**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA**

Ilkovičova 2, 842 16 Bratislava 4

**2022/2023**

Umelá inteligencia

**Zadanie č.1**

**Problém a)**

**Cvičiaci: Ing. Martin Komák, PhD. Vypracoval: Šimon Valíček**

**Čas cvičení: Streda 15:00 – 16:50 AIS ID: 116320**

Šimon Valíček AIS ID: 105345

Obsah

**Bláznivá križovatka................................................................................................................................3**

1. **Úvod........................................................................................................................................3**
2. **Implementácia.........................................................................................................................3**
   1. *Zadefinovanie uzlu............................................................................................................3*
   2. *Zadefinovanie autíčok.......................................................................................................4*
   3. *Zadefinovanie hracieho poľa.............................................................................................4*
   4. *Umiestnenie autíčok na hracie pole..................................................................................5*
   5. *Pohyb autíčok....................................................................................................................5*
   6. *Rekurzívna funkcia createLevel().......................................................................................6*
   7. *Prehľadávanie stromu.......................................................................................................9*
   8. *Výpis jednotlivých krokov.................................................................................................10*
   9. *Main funkcia....................................................................................................................12*
   10. *Prehľad jednotlivých modulov.........................................................................................12*
3. **Testovanie..............................................................................................................................13**
   1. *Úvod.................................................................................................................................13*
   2. *Manuál k testovaniu........................................................................................................14*
   3. *Prvé testovanie................................................................................................................15*
   4. *Druhé testovanie.............................................................................................................16*
   5. *Tretie testovanie.............................................................................................................17*
   6. *Štvrté testovanie............................................................................................................ 19*
4. **Záver......................................................................................................................................20**
   1. *Problémy pri implementácii.............................................................................................20*
   2. *Výsledok pozorovania......................................................................................................20*
   3. *Záverečné slová...............................................................................................................20*

**Bláznivá križovatka**

1. **Úvod**

Úlohou je riešenie problému **Bláznivá križovatka**. Hracie pole predstavuje mriežka o veľkosti 6x6 políčok, pričom políčko môže byť buď prázdne, alebo sa tam môže nachádzať autíčko. Autíčka sú na hracom poli rozmiestnené buď vertikálne, alebo horizontálne, pričom autíčko môže zaberať buď 2 alebo 3 políčka. Každé autíčko je reprezentované nejakou farbou a cieľom hry je dostať červené autíčko von z križovatky.

Úlohu som sa snažil riešiť tak, že som si zadefinoval uzol, ktorý uchováva práve jednu kombináciu rozmiestnení autíčok na hracej ploche. Každý taký uzol ďalej uchováva odkazy na uzly, ktoré predstavujú všetky možné kombinácie pohybov autíčok na hracej ploche. Takýmto spôsobom som zostrojil strom uzlov, ktorý obsahuje všetky možné polohy rozmiestnenia autíčok na hracej ploche, pričom žiadny uzol sa v strome neopakuje. Po vytvorení takéhoto stromu sa vo svojom riešení snažím dohľadať uzol, ktorý prestavuje úspešné vyriešenie tohto algoritmu, pričom takýchto uzlov môže byť viac. Na prehľadanie stromu používam algoritmy prehľadávania do hĺbky a do šírky a porovnávam čas, za ktorý jednotlivé vyhľadávania prebehli. Ďalej vypisujem cestu k takémuto uzlu, a to spôsobom takým, že po nájdení konkrétneho uzla použijem spätné vyhľadávanie, až kým sa nedostanem ku koreňu stromu. V riešení pracujem s viacerými testovacími súbormi.

1. **Implementácia**
   1. **Zadefinovanie uzlu**

Uzol som si vytvoril nasledovne, kde ***data*** predstavuje práve jedno konkrétne hracie pole, ***parent*** predstavuje odkaz na rodiča daného uzla, ***colo*r** predstavuje farbu autíčka, ktoré sa v danom kroku pohlo, ***arrow*** predstavuje šípku, ktorá určuje smer, ktorým sa dané autíčko pohlo a ***counter*** predstavuje počet políčok, o ktoré sa pohlo.

Obrázok, na ktorom je text

Automaticky generovaný popis

Takto vytvorené uzly následne vkladám do stromu. Vytvorenie uzlu, jeho vloženie do stromu, ale aj zostavenie samotného stromu prebieha v module s názvom **nodeSystem.js**

Funkcie z tohto modulu následne volám v testovacom module, ktorý zabezpečuje chod programu.

* 1. **Zadefinovanie autíčok**

V mojej implementácii hlavolamu som s autíčkami pracoval ako s objektmi. Najprv som si vytvoril funkciu **Car** na vytváranie autíčok, kde som si zadefinoval všetky jeho premenné.

Kód vyzerá nasledovne: Obrázok, na ktorom je text

Automaticky generovaný popis

Následne som si vytvoril jednotlivé autíčka a uložil ich do poľa. Jeden objekt autíčko obsahuje nasledujúce premenné:

***color*** – určuje farbu autíčka na hracom poli

***length*** – určuje dĺžku autíčka a teda predstavuje jeho veľkosť na hracom poli (2 alebo 3)

***row*** – predstavuje x-ovú súradnicu začiatku autíčka na hracom poli

***column*** – predstavuje y-ovú súradnicu začiatku autíčka na hracom poli

***horizontal*** – nadobúda hodnotu true alebo false v závislosti od toho, či je autíčko umiestnené horizontálne, alebo vertikálne

Následne som si celé pole autíčok exportoval, aby som s ním mohol pracovať v iných moduloch. Vyššie zobrazený kód sa nachádza v priečinku **Cars**, ktorý obsahuje jednotlivé moduly predstavujúce rôzne rozloženia autíčok na hracom poli. Tento konkrétny kód je príklad zo zadania a nájdeme ho v moduli **cars.js**, ktorý sa nachádza vo vyššie spomínanom priečinku.

* 1. **Zadefinovanie hracieho poľa**

Prázdne hracie pole som si zadefinoval ako dvojrozmerné pole o veľkosti 8x8, nakoľko som si vymedzil aj okraje tohto hracieho poľa.

Kód vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Číslo ***1*** predstavuje okraj hracej plochy, číslo ***-1*** predstavuje východ z križovatky/hracej plochy, ku ktorému sa snažíme dopracovať s červeným autíčkom. Číslo ***0*** predstavujeprázdne políčko na hracej ploche.

Prázdne hracie polia sa nachádzajú v priečinku **Boards**, pričom konkrétne vyššie zobrazené hracie pole je uložené v moduli **board.js** v spomínanom priečinku, odkiaľ ho následne exportujem.

* 1. **Umiestnenie autíčok na hracie pole**

Po zadefinovaní a vytvorení hracieho poľa a jednotlivých autíčok je potrebné umiestniť ich na hraciu plochu. Nakoľko existuje viacero kombinácií autíčok a hracích polí, ktoré k sebe pasujú, vytvoril som si funkciu, ktorá mi ich automaticky umiestňuje na plochu podľa toho, ktoré polia autíčok a hracích plôch importujem do tohto modulu.

Kód vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Táto funkcia sa nachádza v moduli s názvom **startingBoard.js**.

* 1. **Pohyb autíčok**

Ako som už v úvode spomínal, každé autíčko má pevne zadefinované, akým smerom je otočené, a teda tento smer nie je možné meniť. Z toho vyplýva, že ak je autíčko natočené vertikálne, dokáže sa hýbať len smerom hore a dolu. Naopak, ak je autíčko otočené horizontálne, dokáže sa hýbať len vpravo a vľavo. Pôvodne som chcel vytvoriť 4 funkcie, nakoľko autíčka sa môžu všeobecne pohybovať 4 smermi, no nakoniec som si uvedomil, že každé jedno autíčko sa môže pohybovať práve dvomi smermi. Preto som sa rozhodol zostrojiť iba dve funkcie, pričom každá z týchto funkcii kontroluje, akým smerom ja dané konkrétne autíčko otočené. Prvá funkcia **moveCarUpAndRight**, zabezpečuje pohyb autíčka smerom nahor, v prípade, že sa jedná o vertikálne autíčko, alebo smerom vpravo, v prípade, že sa jedná o horizontálne umiestnené autíčko. Vstupnými argumentmi funkcie sú objekt autíčka a konkrétna hracia plocha s aktuálnym rozmiestnením autíčok. Funkcia ďalej pracuje s jednotlivými súradnicami auta a hracej dosky a zabezpečuje vytvorenie ďalších polí, ktoré môžu vzniknúť pohybom práve jedného konkrétneho auta.

Obrázok, na ktorom je text, striebro, snímka obrazovky

Automaticky generovaný popis

Na rovnakom princípe funguje funckia **moveCarDownAndLeft**, akurát s tým rozdieľom, že predstavuje pohyb daného konkrétneho autíčka smerom dolu alebo vľavo, ako z názvu vyplýva. Nakoľko tieto dve funkcie sú veľmi podobné a líšia sa len v detailoch a súradniciach, dovolil som si sem túto spomínanú funkciu nedať v podobe screenshotu.

Obe tieto funkcie sa nachádzajú v module **moves.js**, odkiaľ sú následne volané hlavnou testovacou funkciou, ktorá zabezpečuje chod programu.

* 1. **Rekurzívna funckcia createLevel()**

Všetko čo som doteraz písal by nefungovalo, nebyť funkcie **createLevel.** Tá má za úlohu spojiť všetky vyššie spomínané funkcie dokopy. Jej vstupnými argumentmi sú pole autíčok, a základná hracia plocha v počiatočnom stave. Úlohou tejto funkcie je vo svojej podstate vziať hlavný vrchný uzol a vytvoriť všetky možné kombinácie pohybov autíčka, pričom vždy hýbe naraz práve jedným autíčkom. Tak vytvorí polia, predstavujúce všetky možné pohyby jednotlivých autíčok z počiatočného stavu. Dá sa teda povedať, že táto funkcia zabezpečuje zistenie všetkých možných prvých krokov a následne ich ukladá do jednotlivých uzlov, ktoré sú reprezentované ako „deti“ koreňa stromu. Následne všetky tieto polia ukladá do pola, avšak ukladá tam len tie, ktoré sa neopakujú. To je dôležité pri ďalších rekurzívnych volaniach tejto funckie, nakoľko to zníži čas potrebný na vytvorenie stromu. Následne berie z tohto pola prvý uzol a rekurzívne sa vyvolá znovu, čím vytvorí zas všetky možné kombinácie druhých krokov pri danom prvom kroku a uloží „deti“ tohto uzla ako odkazy na uzol z ktorého vychádzajú. Takto sa volá rekurzívne až pokiaľ nie je vytvorený celý strom, a teda pokiaľ nie sú vytvorené všetky možné kombinácie pohybov autíčok na danej hracej ploche.

Funkcia využíva volania funkcii, ktoré sme si zadefinovali vyššie, avšak aj množstvo pomocných funkcii, ako sú napríklad zistenie súradníc autíčka, prekopírovanie pola po prvkoch a mnohé iné. Funkcia **createLevel** sa nachádza v module **test.js,** ktorého spustením zabezpečíme beh programu.

Funckia vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text

Automaticky generovaný popis

* 1. **Prehľadávanie stromu**

Ako z môjho zadania vyplýva, na prehľadávanie stromu som mal použiť algoritmy na prehľadávanie do hĺbky a do šírky.

Algoritmus prehľadávania do hĺbky vezme koreň stromu a prehľadáva strom z jednej strany na druhú. Je to jeden z najzákladnejších argumentov, ktorý som počas svojho štúdia už neraz implementoval na iných predmetoch, a tak nebudem venovať väčšiu pozornosť jeho vysvetľovaniu, nakoľko sa snažím v dokumentácii zamerať predovšetkým na moju implementáciu daného problému.

Moja implementácia tohto algoritmu vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Nakoľko toto prehľadávanie prebieha od koreňa stromu smerom napravo, ako vstupným argumentom tejto funkcie nie je nič iné, ako koreň stromu. Funkcia využíva volanie funkcie **checkSolution**, ktorú používam ako funkciu na zistenie, či sa daný prehľadávaný uzol zhoduje s uzlom, ktorý predstavuje riešenie tejto úlohy. Táto funkcia vracia hodnotu true, ak sa dané uzly zhodujú, alebo false, ak sa nezhodujú. Ak je návratová hodnota true, riešenie sa našlo a funkcia sa ukončuje, pričom vracia odkaz na daný uzol v strome, ktorý predstavuje riešenie hlavolamu.

Funkcia **checkSolution** sa spolu s inými nachádza v module **helpers.js**, zatiaľ čo funkcia **dfs** sa nachádza v module **depthFirstSearch**.

Ďalším algoritmom na prehľadávanie je algoritmus na prehľadávanie do hĺbky, ktorý zabezpečuje prehľadávanie uzlov po jednotlivých úrovniach podľa toho, ako sú uložené v strome, a teda slúži na prehľadávanie stromu smerom od hora dole. Znamená to, že skôr, než sa začnú prehľadávať „deti“ nejakého uzla, prehľadajú sa najprv „deti“ uzla, ktorý je rodič tohto uzla. Nakoľko je tento algoritmus takisto jedným z najzákladnejších a opäť musím poznamenať, že som ho už neraz implementoval na iných predmetoch, nebudem sa mu ďalej venovať.

Implementácia vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Rovnako tak, ako predchádzajúci algoritmus, aj tento využíva volanie funkcie **checkSolution**. Samotná funkcia **bds** sa nachádza v module **breadthFirstSearch.js.**

* 1. **Výpis jednotlivých krokov**

Na výpis jednotlivých krokov postupnosti k vyriešeniu problému som si zadefinoval funkcie **printBfs** a **printDfs**, nakoľko oba algoritmy môžu nájsť rôzne uzly, ktoré budú predstavovať riešenie úlohy. Obe tieto funkcie sú totožné, avšak každá z nich posiela výsledky do iného poľa. Fungujú na princípe rekurzie, kde vstupnými argumentmi funkcií sú dvakrát koreň stromu a výsledok vyhľadávania - v jednej pomocou algoritmu prehľadávania do hĺbky, v druhom pomocou algoritmu prehľadávania do šírky. Funkcie začnú prehľadávať celý strom na princípe algoritmu prehľadávania do šírky. V prípade, že nájdu uzol, ktorý sa zhoduje sa zhoduje s hľadaným uzlom v strome, pripíšu ho do poľa a prehľadávajú opäť s tým, že rekurzívnym spôsobom hľadajú rodiča tohto uzla. Takto to prebieha, až kým neprídu ku koreňu. V prípade, že v prvej vrstve nenájdu uzol, ktorý by sa zhodoval s uzlom predstavujúcim riešenie, rekurzívne sa opäť volajú, prehľadávajúc ďalšiu úroveň stromu.

Kód vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Tento kód vypíše postupnosť krokov, ktoré predstavujú riešenie danej križovatky. Vyššie uvedený príklad hľadá cestu riešenia, ktoré nám poskytol algoritmus prehľadávania do širky. Druhá funkcia, ktorá vyzerá totožne, avšak využíva iné polia predstavuje zas hľadanie cesty k uzlu, ktorý sa nám vrátil pri prehľadávaní stromu do hĺbky. Nakoľko funkcie sú takmer zhodné, do dokumentácie pridávam iba tento kód. Dá sa dohľadať v moduli **printBfs.js**, zatiaľ čo druhá spomínaná funkcia na výpis sa nachádza v moduli **printDfs.js**.

* 1. **Main funkcia**

Ako už z názvu vyplýva, main funkcia je hlavná funkcia, ktorej volaním spustíme program. Jej telo obsahuje volania rôznych, vyššie spomínaných funkcii a okrem iného aj požadované výpisy do konzole, ako aj meranie času jednotlivých algoritmov. Funkcia **main** sa nachádza v moduli **test.js**, ktorého spustením zabezpečíme beh programu.

Funkcia vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

* 1. **Prehľad jednotlivých modulov**

V tejto podkapitole sa budem snažiť stručne popísať štruktúru celého projektu, pomocou jednotlivých modulov v abecednom poradí, v akom sú zobrazené. Ďalej sa budem snažiť čo najstručnejšie popísať úlohy jednotlivých modulov v mojom kóde.

Priečinok **Boards**

* obsahuje moduly, ktoré predstavujú jednotlivé prázdne hracie plochy

Priečinok **Cars**

* obsahuje moduly, ktoré predstavujú jednotlivé polohy autíčok na hracom poli

Modul **breadthFirstSearch.js**

* v tomto moduli je zabezpečené prehľadávanie stromu do šírky

Modul **coordinates.js**

* tento modul zabezpečuje výpočet jednotlivých súradníc autíčka a hracej plochy

Modul **copies.js**

* tento modul zabezpečuje kopírovanie polí a objektov

Modul **depthFirstSearch.js**

* modul zabezpečujúci prehľadávanie stromu do hĺbky

Modul **helpers.js**

* modul obsahujúci pomocné funkcie, ktoré nie sú pre program ako taký podstatné, avšak bez nich by nefungoval

Modul **moves.js**

* v tomto moduli sú uložené funkcie na výpočet pohybu autíčok

Modul **nodeSystem.js**

* udáva štruktúru stromu a jednotlivých jeho uzlov

Modul **printBfs.js**

* slúži ako pomoc pri výpise postupnosti jednotlivých krokov, ktoré sme získali prehľadávaním do šírky

Modul **printDfs.js**

* slúži ako pomoc pri výpise postupnosti jednotlivých krokov, ktoré sme získali prehľadávaním do hĺbky

Modul **startingBoard.js**

* zabezpečuje automatické umiestnenie autíčok na hraciu plochu

Modul **test.js**

* modul, ktorého spustením spustíme aj program, okrem iného obsahuje **main** funkciu
* v tomto moduli sa toho deje najviac, nakoľko sa z neho volá väčšina funkcii zadefinovaná v iných moduloch

1. **Testovanie**
   1. **Úvod**

V tejto kapitole sa pozrieme na výsledky testovania jednotlivých hracích polí a porovnáme si výsledky prehľadávania pomocou algoritmov prehľadávania do hĺbky a do šírky pre jednotlivé hracie polia.

* 1. **Manuál k testovaniu**

Ako som už v predošlej kapitole spomínal, testovacie polia a objekty autíčok, ktoré sa následne na ne umiestnia sú uložené v súboroch **Boards** a**Cars.** V tejto podkapitole sa posnažím v krátkosti vysvetliť, ako toto poznanie využiť v praxi. Na začiatku si zvolíme súbor s autíčkami, ktoré chceme umiestniť na hraciu plochu. Máme na výber zo štyroch, avšak dajú sa v budúcnosti vytvoriť aj ďalšie. Takže v prvom kroku vyberieme jeden z modulov v priečinku **Cars**. Na výber sú tu moduly **cars1.js, cars2.js, cars3.js, cars4.js**. Po výbere jedného z modulov prejdeme do modulu **startingBoard.js** a v hlavičke tohto modulu importujeme modul predstavujúci konkrétny modul obsahujúci pole s autíčkami, ako môžeme vidieť na obrázku nižšie (tam kde ukazuje červená šípka).

Obrázok, na ktorom je text

Automaticky generovaný popis

V našom prípade teda stačí prepísať číslo na niektoré z číslic 1 až 4, nakoľko modulov s autíčkami máme 4.

Toto isté zopakujeme v hlavičke modulu **test.js**. Opäť nám stačí prepísať iba číslo.

Následne chceme, aby bolo červené autíčko v rovnakom riadku ako je východ z križovatky (z poľa), preto potrebujeme, aby sme mali hracie pole kompatibilné s rozmiestnením autíčok.

Vyberieme si hracie pole zo súboru **Boards**, kde máme na výber z dvoch polí, a to sú **board1.js** a **board2.js**. Následne potrebujeme tento modul s hracím poľom, ktoré sme si vybrali opäť importovať do modulu **startingBoard.js**. Urobíme to tak, ako sme to robili s autíčkami. Opäť nám stačí prepísať len jedno číslo, ako je to uvedené na obrázku nižšie.

Obrázok, na ktorom je text

Automaticky generovaný popis

Musíme dať však pozor, aby boli moduly autíčok a hracích plôch kompatibilné, inak nám program vypíše error.

Kompatibilné kombinácie v tomto prípade znamenajú:

**boards1.js** je kompatibilné s modulmi **cars1.js** a **cars2.js**

**boards2.js** je kompatibilné s modulmi **cars3.js** a **cars4.js**

Keď sme si istí, že jednotlivé moduly sú kompatibilné, môžeme spustiť program volaním modulu test.js.

* 1. **Prvé testovanie**

Základné hracie pole pre tento test vyzerá nasledovne, pričom farby jednotlivých políčok udávajú základné rozmiestnenie autíčok na hracom poli. Predpokladá sa, že východ z križovatky je zobrazený dostatočne jasne.

Dané hracie pole je definované nasledovne:

****

Výpis môjho programu vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Z postupnosti krokov riešenia vyplýva, že oba algoritmy našli ten istý uzol, predstavujúci riešenie pre dané hracie pole, avšak algoritmus prehľadávania do šírky ho našiel skôr, ako algoritmus prehľadávania do hĺbky. Riešenie obsahuje 7 krokov.

* 1. **Druhé testovanie**

Pre druhý test som si dovolil použiť hracie pole zo zadania, ktoré je zadefinované nasledovne. Východ z križovatky predstavuje pohyb vpravo z políčka [3,6].



Výpis z môjho programu vyzerá nasledovne:

Obrázok, na ktorom je text

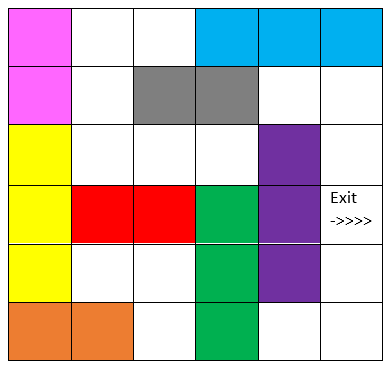
Automaticky generovaný popis

Z uvedeného opäť vyplýva, že oba algoritmy našli rovnaký uzol, avšak narozdieľ od predošlého prípadu, tentokrát bol algoritmus prehľadávania stromu do hĺbky efektívnejší. Riešenie obsahuje 8 krokov.

* 1. **Tretie testovanie**

Pre nasledujúci test som si zvolil hracie pole s východom umiestneným inde ako doposiaľ.

Zadefinované je nasledovne:



Výpis z programu vyzerá nasledovne:

Obrázok, na ktorom je text

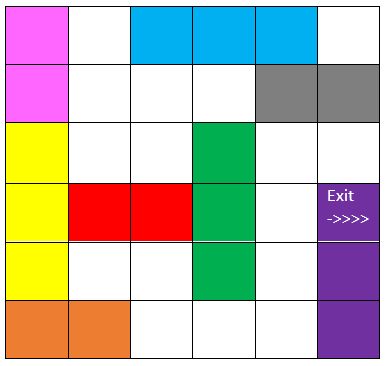
Automaticky generovaný popis

V tomto prípade si môžeme všimnúť, že výpis jednotlivých krokov je rozdielny. Zatiaľ čo algoritmus pre prehľadávanie do šírky prebehol pomalšie, môžeme povedať, že našiel efektívnejšie riešenie, nakoľko obsahuje menej krokov. Na druhej strane, pomocou prehľadávania do hĺbky sme našli výsledok skôr, avšak nie ten najoptimálnejší. Výsledkom teda je, že algoritmus prehľadávania do šírky našiel cestu na 8 krokov, zatiaľ čo algoritmus prehľadávania do hĺbky našiel cestu na 9 krokov, avšak našiel ju skôr.

* 1. **Štvrté testovanie**

Pre štvrtý a zároveň posledný test som si opäť zadefinoval hracie pole s východom takým, ako v predošlom teste.

Hracie pole vyzerá nasledovne:



Výpis z programu vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Výsledkom pozorovania sme zistili, že oba algoritmy prehľadávania našli v tomto prípade rovnakú cestu, avšak algoritmus prehľadávania do hĺbky ju našiel skôr. Cesta má v oboch prípadoch 8 krokov.

1. **Záver**
   1. **Problémy pri implementácii**

Za najväčší problém svojho programu považujem vytváranie stromu. To obvykle zaberie niekoľko desiatok sekúnd, v závislosti od toho, koľko uzlov sa musí vytvoriť. Dalo by sa to zefektívniť zredukovaním premenných v danom uzle. Pri prehľadávaní takého stromu však už problém nie je, nakoľko tento čas je meraný v milisekundách a vykazuje veľmi nízke hodnoty.

* 1. **Výsledok pozorovania**

Výsledkom môjho pozorovania je zistenie, že v danom konkrétnom prípade sa z algoritmov prehľadávania do hĺbky a do šírky nedá vopred presne určiť, ktorý bude optimálnejší. Závisí to od konkrétneho hracieho poľa. Zatiaľ čo v mojich testovacích sadách sa algoritmus prehľadávania do hĺbky ukázal ako efektívnejší, čo sa časovej náročnosti týka, nie vždy našiel najoptimálnejšie riešenie. Na druhej strane, pri algoritme prehľadávania do šírky nájdeme optimálny výsledok vždy, avšak časová náročnosť je vo väčšine prípadov väčšia. Počas testovania svojho programu som však zistil, že čím viac riešení obsahuje daný problém, tým sa znižuje časová náročnosť prehľadávania do hĺbky, avšak ako vravím, na úkor nájdenia najoptimálnejšieho výsledku. Ak by som to mal zrhnúť, ako metódu prehľadávania by som v tomto prípade vždy volil algoritmus prehľadávania do šírky, napriek jeho vyššej časovej náročnosti, ktorá však nemusí byť vyššia vo všetkých prípadoch.

* 1. **Záverečné slová**

Cieľom úlohy bola práca s algoritmami prehľadávania do hĺbky a do šírky. Nakoľko táto téma je pre mňa osobne veľmi zaujímavá a vzhľadom na to, že s podobnými problematikami som pracoval už v minulom semestri na niektorých zadaniach, dovolím si skonštatovať, že úloha ma veľmi bavila. V neposlednom rade by som chcel dodať, že oceňujem možnosť voľby pri výbere jazyka, nakoľko som sa pri riešení tejto úlohy naučil opäť niečo viac z JavaScriptu.