Bernadeta Okołowicz
Paulina Kusztelak
Bioinformatyka, II rok
Paradygmaty Programowania

Symulator gry w Blackjack

Blackjack jest popularna grą karcianą w kasynach. Celem gracza jest pokonanie krupiera, osiągając sumę punktów jak najbliższa 21, ale nie przekraczającą tej wartości. Zasady gry są ściśle ustalone - każda karta ma przypisaną wartość. Jeśli suma kart na ręce danego gracza przekroczy liczbę 21 gracz automatycznie przegrywa (niezależnie od sumy kart drugiej osoby), a gra się kończy. Najbardziej porządnym układem kart jest tzw. „Blackjack", czyli suma kart równa 21. Jeżeli któremuś z graczy uda się uzyskać Blackjack to wygrywa, a gra kończy się automatycznie.

Do gry w Blackjacka wykorzystywana jest standardowa talia kart. Karty o numerach od 2 do 10 posiadają wartość równą ich numerowi. Każda z kart - walet, dama i król - ma wartość 10. As może mieć wartość 1 lub 11, w zależności, co jest korzystniejsze dla gracza. Początkowo as zawsze ma wartość 11, a w przypadku, gdy suma na ręce gracza będzie wynosić więcej niż 21 i będzie on posiadał na ręce asa o wartości 11, to program zamieniać będzie wartość asa na 1. Kolor karty nie ma wpływu na jej wartość, dlatego nie bierzemy go pod uwagę.

W naszym symulatorze gry udział bierze gracz i krupier (komputer). Obydwaj gracze dostają po dwie karty. Wartość pierwszej karty obu uczestników jest widoczna dla wszystkich, natomiast wartość drugiej karty jest jawna tylko dla jej posiadacza. Po otrzymaniu obu kart gracz podejmuje decyzje, czy chce dobrać kolejna kartę. Po każdym dobraniu karty gracz ma możliwość podjęcia decyzji, czy chce dobrać następna. Podczas gry krupier nie podejmuje żadnych decyzji. Od działań gracza zależy jak potoczy się cała gra. Jeśli gracz nie będzie chciał dobrać karty, następuje podsumowanie. Jeśli suma kart krupiera jest mniejsza niż 17 to dobiera on następną kartę. Gra kończy się remisem, jeśli obydwaj gracze maja taka samą sumę kart. Wygrywa gracz który jest bliżej liczby 21, nie przekraczając jej.

Nasz symulator gry jest uruchamiany w terminalu poprzez otwarcie skryptu naszego programu. W grę można zagrać kolejny raz bez konieczności ponownego uruchamiania programu.

Opis funkcji użytych w kodzie z algorytmem działania

Funkcja „tasuj_talie()" posiada zinicjalizowaną talię kart od 2 do Asa, a następnie tasuje ją losowo przy użyciu funkcji „.shuffle()" z modułu random. Następnie funkcja zwraca przygotowaną i potasowaną talię.

Funkcja „rozdaj_karty(talia)” przyjmuje talie zwróconą z funkcji tasującej jako argument. Następnie za pomocą metody „.pop()” usuwa dwie losowe karty z talii(listy) i zwraca je jako listę za pomocą zapisu [.pop(), .pop()].

Funkcja „reprezentuj_reke(reka)” używamy do konwersji listy kart, zwróconych przez funkcję rozdającą na string, używamy do tego funkcji „join” używając oddzielenia jako przecinek. Funkcja przyjmuje listę kart rozdanych i zwraca string, który reprezentuje rękę w bardziej czytelny dla użytkownika sposób.

Funkcja „oblicz_sume(reka, wartosci_kart)” przyjmuje listę rozdanych kart oraz słownik z wartościami kart przypisany do zmiennej „wartosci_kart”. Oblicza ona sumę punktów zgodnie z zasadami blackjacka, uwzględniając elastyczność asa (1 lub 11). Sumę obliczamy za pomocą funkcji „reduce()”, jest to funkcja z modułu „functools”. Jej działanie polega na aplikowaniu funkcji redukującej do elementów sekwencji w celu uzyskania jednej wartości. W tym przypadku sekwencją są karty na ręce gracza.

Następnie w funkcji „reduce()” używamy funkcji anonimowej (lambda), która opisuje, jak dokładnie redukować (kumulować) elementy sekwencji. Ma dwie zmienne, acc (akumulator - zmienna, która przechowuje wynik kumulacji. W kontekście tego wyrażenia, jest wartością, która rośnie w miarę iteracji przez karty. Początkowo jest to 0, a następnie dla każdej karty dodawana jest jej wartość, aktualizując acc) oraz karta (lista kart na ręce gracza, przez którą funkcja „reduce” będzie iterować), a następnie zwraca sumę wartości związanej z kartą zdefiniowaną w słowniku „wartosci_kart”. Funkcja ta jest wywoływana dla każdej karty na ręce gracza.

Jeśli suma na ręce gracza przekracza 21 i posiada on jednocześnie asa o wartości 11, to funkcja zamienia jego wartość na 1.

Funkcja decyzja_gracza() pyta gracza, czy chce dobrać kolejną kartę. Wartością zwracaną jest wartość typu boolean - „True”, tylko w przypadku, gdy odpowiedź rozpoczyna się od 't', stosujemy tutaj metodę startswith(‘t’). W przeciwnym wypadku funckja zwraca „False”.

Funkcja ponownie() dopytuje gracza, czy chce zagrać ponownie. W tym przypadku również przy użyciu metody startswith(‘t’) sprawdzamy, czy gracz wybiera odpowiedź „tak” i zwraca wartość „True” , jeśli odpowiedź jest „nie”, to zwracane jest „False”.

W funkcji graj() pobieramy jako argumenty słownik z wartościami kart, potasowaną talię oraz rękę gracza i krupiera. Na początku rozgrywki okazujemy graczowi pierwszą kartę krupiera pobierając pierwszą kartę z listy „reka_krupiera”, pokazujemy również karty samego gracza za pomocą funkcji konwertującej listę na string przyjmującej jako argument zmienną z funkcją rozdającą karty dla gracza, wyświetlana jest także suma kart gracza. Tutaj sprawdzamy również, czy w tym

momencie suma na ręce gracza nie jest równa 21, gdyż oznaczałoby to natychmiastową wygraną, a także czy suma nie wynosi więcej niż 21, gdyż to oznaczałoby przegraną. Następnie za pomocą funkcji „decyzja_gracza()” pytamy go czy chce dobrać kolejną kartę oraz decydujemy o dobieraniu kart przez krupiera, jeśli suma na jego ręce jest mniejsza niż 17. Na koniec wyświetlamy wynik gry oraz pytamy, czy gracz chce zagrać ponownie za pomocą wywołania funkcji ponownie() w bloku `if __name__ == "__main__".

Pętla Główna: w bloku „if __name__ == "__main__"” sprawdzamy, czy plik jest uruchamiany jako program główny (oznacza to, że plik jest bezpośrednio uruchamiany jako samodzielny program) czy importowany jako moduł. Dzięki temu zapisowi możemy wykorzystać nasz program w innych plikach. Uznałyśmy to za przydatny zapis, w przypadku gdybyśmy chciały w przyszłości dalej pracować nad kodem, ale w kontekście innej gry. Jeśli z kolei uruchamiamy plik jako program główny, pętla „while True” umożliwia graczowi kontynuację gry lub wyjście z programu.

Algorytm Gry

1. Inicjalizacja talii kart i tasowanie:

Funkcja „tasuj_talie()” inicjalizuje talię kart, które są reprezentowane jako lista wartości od 2 do A powtórzone czterokrotnie. Tasowanie(mieszanie) kart odbywa się przy pomocy funkcji „shuffle()” z modułu random, która losowo przemieszcza elementy w sekwencji takiej, jak lista. Działa na miejscu, co oznacza, że modyfikuje oryginalną sekwencję i nie zwraca nowej.

2. Rozdanie kart graczowi i krupierowi:

Funkcja „rozdaj_karty(talia)” przyjmuje talię kart, a następnie przekazuje dwie karty dla każdego z uczestników gry, zwracając listę dwóch kart (do wyciągania z talii dwóch kart użyta jest metoda „.pop()” powtórzona dwukrtonie).

3. Obliczanie sumy punktów dla ręki gracza:

Funkcja „oblicz_sume(reka, wartosci_kart)” przyjmuje rękę gracza lub krupiera(reka) oraz słownik wartości kart(wartosci_kart). Sumuje punkty dla danej ręki, a jeśli suma przekracza 21 i ręka zawiera asa o wartości 11, zmienia wartość asa na 1. Do sumowania używana jest funkcja reduce, która zawiera w sobie funkcję anonimową lambda z określonym akumulatorem(acc). Działanie tego zapisu zostało szczegółowo wyjaśnione w opisie funkcji.

4. Decyzja gracza o doborze kolejnej karty:

Funkcja „decyzja_gracza()” prosi gracza o podjęcie decyzji, czy chce dobrać kolejną kartę. Zwraca `True`, jeśli gracz chce dobrać, a `False` w przeciwnym razie.

Uzyskujemy to przy pomocy metody, która jest używana do sprawdzania, czy dany ciąg znaków rozpoczyna się od określonego prefiksu (innej sekwencji znaków) - „startswith()".

5. Odbycie rozgrywki:

W funkcji „graj()" inicjalizuje się słownik wartosci_kart, który przechowuje oznaczenie karty, np. '2' oraz jej wartość jako klucz, np. 2 - '2':2, talia, której wartością jest wywołana funkcja „tasuj_talie()", ręka gracza i ręka krupiera, których wartościami jest ta sama funkcja „rozdaj_karty(talia)", z argumentem talia.

Następnie przy użyciu głównej pętli „while True", która będzie działała, dopóki nie zostanie przerwana przez jedno z warunków `break`, wyświetlamy karty gracza przy użyciu funkcji „reprezentuj_reke(reka)", która wykorzystuje funkcję „join()" do łączenia elementów z listy, iterując po jej elementach (pętla for), w jeden ciąg znaków, ciąg ten oddziela przecinkiem. Dzięki temu zamiast wyniku w postaci ['2', '3'] otrzymujemy go w postaci 2,3, wyświetlamy tu również sumę kart gracza wywołując funkcję „oblicz_sume" i jako argumenty podajemy rękę gracza (reka_gracza) oraz słownik wartosci_kart. Potem wyświetlana jest pierwsza karta krupiera, która z listy reka_krupiera pobiera pierwszy element z listy (indeks 0 - [0]). Następnie przy użyciu warunków (if, elif) sprawdzamy, czy suma na ręce gracza („oblicz_sume(reka_gracza, wartosci_kart") jest równa 21 (==21), jeśli tak, to wyświetlamy informację o wygranej i funkcja kończy swoje działanie, w drugim warunku sprawdzamy, czy suma kart gracza („oblicz_sume(reka_gracza, wartosci_kart") przekroczyła 21 (>21), jeśli tak, wyświetlana jest informacja o przegranej i funkcja kończy działanie. W kolejnym warunku (if) pytamy gracza, czy nie chce już dobrać więcej kart (`decyzja_gracza()` zwraca `False`), jeśli tak, pętla jest przerywana. Jeśli gracz chce dobrać kartę, dodawana jest nowa karta do ręki gracza („reka_gracza.append(talia.pop())").

Po wyjściu z pętli gracza, następuje ruch krupiera. Krupier dobiera karty dopóki suma kart w jego ręce jest mniejsza niż 17 - „oblicz_sume(reka_krupiera(talia.pop())". Wykonujemy to za pomocą pętli „while", dopóki warunek nie zostanie spełniony.

Po ruchach krupiera, wyświetlane są ostateczne karty gracza („reprezentuj_reke(reka_gracza"))i krupiera („reprezentuj_reke(reka_krupiera)") oraz ich sumy („oblicz_sume(reka_gracza/reka_krupiera, wartosci_kart)"). Następnie przy użyciu warunków (if, elif, else) sprawdzane są różne scenariusze:

- Czy krupier 21 punktów („oblicz_sume(reka_krupiera, wartosci_kart) > 21"), jeśli tak, to gracz otrzymuje informację, że wygrał.
- Czy gracz ma wyższą sumę kart na ręce niż krupier (or „oblicz_sume(reka_gracza, wartosci_kart) > oblicz_sume(reka_krupiera, wartosci_kart)"), jeśli tak, to gracz otrzymuje informację, że wygrał.

- Czy jest remis, czyli czy suma kart gracza na ręce jest równa sumie kart krupiera („oblicz_sume(reka_gracza, wartosci_kart) == oblicz_sume(reka_krupiera, wartosci_kart”), jeśli tak, to gracz widzi informację o remisie.
- W innym wypadku gracz otrzymuje informację, że przegrał rozgrywkę.

Na końcu używany jest standardowy idiom („if __name__ == "__main__":”) używany do sprawdzania, czy plik jest uruchamiany jako program główny, czy jest importowany jako moduł w innym skrypcie. Działa to na zasadzie, że jeśli plik jest uruchamiany bezpośrednio (jako program), to blok kodu wewnątrz tego warunku zostanie wykonany.`__name__`: To specjalna zmienna w Pythonie, która zawiera nazwę bieżącego modułu. Gdzie: `__main__`, to nazwa specjalna, która jest przypisana do zmiennej `__name__`, gdy plik jest uruchamiany jako program, a nie importowany jako moduł. Po sprawdzeniu, czy plik jest uruchamiany jako program główny wykorzystujemuy pętlę „while True”, co oznacza, że poniższy blok kodu będzie wykonywany w nieskończoność, dopóki warunek pętli będzie spełniony (tutaj zawsze spełniony, ponieważ warunek to po prostu `True`). W pętli z kolei wywołujemy funkcję „graj()”, której deziałanie zostało objaśnione w poprzednim kroku, następnie dzięki „if not ponownie()” sprawdzamy, czy użytkownik nie chce zagrać ponownie. Funkcja „ponownie()” zwraca wartość logiczną (True lub False) w zależności od decyzji użytkownika, jeśli użytkownik nie chce grać ponownie („not ponownie()” jest True), to instrukcja „break” kończy pętlę while i tym samym kończy działanie programu.

Warto zwrócić uwagę, że ważną koncepcją w tym kodzie jest nieskończona pętla „while True”, która umożliwia powtarzanie rozgrywki, dopóki gracz nie zdecyduje się przerwać. To zapewnia interaktywny i ciągły charakter gry, pozwalając graczowi na kontynuację rozgrywki bez konieczności ponownego uruchamiania programu.

Przykład przejścia przez kod:

1. Inicjalizacja i potasowanie talii.
2. Rozdanie dwóch kart graczowi (np. ‘6’ i ‘9’) i krupierowi (np. ‘10’ i ‘As’).
3. Gracz decyduje, czy dobrać kolejną kartę (np.”t”).
4. Dobranie kolejnej karty (np. ‘7’).
5. Gracz ma teraz na ręce karty: 6, 9, 7 (suma 22 > 21).
6. Wyświetlenie informacji o przegranej gracza.
7. Zapytanie, czy zagrać ponownie (np. ‘n’).

To jest jedno możliwe przejście przez kod. Oczywiście, istnieje wiele scenariuszy w zależności od losowości tasowania talii i decyzji gracza, co wprowadza różnorodność w przebiegu gry.

Cały nasz algorytm programu starałyśmy się zamknąć w idei programowania funkcyjnego, tzn. bez używania zmiennych globalnych i lokalnych, używając superpozycji i czystych funkcji. Niestety idea naszego programu nie pozwalała na całkowite zastosowanie się do wszystkich zasad.

Uzasadnić należy użycie zmiennych lokalnych głównie w funkcji „graj()”., ponieważ tam jest ich najwiecej. Próbowałyśmy usunąć zmienne lokalne na rzecz stworzenia funkcji, które odwzorowałyby ich sens, jednak nie przyniosło to pożądanego rezultatu. Użycie funkcji zamiast przypisania do zmiennych w przypadku „talia”, „ręka_gracza” i „ręka_krupiera” powoduje ponowne wywołanie funkcji przy każdym użyciu - czyli z każdym użyciem funkcji tworzy się nowa talia, inne karty znajdują się na ręce gracza oraz krupiera. Mija się to z celem naszej gry.

W funkcji „tasuj_talie()” również użyłyśmy zmiennej lokalnej „talia”. Nie znalazłyśmy żadnego innego sposobu, aby ta funkcja działa poprawnie bez tego elementu. Tutaj potrzebujemy aby argumentem tej funkcji była lista, ponieważ opieramy się w innych częściach programu na zmianach dokonywanych właśnie na listach np „.pop”.

Aby nasz symulator gry mógł działać poprawnie konieczne również było użycie użycie warunków oraz pętli. Pętle są konieczne, aby w grę można było grać bez konieczności ponownego uruchomienia programu. Instrukcji warunkowych używałyśmy głównie, aby sprawdzać wartość kart na ręce gracza. Jest to warunek konieczny do gry w blackjacka, która opiera się na nie przekroczeniu sumy 21.

Podsumowanie

Podsumowując projekt z programowania funkcyjnego, jesteśmy zdania, że jest to ciekawy i czytelny sposób na pisanie programów. Definiowanie małych i prostych funkcji ułatwia zrozumienie pojedynczych kroków działania algorytmu. Jest to również sposób bardziej uniwersalny, taką funkcje można użyć w każdym przyszłym programie. Nie korzysta ona ze zmiennych lokalnych, tylko pobiera parametry i zwraca argumenty. Niestety nie każdą funkcję da się tak zamienić, aby zachowała swój pierwotny sens. Ponadto musimy tutaj uwzględnić, że w niektórych przypadkach brak używania zmiennych loklanych może prowadzić do zmniejszenia czytelności kodu - należy przyznać, że zapis „if suma < 21” jest bardziej czytelny niż zapis „if oblicz_sume(rozdaj_karty(talia), wartosc_karty(talia) <21”. Niemniej jednak uważamy, że zaprezentowany kod można byłoby poprawić w taki sposób, aby w większym stopniu spełniał zasady programowania funkcyjnego, lecz uważamy, że dalsze próby naszego rozwiązania były mniej funkcjonalne i czytelne, a ponadto powodowały dużo niezrozumiałych dla nas, na tym etapie, bugów. Po kilkugodzinnych testach różnych wersji kodu podjęłysmy decyzję o tym, że obecna w dużym stopniu spełnia zasady przedstawionego zadania, a przy tym kod jest czytelny i działa w oczekiwany przez nas sposób.