaj działanie opracowanej przez siebie aplikacji z filmem w pierwszym podrozdziale. Czy obie aplikacje stosują ten sam algorytm znajdowania drogi wyjścia z labiryntu? Łatwo zauważysz, że w twojej aplikacji duszek znajduje drogę do wyjścia, ale wcale nie najkrótszą, często wracając. Takie algorytmy nazywane są algorytmami z powrotami i często są zapisywane rekurencyjnie.
Natomiast w przedstawionej na filmie aplikacji duszek znajduje najkrótszą drogę do wyjścia. Rozwiązuje tak naprawdę bardziej ogólny problem znalezienia najlepszej drogi (w tym przypadku najkrótszej) pomiędzy dwoma miejscami (polami). Z problemem znalezienia najlepszej drogi (najkrótszej, najszybszej) możesz się zetknąć choćby korzystając z nawigacji satelitarnej lub map internetowych.
W przyszłości, jeśli będziesz kontynuować swoją przygodę z algorytmami, poznasz zapewne algorytm znajdowania najkrótszej drogi przejścia, a także inne problemy rozwiązywane algorytmami z powrotami.

# Zadania uzupełniające

**Ćwiczenie 7**

Popraw planszę tak, aby nie było przejścia z pola startowego do wyjścia lub usuń wyjście. Możesz w tym celu skopiować tło sceny i je nieznacznie poprawić w edytorze graficznym. Na którym polu duszek zakończy poszukiwanie drogi? W którym miejscu i którego skryptu można rozpoznać taką sytuację? Popraw właściwy skrypt, niech duszek poinformuje w dymku komiksowym, że nie ma przejścia do wyjścia.

**Ćwiczenie 8**

Zmodyfikuj tak aplikację, aby duszek próbę wykonania kolejnego ruchu rozpoczynał zawsze od tego samego kierunku (np. prawo), a nie kierunku, w którym patrzy.

> **Wskazówka**
>
> Zadbaj o to, aby duszek wycofując się z ruchu, patrzył w tym samym kierunku, w którym wykonał ruch.

**Ćwiczenie 9**

Zajrzyj ponownie do projektu „Ułamki zwykłe” i do podrozdziału dotyczącego algorytmu Euklidesa. Własność NWD została zdefiniowana rekurencyjnie: NWD(*a,b*)=NWD(*a-b,b*) przy założeniu, że *a*>*b*. Spróbuj utworzyć własny blok obliczający NWD rekurencyjnie i stwórz prostą aplikację sprawdzającą, czy blok działa poprawnie.

**Ważne!**

Dla wielu problemów można podać rozwiązanie zarówno rekurencyjne, jak i iteracyjne (czyli z wykorzystaniem pętli). W szczególności, jeśli w algorytmie rekurencyjnym następuje dokładnie jedno wywołanie rekurencyjne, zawsze można podać rozwiązanie z jedną pętlą powtarzaj aż. W takiej sytuacji lepiej jest stosować rozwiązanie iteracyjne.