Struktury na zapamatanie si instrukcii:

Mame mnozstvo funkcii, ktore treba zavolat -> zapamatne v nejakom strome pre lahsi pritup, spolu s premennymi.
Ked hladam funkciu -> v poslednom uzle je odkaz na strukturu, ktora drzi vsetky instrukcie, ktore sa maju vykonat, defaultne null, aby sa odlisila od toho, ked bude nahodou rovnakym menom oznacena funkcia aj premenna.
Chcem volat funkciu : vyhladam si strome dane meno a zistim, ze nie je struktura NULL -> potom ju naloadujem – nemiesto volania CALL skopirujem kompletne jej instrukcie na stack
chcem zavolat premennu (naloadovat) -> vyhladam si v strome a zistim, ci je 'aktivna', to jest bola v danom bloku deklarovan, ak nie, tak ju v danom bloku deklarujeme t.j. 'var' nam vobec netreba, lebo akonahle nejaky identifikator pouzijeme, vytvori sa a priradi sa mu typ podla toho, co donho nacpeme. Pricom plati, ze prvy vyskyt je rovnake ako deklarovanie, takze existuju lokalne premenne, ktore sa na konci kazdeho bloku ({...}) vymazu.
Loadovanie znamena, ze sa n ciselny stack prida pointer na danu strukturu premennej.
V pripade volania pola sa zisti to iste, akurat ze moze mat ako vedlajsi parameter cislo elementu (v pripade viacnasobneho pola sa toto cislo dopocitava, jelokoz to v strukture ukladam ako jedno velke pole)
Preddefinovana funkcia je funkcia main, ktora je unikatna a neprepisatelna.

Co su instrukcie:

Najelementarnejsie akcie, ktore sa daju vykonat, t.j. napriklad vyzdvihnutie premennej, pricitanie, ulozenie a pod.
Poznam dva typy instrukcii :
- nepotrebuju nic (ako napriklad move, shoot, tie si svoje parametre zoberu zo ciselneho stacku, ktory je oddelene a obsahuje len aktualne hodnoty a mena tych premennch, co boli naloadovane), pretoze to, co potrebuje, si musi dynamicky naloadovat(hodnoty inych premenncyh)
- maju staticke parametre (napriklad aku funkciu zavolat alebo aky premennu naloadovat, potrebujem to vediet, skor ako s tym zacnem nieco robit)

za povsimnutie stoji, ze u instrukcii teda potrebujem iba cislo, kam by sa eventualne skocilo (if, repeat...), samotnu instrukciu a string, potazne cislo (takze staci instrukcia, cislo, string ako struktura)

Instrukcie

Vsetky operacie s cislami + vsetky funkcie -> nepotrebuju nic pridavne, vsetko si beru zo ciselneho stacku
loadovanie premennych a ich ulozenie -> string, potrebuju vediet, co loaduju
zistovanie vlastnosti objektu -> podobne : (see(3).IsMoving())
Dalsie zvlastne instrukcie:
-label
-jump (pre if, else, while, repeat, for -> potrebuju vediet, kam skocit)
-clean_local_variable - po kazdom skonceni sa vymaze premenna, ktora bola prvy krat pouzite v tomto bloku, aby sa zamedzilo zbytocnemu hlteniu pamate bota, ktory ju ma obmedzenu.

Uchovavanie premenych:
Strom, vytvori sa pri inite a nemaze sa, pokial nepresiahne urcitu velkost, ktora sa pocita externe (teda bot si pamata uz moc premennych a akoby nema 'miesto' pre dlasiu, hoc by aj fyzicky

mohlo byt) – bot si proste nieco zmaze a tym strati informacie, co bolo v tej premennej,co sa zmazala, popripade strati funkciu a nebude ju uz vykonavat (ta sa neda loadnut podobne ako premenna, resp. nepozname 'neinicializovanu funkciu')

Pri pristupe na tieto premenne bud bot skonci alebo (lepsie) sa mu dostane neinicializovana hodnota.

U kazdeho uzlu je napisane: - ci je premenna aktivna (podal toho sa pozna, aky typ to je (point, object alebo integer, potazne este neinicializovana) -> globalna premenna bude aktivna stale,

- velkost zanorenia, v ktorom bola tato premenna vytvorena (dolezite pre mazanie lokalnych premennych), pred kazdym blokom bude specialna intrukcia, ktora nestoji vlastne nic, a to zvys_zanorenie a zniz_zanorenie;

- samotne hodnoty premennej (odtial sa bude loadovat)

Rozoznavanie vnorenej premennej -> v pripade, ze rekurzivne volam istu funkciu, v strome sa mi prida jej strom premenych pod funkciu, v ktorej som prave zanorena – pri vystupe sa tento strom nemusi mazat, iba oznacit za 'neplatny', spoplatni sa zase v okamihu, ked danu funkciu v rovnakom kontexte zavolame.
V pripade skoncenia bloku sa vsetky premenne tu deklarovane musia znehodnotit -> to sa spravit tak, ze ako poslednu instrukciu po bloku este pridam instrukciu 'clear_local', ktora od korena danej funkcie prejde vsetky premenne, ktore sa vytvorili s indexom 'vnorenia', aky je maximalne nastaveny (pri kazdom zaciatku bloku sa zvysuje, po konci znizuje, udrzuje sa ako globalna premenna)

Strom sa bude dotvarat aj pocas behu bota -> pri volani nejakej uzivtalom definovanej funkcie sa do stromu prida (ak tam este nie je) prefix aktualnej funkcie, v ktorej sme (okrem main) , + '.', a deep_copy stromceka funkcie-> presne kopia toho stromu. Premenne vo funkcii su oznacene teda v hlavnom strome ako 'nazov_funkcie.meno_premennej'. Predide sa tak zistovaniu, ci je premenna lokalna, potazne pri rekurzivnom volani tej istej funkcie nie je problem s menom premennej.

Ako sa hlada premenna, ktoru ma loadovat -> bude je to lokalna vo funkcii alebo globalna uplne -> bud zacnem uplne od korena a idem po strome alebo zacnem vnutri funkcie. Ak sa hlada funkcia, staci mi prehladat iba koren, jelikoz vnorene funkcie nepodoporujem (ani nevidim velky zmysel)

Samotne implementacia priebehu boja:
fronta poziadavkov na plochu, co sa deje -> pouzitie ale viacero threadov, aby pri pryslani ineho bota sa mohli premyslat / konat aj ostatni.

Rekurzia:
pocet rekurzii radsej bude odmedzeny tym sposobom, ze bude obmedzena velkost stacku instrukcii -> povedzme, ze 1000 instrukcii v hlavnom kode, tym sa omedzi rekurzia a nebude sa vsetko pchat do globanych premennych, jelikoz te by zabrali

Samotne prevadzanie instrukcii:
stack bude vzdy naalokovany na maximalnu moznu velkost. najskor sa do stacku prekopiruje kompletne cela funkcia 'main' a zacnu sa vykonavat instrukcie.
V pripade, ze sa bude odovzdavat aj parameter referenciou, bude na konci kazdej uzivatelom definovanej funkcie alebo procedury dalsia instrukcia, ktora hovori, zev pripade,ze boli parametre var-ovane, nech sa zisti, ako boli volane potom tam daj hodnotu.