Pretrage

Potrebno je izgraditi graf stanja i akcija za zagonetku “Vuk, Ovca, Kupus”. Zagonetka ima početno stanje gdje su sva tri objekta (Vuk, Ovca i Kupus) na lijevoj strani obale rijeke. Na lijevoj strani se nalazi i brod koji može prebacivati jedan po jedan objekt preko rijeke. Vuk i Ovca ili Ovca i Kupus ne smiju u nijednom stanju biti ostavljeni na jednoj strani obale (bez prisutnosti broda). Ovakva stanja se smatraju konačnim i gubitničkim stanjem. Rješenje zagonetke je da se dosegne konačno stanje gdje su sva tri objekta na desnoj strani rijeke.

U grafu će čvorovi biti stanja zagonetke odnosno detaljan opis situacije: gdje su objekti (lijeva obala, desna obala, u brodu) i gdje je brod (lijeva ili desna obala). Veze među čvorovima su mogućnost prelaska iz jednog stanja u drugo pomoću jedne od akcija:

- ukrcaj objekt sa bliže obale
- prebaci brod na drugu obalu
- iskrcaj objekt na bližu obalu

Stanje treba predstaviti klasom (Stanje) koja sadrži opis stanja i iduće metode:

- Konstruktor koji inicijalizira stanje na početno stanje iz opisa stringom “VOKB || ----“
- as_string() ili __str__() – vraća opis stanja stringom, prikladan za ispis na ekran, a može se koristiti i kao ključ u algoritmima
- all_actions() – vraća sve moguće akcije za trenutno stanje
- next_states() – vraća listu svih idućih stanja koja se mogu doseći iz trenutnog stanja pomoću akcija
- is_solved() – vraća True ako je stanje rješenje zagonetke (“---- || VOKB”)
- is_terminal() – vraća True ako je stanje konačno (izgubljeno ili riješeno)
- action() – izvodi akciju i mijenja stanje (zagonetke)
- undo_action() – poništava akciju i vraća stanje u stanje prije akcije
- copy() – vraća kopiju stanja (možete koristiti *deepcopy* iz *copy* bilioteke)

Najbolje je opisati stanje pomoću što manje varijabli imajući u vidu da će trebati implementirati gornje metode. Gornjim metodama dodajte prikladne parametre, a u klasu dodajte članove i metode koje smatrate prikladnim.

Zatim treba implementirati iduće algoritme:

a) Napisati funkciju generate() koja rekurzijom generira sva moguća stanja i sprema ih u rječnik koji opisuje graf stanja. Svaki par u rječniku će imati opis stanja stringom kao ključ i objekt klase Stanje kao vrijednost. Kako je ukupan broj stanja mali, možete ispisati rječnik da bi provjerili ispravnost implementirane igre. Koristite action() / undo_action() za šetanje po stanjima. Ovaj graf služi samo kao provjera ispravnosti implementacije same igre, a preostali algoritmi se pišu neovisno o njemu.

b) Napisati funkcije solution_dfs() i solution_bfs() koje pretražuju prostor stanja (pretragom u širinu i u dubinu) i generira stazu od početne pozicije do rješenja (lista stanja). Da bi se na kraju mogla generirati lista stanja potrebno je, u pomoćnoj strukturi podataka (stog ili red) i u skupu posjećenih stanja, pamtiti roditelja svakog stanja. Pogledajte primjere s predavanja za sličnu zagonetku.

c) Napisati funkciju za Best First Search. BestFS pretraga će se oslanjati na heuristiku koja jednostavno broji koliko je objekata (vuk, ovca, kupus) na desnoj strani rijeke.

Samu igru možete vidjeti na:

http://coolmath-games.com/Logic-wolfsheepcabbage/index.html