Games 3: 2048

1. 2048

2048 je računalniška *puzzle* igra, ki jo je izumil italijanski internetni razvijalec Gabriele Cirulli. Igra je bila originalno napisana v programskem jeziku `Javascript`.

Igro začnemo na prazni razpredelnici (navadno je velikosti 4×4). Računalnik naključno ustvari številko 2 (z verjetnostjo 90%) ali številko 4 (z verjetnostjo 10%). Postavi jo na naključno mesto na razpredelnici. Igralec ima nato možnost premakniti številko v štiri različne smeri. Če med premikom združi dve enaki števili, dobi eno število, ki je dvakrat večje. Če na primer združi dve osmici, nastane število šestnajst. Nato računalnik spet ustvari naključno številko (2 ali 4) in jo postavi na prazno mesto na razpredelnici. Igra se tako nadaljuje, dokler:

- računalnik ne more več ustvariti naključne številke, saj je celotna razpredelnica zapolnjena (igralec izgubi)
- igralec doseže število 2048 (zmaga)

Združevanje števil in pravila igre so bolj podrobno razložena na Wikipediji.

2. VARIANTE

Poznano je tudi več variant igre:

- klasična (opisana zgoraj)
- neskončna (igra se ne konča, ko igralec doseže 2048, temveč je cilj doseči čim večje število)
- različne velikosti razpredelnice (3x3, 5x5, 6x6, 8x8)

3. CILJI PROJEKTA

Cilj projekta je:

- implementacija igre v programskem jeziku `Java` (vizualizacija)
- nastavitve igre, različna izbira igralnih načinov, barvne sheme
- omogočiti igranje igre igralcu
- zasnovati nekaj računalniških algoritmov, ki sami poskušajo (uspešno) reševati problem in premagati igro

4. TEHNIČNI PODATKI

Igro sem se odločil zasnovati v programskem jeziku Java, ki ima s knjižnico `Swing` dobro podlago za risanje na platnu (`JPanel`) in vizualizacijo v oknu (`JFrame`). Igro sem predstavil kot razred `Game`, ki ima na sebi implementirane različne metode, vsaka s svojo uporabnostjo. Sama igra je sestavljena kot matrika (seznam seznamov) velikosti $n \times n$ (v klasičnem primeru 4x4). Za več informacij sem zraven vsake metode komentiral njen namen in kaj naredi (glej kodo).

Kot metode na igri so definirani tudi računalniški algoritmi (metoda je sestavljena tako, da algoritem najde potezo in jo odigra na igri). Zaenkrat sem definiral tri računalniške algoritme, ki so opisani v spodnjem poglavju.

V orodni vrstici igre lahko izberemo njeno velikost ali barvno shemo.

5. RAČUNALNIŠKI ALGORITMI ZA REŠEVANJE IGRE

5.1. NAKLJUČNE POTEZE

Algoritem izbira naključne poteze iz nabora vseh potez. Če izbrane poteze ni med možnimi, algoritem naključno izbere še enkrat. Algoritem za reševanje ni prav uspešen in potrebuje vsaj 6x6 veliko mrežo, da premaga igro.

5.2. SIMULATOR

Algoritem vsako možno potezo k-krat odigra na kopiji igre (novi igri). Nato od naslednje poteze naprej igra naključne poteze, dokler igre ne premaga ali izgubi. Za vsako možno potezo nato izračuna povprečen rezultat in izbere potezo z največjim.

Na primer: možni potezi sta premik gor in premik levo. Algoritem ustvari 25 novih iger (kopij originalne igre), na katerih odigra premik gor in 25 novih iger, na katerih odigra premik levo. Nato vsako igro odigra do konca z naključnimi potezami. Za dejansko potezo izbere tisto, za katero imajo igre večji povprečen rezultat.

Algoritem se odlično obnese za velike k, kjer vedno premaga igro, vendar je potrebno vedeti, da se s povečevanjem k-ja poveča tudi čas za potezo. Kljub temu, za $k = 10$, algoritem rabi slabo sekundo za potezo, je pa dokaj uspešen pri premagovanju igre (približno 50%). Za $k = 100$ je 80% uspešen, za $k = 10000$ pa 100%.

Algoritem je zelo dober, saj lahko glede na želje prilagodimo število simulacij (za večjo zanesljivost več simulacij, za večjo hitrost a manjšo zanesljivost pa manj simulacij).

5.3. MINIMIZIRANJE PRAZNIH POLJ

Algoritem, ki za vsako potezo pogleda, koliko bo po njej praznih polj in izbere tisto potezo, za katero bo praznih polj po njej minimalno. Če ima minimalno število praznih polj več potez, naredimo preferenčno lestvico, kako naj algoritem izbira. Ta algoritem je sicer bolj uspešen pri reševanju kot naključne poteze, precej manj pa kakor simulator.

Opomba: Na misel sicer takoj pade ideja, da bi namesto naključnih potez pri simulatorju minimizirali prazna polja, a ta ideja ni najbolj praktična, saj potem simulator porabi precej (preveč) več časa.

6. NADALJNJO DELO

Večino dela sem sicer že opravil, saj je vse zgoraj napisano že implementirano in tudi dela. Je pa v planu nadaljnjega dela še:

- primerjanje uspešnosti različnih algoritmov za različne globine (pri simulatorju) in različne velikosti igre (grafi)
- odpravljanje trenutnih hroščev
- poljubna izbira globine za simulator algoritem
- če bo čas, mogoče poskusno še kakšen algoritem

7. VEČ INFORMACIJ

Za več informacij (v angleškem jeziku) glej datoteko `README.md` na repozitoriju.

8. POSNETKI ZASLONA