Oliver Leontiev, ID: 103027
DSA, Zadanie 3
2020

Popolvár

Dokumentácia k zadaniu 3

Úvod

Program pomáha popolvárovi zabiť draka a zachrániť všetky princezné v najkratšom možnom čase. Toto je dosiahnuté pomocou Dijkstrovho algoritmu s využitím minimálnej haldy.

Program sa skladá zo štyroch častí: reprezentácia grafu, minimálna binárna halda, Dijkstrov algoritmus a permutovanie princezien s použitím Heapovho algoritmu.

Časti programu

Reprezentácia grafu

Graf je reprezentovaný ako 2D pole smerníkov na štruktúry vrcholov, pričom neprechodné políčka (skaly, na mape „N“) sú v tomto poli NULL.

Zároveň je obalený vo vlastnej štruktúre, v ktorej si pamätá výšku a šírku grafu / mapy, počet princezien na mape, a smerník na draka. Tento smerník sa nachádza aj v poli všetkých vrcholov, ale tu je tiež pre ľahší prístup a prenos informácií medzi funkciami (funkcie majú o jeden parameter menej).

```
typedef struct graf {
	VRCHOL*** vrcholy;
	int vyska;
	int sirka;
	int pocet_princezien;
	VRCHOL* drak;
}GRAF;
```

V štruktúre vrcholu si pamätáme jeho **súradnic**e (x, y), **terén** reprezentovaný 3-bitovým číslom (0 = cesta, 1 = les, 2 = drak, 3-7 = princezné), **predošlý vrchol**, z ktorého sem vedie najkratšia cesta (ak je *pred* NULL, je to informáciu pre Dijkstru, že sa sem dostal cez prekážku a teda vrchol je zablokovaný), dĺžku zatiaľ najlacnejšej **cesty** sem (pre Dijkstru) a **index vrcholu v halde**, pre jednoduché zachovávanie haldovej vlastnosti pri zmenách (umožňuje okamžitý prístup ku vrcholu v halde).

```
typedef struct vrchol {
	unsigned int halda_index;
	unsigned char x;
	unsigned char y;
	unsigned int teren : 3;
	struct vrchol* pred; // predosly vrchol, pre vypis od konca
	unsigned int cesta;
}VRCHOL;
```

Minimálna binárna halda

Oliver Leontiev, ID: 103027
DSA, Zadanie 3
2020

Minimálna halda je reprezentovaná dynamickým poľom, kde pre prvok s indexom *i* je rodič na indexe (*i* - 1) / 2 a deti na indexoch 2 * *i* + 1 a 2 * *i* + 2. Je obalená v štruktúre, kde si pamätá aj svoju dĺžku.

Funkcie heapify_up() a heapify_down() slúžia na obnovenie haldovej vlastnosti v príslušných smeroch. Haldová vlastnosť je, že rodič je menší alebo rovnaký ako dieťa.

Funkcia vlozHalda() zväčší haldu o jedno miesto, vloží vkladaný prvok na koniec a obnoví haldovú vlastnosť smerom hore od konca. Funkcia vytiahniHalda() vytiahne prvý prvok v halde (minimum), na jeho miesto dá posledný, zmenší haldu o jedno miesto a obnoví haldovú vlastnosť smerom dole.

V halde uchovávame vrcholy grafu a udržiavame haldu na základe premennej *cesta*.

Vytvorenie grafu a haldy má **časovú zložitosť O(*n*)** kde *n* je počet políčok na mape a **priestorovú zložitosť O((m + n) * smerník + m * vrchol(štruktúra))**, kde n je počet políčok na mape a m je počet políčok, rôznych od skaly. Štruktúry pre vrcholy sú vytvorené iba raz a iba pre platné políčka. Graf aj halda má alokovanú pamäť pre smerníky na tieto vrcholy, pričom halda iba na platné a graf aj na NULL, teda skaly.

Odstránenie minima z haldy a vloženie prvku má časovú zložitosť O(log *n*), kde *n* je počet prvkov v halde.

Dijkstrov algoritmus

Na zistenie najlacnejšej cesty medzi dvoma vrcholmi (štart – koniec) je použitý Dijkstrov algoritmus v štandardnej podobe. Postupne vyťahuje vrcholy s najlacnejšou cestou z min. haldy a vyhodnocuje cestu do všetkých susedov vrcholu, ktorý vytiahol. Ak zistí, že cez vytiahnutý vrchol vedie do nejakého suseda lacnejšia cesta, ako doteraz, tak ju v susedovi prepíše. Po každé obnoví haldovú vlastnosť (stačí smerom hore, keďže cestu prepisujeme iba na lacnejšiu). Taktiež zapisuje do vrcholov, z akého predošlého vrcholu vedie doteraz najlacnejšia cesta. Cena cesty medzi dvoma vrcholmi, je súčet ceny terénov v oboch vrcholoch.

Na konci vráti posledný vrchol (koniec), z ktorého sa cez predošlé vrcholy da vyskladať najlacnejšia cesta. Ak Dijkstra vytiahol z haldy vrchol, ktorý nemá predošlý vrchol (má na jeho mieste NULL) alebo vytiahol všetky vrcholy (vytiahol NULL), tak vráti NULL a do konca sa nedá dostať.

Dijkstrov algoritmus spúšťame raz pre draka ([0,0] – drak) a potom pre každú princeznú, v každej možnej postupnosti (permutácii) princezien. Pred každým spustením resetujeme haldu a meníme iba prvý vrchol v halde, teda začiatok. Nový koniec dávame priamo ako parameter Dijkstrovému algoritmu, pričom rozhoduje terén vo vrchole, keďže každá princezná ma priradený vlastný terén (3-7).

Časová zložitosť je O(n), kde n je počet vrcholov, ktoré sú bližšie ako hľadaný cieľ. Priestorová zložitosť je taká ako grafu a haldy, keďže obe štruktúry využívame.

Permutovanie princezien

Po zabití draka je potrebné vyskúšať zachrániť princezné vo všetkých poradiach. Princezné teraz reprezentujú ich terény, teda čísla 3-7 pre 5 princezien. Tieto čísla permutujeme

a s každou novou permutáciou zavoláme funkciu skusPermutaciu(), ktorá spustí Dijkstru niekoľkokrát (raz pre každú princeznú) počínajúc drakom, a uloží najlacnejšie poradie princezien.

Samotné permutácie sa vytvárajú v rekurzívnej funkcii permutuj(), ktorá je založená na Heapovom algoritme.

Heapov algoritmus

Heapov algoritmus funguje na dvoch princípoch. Prvý je, že pre každú dĺžku ***k*** podmnožiny poľa vrátane, máme **cyklus od 0 po *k*-1** a ak je dĺžka poľa, ktoré permutujeme párna, tak postupne vymieňame posledné prvky a prvky na indexe rastúcom s cyklom. Ak je dĺžka poľa nepárna, tak v cykle len vymieňame posledný prvok za prvý a následne aplikujeme prvé pravidlo na párnu podmnožinu (o jedno menšiu a s novým, práve vymenením prvkom), čím vymeníme posledný prvok tiež za všetky pred ním. Pri každej zmene sa rekurzívne vrátime na najnižšiu úroveň (dĺžka poľa = 1), kde vieme, že sme získali novú permutáciu. Následne každá vyššia úroveň zbehne znova, keďže sme vymenili prvky a do podmnožiny sa dostal nový prvok.

Druhý princíp je práve toto rekurzívne skákanie medzi úrovňami, kde každú úroveň reprezentuje dĺžka podmnožiny pôvodného poľa, s ktorou práve pracujeme. Rekurzívne volania sú dve, jedno na začiatku, ktoré nám umožňuje vždy sa vrátiť na najnižšiu úroveň a spracovať novú permutáciu. Druhé je na konci cyklu v ktorom vymieňame prvky v poli. Toto druhé volanie nám umožňuje presunúť sa o úroveň nižšie (a teda aj hneď na najnižšiu, vďaka prvému volaniu) potom, čo sme vykonali nejakú výmenu a teda na konci poľa vo vyššej úrovni sa objavilo nové číslo, pre ktoré všetky permutácie podmnožiny sú ešte nezískané.

Algoritmus končí, keď na najvyššej úrovni skončí cyklus. Podstata je minimalizácia výmen prvkov a keďže každú permutáciu získame jednou výmenou, a teda v konštantnom čase (ak neberieme do úvahy rekurzívne volania a vyhodnocovanie podmienok, pričom aj tých je n len v najhoršom prípade a to na začiatku), tak celková časová zložitosť je $O(n!)$ a priestorová zložitosť je $O(n)$, kde n je počet prvkov.

**Poznámka: tento algoritmus sa mi zdal zaujímavý a chcel som ho použiť, toto vysvetlenie uvádzam ako dôkaz, že mu rozumiem.*

Zdroj: https://en.wikipedia.org/wiki/Heap%27s_algorithm

Záver programu

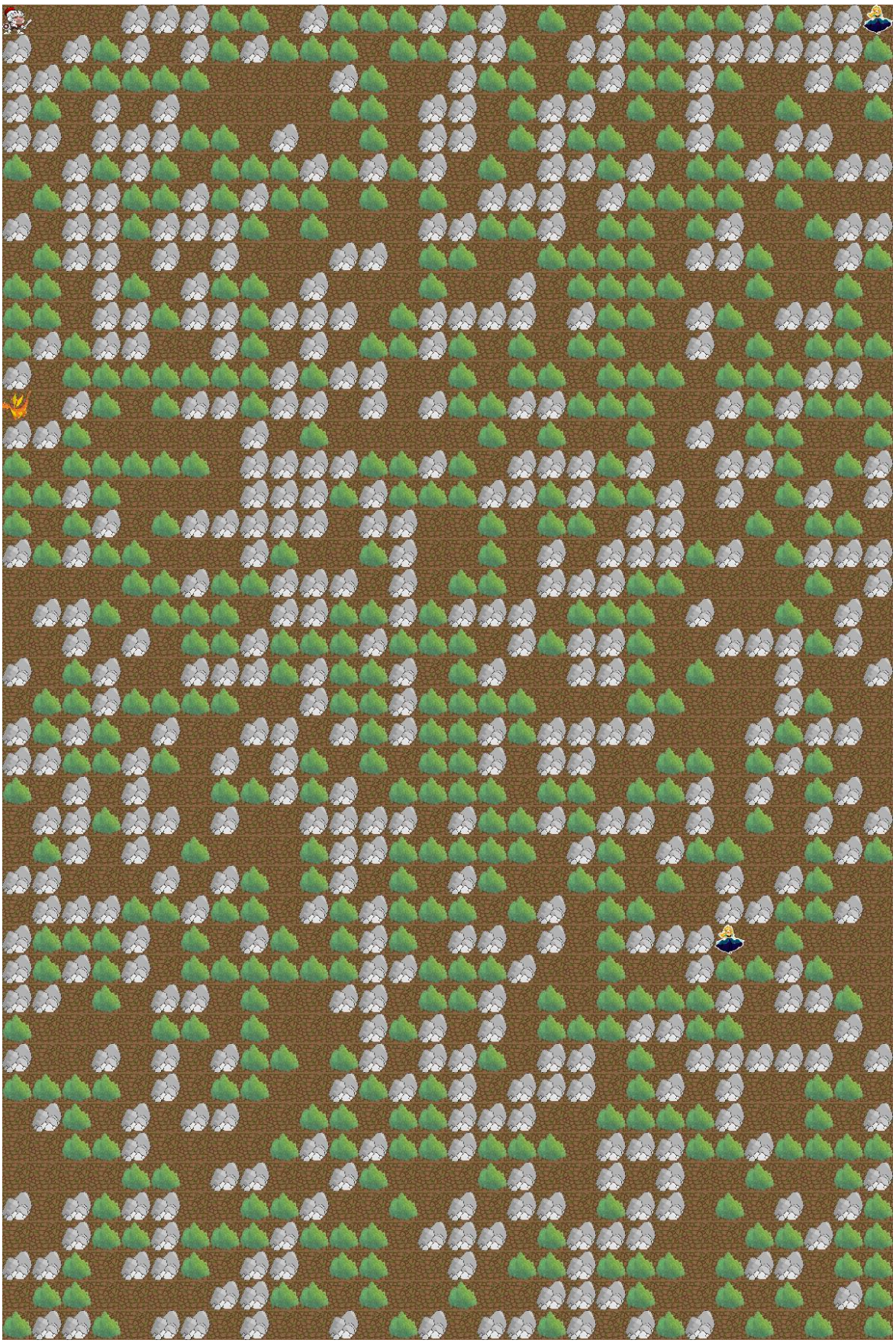
Na záver sa spojí cesta ku drakovi a najlacnejšia od draka do princezien. Tento výsledok sa odovzdá testovaču, ktorý už ho sám vypíše a zistí či sme stihli zabiť draka včas.

Testovanie

Na testovanie bol využitý poskytnutý testovač, pri čom bol vytvorený nový case (3), ktorý pracuje zo súborom *moj_test.txt*, v ktorom je niekoľko testov za sebou. Po každom zadaní čísla 3 do programu, zbehne ďalší (nasledujúci test) zo súboru. Testy sú zamerané na hraničné prípady a 3 sú veľké (100x100), kde je otestovaná celková funkčnosť programu. Časy sú pri testoch nepodstatné a preto nastavené arbitrárne.

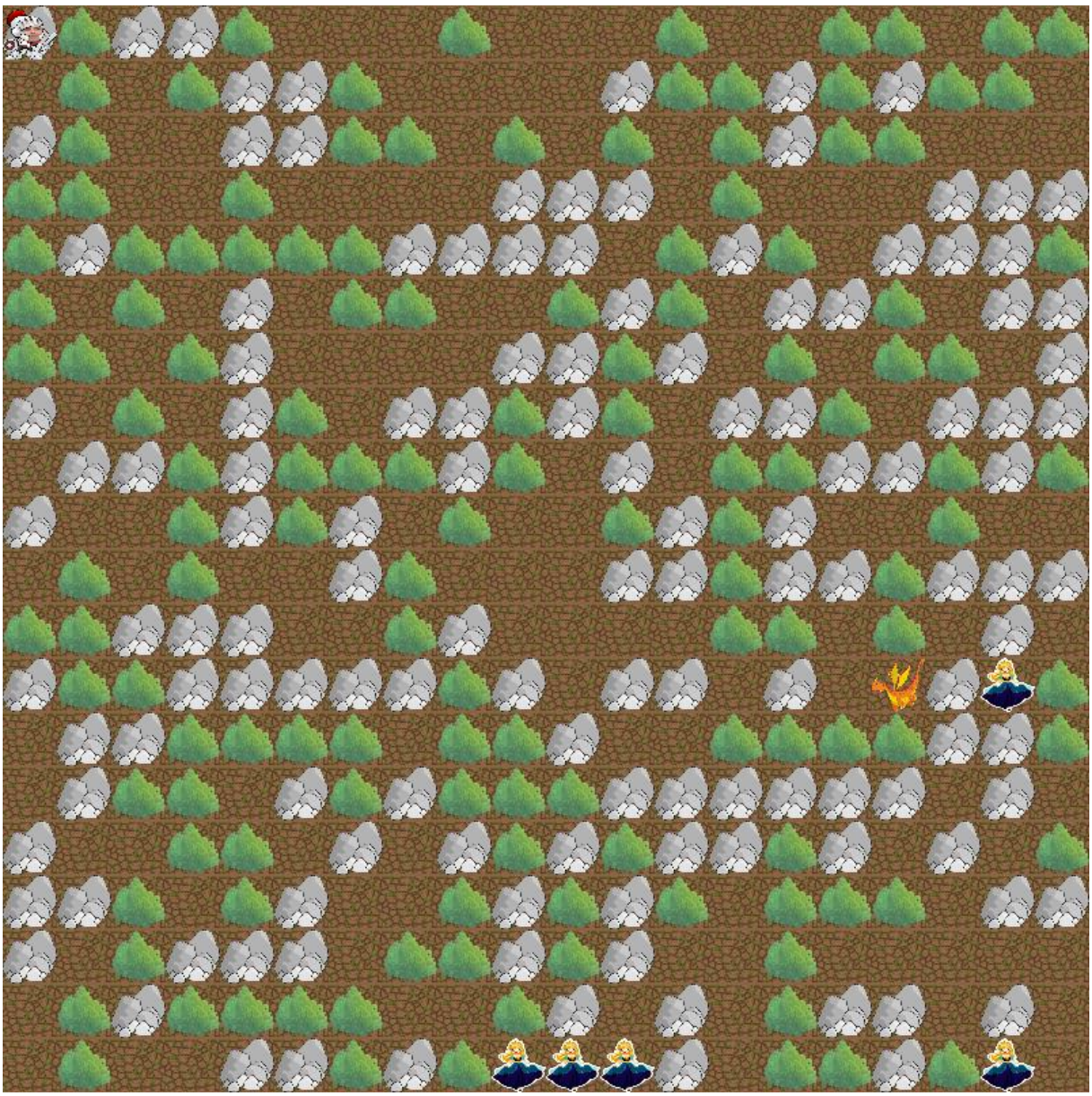
Oliver Leontiev, ID: 103027
DSA, Zadanie 3
2020

Test 1 – zablokovaná princezná



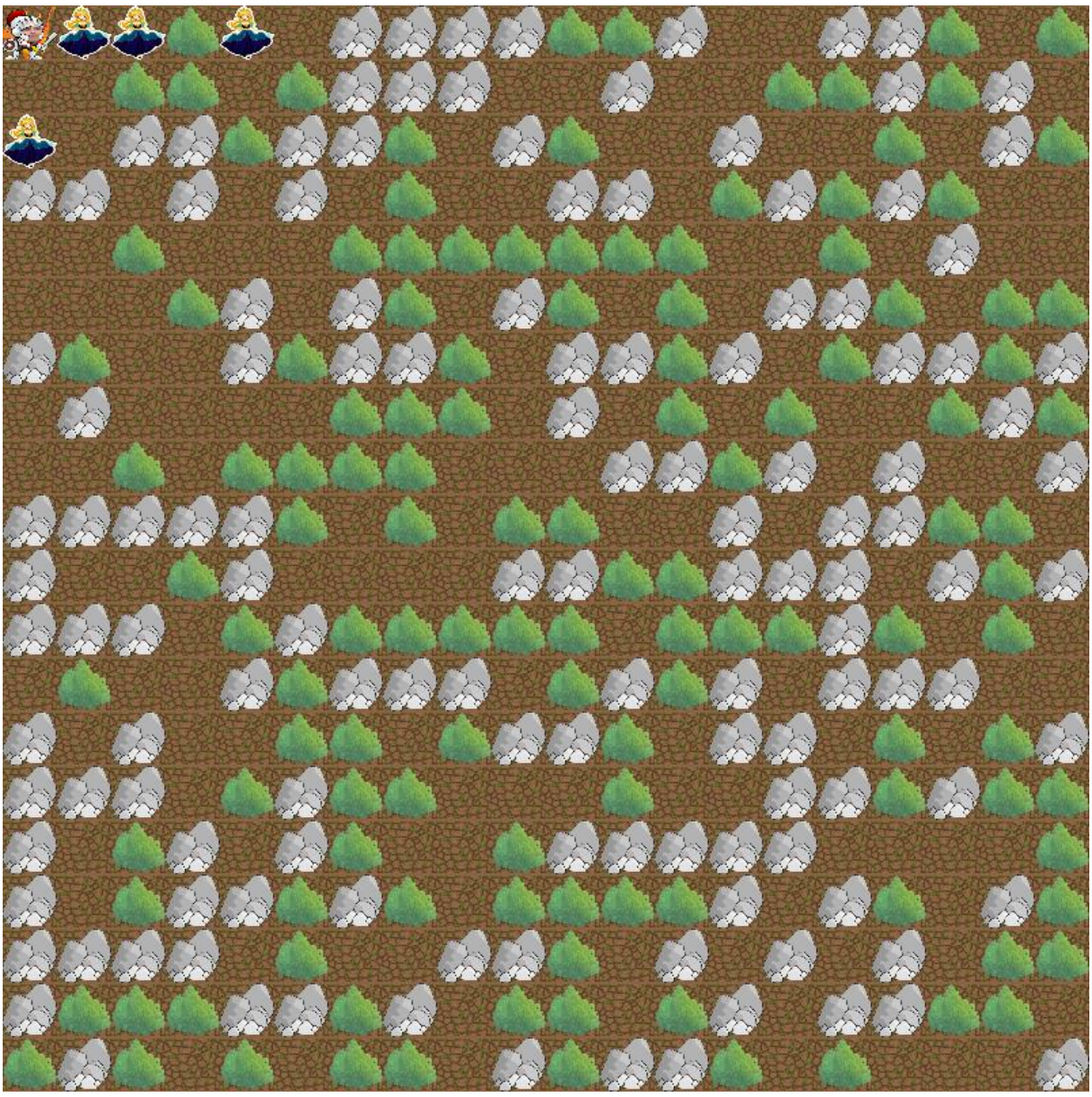
Oliver Leontiev, ID: 103027
DSA, Zadanie 3
2020

Test 2 – skala na štarte



Oliver Leontiev, ID: 103027
DSA, Zadanie 3
2020

Test 3 – drak na štarte, správne poradie princezien



Oliver Leontiev, ID: 103027
DSA, Zadanie 3
2020

Test 4 – zablokovaný drak (popolvár)

Testy 5,6,7 sú 100x100 riešiteľné mapy. Všetky testy zbehli úspešne.