# Wykrywanie ilości oczek na kostkach

Autorzy:

Patryk Chmielecki 145190 I7.1
Bartek Demut 145324 I7.1

Kod źródłowy:
https://github.com/Chmielek2137/KCK_Obrazy

## Wstęp

Celem projektu było zaimplementowanie mechanizmu wykrywania liczby oczek na kostkach znajdujących się na zdjęciach. Algorytm wykrywa kostki które znajdują się na fotografii, oznacza je prostokątem oraz wyświetla w ich pobliżu liczbę oczek która się na nich znajduje.

## Opis działania algorytmu

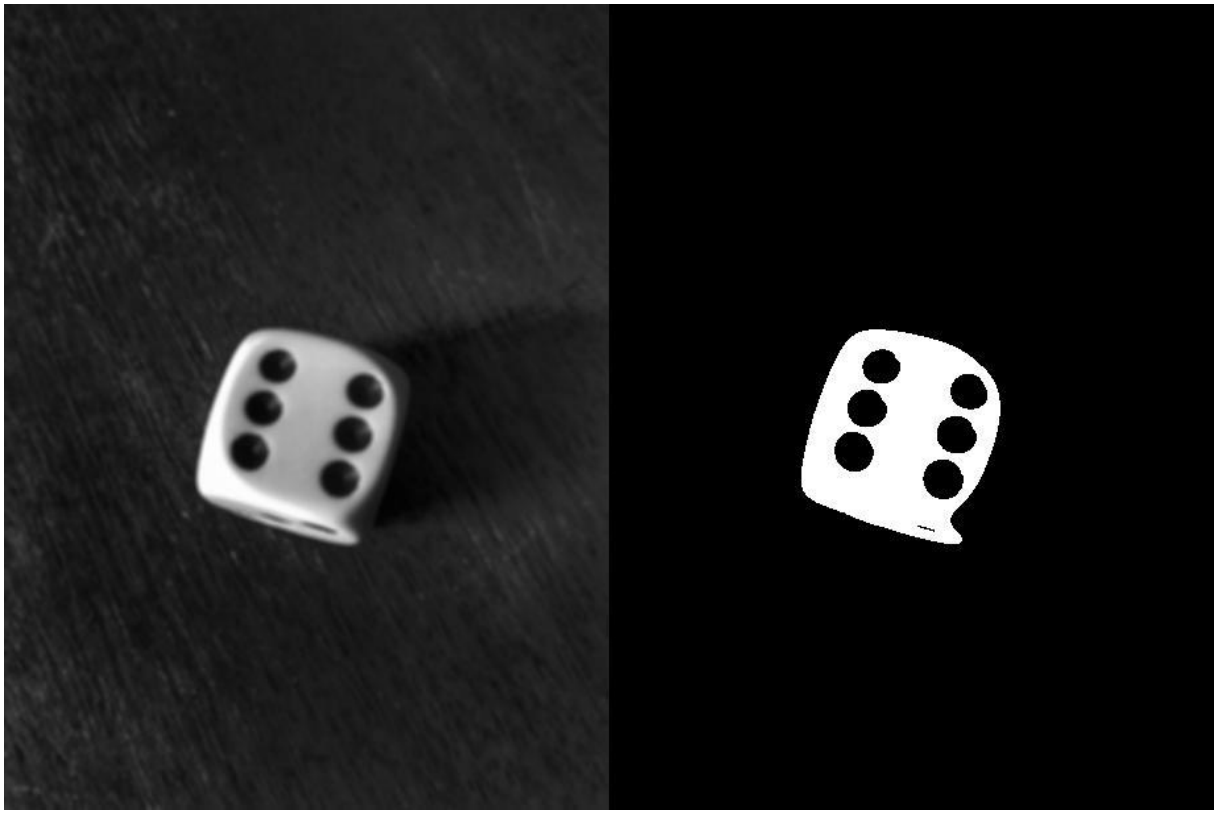
Program został w całości wykonany w języku python z wykorzystaniem biblioteki OpenCV w celu przetwarzania obrazów.

Algorytm przetwarza obraz i wykrywa oczka na kostkach wykonując kolejne kroki:

- Wczytanie zdjęci w odcieniach szrości
- Zastosowanie filtru Gaussa dla przetwarzanego obrazu
- Konwersję obrazu do postaci binarnej z wykorzystaniem wartości progowej wyznaczonej metodą Otsu
- Znalezienie konturów oraz ich hierarchii na zdjęciu przy wykorzystaniu funkcji findContours() z biblioteki OpenCV
- Znalezieniu minimalnego prostokąta przybliżającego kształt każdego znalezionego konturu
- Wyznaczenia współczynnika proporcji obrazu i odrzuceniu tych znalezionych prostokątów których współczynnik proporcji jest zbyt duży lub zbyt mały co oznacza że zbytnio odbiegają kształtem od kształtu kostki
- Nałożenia na zdjęcie obrysów prostokątów których współczynnik proporcji mieści się w zadanym przedziale
- Wykorzystaniu hierarchii konturów w celu znalezienia takich, w których znajduje się od jednego do sześciu innych które są kropkami oraz zliczenie oczek
- Nałożeniu na obraz liczby znalezionych oczek

Prześledzimy proces przetwarzania dla przykładowej fotografii przedstawionej poniżej.

Pierwszy etap polega na zastosowaniu filtru Gaussa (po lewej) a następnie zmianie obrazu wejściowego na postać binarną wyznaczając wartość progową metodą Otsu (po prawej).



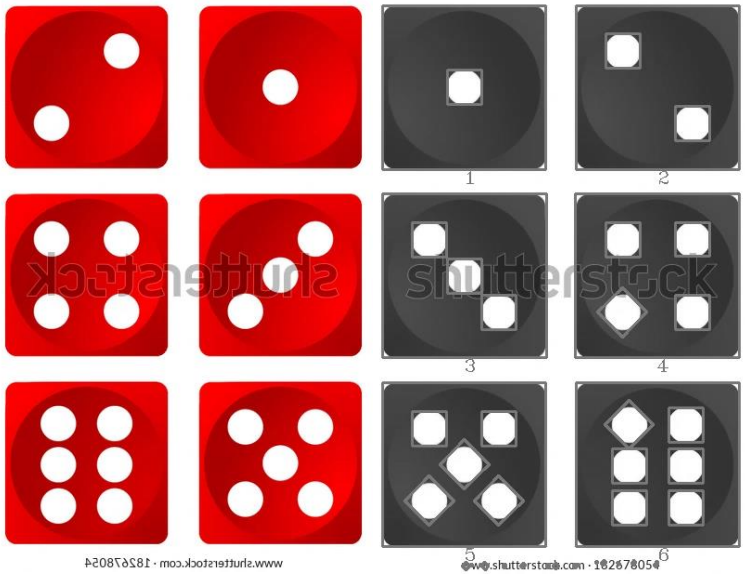
Kolejnym krokiem było wyznaczenie konturów oraz naniesienie ich na obraz. Kontury wyznaczane są przy pomocy funkcji findContours() z biblioteki OpenCV wraz z hierarchią konturów. Następnie w pętli dla każdego z konturów wyznaczany najmniejszy prostokąt okalający kontur przy pomocy funkcji minAreaRect() oraz współczynnik proporcji każdego z prostokątów. Kolejnym krokiem jest wyznaczenie współrzędnych wierzchołków każdego z prostokątów przy pomocy funkcji boxPoints(). W tym miejscu jest także wyznaczane miejsce w którym znajdzie się wyświetlona liczba oczek jeśli okaże się że w danym prostokącie znajduje się kostka. Potem sprawdzany jest współczynnik proporcji prostokąta okalającego przetwarzany kontur i w przypadku gdy jest zbyt różny od współczynnik proporcji kwadratu, tzn. mniejszy od 0,7 lub większy od 1,3, oznaczamy go jako nie spełniający kryteriów i przechodzimy do kolejnej iteracji pętli. W przeciwnym przypadku rysowany jest prostokąt na zdjęciu wynikowym.

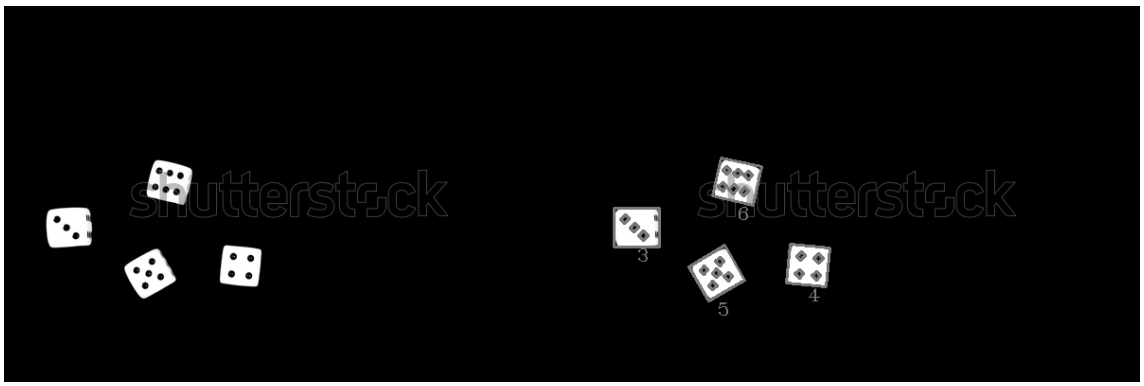
Następnym krokiem jest sprawdzenie ile innych konturów znajduje się w każdym z wyznaczonych konturów. W tym celu wykorzystujemy wcześniej przygotowaną tablicę w której oznaczyliśmy prostokąty okalające których współczynnik proporcji nie spełnia przyjętych przez nas założeń i hierarchię konturów. Sprawdzamy dla każdego z konturów czy jego prostokąt okalający spełnia założenia co do współczynnik proporcji, jeśli tak to z tablicy hierarchii odczytujemy ostatnią pozycję która oznacza indeks konturu w którym się znajduje i w tablicy tymczasowej inkrementujemy wartość znajdującą się pod odczytanym indeksem.

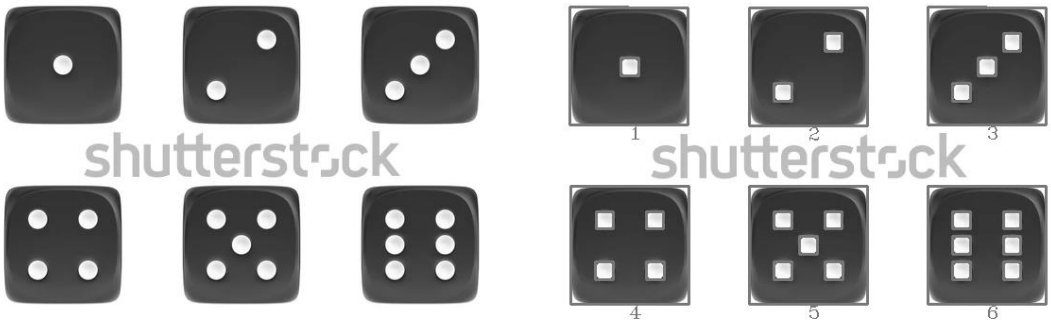
Ostatnim krokiem jest sprawdzenie dla każdego z konturów czy ilość innych konturów które się w nim znajdują mieści się w zakresie od jednego do sześciu. Jeśli tak to w pobliżu danego konturu wypisujemy liczbę tych konturów ponieważ możemy przypuszczać że dany kontur to kontur szukanej kostki.

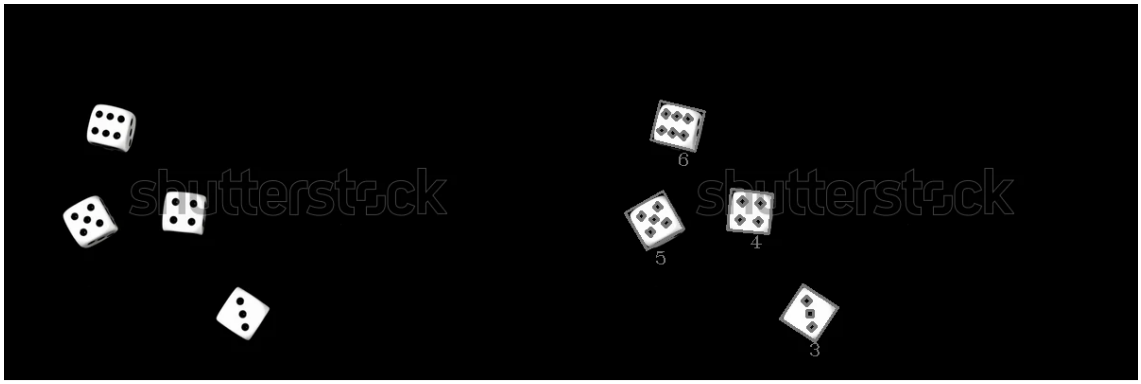
## Wyniki uzyskane przez program

- Przykładowe wyniki dla łatwych obrazów



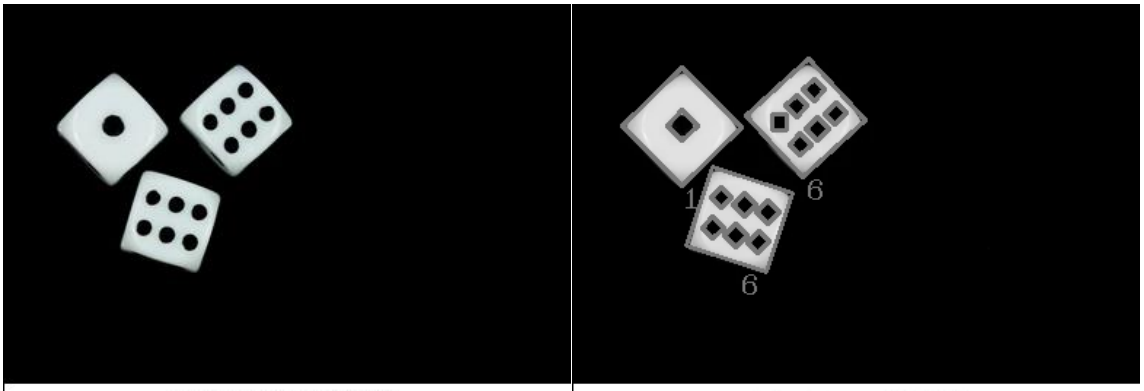



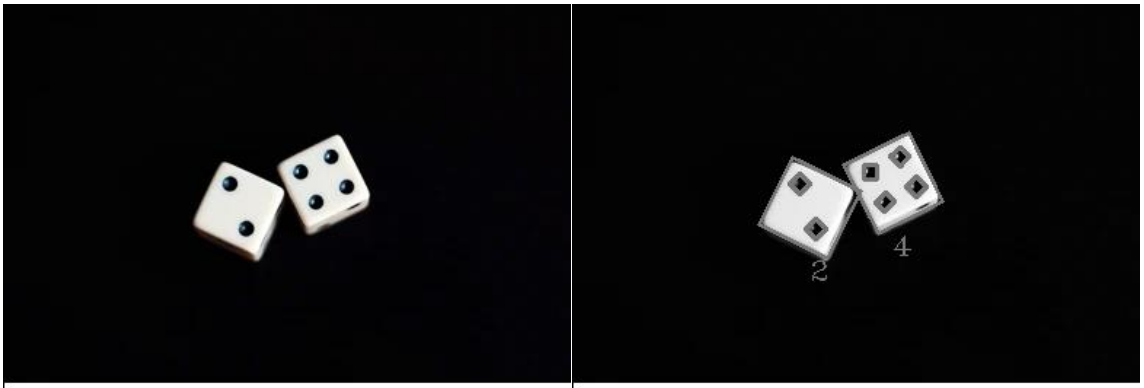


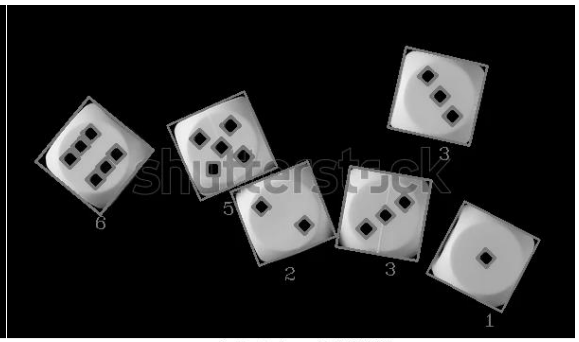













Kostki na jednorodnym tle, odrużniającym się od koloru kostek są z łatwością rozpoznawane. Trudnością też nie jest zazwyczaj lekkie pochylenie kostki.

- Przykładowe wyniki dla obrazów o średnim stopniu skomplikowania

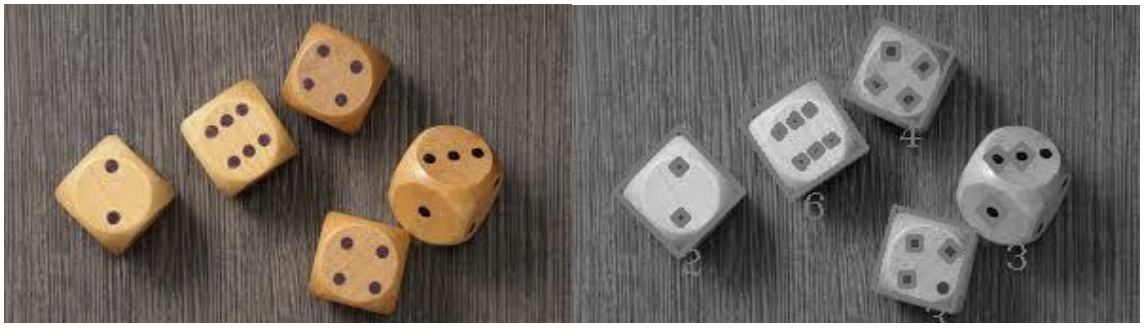




www.shutterstock.com · 1784454665

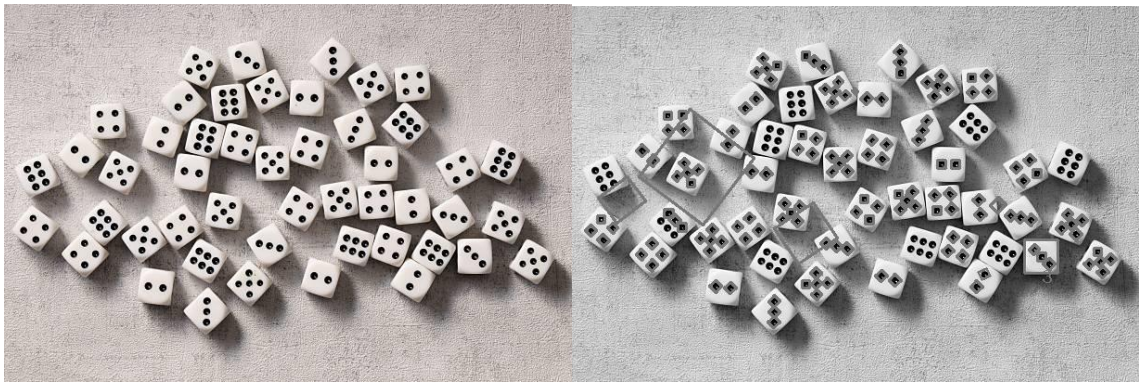

www.shutterstock.com · 1784454665

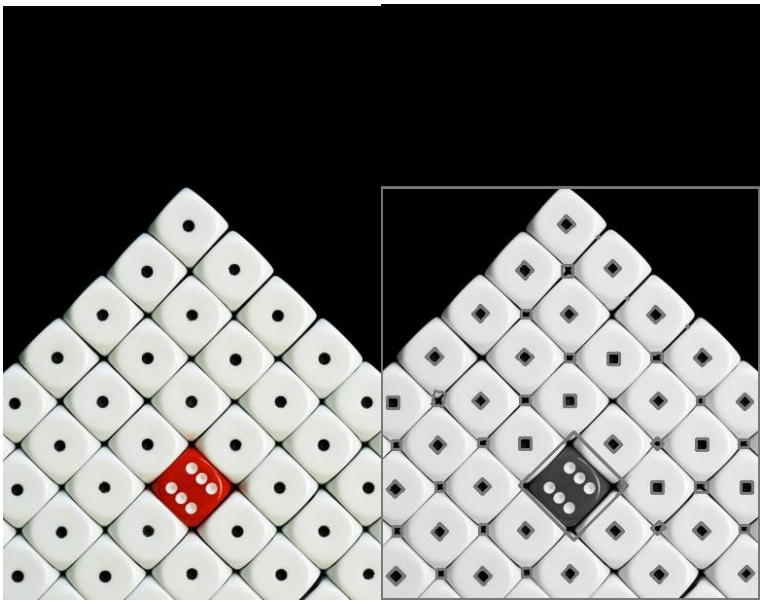




Lekko zróżnicowane podłoże oraz oświetlanie nie było problemem dla programu. Trudności natomiast sprawiało otoczenie w kolorze bradzo podobnym do kostki lub bardzo zróżnicowane a także kostki stykające się bokami.

- Przykładowe wyniki dla obrazów o wysokim stopniu skomplikowania




gettyimages
Herwin Bahar / EyeEm


gettyimages
Herwin Bahar / EyeEm


£20
£5
£10
£50
Bank of England
FIVE Pounds

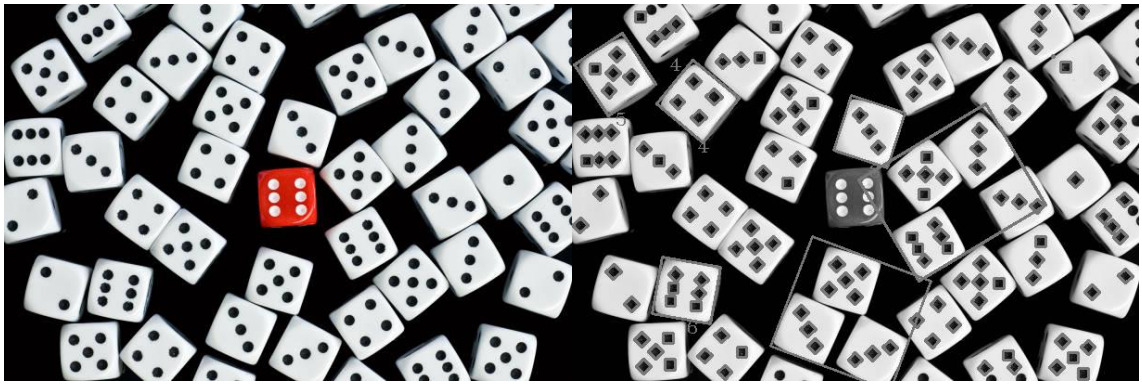

£20
£5
£10
£50
Bank of England
FIVE Pounds

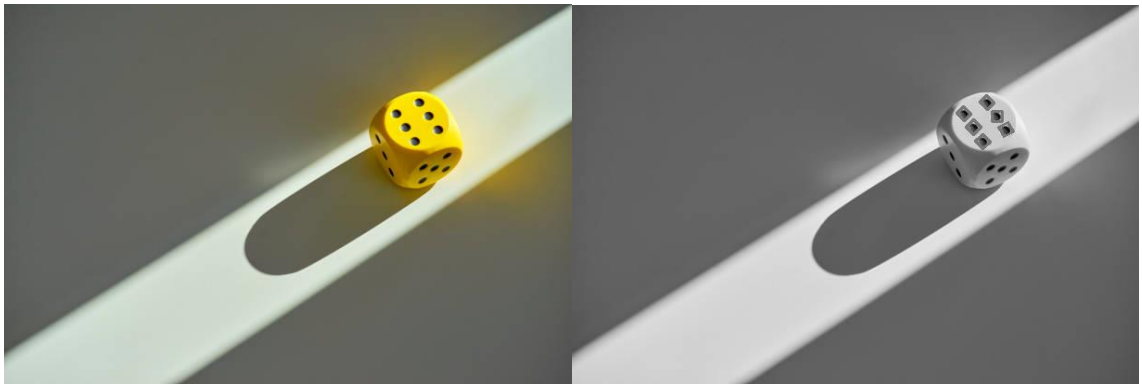

gettyimages
Peter Dazeley


gettyimages
Peter Dazeley


gettyimages
antpartner-images


gettyimages
antpartner-images





W tym przypadku program miał problemy z rozpoznaniem kostek stykających i nakładających się na zdjęciu oraz o różnych kolorach a także na tłach bardzo zróżnicowanych np. trzymanych w ręku.

## Podsumowanie

Stworzony algorytm jest w stanie rozpoznać kostki będące na w miarę jednorodnych powierzchniach, zwrócone w stronę obserwatora i nie stykające się. Problemy zaczynają się pojawiać gdy w tle zaczynają się pojawiać inne obiekty lub cienie a także gdy nie ma wystarczających odstępów pomiędzy nimi. Częstokroć algorytm jest w stanie zaznaczyć kropki na kostkach lecz nie jest w stanie zinterpretować, że dany obiekt to kostka gdyż zlewa się ona np. z obiektami w otoczeniu.