PIPR Projekt “Kółko i krzyżyk na sterydach”

Cel projektu

Aplikacja “Kółko i krzyżyk na sterydach” ma pozwalać zagrać z graczem komputerowym, którego ruchy są losowe lub takim, który wykonuje ruchy według prostych reguł.

Opis projektu

To gra planszowa składająca się z dziewięciu plansz takich jak w standardowym kółko i krzyżyk ułożonych w siatkę 3 × 3. Gracze na zmianę grają na mniejszych planszach w kółko i krzyżyk, dopóki jeden z nich nie wygra na większej planszy.

Zaimplementowane klasy

SingleBoard – plansza zawierająca pola, w których możliwe jest wstawienie znaku X lub O

BigBoard - jest to tablica złożona z obiektów klasy Single_Board i ma swoją reprezentację w tej klasie

Game – przyjmuje atrybuty (gracza i gracza komputerowego), ustala kto ma zaczynać i przypisuje znak, który nie został wybrany przez człowieka komputerowi

Player – klasa reprezentująca gracza, posiada atrybuty takie jak znak czy nazwę, która domyślnie przyjęta jest jako ‘gracz’

AI – dziedziczy po klasie player; wykonuje ruchy na podstawie prostych reguł, posiada funkcje obronną i atakującą, decyzja, którą funkcję ma wybrać jest podejmowana na podstawie ostatniego ruchu wykonanego przez użytkownika

Sign – zawiera znaki do wyboru przez gracza

Najważniejsze funkcje

Make_dict_of_answers – tworzy słownik odpowiedzi AI na ruchy gracza

Stałe

DIMENSION – odpowiada za “wielkość” planszy, to od niej uzależnionych jest znaczna część funkcji, po edycji tej zmiennej możemy uzyskać np. planszę 16x16 czy 25x25

Instrukcja użytkowania

Uruchomienie programu:

Na początku należy zainstalować bibliotekę colored (np. poprzez wpisanie komendy pip install colored w terminalu). Po wykonaniu tej czynności uruchomienie pliku play.py spowoduje rozpoczęcie działania programu.

Przed rozgrywką:

Po uruchomieniu programu należy wprowadzić swoje imię i zatwierdzić je enterem, naciśnięcie entera bez wpisania nazwy ustawi ja na domyślną (‘gracz’). Po wprowadzeniu nazwy należy wybrać znak, którym użytkownik chce się posługiwać - czyli kółko lub krzyżyk. Wprowadzenie błędnego znaku spowoduje powtórzenie zapytania, natomiast znak wpisany jako mały zostanie zamieniony na duży. Następnie użytkownik zostanie poproszony o wprowadzenie ‘losowy’ jeśli chce grać z użytkownikiem losowym lub dowolnej innej sekwencji znaków, jeśli chce zagrać z graczem, który stawia opór.

Cel gry:

Celem gry jest wygranie (zgodnie z zasadami kółka i krzyżyk) 3 dużych kwadratów (3x3 oddzielonych grubymi liniami) w jednej kolumnie, rzędzie lub po przekątnej na dużej planszy.

Rozgrywka:

Użytkownik zostanie poproszony o wprowadzenie numeru dużego pola (kwadratu 3x3, są one oddzielone grubymi liniami), a później małego pola w wybranym kwadracie. Zostanie wstawiony znak gracza w wybrane pole i znak komputera w pole wybrane przez gracza komputerowego. Znak gracza jest zawsze w kolorze zielonym, natomiast komputera w czerwonym. Gra kończy się w momencie, w którym zgodnie z zasadami standardowego kółka i krzyżyk, gracz w jednej kolumnie, wierszu lub po przekątnej ułoży 3 duże pola (oddzielone pogrubionymi liniami). Wtedy zwycięzcą zostaje gracz, któremu udało się tego dokonać i wyświetla się komunikat: “Zwycięzcą został/a *nazwa gracza*”

Refleksje

Zgodnie z początkowymi założeniami utworzyłem klasy opisane powyżej. W stosunku do planowanego rozwiązania zmieniłem to, że zdecydowałem się na rozbicie dużej planszy na 9 małych. Umożliwia to o wiele łatwiejsze sprawdzanie i wybranie właściwego kwadratu, w którym ilość pól zajętych przez gracza komputerowego jest największa. Podczas tworzenia programu okazało się, że przy pewnych ustawieniach gracz komputerowy nie wykonuje ruchu, co wywoływało zatrzymanie programu. Problemem w tym przypadku było to, że pominąłem dodawanie zapełnionych, ale nie wygranych przez żadnego z graczy pól jako pełne. W związku z tym po wybraniu dużego pola, nie było możliwości wstawienia znaku w żadne pole, więc komputer nie wykonywał ruchu. Rozwiązaniem okazało się wpisanie w reprezentacji dużego kwadratu jako małego w pole, które jest zapełnione ‘full’.

Gdybym teraz podchodził do napisania projektu nie zdecydowałbym się na zwracanie ilości pól w danej sekwencji (kolumna, rząd, przekątna) w postaci listy, ponieważ utrudnia to sprawdzanie zależności między polami, które należą do różnych sekwencji. Z tego powodu w

funkcji obronnej AI, gdy gracz zostawi pole, które należy do obu sekwencji, gracz komputerowy nie stawia znaku, aby go zablokować, bo uważa, że nie jest w stanie obronić już tej małej planszy. Zamiast tego zdecydowałbym się na słownik list z dopasowanymi sekwencjami do danego pola, co ułatwiłoby mi identyfikację i pozwoliło wyeliminować niedoskonałość.