VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PROGRAM PRO AUTOMATICKÉ HRANÍ HER SOLITAIRE A MINY

PROGRAM FOR AUTOMATIC PLAYING OF SOLITAIRE AND MINES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN MORES
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV ROZMAN, Ph.D.
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

155350

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Mores Martin**
Program:	Informační technologie
Název:	**Program pro automatické hraní her Solitaire a Miny**
Kategorie:	Umělá inteligence
Akademický rok:	2023/24

Zadání:

1. Nastudujte pravidla her Solitaire a Miny. U obou her proveďte analýzu zákonitostí, které z pravidel vyplývají. Nastudujte nástroje pro detekci ovládacích prvků na obrazovce počítače a emulaci uživatelských akcí (např. kliky myší, stisky kláves).
2. Navrhněte program v jazyce Python, který umožní obě hry hrát automaticky, tzn. detekovat jednotlivé části hracího pole a klikat a přetahovat karty myší.
3. Navržený program implementujte.
4. Oba implementované programy otestujte. U Min se zaměřte na statistiku toho, po jaké akci byla hra ukončena. U Solitaire, jestli je možné každou hru dohrát, případně, jak moc je potřeba se ve hře vracet, aby ji bylo možné dokončit. Navrhněte také možné způsoby, jak hraní zrychlit ve srovnání s předchozí prací.

Literatura:

- Přikryl, S., Program pro automatické hraní her Solitaire a Miny, bakalářská práce, VUT FIT, 2023
- Russell, S. a Norvig, P., *Artificial Intelligence: A Modern Approach* , Prentice Hall, 2010

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	6.11.2023

Abstrakt

Táto práca je zameraná na navrhnutie a na naimplementovanie dvoch programov pre automatické hranie hier Hľadanie Mín a Solitaire. Teoretická časť je zameraná na pravidlá o hrách a aké zákonitosti z nich vyplývajú a o teórii o prehľadávaní stavového priestoru. Práca v ďalších kapitolách popisuje implementáciu oboch programov a popisuje rôzne veci, ktoré viedli k prípadnému zrýchleniu časov oproti minulým prácam. Práca taktiež poskytuje čitateľovi rôzne postupy, ktoré môže využiť pri vlastnej implementácii. V závere práce je zhrnuté zhodnotenie implementácie a sú navrhnuté potencionálne vylepšenia.

Abstract

This thesis focuses on the design and implementation of two programs for automatic game play, Mine Search and Solitaire. The theoretical part focuses on the rules about the games and what regularities follow from them, and on the state space search theory. The work in the following chapters describes the implementation of the two programs and describes various things that led to the eventual speedup of the times compared to past work. The thesis also provides the reader with various techniques that can be used in their own implementation. And at the end of the thesis, an evaluation of the implementation is summarized and potential improvements are suggested.

Kľúčové slová

Solitaire, Hľadanie Mín, Python, DFS, Umelá inteligencia

Keywords

Solitaire, Minesweeper, Python, DFS, Artificial intelligence

Citácia

MORES, Martin. *Program pro automatické hraní her Solitaire a Miny.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Program pro automatické hraní her Solitaire a Miny

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jaroslava Rozmana, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

. . . . . .
Martin Mores
8. mája 2024

Poďakovanie

Týmto by som chcel poďakovať všetkým, ktorý mi pri písaní práce pomohli. Hlavne by som týmto chcel poďakovať pánovi Ing. Jaroslavovi Rozmanovi Ph.D. za odborné znalosti, spätnú väzbu a podporu, ktorú mi venoval pri písaní tejto práce. A taktiež aj svojím rodičom, ktorí pri mne stáli a boli mi podporou.

Obsah

1 Úvod **4**

2 Pravidlá hier Solitaire a Míny **5**
2.1 Solitaire 5
2.1.1 Plán hry Solitaire 5
2.1.2 Pravídla hry Solitaire 6
2.2 Hľadanie Mín 8
2.2.1 Plán hry Hľadania Mín 8
2.2.2 Pravidlá hry Hladania Mín 9
2.2.3 Algoritmus pre Hľadanie Mín 10

3 Stavový priestor **12**
3.1 BFS 12
3.2 DFS 13
3.3 A* 13

4 Predchádzajúci výskum a podobné nástroje **14**
4.1 Predchádzajúce poznatky 14
4.2 Existujúce programy 15

5 Nástroje na detekciu ovládacích prvkov **17**
5.1 PyAutoGUI 17
5.1.1 Pohyby myšou 17
5.1.2 Lokalizácia okna 18
5.1.3 Snímok obrazovky 18
5.1.4 Stlačenie klávesnice 18

6 Návrh programu pre automatické hranie hier **19**
6.1 Spustenie hier 19
6.2 Detekcia aktívneho okna 20
6.3 OpenCV 20

7 Implementácia programu **25**
7.1 Implementácia pre hru Hľadanie Mín 25
7.2 Implementácia pre hru Solitaire 29

8 Testovanie a záverečné štatistiky **34**
8.1 Testovanie hry Hľadanie Mín 34
8.2 Testovanie hry Solitaire 36

9 Záver **37**

Literatúra **38**

A Obsah priloženého pamäťového média **40**

Zoznam obrázkov

2.1 Herný plán [1] 6
2.2 Druhy kariet 7
2.3 Začiatok hry 7
2.4 Priebeh hry 8
2.5 Obtiažnosť: začiatočník 9
2.6 Výbuch míny pri obtiažnosti Expert 10
2.7 Lineárne algebraické rovnice pre hru Hľadanie Mín 11

3.1 BFS 13
3.2 DFS 13

4.1 Snímok z hry `minesweeper-solver` 15
4.2 Snímok z hry `Minesweeper Probability Calculator` 16
4.3 Snímok z hry `Solvitaire` 16

6.1 Ukážka ako by malo fungovať `Template Matching` 21
6.2 Prvý klik 22
6.3 Políčko s mínou 23
6.4 Bezpečný klik 23
6.5 Vyloženie esa 24

7.1 Kód na detekciu hracej plochy 26
7.2 Detekcia hracieho poľa 27
7.3 Región so zásobníkom 30
7.4 Región s Tableue 31

Kapitola 1

Úvod

Nachádzame sa na prelome rokov 2023/2024, kde sa ľudstvo snaží posúvať dopredu a všetko zautomatizovať pre ľahšie fungovanie domácnosti, dopravy, spoločnosti ale aj práve pri pomoci pri hraní počítačových hier. Jednou z neoddeliteľných súčastí hrania hier na počítači sa stali hry ako Solitaire a Hľadanie Mín, ktoré vznikli ešte pred 21. storočím. Tieto hry si získali srdcia mnohých ľudí, či už kvôli tomu, že to bolo niečo nové v tej dobe alebo aj preto, lebo ľudia si chceli skrátiť čas pri nejakej strategickej hre. Zábavné, ale častokrát aj náročné hry si ľudia mohli zahrať už na operačnom systéme Windows 3.0 a Míny sa pridali už o dva roky neskôr na verzii 3.1.

Táto práca je v danej oblasti dôležitá hlavne pre nový pohľad a výskum na problematiku automatického hrania hier Solitaire a Míny v jazyku Python a pre inováciu a zlepšenie existujúcich riešení na danú problematiku. Dozvieme sa, či sa každá hra Mín dá dohrať, a ak áno ako dlho to bude programu trvať a pri Solitaire sa v práci okrem implementácie inovovaného riešenia zameriame na samotný program a ukážeme si na algoritme ako, a ktoré karty musí program vyberať, aby bol schponý sa dohrať, teda respektívne, či toho bude schopný. Môj vzťah k tejto práci je taký, že dobre poznám pravidlá oboch hier a rád si ich zahrám ešte aj v tejto dobe, keď je na výber veľké množstvo iných hier. A z mojích znalostí o oboch hrách by som chcel naimplementovať program, ktorý zrýchli a zdokonalí predchádzajúce riešenia.

Cieľom práce je, aby pri hre Hľadanie Mín vedel program detekovať okno programu, vedel rozpoznať políčka a čísla , musí vedieť odsimulovať pozície mín a vybrať také, kde sa mína nebude nachádzať a ak sa dostane do stavu, kde nebude vedieť ako ďalej postupovať, musí vedieť vypočítať pomocou pravdepodobnosti možné políčka a vybrať si to najlepšie pre dohranie programu. Pri hre Solitaire musí tiež program vedieť detekovať okno, vedieť hodnoty kariet a byť schopný prehľadávať stavový priestor tak, aby hra mohla byť dohratá.

Práca je štrukturovaná do niekoľkých kapitol. 2. kapitola sa zaoberá teóriou hier Solitaire a Hľadanie Mín a pravidlami pre hranie týchto hier. Nasledujúca 6. kapitola sa venuje existujúcim riešeniam, ktoré možno využiť pri implementácii a návrhu programu v jazyku Python, ktorý umožní automatické hranie. V 7. kapitole sa čitateľ dočíta o samotnej implementácii navrhnutého riešenia a v 8. kapitole si čitateľ nájde testy a štatistiky, ktoré vyplynuli z testovania implementovaného programu.

Kapitola 2

Pravidlá hier Solitaire a Míny

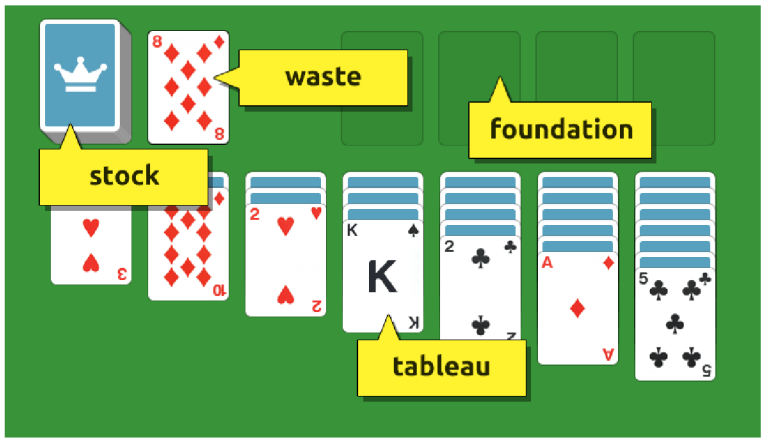
Táto kapitola sa bude venovať pravidlám a zákonitostiam, ktoré z týchto pravidiel vyplývajú. Pravidlá o hrách ako sú Solitaire a Hľadanie Mín. V prvej časti si povieme o histórii hry Solitaire. Následne si povieme ako vyzerá hracie pole tejto hry a aké majú jednotlivé elementy plochy funkcie, ktoré karty sa môžu premiestňovať a kedy je možné hru dohrať. V ďalšej časti sa zameriame na hru Hľadanie Mín, pri ktorej si zasa povieme, čo znamenajú jednotlivé čísla na hracom poli. Čo znamená, keď stlačíme na políčko, ktoré skrývalo mínu, ale aj to, aké algoritmy môžeme využiť aby sa nám podarilo hru dohrať.

2.1 Solitaire

Hra Solitaire Klonidke je jednou verziou z vyše 150 možných variácií hry. Hra sa na obrazovky počítačov dostala v roku 1990 na verzii operačného systému Windows 3.0 [6]. Jedná sa o strategickú kartovú hru, ktorá je určená predovšetkým len pre jedného hráča. Ale existujú aj varianty pre viacerých hráčov. Princípom hry je poskladať štyri rôzne kopy kariet, ktoré budú postupne umiestnené do základnej hromádky. Tieto kopy budeme skladať z kariet, ktoré nám hra rozdá do sekcie tableau a zásobníka. Hra končí tým, že každá jedna hromada v základnej hromádke bude tvorená postupnosťou kariet od hodnoty karty eso až po výslednú kartu kráľa. Táto sekcia čerpá infomácie z literatúri[12], [6].

2.1.1 Plán hry Solitaire

Pred začatím hry, každý hráč musí poznať rozdelenie herného plánu hry a poznať, ktorá časť má akú funckiu. Pri hre Solitaire rozďeľujeme hraciu plochu na 4 základné časti. Ľudia to môžu poznať pod inými pojmami, ale my sa tu budeme držať označenia ako tableau [12], základná hromádka (po anglicky Foundations), zásobník (po anglicky Stock) a odpadový kôs (po anglicky Waste) viď obrázok 2.1.



Obr. 2.1: Herný plán [1]

Funkcie jedonotlivých častí sú následovné[12]:

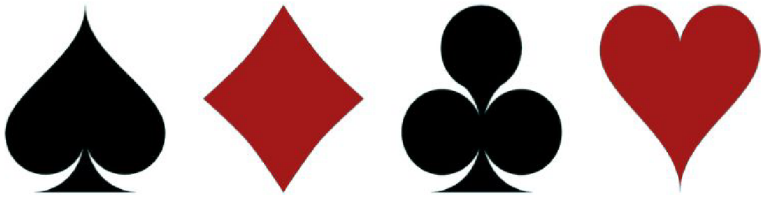
- **Tableau**
  - Tableau je hlávná časť, kde sú karty rozmiestnené do siedmych hromádok. Pričom karty z jednej hromádky sa môžu premiestňovať na iné hromádky. Keď je karta premiestnená a pod touto kartou sa nachádzala ešte neodokrytá karta, tak sa táto karta odokryje a máme ďalšiu možnosť, kde premiestniť kartu. Ak sa pod kartou už nenachádza ďalšia karta, tak nám vznikne prázdna hromádka, na ktorú môžeme umiestniť kartu s typom kráľa. K vyhratiu hry potrebujeme všetky karty z tableau umiestniť do vykladacej alebo teda základnej hromádky.
- **Základná hromádka**
  - Časť, ktorá má štyri na začiatku prázdne hromádky. Karty, ktoré sem budeme dopĺňať, či už z tableua alebo zásobníka, musia tvoriť postupku. Postupka musí byť tvorená z rovnakej farby a každá hromádka musí začať hodnotou eso a skončiť kráľom.
- **Zásobník**
  - Obsahuje zvyšné karty, ktoré potrebuje k dohratiu hry. Karty môžu byť ľubovoľne rotované, či už po 3 alebo 1 karte (závisi na vývojárovi). Karty sa zo zásobníka premiestňujú na tableau časť alebo rovno do základnej hromádky.
- **Odpadový kôš**
  - Obsahuje karty, ktoré boli v zásobníku, ale nebolo ich potreba využiť ešte v hre. S kartami odloženými v odpadovom koši sa v priebehu hry ešte bude môcť pracovať.

2.1.2 Pravídla hry Solitaire

Hra začína zamiešaním klasického balíka kariet, ktorý obsahuje 52 kariet rozdelených do 4 druhov podľa farby a vzoru. Vizuálne zobrazenie kariet môžeme vidieť na obrázku 2.2. Druhy skupín kariet sú nasledovné (z ľava do prava):

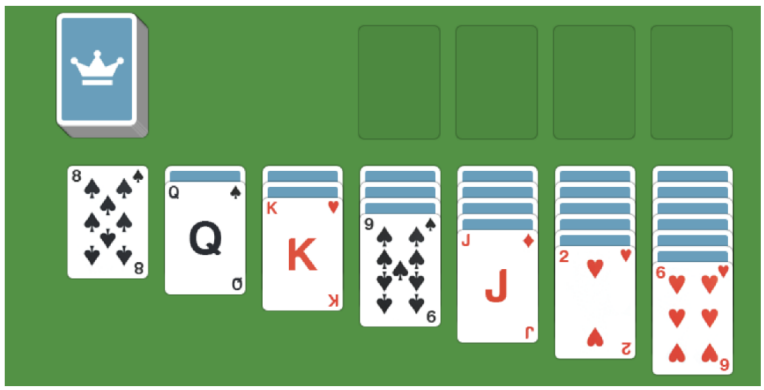
[1] Prevzaté a upravené z: https://www.solitairecardgames.com/how-to-play-solitaire

- Listové karty
- Kárové karty
- Pikové karty
- Srdcové karty



Obr. 2.2: Druhy kariet

Následne, keď sú karty zamiešané, tak sa rozdelia do dvoch skupín. Prvá skupina, tableau, pozostáva z 28 kariet rozdelených do siedmych častí. A druhej skupiny , zásobnika, kde je zvyšných 24 kariet, ako môžeme vidieť na obrázku 2.3.



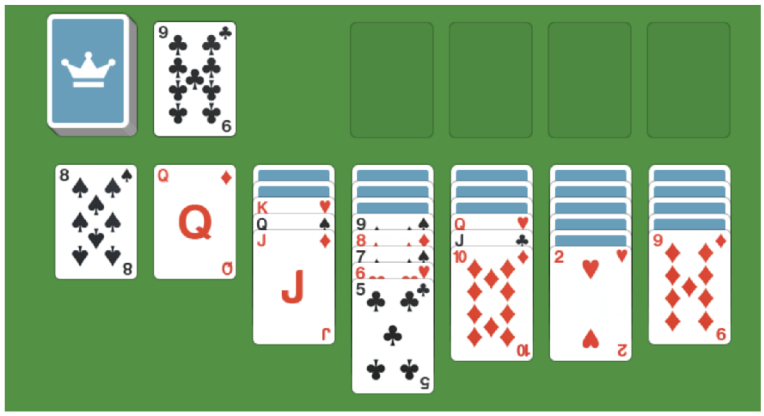
Obr. 2.3: Začiatok hry

Hra začína tým, že presúvame karty zo zásobníka na tableau alebo medzi jednotlivými časťami tableau medzi sebou. Ale musíme dodržať dve podmienky. Prvé pravidlo je, že presúvana karta bude presne o jednu hodnotu menšia ako karta, na ktorú ju budeme prikladať. A druhé pravidlo je, že presúvana karta musí mať ešte aj inú farbu ako karta, na ktorú ju premiestňujeme ako môžeme vidieť na obrázku 2.4. Ak sa nám na tableau uvoľní celá rada, tak na prázdne políčko môže ísť len karta kráľ. A potom už ako podľa podmienok kráľovná opačnej farby.

Karty v zásobníku môžeme rotovať podľa seba, ak danú kartu, ktorú nám zásobník vyhodil nepotrebujeme , tak ju vložíme do odpadového koša. S tou kartou budeme môcť pracovať neskôr.

Pravidlo, aby sme mohli karty vykladať do základnej hromádky spočíva v tom, že potrebujeme medzi všetkými kartami nájsť eso. Eso vložíme na prázdne miesto a potom postupne dokladáme na eso rovnaký druh kariet ako eso. Vkladané karty musia mať teraz presne o jednu hodnotu viac ako karta, na ktorú ju vkladáme.

Cieľom hry je zbaviť sa všetkých kariet v tableau a na zásobníku. Všetky karty musia byť poukladané postupne v základnej hromádke.



Obr. 2.4: Priebeh hry

2.2 Hľadanie Mín

Hľadanie mín je strategická hra určená pre jedného hráča, ktorá sa preslávila na operačnom systéme Windows a svoju popularitu získala hlavne v roku 1989 [8]. Cieľom hry je odhaliť všetky políčka na hracej ploche, ktoré nie sú míny a vyhnúť sa všetkým mínam. Ak hráč stlačí políčko, na ktorom sa nachádza mína, hra sa reštartuje a hráč musí začať hrať hru od začiatku.

2.2.1 Plán hry Hľadania Mín

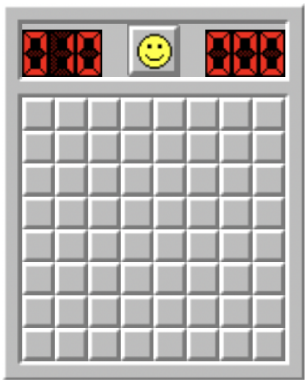
Hrací plán v hre sa odvíja od zvolenej obtiažnosti, pričom existujú tri hlavné úrovne náročnosti, ktoré ukazujú rozloženie herného poľa. Podľa klasickej verzie, ktorá bola na Windowse poznáme obtiažnosti ako začiatočník (Begginer), mierne pokročilý (Intermediate) a pokročilý (Expert) [7].

V prípade obtiažnosti začiatočník, viď obrázok 2.5, je hrací plán koncipovaný s 81 políčkami, na ktorých sa nachádza celkovo 10 mín [9].

Náročnosť mierne pokročilý zahrňuje komplexnejší hrací plán, ktorý obsahuje celkovo 256 políčok a 40 mín.

A pri poslednej obtiažnosti, označovaná ako pokročilý, je hrací plán rozložený nie do štvorca ako pri predchádzajúcich obtiažnostiach, ale do obdĺžnika, ktorý obsahuje 16 * 30 možných políčok, kde sa nachádza už 99 mín [8].

Výber obtiažnosti v hľadaní mín nielen determinuje veľkosť hracieho plánu, ale aj množstvo mín, čím hráčom poskytuje hrať hru podľa ich preferencií a schopností dohrať hru.

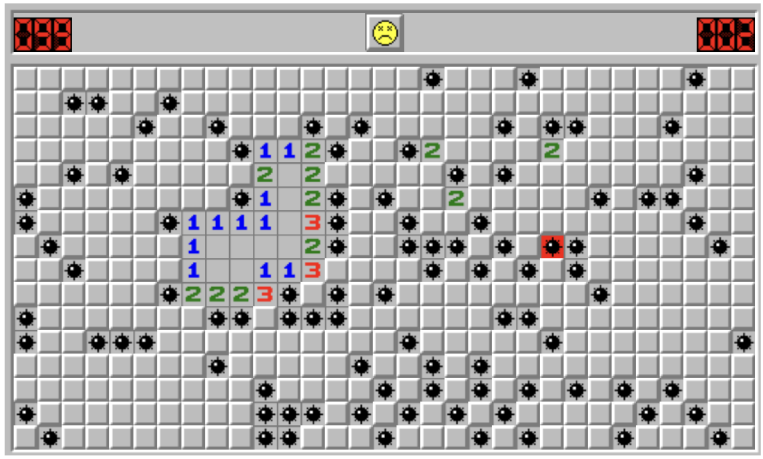


Obr. 2.5: Obtiažnosť: začiatočník

2.2.2 Pravidlá hry Hladania Mín

Pravidlá a samotný priebeh je nasledovný[9]:

- **Začiatok hry:**
  - Hra začina na ploche, kde sú všetky políčka zakryté. Takže hráč nevie, kde sú umiestnené jednotlivé míny a musí spraviť prvý krok. Pri prvom kroku nikdy hráč nemôže stlačiť na mínu, lebo míny sa rozmiestnia až po tomto kroku. Po stlačení prvého kroku môžu nastať dve situácie. Prvá je, že sa hráčovi zobrazí väčšia časť plochy s odokrytými políčkami. Toto nastane, keď stlačíme na políčko, ktoré neobsahuje číslo. Alebo pri druhej situácii sa hráčovi zobrazí len jedno číslo.
- **Priebeh hry:**
  - Hráč pri každom svojom ťahu môže odhaliť políčko, pod ktorým sa skrývalo číslo. Toto políčko udáva, koľko mín sa nachádza okolo neho. Takže, ak by hráč odkryl číslo 2, tak to znamená, že okolo tohto políčka sa vyskutujú práve dve míny. Tieto míny sa nevyskytujú len napravo, naľavo, dole a hore pod políčkom, ale môžu byť schované aj na jeho diagonále.
- **Odhalenie míny:**
  - Ak hráč zistí, kde sa má mína nachádzať, tak pomocou pravého tlačítka na myši môže toto políčko označiť symbolom vlajky a bude vedieť, že nemá toto označené políčko stláčať, lebo pod ním by sa mala nachádzať mína.
- **Koniec hry:**
  - Hra končí práve vtedy, keď sa na hracej ploche nachádzajú len také nevyhodnotené políčka, pod ktorými vieme povedať, že sa musí nachádzať mína. Vtedy je hra výherná. Ale hra môže skončiť aj skôr, ak hráč stlačí na políčko, ktoré obsahuje mínu a vybuchne, viď obrázok 2.6.



Obr. 2.6: Výbuch míny pri obtiažnosti Expert

2.2.3 Algoritmus pre Hľadanie Mín

Pre správne fungovanie programu a hrania Hľadania Mín existuje niekoľko rôznych algoritmov ako pristupovať k jednotlivým bunkám hracieho poľa pre dohranie hry. Od základných algoritmov až po výpočetne náročnejšie. Ako sú napríklad algoritmy spomenuté nižšie.

Prvý krok

Je prvý klik v hre na bunku s nulovými súradnicami, teda políčko, ktoré je úplne najvyššie a úplne naľavo. Pri tomtu kroku nikdy nemôže nastať situácia, že sa stlačí políčko s mínou. Najlepšie pri tomto kroku je začať v nejakom políčku, ktoré sa nachádza v rohu, pretože, takéto políčko má menej susedných mín a kvôli tomu môžeme ľahšie nájsť políčko bez míny. A keď začneme v rohu, máme viacej priestoru na rozhodovanie sa, čo nám zabezpečuje viac možností pri výpočte umiestnenia míny

Zhluk buniek

Pri zhlukovaní jednotlivých buniek funguje pravidlo, že ak máme tri pri sebe nachádzajúce sa políčka napríklad A,B a C, tak vieme, že ak políčka A a B obsahujú jednu mínu a políčka A, B a C majú tiež jednu mínu, tak vieme určiť, že políčko C neobsahuje mínu a môžeme na neho kliknúť. Takže vieme povedať, ak políčka A a B majú jednu mínu, políčka A, B a C majú už dve míny, tak vieme určite povedať, že políčko C bude obsahovať mínu.

Ale platí to aj naopak. Ak máme skupinku políčko A,B a C a obsahujú jednu mínu, tak skupiny AB a AC a BC neobsahujú viacej ako jednu mínu.

Počet mín

Keď poznáme koľko neodhalených políčok celkovo nám ešte zostalo a to číslo sa zhoduje s číslom koľko celkovo mín nám treba ešte odhaliť, tak vieme, že všetky neodhalené políčka budú míny. A zasa vieme povedať že ak budeme mať len neodokryté políčka a budeme vidieť, že sme už odhalili všetky míny v hre, tak vieme, že všetky políčka budú bezpečné a môžeme na ne kliknúť.

Alebo ak máme skupinu buniek a podľa odhaleného políčka s číslom 3 vieme, že okolo neho musia byť práve tri míny a nemáme odhalenú ešte žiadnu a okolo sú práve tri políčka,

tak vieme, že všetky políčka sú míny. A zasa, ak už okolo políčka s číslom 3 máme odhalené všetky míny a nachádzajú sa okolo neho ešte neodhalené bunky, tak vieme, že všetky takéto bunky budú bezpečné.

Pravdepodobnosť

Pri pravdepodobnosti vieme, že ak sme odokryli políčko napríklad s číslom 3 a pri tomto políčku mamé odokrytú len jednu bunku a zvyšných sedem nie, tak poďla pravdepodobnosti vieme zistiť, s akou pravdepodobnosťou sa mína bude nachádzať na políčkach vedľa bunky s číslom 3. Pri tomto príklade by malo každé políčko pravdepodobnosť 3/7, že sa tam objaví mína.

Lineárne algebraické rovnice

K viacej zložitejším algoritmom pre dohranie hry Hľadanie Mín patrí, algoritmus pomocou lineárnych algebraických rovníc.

Využitím lineárnej algebry môže byť hra Minesweeper prístupná ako systém rovníc reprezentovaných maticami. Každému neodokrytému políčku je pridelená premenná x_i, s hodnotou 1, ak sa jedná o políčko, kde sa nachádza mina, a 0, ak tam mína nie je [10]. Číselné políčka poskytujú informácie o počte susedných mín. Analýzou vzťahov medzi týmito premennými a obmedzeniami poskytnutými číselnými políčkami môžeme zostaviť systém lineárnych rovníc. Tieto rovnice možno reprezentovať v maticovom tvare a riešiť pomocou techník ako gaussová eliminácia.

Týmto spôsobom môže byť Hľadanie Mín systematicky riešené pomocou matematických princípov namiesto toho, aby sme sa spoliehali iba na intuíciu alebo pokus omyl. Použitím lineárnej algebry môžeme efektívne zistiť prítomnosť alebo neprítomnosť mín v každom políčku hracej plochy Hľadanie Mín.

$$
\begin{aligned}
1x_1 + 1x_2 + 0x_3 + 0x_4 + 0x_5 &= 1 \\
1x_1 + 1x_2 + 1x_3 + 0x_4 + 0x_5 &= 2 \\
0x_1 + 1x_2 + 1x_3 + 1x_4 + 0x_5 &= 2 \\
0x_1 + 0x_2 + 1x_3 + 1x_4 + 1x_5 &= 2 \\
0x_1 + 0x_2 + 0x_3 + 1x_4 + 1x_5 &= 1
\end{aligned}
$$

Obr. 2.7: Lineárne algebraické rovnice pre hru Hľadanie Mín

Kapitola 3

Stavový priestor

Stavový priestor patrí medzi základné požiadavky k dosiahnutiu umelej inteligencie [3]. Stavový priestor je reprezentovaný ako orientovaný graf, ktorý v skutočnosti tvorí strom. A skladá sa z uzlov a hrán. Kde hrany reprezentujú prechody medzi jednotlivými stavmi systému a ulzy predstavujú jednotlivé stavy daného systému.

Prehľadávanie stavového priestoru môžeme rozdeliť do dvoch základných skupín [17]. Prvá skupina je prehľadávanie pomocou takzvanej neinformovanej metódy, kde patria algoritmy ako sú napríklad prehľadávanie do šírky (BFS) alebo prehľadávanie do hĺbky (DFS). Neinformované metódy sú také metódy, kde algoritmus nepozná žiadne informácie o cieľovom stave a musí prehľadávať každý uzol.

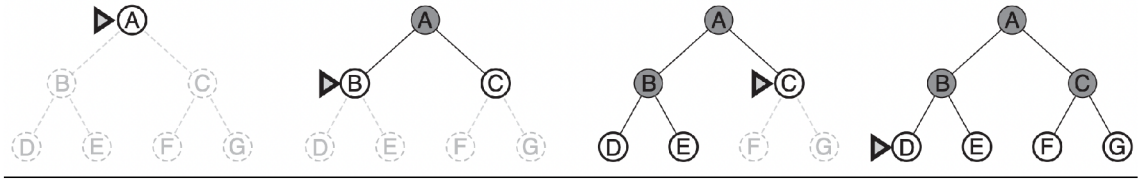
Naopak infomované metódy ako je napríklad A star (A*), poznajú potrebné informácie o cieľovom stave.

3.1 BFS

Prehľadávanie do šírky alebo po anglicky Breadth-First Search 3.1, je algoritmus, pri ktorom je ako prvý prehľadávaný koreňový uzol. Ak je koreňový uzol aj uzol cieľový, algoritmus končí. Ak nie, tak následne sú prehľadávaní všetci jeho následníci [15]. Platí, že prehľadávané sú najskôr všetky uzly v určitej hĺbke. BFS algoritmus funguje na princípe FIFO fronty, first in first out, čiže to znamená uzol, ktorý prišiel do fronty ako prvý bude prehľadaný a spracovaný ako prvý a prvý aj opustí frontu.

BFS algoritmus podľa fungovania môžeme rozdeliť do kritérií, že je úplný. Jeho úplnosť môžeme zistiť tak, že ak je jeho najspodnejší uzol, uzol cieľový na nejakej finálnej hĺbke, tak ho algoritmus nájde, keď prehľadá všetky uzly vo FIFO fronte nad ním. To znamená, že algoritmus BFS nájde vždy riešenie problému. Ale jeho úplnosť platí len v prípade, keď je jeho vetvenie uzlov konečné.

Ďalšia vlastnosť algoritmu je, že BFS je aj optimálny algoritmus. To znamená, že algoritmus nájde najkratšiu cestu k cieľovému riešeniu. A to tak, že postupne prehľadáva uzly na jednotlivých hĺbkach a až po vyčerpaní všetkých uzlov na danej úrovni, algoritmus prejde o hĺbku nižšie.

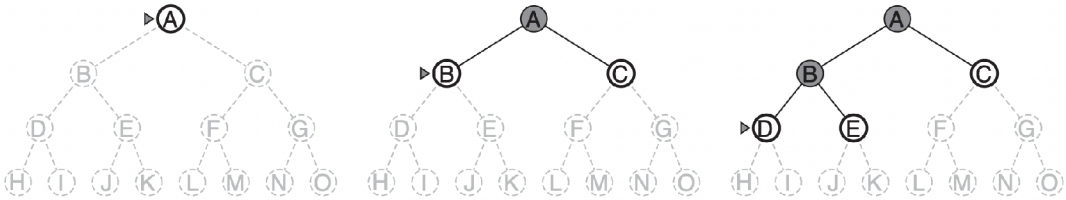


Obr. 3.1: BFS

3.2 DFS

Depth-First Search 3.2 alebo po slovensky prehľadávanie do hĺbky. Algoritmus ako pri BFS začína pri koreňovom uzly [15]. Ak je koreňový uzol výsledkom vyhľadávania, algoritmus končí prehľadávanie. Ale ak nie je, tak narozdiel od BFS, algoritmus neprehľadáva uzly na rovnakej hĺbke, ale každy ďalší prehľadávaný stav musí byť o úroveň nižšie ako uzol pred ním. Ak nie je prehľadávaný uzol výsledkom, tak sa vyhodí z fronty a prehľadávanie pokračuje od najvyššej hĺbky, ktorá ešte nebola prehľadaná. K tomuto DFS využíva typ fronty LIFO, last in first out. Čiže posledný uzol, ktorý prišiel do fronty bude spracovaný a vyhodnotený ako prvý.

Základná verzia DFS nie je optimálna a ani nie je úplná. Nie je optimálny, lebo najskôr bude prehľadávať jeden uzol do hĺbky, pričom prehliadne uzol, ak by sa nachádzal na takej istej úrovni ako skúmaní uzol. Čo by sa pri BFS algoritme nestalo. A úplný môže byť, ak sa prehľadáva konečný graf, ale pri nekonečnom grafe môže skúmať nekonečne dlhú vetvu a nikdy nepríde na výsledok, ak sa výsledok nachádza v inej vetve.



Obr. 3.2: DFS

3.3 A*

A star patrí medzi najznámejšie používané algoritmy[15]. Algoritmus funguje na princípe sčítania dvoch hodnôt $g(n)$ a $h(n)$. $G(n)$ označuje cenu k dosiahnutiu uzla a hodnota $h(n)$ označuje cenu, ktorú potrebujeme prejsť od uzla k cieľu. Takže platí vzťah $f(n) = g(n) + h(n)$. Kde $f(n)$ označuje odhadovanú cenu najlacnejšieho riešenia. Takže, ak sa snažíme nájsť najlacnejšie riešenia problému, je najlepšie začať s uzlom, ktorý má najmenšiu cenu pri $g(n) + h(n)$.

Algoritmus je úplný a je aj optimálny[17]. Jeho optimálnosť zaisťuje, že jeho heuristika nikdy neprecení cenu na dosiahnutie uzla.

Kapitola 4

Predchádzajúci výskum a podobné nástroje

V tejto kapitole sa budeme venovať predchádzajúcim poznatkom, ktoré priniesli dôležité poznatky pri implementácii automatického hrania hier Hľadanie Mín a Solitaire. Poznatky, ktoré priniesli pri písaní a implementácii minulej bakalárskej práce. Povieme si, koľko to jednotlivým hrám trvalo, kým sa im podarilo dohrať. A aj to aké knižnice a návrhy boli použité pri implementácii. V ďalšej časti si povieme, či existujú na internete programy, ktoré dokážu dohrať tieto hry a ak takéto programy existujú, povieme si koľko to jednotlivým programom trvá a aj to aké algoritmy boli pri jednotlivých hrách použité.

4.1 Predchádzajúce poznatky

Táto práca je postavená na predchádzajúcich poznatkoch z bakalárskej práce, vypracovanej študentom Stanislavom Přikrylom, počas bakalárskeho štúdia na tému "Program pro automatické hraní her Solitaire a Miny". Táto kapitola čerpá z literatúry [14]. Cieľom tej práce bolo čitateľa oboznámiť o teórii a pravidlách hier Solitaire a Míny, následne navrhnúť, naiplementovať a otestovať dva programy pre automatické hranie.

Prvý program sa mal zaoberať hrou Hľadanie Mín. Detekciou okna a rozpoznania jednotlivých hracích políčok. Na tento problém si študent vybral Python knižnicu python-xlib[1] a knižnicu wnck7[2]. Výstupom tejto aplikácie mal byť program, ktorý dokáže čo s najväčšou presnosťou dohrať automaticky hru Hľadnie Mín. A to sa v závere otestovalo a dokázalo, že program bol schopný to dohrať s pravdepodobnosťou 91% bez dotyku programu na mínu, ale to programu pri tej najľahšej obtiažnosti trvalo v priemere až 16,63 sekundy pri strednej obtiažnosti to bolo 67,85 sekundy a pri obtiažnosti expert to bolo až 174,84 sekundy.

Druhý program rieši hru Solitaire, pri ktorej sa taktiež musel zaoberať detekciou okna, aby bol program schopný a vedel rozlíšiť o aký druh karty sa jedná. K tomuto autor využíva knižnicu `OpenCV` a z tejto knižnice používa funkciu `matchTemplate`, ktorá mu poskytne všetky potrebné informácie o detekovanej karte. Tým pádom si musel študent navrhnúť vhodnú dátovú štruktúru pre reprezentáciu celej hernej plochy a jednotlivých herných prvkoch ako sú zásobník kariet na hranie a zásobník na odkladanie kariet, sĺpce kariet na vykladanie a tableau. K detekcii o akú časť hracej plochy ide, autor rozdelil hraciu plochu do takzvaných regiónoch. A vďaka tomuto vždy vedel aká karta sa kde na ploche nachá-

[1] https://python-xlib.github.io/python-xlib.html
[2] https://lazka.github.io/pgi-docs/Wnck-3.0/classes/Window.html

dza. Autor v sekcii testovanie uvádza, že dohranie tejto hry môže trvať až pár hodín. Kvôli tomu, že hra musela prejsť veľmi veľa stavmi, ktoré musela preskúmať, až sa dokázala dohrať. A uvádza aj to, že nie všetky partie hry Solitaire môžu byť dohrané.

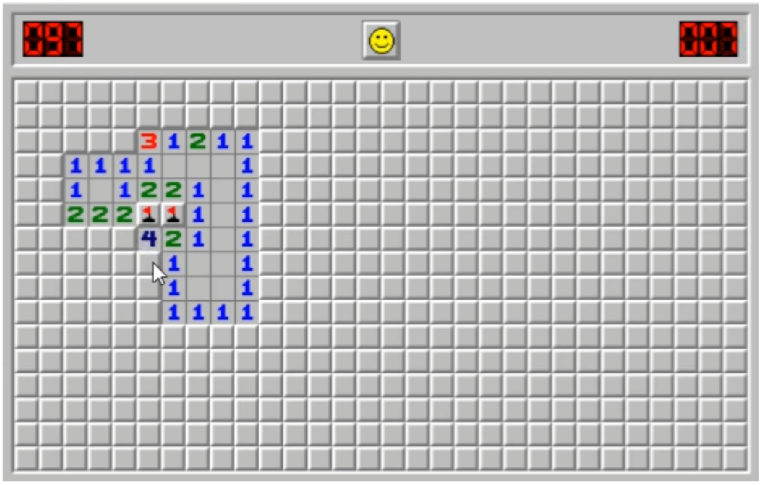
Mojím cieľom tejto práce je pokúsiť sa obe hry zrýchliť, aby dosahovali čo najlepšie časy pri úspešnom dohraní hier.

4.2 Existujúce programy

V čase písania tejto práce na internete existuje veľké množstvo podobných programov na automatické hranie a dohranie hier Hľadanie Mín a Solitaire. Ako sú napríklad:

minesweeper-solver

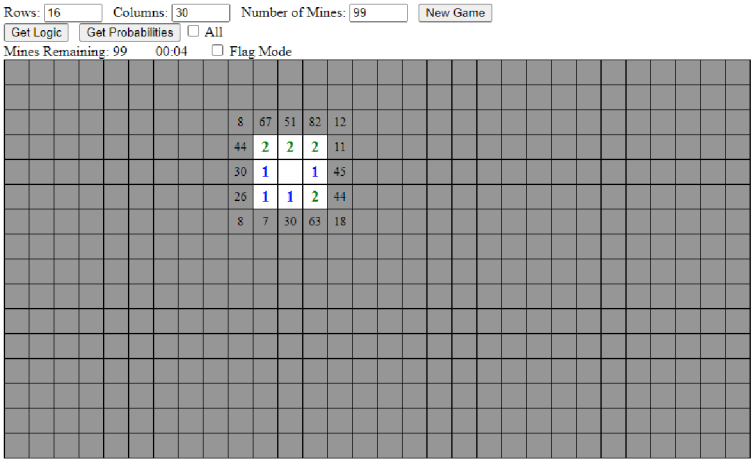
Jedným z programov na hranie Hľadanie Mín je program `minesweeper-solver` [5]. Tento program bol vyvinutý v programovacom jazyku Python. Pre fungovanie programu, aby vedel detekovať jednotlivé políčka alebo celú plochu hry autor, Gamescomputersplay , využíva snímky obrazovky. Program funguje na princípe, že k dohratiu hry a teda dosiahnutiu, ktoré políčko je najvýhodnejšie stlačiť, autor využiva sedem rôznych metód. Ako sú napríklad prvý klik, zoskupovanie alebo využíva pravdepodobnosť. Úspešnosti tohoto programu sú pri začiatočníckej obtiažnosti až 81% , pri strednej obtiažnosti to je 75% a pri tej najtažšej to je len okolo 30% pri spustení až 1000 hier.



Obr. 4.1: Snímok z hry `minesweeper-solver`

Minesweeper Probability Calculator

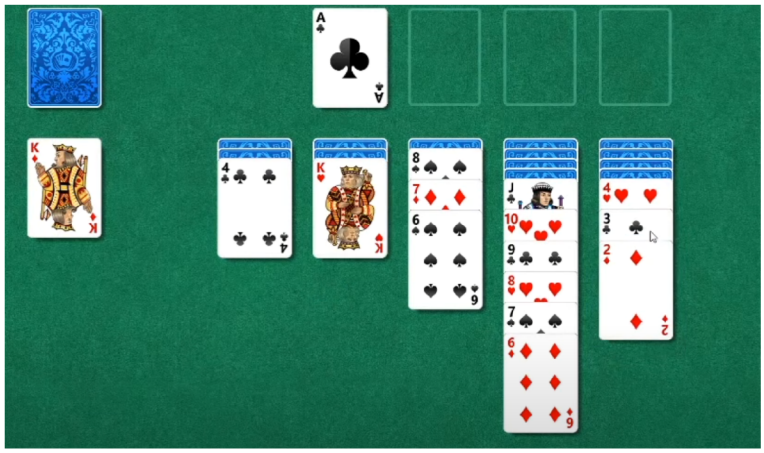
Ďalším program na automatické hranie hry Hľadanie Mín je `Minesweeper Probability Calculator`, ktorý funguje na princípe iba pravdepodobností [11]. Tento program funguje tak, že dostane plochu, kde sa nedá určíť mína na 100%, tak pre túto plochu vyráta všetky možné kombinácie, kde sa môže mína nachádzať. Následne sa všetky tieto možné pozície míny skombinujú a vznikne výsledná pravdepodobnosť, kde sa môže mína nachádzať, ale to platí len pre aktuálnu plochu.



Obr. 4.2: Snímok z hry `Minesweeper Probability Calculator`

Solvitaire

Pre hranie hry Solitaire existujú tiež podobné programy. Jeden program, ktorý dokáže dohrať automaticky hru Solitaire sa nazýva `Solvitaire`[1]. Solvitaire funguje len na operačnom systéme Windows. `Solivitaire` je schopný dohrať jedno kolo hry aj pod 4 minúty.



Obr. 4.3: Snímok z hry `Solvitaire`

Kapitola 5

Nástroje na detekciu ovládacích prvkov

V oblasti automatického hrania hier existujú prostriedky a nástroje, ktoré zjednodušia návrh aplikácie, ktorá je schopná automatizovať proces hrania hier. Nástroje, ktoré sú schopné emulovať uživatelské potreby ako sú kliky myšou, pohyb myšou po obrazovke, preťahovanie kariet z jednej kopy do druhej a stláčanie jednotlivých kláves na klávesnici. Na takéto úkony slúži knižnica `PyAutoGUI` [16].

Ďalším nástroj pre detekciu aktívneho okna počítača je knižnica `PyGetWindow`. A treťou knižnicou pre detekciu okna aplikácie a detekciu jednotlivých častí herného poľa je knižnica `OpenCV`.

Tieto knižnice umožnujú vývojarom vytvárať rôzne automatizované programy a skipty.

5.1 PyAutoGUI

Jeden z nástorojov pre detekciu odvládacích prvkov na obrazovke a emuláciu uživatelských akcií je nástroj `PyAutoGUI`. PyAutoGUI pracuje na operačných sytémoch ako je Windows, Linux alebo macOS [16].

PyAutoGUI je nástroj, ktorý dokáže automaticky potiahnuť našu myš na požadované súradnice, automaticky stlačiť tlačítka na tejto myši. Táto knižnica obsahuje aj funkcie na spravenie snímku obrazovky. Pre prácu len s určitým aktívnym oknom alebo časťou plochy. A jednou z ďalších funkcií je, že dokáže aj stláčať klávesy, ktoré mu zadáme.

Pre fungovanie nástroja je potrebné mať nainštalovaný buď Python 2 alebo Python 3 a nainštalovanú samotnú knižnicu PyAutoGUI. PyAutoGUI sa inštaluje pomocou príkazu `pip install pyautogui`.

5.1.1 Pohyby myšou

PyAutoGui umožňuje pohybovať alebo robiť úkony s myšou. Pri zadaní príkazu `moveTo()` [16], program posunie kurzor myši na obrazovke na zvolené súradnice X,Y. Ďalším príkazom pre stlačenie myši je `pyautogui.click(X,Y, button='left')`, kde program posunie myš na súradnice na obrazovke a namiesto toho, aby tam len prišiel, tak aj stlačí ľavé tlačítko na myši. Tlačítka, ktoré môže na myši stlačiť sú pravé (right), ľavé (left) a stredné tlačítko (middle) Nasledujúci príkaz je príkaz `pyautogui.dragTO(X,Y)`, kde myš bude potiahnutá na súradnice.

5.1.2 Lokalizácia okna

Pre detekciu okna, využíva tento nástroj príkaz `pyautogui.size()`. Táto funkcia vracia výšku a šírku monitora v pixeloch. Ľavý horný roh má súradnice 0,0 a vpravo dole sú súradnice, aký veľký je monitor. Pri práci s retina[1] displejom, ktoré majú predovšetkým macOS zariadenia , PyAutoGUI vracia pri všetkých funkciách súradnice dvakrát také veľke[13]. Čo môže byť pri práci nevýhoda.

Táto funkcia ale neumožňuje lokalizovať okno, ktoré potrebujeme. Dokáže to detekovať len veľkosť obrazovky.

5.1.3 Snímok obrazovky

Pre vytvorenie snímku obrazovky PyAutoGUI využíva viacej príkazov ako sú pyautogui.screenshot(), pyautogui.locateOnScreen(), pyautogui.locateCenterOnScreen() [16].

Prvý príkaz slúži na vytvorenie snímku obrazovky celého displeja. Druhý príkaz sa používa, keď potrebujeme nájsť vzorku obrázku na obrazovke. Táto funkcia, keď nájde obrázok na displeji vracia súradnice, výšku a šírku tohto obrázku a keď sa taký vzor nenájde funkcia vracia `NONE`. Tretí príkaz vracia stredné súradnice bodu X a Y, kde sa na displeji nachádza požadovaný obrázok.

A príkaz `pyautogui.locateAllOnScreen()` vracia súradnice, výšku a šírku všetkých nájdených obrázkov na displeji.

5.1.4 Stlačenie klávesnice

Pre prácu s klávesnicou PyAutoGUI používa príkaz `pyautogui.press()`. Ktorý stlačí požadovanú klávesu. Tento príkaz už obsahuje v sebe zabudované funkcie ako stlačenie klávesy a následne pustenie tejto klávesy.

Ale ak by sme chceli iba stlačiť tlačítko a nepustiť ho, tak PyAutoGUI využíva príkazu `pyautogui.keyDown()`. Následne ak by sme chceli klávesu už uvoľniť, tak sa použije príkaz `pyautogui.keyUp()`.

[1] Displaje spoločnosti Apple, ktoré majú väčšiu hustotu pixelov

Kapitola 6

Návrh programu pre automatické hranie hier

V tejto kapitole sa pozrieme na návrhy programov pre automatické hranie hier Solitaire a Hľadanie mín. Pozrieme sa na návrh, ktorý bude napísaný v programovacom jazyku Python s použitím knižníc ako sú napríklad `PyGetWindow` alebo `OpenCV`.

Najskôr sa zameriame na návrh spustenie hry. Pri Hľadaní Mín napríklad ako sa bude vyberať obtiažnosť pri spustení hry. Alebo pri hre Solitaire aby program bol schopný dať hru do full–screen rozlíšenia.

Následne sa zameriame na samotný návrh na detekciu hracieho poľa pre jednotlivé hry. Teda ako by bol program schopný pri hre Hľadanie Mín detekovať napríklad celú hraciu plochu, jednotlivé bunky hracieho poľa alebo keď hru program prehrá, aby vedel nájsť možnosť pre reštartovanie hry. Pri hre Solitaire bude treba detekovať každú časť hracieho poľa zvlášť. Či sa bude jednať o tableau, základnú hromádku, zásobník alebo odpadový kôš.

Cieľom tejto kapitoly je pripraviť návrh, aby výsledný program bol schopný spustenia sa, vedel, ktoré časti potrebuje pre automatické hranie. Aby program vedel pracovať bez pomoci, či už s myšou alebo klávesnicou. Z tejto kapitoly sa následne bude odvíjať samotná implementácia a záverečné testovanie programu.

6.1 Spustenie hier

Rozhodol som sa, že obidve hry budem spúšťať pomocou využívania Python knižnice `subprocess`, ktorá nám umožňuje používať príkaz ako je napríklad `subprocess.Popen([ ])`. Pomocou tohto príkazu by sme mali byť schopní spustiť hru aj pre algoritmus na dohranie Mín aj pre algoritmus na dohranie hry Solitaire.

Hra Hľadanie Mín by na začiatku mala byť schopná pomocou argumentov, ktoré program dostane si vedieť zvoliť, ktorú obtiažnosť má riešiť. A tak isto by som chcel spraviť pomocou knižnice spomínanej v kapitole 5. Pomocou príkazov buď `locateOnScreen()` alebo `locateCenterOnScreen()`, ktoré by mali byť schopné nájsť pomocou obrázkov uložených v adresári, kde sa nachádzajú a následne stlačit nájdené súradnice. A to tak, že pomocou snímku nájde, kde sa nachádza menu aplikácie. To následné stlačí a potom podľa zvolenej obtiažnosti v tom menu nájde príslušnú plochu, ktorá sa zhoduje s daným obrázkom a stlačí túto plochu.

A zasa pri hre Solitaire pri spustení hry bude treba spraviť to, aby bola hra na celej obrazovke, a to taktiež po využití funkcie z PyAutoGUI knižnice. Volaním funkcie by sa hra mala vedieť dať do full-screen rozlíšenia.

6.2 Detekcia aktívneho okna

Detekcia aktívneho okna je jednou z najpodstatnejších častí pri návrhu programu. Zaisťuje nám prácu len s oknom, ktoré program potrebuje. Na detekciu aktívneho okna v Python prostredí bolo vyvinutých niekoľko knižníc ako sú napríklad `PyGetWindow` alebo `Win32GUI`. Ale pre túto prácu som zistil, že najlepšie bude použiť knižnicu `PyGetWindow`.

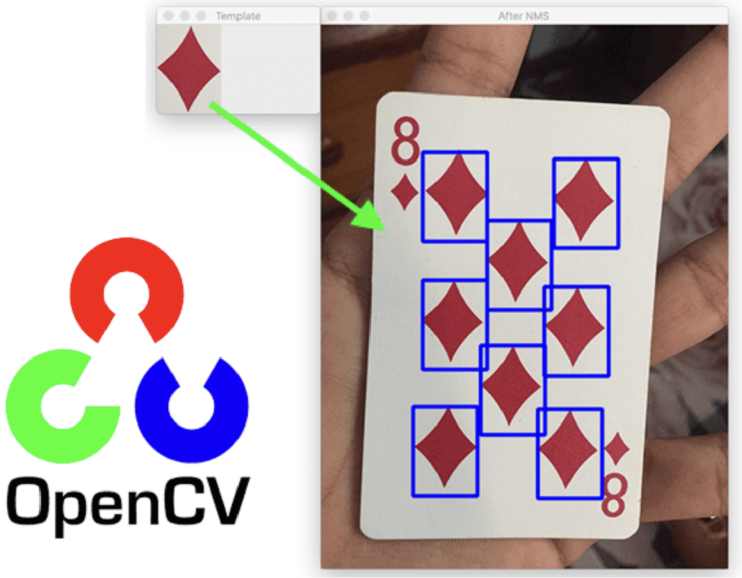
PyGetWindow

PyGetWindow je jedna z knižníc, ktoré nám umožňujú detekovať aktívne okno programu. Dovoľuje nám získať jeho rozmery ako sú jeho začiatočné súradnice bodov X a Y ale aj jeho výšku a šírku, ktorú budeme potrebovať pri implementácií programu. Táto knižnica funguje na všetkých zariadeniach, či už majú Windows, Linux alebo dokonca aj MacOS [2]. Túto knižnicu chcem hlavne využívať pri hre Hľadanie Mín. Aby som presne vedel, kde sa bude hra na ploche nachádzať. Pri hre Solitaire túto knižnicu asi potrebovať nebudem. Lebo tam bude program pracovať s celou plochou obrazovky a nie len s nejakou určitou časťou.

6.3 OpenCV

OpenCV patrí medzi knižnice, ktoré sú taktiež podporované pri programovacom jazyku Python. Táto knižnica je veľmi užitočný nástroj pre programátorov, ktorí pracujú v oblasti počítačového videnia. OpenCV sa používa v oblastiach ako sú napríklad detekcia objektov, spracovanie objektov, analýza videí alebo aj kalibrácia kamier [4]. OpenCV knižnicu som sa rozhodol používať pri detekcii jednotlivých hracích elementov pri hre Hľadanie Mín. Táto knižnica by nám mala byť schopná povedať podľa farby pixelov o akú číslicu ide. Napríklad ak tam nájde pixel s modrou farbou, tak by nám program mal byť schopný povedať, že sa okolo tejto bunky bude nachádzať práve jedno políčko s mínou.

A pri hre Solitaire by som chcel pomocou tejto knižnice a funkcie `Template Matching` navrhnúť funkciu, ako môžeme vidieť na obrázku 6.1, ktorá dokáže porovnať pomocou snímku z regiónu jednotlivé karty, ktoré budú uložené v zložke a povie nám o akú kartu sa jedná, aká je farba karty, aké má karta číslo a aj na akých súradniciach sa karta bude nachádzať.



Obr. 6.1: Ukážka ako by malo fungovať `Template Matching`

Detekcia políčok pri hre Hľadanie Mín

Na rozpoznávanie o aké políčko pôjde existuje viacero spôsobov. Jedným z nich je využiť knižnicu `PIL` a funkciu z tejto knižnice `getpixel`, ktorá bola využívaná pri minulej bakalárksej práci. Ale z dôvodu, že chcem zrýchliť detekovanie pre rýchlejší výsledný čas, tak navrhujem využiť knižnicu `OpenCV`, ktorá dostane všetky možné farby pixel, ktoré sa budú môcť nájsť na ploche. Následne zavediem funkciu `prechodHracouPlochou`, ktorá by mala byť schopná prechádzať celé hracie pole, políčko po políčku, a keď na danom políčku nájde farbu, ktorá prislúcha danému číslu. Tak toto číslo uloží do globálnej premennej `global hraciePole`.

hraciePole[x][y] = farba_pixelu

A vďaka tomuto by som mal byť schopný presne vedieť, kde sa aké číslo nachádza. A algoritmus bude vedieť, či dané políčko už je stlačené alebo nie, a aj to koľko mín sa tam bude nachádzať.

Detekcia častí hry Solitaire

Pri návrhu ako budem detekovať, o ktorú časť hracieho poľa ide pri hre Solitaire, či už ide o tableau plochu alebo odpadový kôš alebo zásobník. Som sa rozhodol navrhnúť, že hru rozdelím do niekolkých častí. Napadlo mi, že najlepšie by to bolo rozdeliť do troch hlavných častí. Prvá časť alebo región, by bol región, kde sa bude nachádzať zásobník a odpadový kôš. Ďalším regiónom by bol región s tableau plochou a všetkými kartami v tej ploche. A posledný región by bola časť, kde sa budú karty vykladať.

Toto by malo predstavovať zjednodušenie pre porovnávanie kariet, kde sa ktorá a aká karta bude nachádza. Budú sa robiť snímky obrazovky práve regiónov, kde sa bude nachádzať tableau plocha a druhý snímok bude snímok, kde sa bude nachádzať zásobník

a odpadový kôš. A karty by sa podľa toho, kde ich program nájde, uložia do daného pola kariet, či už pole zásobníku alebo pole tableau kariet.

Detekcia kariet pri hre Solitaire

Pri detekcii jednotlivých kariet v celej Solitaire hre musím navrhnuť funkciu, ktorá by mala dostať región, kde má karty detekovať. Keď bude funkcia vedieť, v ktorej časti má karty detekovať tak zavolá funkciu `Template Matching`, ktorá by mala dostať snímok toho regiónu a všetky karty, ktoré sa v hre môžu nachádzať. A k tomuto by som mal navrhnúť nejakú funkciu, ktorá buď podľa mena karty detekuje, o ktorú kartu sa jedná. Že nám napíše číslo karty, znak karty a aj farbu karty. Alebo uložím do nejakého súboru čísla a znaky farieb. A pri porovnávaní o akú kartu sa jedná, by sa porovnávalo aj aké číslo a znak našiel na karte. A podľa znaku by sa zistila farba karty. A v závere by nám vzniklo nejaké pole kariet, ktoré by obsahovalo všetky potrebné informácie pre hru.

poleKariet = [(farba_karty, znak_karty, číslo_karty, lokácia_karty)]

Návrh dostupných ťahov pre hru Hľadanie Mín

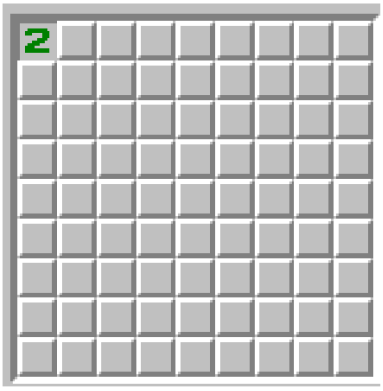
Pred samotnou implementáciou algoritmov pre hru Hľadanie Mín, musím navrhnúť ako budú algoritmy postupovať, aby vedeli hru automaticky dohrať, a aby vedel, keď program už, zistí, aké políčko sa tam nachádza, a vedel čo má s ním robiť. Ako som spomínal v kapitole 2, máme k dispozícii 5 algoritmov, ktoré by som chcel naimplementovať.

Program bude zakaždým stláčať horný ľavý roh. Ak v tomto rohu nájde číslo, ako môžeme vidieť na obrázku 6.2, tak ho zapíše do premennej `hraciePole[x][y]` a následne by mal program byť schopný pomocou nejakého `for cyklu` ako napríklad:

for i in range(max(0, row - 1), min(hraciePole[x], row + 2))

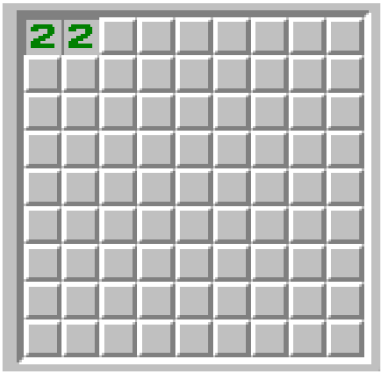
for j in range(max(0, col - 1), min(hraciePole[y], col + 2))

pomocou tohoto cyklu, by mal program byť schopný prejsť všetkých susedov tohoto čísla a keď tam, ako na obrázku nájde iba neodokrytých susedov, tak zistí koľko mín sa ešte môže nachádzať v hre. V tomto prípade to môže byť viac ako susedov, tak každému susedovi priradí pravdepodobnosť 66,67, že sa tam mína bude nachádzať.



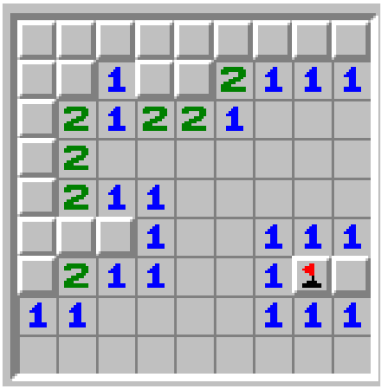
Obr. 6.2: Prvý klik

A ak program odokryje ďalšie políčko pri tomto čísle 6.3, tak sa program vráti k prvému odokrytému číslu 2 a zasa porovná, či sa počet neodokrytých políčok rovná hodnote skúmaného čísla. V tomto prípade sa čísla rovnajú, tak program by mal do hraciePole[x][y] na pozíciu týchto dvoch neodokrytých políčok zapísať číslo -2, čo by program vedel, že sa jedná o mínu



Obr. 6.3: Políčko s mínou

A posledný problém, ktorý by mal program vyriešiť pomocou for cyklu, korý nájde všetkých neoodkrytých susedov je 6.4. V tomto prípade algoritmus príde napríklad na políčko, ktoré sa nachádza v šiestom riadku a ôsmom stĺpci a zistí, že ako susedov už obsahuje taký počet mín, ktorý sa rovná hodnote políčka. Následne skontroluje, či sa pri tomto políčku nachádzajú ešte nejaké neodokryté políčka. V tomto prípade áno, tak na všetky takéto políčka algoritmus bude môcť stlačiť, lebo bude vedieť, že sa pod takýmito políčkami nebude nachádzať mína na 100%.



Obr. 6.4: Bezpečný klik

Návrh dostupných ťahov pre hru Solitaire

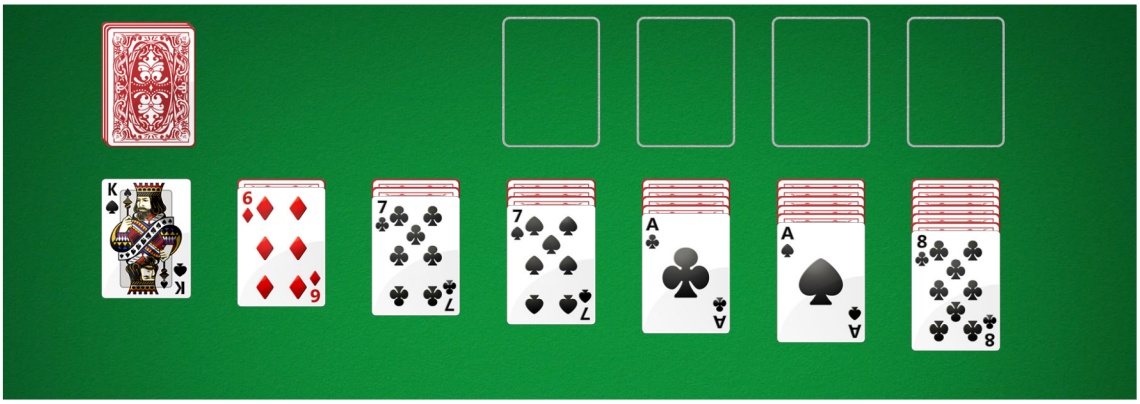
V tejto časti sa zameriame na návrh možných ťahov pri akutuálnej konfigurácii pri hre Solitaire. Hľadanie týchto ťahov zahŕňa dôkladné preskúmanie herného plánu a identifikáciu kariet, ktoré môžeme presunúť. Pri tomto kroku musíme brať do úvahy pravidlá hry, ktoré určujú obmedzenia pre presuny kariet medzi stĺpcami na tableau, manipuláciu s kartami v zásobníku a ich presuny do vykladacej hromádky.

Prvým krokom navrhujem pred začatím každej hry načítať celý obsah zásobníka do globalnej premennej, aby sa nám uľahčila práca so zásobníkom. To nám umožní presne vedieť koľko kariet musíme v zásobníku stlačiť, aby sme sa dostali na požadovanú kartu. Potom by mal byť program schopný, ak nájde v tomto zásobníku kartu eso, tak by ju mal hneď vyložiť do vykladacej hromádky.

Keď bude zásobník zapísaný, tak by hra mala pokračovať na tableau plochu. Najskôr by mal program byť schopný, ak sa tam nájde karta eso, ako na obrázku 6.5, tak ju hneď vyloží do vykladacej hromádky. Keď vyloží všetky esá, začne uplatňovať pravidlá ako sú, že karta musí byť o jedno menšia a musí byť opačnej farby. Týmto pravidlom by sa mali premiestniť všetky karty, ktoré sa nachádzajú v zásobníku a aj na tableau.

Hneď ako prejde cyklus, ktorý premiestňuje karty, tak sa zavolá funckia, ktorá dokáže vykladať karty do vykladacej hromádky, aby sa hra dohrala. A to tak, že budem musieť naimplementovať funkcie, ktoré budú vedieť, že karta je o jedno väčšia a je aj rovnakej farby ako karta, ktorá sa už nachádza vo vykladacej hromádke.

Program bude obsahovať počítadlo, koľko a akých ťahov môže vykonať. Ak toto počítadlo, už nebude obsahovať žiadny možný ťah, tak sa bude musieť naimplementovať funkcia, ktorá celú hru vráti do bodu, kde sa program mohol rozhodnúť inak a následne program vyskúša iný smer premiestňovania kariet.



Obr. 6.5: Vyloženie esa

Kapitola 7

Implementácia programu

V tejto kapitole sa zameriame na opis implementácie oboch programov ako Hľanie Mín tak aj Solitairu. Ukážeme si, aké techniky som využil pri detekovaní obrazovky, detekovaní jednotlivých hracích elementov a aj aké algoritmy a pravidlá som uplatnil.

Prvá časť kapitoly sa bude zaoberať hrou Hľadanie Mín. Pri tejto hre si ukážeme ako som detekoval hraciu plochu, ako som detekoval jednotlivé hracie políčka a aj ako je schopný algoritmus dohrať hru. V druhej časti kapitoly si povieme ako som rozdelil hru na regióny, aby som bol schopný vedieť, v ktorej časti hracej plochy sa jednotlivé karty nachádzajú, ako som detekoval jednotlivé hracie karty a aké prehľadávanie stavového priestoru som využil, aby algoritmus bol schopný dohrať hru skoro vždy.

Cieľom tejto kapitoly je popísať ako sú obidva programy implementované, vysvetliť ťažšie veci pri implementácií a ukázať ako obidva programy pracujú a vo výsledku sa dajú dohrať v čo najlepšom čase.

7.1 Implementácia pre hru Hľadanie Mín

Program je napísaný v programovacom jazyku Python, ktorý využíva Python knižnice ako sú PyautoGUI, OpenCV, Numpy, PyGetWindow alebo Random. V tejto časti si popíšeme ako som naimplementoval túto hru a čo ma viedlo použiť veci, ktoré som pri implementácii použil.

Štruktúra programu

Tento program Hľadanie Mín je štrukturovaný do troch základných súborov:

- **mine_field.py**
  - Súbor `mine_field.py` predstavuje všetky potrebné funkcie, pre detekciu hracej plochy, detekciu celej hry Hľadanie Min, funkcie potrebné pre detekciu jednotlivých hracích políčok, ale aj funkcie, ktoré zabezpečujú výber obtiažnosti a samotný začiatok hry.
- **algorithm.py**
  - Druhý súbor, súbor `algorithm.py`, obsahuje všetko potrebné, aby sa hra v čo najlepšom čase dala dohrať. Je v ňom napísaný súbor pravidiel a algoritmov pre hru.

- **play.py**
  - Posledným súborom je súbor `play.py`, ktorý slúži na to, že obsahuje cyklus, ktorý hrá hru, meria čas za koľko bude hra dohratá a hovorí nám, či sa nám hru podarilo dohrať alebo sme stupili na mínu.

Spustenie hry a výber obtiažnosti

Ako som spomínal v kapitole 6 samotná hra sa spúšťa pomocou funkcie `subprocess.Popen`. A následne algoritmus pre dohranie hry sa spúšťa príkazom `\play` a za tým nasleduje výber obtiažnosti od jedna po tri. Takže napríklad hra expert, ktorá obsahuje 16 krát 30 políčok a v nich až 99 mín, by sa zapla ako `\play 3`.

Detekcia hry

V tejto časti sa zameriame na potrebnú detekciu pri hre. Na detekciu aktívneho okna hry, ktoré sa pri spustení zapne som využil knižnicu `PyGetWindow`, ktorá dokáže pomocou funkcie:

active_window = gw.getActiveWindow()

nájst horné súradnice bodov X a Y a dokáže aj zmerať dĺžku a výšku celej plochy hry. Tak vďaka týmto súradniciam sme schopní povedať algoritmu, kde má na obrazovke klikať.

Detekcia hracej plochy

K detekcii hracej plochy využívame snímku celej hry a musíme nájsť potrebné kontúry v tomto snímku, aby sme boli schopní vedieť, kde sa nachádza iba hracie pole. A to dokážeme získať pomocou farby pixel štvorca, ktorý sa nachádza okolo hracieho poľa. Tento štvorec ale pozostáva až z dvoch farieb. Naľavo a hore sa jedná o bielu farbu a napravo a dole ide o farbu šedú. Takže k detekcii bolo treba napísať tento kód viď 7.1:

```
105     colorBoardMaskGrayLower = np.array([125, 125, 125])
106     colorBoardMaskGrayUpperr = np.array([130, 130, 130])
107
108     colorBoardMaskWhiteLower = np.array([250, 250, 250])
109     colorBoardMaskWhiteUper = np.array([255, 255, 255])
110
111     mask1 = cv2.inRange(scanBoard, colorBoardMaskGrayLower, colorBoardMaskGrayUpperr)
112     mask2 = cv2.inRange(scanBoard, colorBoardMaskWhiteLower, colorBoardMaskWhiteUper)
113
114     WhiteGray = cv2.bitwise_or(mask1, mask2)
115
116     contours, _ = cv2.findContours(WhiteGray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
117
118     square_contour = max(contours, key=cv2.contourArea)
119
120     cv2.drawContours(scanBoard, [square_contour], -1, (0, 255, 0), 2)
```

Obr. 7.1: Kód na detekciu hracej plochy

Popis jednotlivých funkcií kódu:

- **colorBoardMask**

  - Hodnota `colorBoardMask` obsahuje hodnoty pixelov farby, ktoré má program hľadať na snímku obrazovky celej hry. Ako som spomínal vyššie, štvorec obsahuje dva rôzne druhy farieb, tak bolo treba pracovať aj pixelmi s bielou farbou aj s pixelmi svetlo šedou farbou.

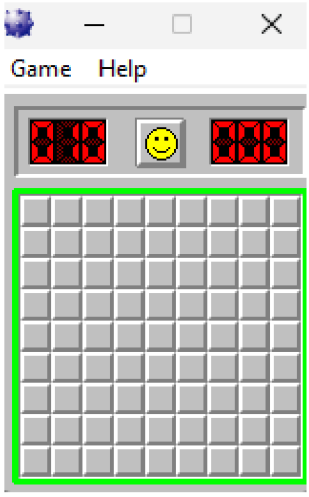
- **cv2.inRange**
  - Funkcia `cv2.inRange` vytvára binárnu masku , ktorá prijíma tri parametre. Prvý parameter je vstupný obraz, čiže pri nás je to snímok celej hracej plochy. Druhým a tretím parametrom sú dolná a horná medza, ktoré sa majú na obrázku hľadať.

- **cv2.bitwise_or**
  - Táto funkcia dostáva dve hodnoty, hodnoty dvoch masiek, a vytvára kombináciou výsledný nový obraz.

- **cv2.findContours**
  - Pomocou tejto funkcie a parametrov `cv2.RETR_EXTERNAL`, ktorý bude hľadať iba vonkajšie kontúry a `cv2.CHAIN_APPROX_SIMPLE` pomocou, ktorého sa uchovávajú iba krajné body kontúry. Sme schopní na snímku z obrazovky nájsť, kde sa presne nachádza len hracie pole.

- **cv2.drawContours**
  - `cv2.drawContours` slúži len pri debuggovaní, či sa naozaj našla plocha, ktorú sme potrebovovali, viď obrázok 7.2.

- **square_contour**
  - Pomocou tejto hodnoty, keď ju dosadíme do funkcie `cv2.boundingRect` sme schopní dostať súradnice hracieho poľa, a keď tieto súradnice skombinujeme so súradnicami, kde sa nachádza celá hra, dostaneme súradnice hracieho poľa, s ktorými bude algoritmus už pracovať.



Obr. 7.2: Detekcia hracieho poľa

Detekcia políčok

Na detekciu políčok, či už ide o políčko odokryté alebo neodokryté, políčko, kde vybuchla mína alebo políčko len s číslom, som taktiež využil knižnicu `OpenCV` a s pomocou nástroja `GIMP` som zistil, aké farby majú jednotlivé políčka pixeli.

Pri detekcii, keď `cv2.findContours` nájde o akú farbu sa jedná, je zavedená globálna premenná `global matrix`, do ktorej sa na presnú pozíciu, kde sa farba v poli nachádza uloží číslo podľa farby pixelu. Pričom číslo s hodnotou -1 značí, že políčko ešte nebolo odhalené a číslo s hodnotou -2 znamená, že bolo stlačené políčko s bombou.

Využité algoritmy

Pri implementovaní programu bolo treba navrhnúť algoritmy, ktoré dokážu dohrať automaticky túto hru. Tento program využíva štyri rôzne druhy algoritmov pre nájdenie mín, ktoré sú napísané v súbore `algorithm.py`. Algoritmy sú to nasledovné:

Prvý klik

Začiatok hry a teda prvý klik algoritmu je pri každej hre políčko, ktoré sa nachádza naľavo a úplne hore. Z dôvodu toho, že pri náročnejších obtiažnostiach redukuje množstvo mín, ktoré sa môžu nachádzať pri políčku.

Odhalenie míny

Prvým algoritmom, ktorý využíva program, je taký algoritmus, ktorý nám s určitosťou povie, že sa pod neodhaleným políčkom nachádza mína. A tento algoritmus funguje na princípe, že keď sa nám podarí odhaliť väčšia plocha alebo algoritmus má k dispozícií väčšiu plochu. Tak pomocou detekcie spomínanej vyššie nájde napríklad číslo 2. Algoritmus pomocou funkcie spočíta, koľko neodhalených políčok tvorí susedov tohto čísla 2. A ak sa to číslo zhoduje s čislom v našom prípade dva, tak s určitosťou algoritmus vie, že pod obidvoma neodhalenými políčkami sa nachádzajú míny a tieto políčka nebude algoritmus stláčať.

Odhaľovanie bezpečného políčka

Ďalším podobným algoritmom je `odhaľovanie bezpečného políčka`, ktorý funguje v princípe podobne ako algoritmus `odhaľovanie míny`, ale narozdiel od míny nám povie, kde má algoritmus stláčať. Funguje presne podobne, ale narozdiel od súčtu všetkých neodhalených susedov políčka s číslom dva, ho zaujíma koľko jeho susedov tvoria míny. Takže v našom prípade, ak by pri čísle 2 našiel presne dvoch susedov označených ako míny a ďalší dvaja susedia by boli ešte neodhalení, tak algoritmus s určitosťou vie zistiť, že práve tieto dve neodhalené políčka sú bezpečné a môže ich stlačiť.

Matematické riešenie

Ak algoritmus už nemože použiť prvé dve spomínané pravidlá, tak využíva takzavné matematické riešenie, ktoré funguje na princípe množín. Tento algoritmus iteruje cez všetky možné dvojice, pre každé políčko z dvojice nájde a uloží súradnice všetkých neodokrytých susedov. Následne zistí hodnoty aké majú obidve iterované políčka a od týchto hodnôt odčíta veľkosť políčok, ktoré sú označené ako mína, ale stále musia byť susedia a uloží tieto

hodnoty do dvoch premenných. A v závere porovná, ak sa rozdiel týchto dvoch premenných rovná veľkosti rozdielu súradníc s neodokrytími poličkami pri jednom a pri druhom iterovanom políčku. Ak sa hodnoty rovnajú, tak vieme s určitosťou povedať, že rozdiel množiny súradníc prvého a druhého políčka, budú políčka, ktoré sú míny. A naopak rozdiel množiny druhého iterovaného políčka od prvého nám povie súradnice, kde má algoritmus stlačiť, kde sa nenachádza mína.

Pravdepodobnosť

A posledný algoritmus, ktorý je, keď sa program už nevie rozhodnuť, kde má stlačiť a nevie použiť žiadny z vyššie spomínaných algoritmov, je výber pomocou pravdepodobnosti. Pravdepodobnosť v tomto programe funguje, že keď máme skúmané políčko s číslom dva a máme vedľa neodhalených troch susedov a ani jeden zo susedov zatiaľ nie je známy ako mína, a ani jeden algoritmus sa nedá už využiť, tak každému neodhalenému susedovi nastaví pravdepodobnosť 66% v tomto prípade, že sa tam bude nachádzať mína. A program po všetkých kombináciach zistí, kde má najmenšiu šancu stlačiť mínu, tak na to políčko stlačí.

Ukončenie hry

Celý proces dohrania hry beží vo `While cykle`, ktorý funguje na princípe, že bude stále bežať, dokým nenastane jedna z dvoch udalostí. Prvá udalosť je, že hra bola úspešne dohraná, ak program nenájde žiadne ďalšie možné súradnice v poliach `mine_cells` a `notmine`, kde má stlačiť, počítadlo mín sa bude rovnať celkovému počtu mín na začiatku a nebude musieť využiť funkciu na počítanie pravdepodobností, kde má stlačiť.

Ďalšia udalosť je podobná ako tá prvá, narozdiel od toho, že program nevie využiť algoritmy, ktoré vedia mínu odhaliť na 100%. Tak využije pravdepodobnosť. Ale pri tejto udalosti môžu vzniknúť až tri závery. Poprvé, že sa stlačilo políčko, kde sa nenachádzala mína a tým pádom by sa znovu išli použiť algoritmy, ktoré mínu odhalia na 100%. Alebo sa stlačilo políčko bez míny a počet označených políčok sa rovná počtu mínam na ploche, tak sa hra ukonči tak, že sa ju podarilo vyhrať. Alebo sa nájde mína a program sa ukonči s negatívnym výsledkom.

Program sa vie prehrať len vtedy, keď musí využiť počítanie pomocou pravdepodobnosti.

7.2 Implementácia pre hru Solitaire

Program pre automatické hranie hry Solitaire je taktiež naimplementovaný v jazyku Python. Za pomoci tiež knižníc ako sú `OpenCV`, `PyAutoGUI` alebo `Time`. V následujúcej časti popíšem ako som implementoval tento program, ako som dokázal rozoznať jednotlivé karty na hracom poli, ako som vedel, kde sa tie karty na hracom poli nachádzajú a aj to, aké prehľadávanie stavového priestoru som využil.

Štruktúra programu

Štruktúra celého programu pre automatické hranie hry Solitaire pozostáva z troch súborov, ktoré vzájomne využívajú svoje funckie. Štruktúra programu vyzerá následovne:

- **detection.py**

- `Detection.py` slúží k rozpoznaniu, s akou kartou program pracuje. Určuje číslo karty, znak karty a aj to, či sa karta nachádza v zásobníku alebo v tableau regióne.
- Ďalej tento súbor zabezpečuje aj to, že hra sa bude vždy hrať v full-screen rozlíšení.

- **algorithm.py**
  - Súbor `algotihm.py` tvorí všetko, čo hra potrebuje, aby sa vedela dohrať. Tvorí zbierku základných pravidiel, ktoré hra potrebuje. Keď hra už nemá dalšie možnosti, kde má algoritmus pokračovať s výberom a premiestnením karty, obsahuje funkciu na prehľadávanie stavového priestoru, kde sa má program vrátiť a pokračovať v inej vetve prehľadávania.

- **solve.py**
  - Posledný súbor, súbor `solve.py` obsahuje len funckiu, ktorá spúšťa celé hranie a meria výsledný čas.

Spustenie hry

Automatické dohranie sa spúšťa pomocou príkazu `python3 solve.py`. Hra sa taktiež spúšta pomocou príkazu `subprocess.Popen([])`. Kde sa najskôr hra musí dať do full-sreen rozlíšenia a to pomocou funkcie z knižnice `PyAutoGUI`. Funkia locateCenterOnScreen() nájde pomocou prednastaveného obrázku súradnice, kde sa nachádza zväčšenie obrazovky, pretože samotná hra nedovoľuje zväčšiť obrazovku pomocou stlačenia klávesových skratiek.

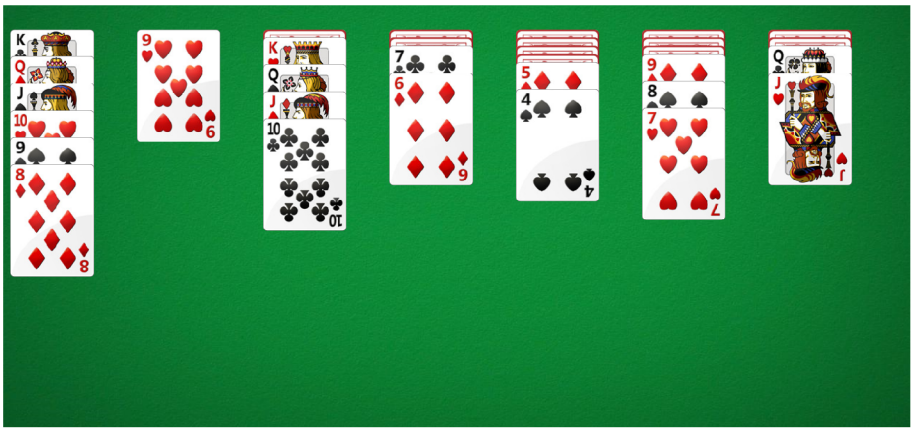
Detekcia hracej plochy

Hracia plocha je rozdelená pri detekcii v programe do troch základných regiónov.

Prvý región viď obrázok 7.3 je časť, kde sa nachádza zásobník. Tento región sa detekuje pomocou uloženého obrázku prvej neotočenej karty v zásobníku. Program, keď nájde súradnice tejto karty, tak sa pomocou týchto súradníc a dĺžky a výšky karty, ktoré tiež dostaneme, keď sa nájde karta, sa vypočíta región, kde sa nachádza zásobník a aj odpadový kôš, lebo ten sa nachádza hneď vedľa zásobníka. Tak pomcou tejto logiky dostaneme prvý región, kde sa budú detekovať karty, ktoré môžu byť vhodené do tableau regiónu alebo rovno byť uložené do základnej hromádky.

Obr. 7.3: Región so zásobníkom

Druhým regiónom viď obrázok 7.4 je región, kde sa nachádza tableau plocha. Tento región sa získa taktiež pomocou karty z prvého regiónu a teda pomocou tých súradníc sa vypočíta, kde sa nachádza ľavý spodný koniec tejto karty a podľa toho, kde sa nachádza, sa tieto súradnice uložia do bodu $X = 0$ a $Y = 0$. Takže snímok tohto regiónu bude začinať v týchto súradniciach a bude končiť tam, kde je pravý a spodný bod celej hracej plochy. Takže pomocou tohto výpočtu dostaneme len plochu, kde sa budú nachádzať len karty, určené na detekciu, ktoré sa nachádzajú v tableau časti.



Obr. 7.4: Región s Tableue

Posledný regón je ten, kde sa budú karty vyhadzovať na dohranie hry, ten sa bude detekovať úplne inak ako predchádzajúci a to len pomocou uložených obrázkov. Takže keď program nájde všetky možné pozície týchto obrázkov, tak ich súradnice uloží do globálnych premenných a vďaka tomu sme schopní premiestňovať karty do základnej hromádky.

Detekcia a rozpoznávanie kariet

Pre detekciu kariet z určitého regiónu využívam knižnicu `OpenCV`. Z tejto knižnice používam funkciu `matchTemplate()`. Táto funkcia dostane tri parametre. Prvým parametrom je obrázok regiónu s kartou, ktorú chceme detekovať. Ďalším parametrom sú všetky karty, ktoré snímky sú uložené v súbore `cards`. Posledný parameter je `cv2.TM_CCOEFF_NORMED`. Tento parameter je spôsob, ktorý zaobstaráva porovnávanie vzorov s obrazom.

Keď sa nájde karta, tak sa zavolá funkcia `get_number_and_symbol_color`, ktorá dostane názov karty ako je pomenovaná v súbore. A ona rozdelí ten názov na časti a podľa toho aké písmeno dostane, táto funkcia vráti názov, číslo a farbu karty, ktoré potom aj s lokáciou, kde sa nachádza uloží do poľa a vráti.

Základné algoritmy

K základným algoritmom, ktoré som pri hre Solitaire musel využiť sú pravidlá, ktoré nevyžadujú prehľadávanie stavového priestoru a sú to nasledovné pravidlá:

- **Eso**
  - Pred začatím každej hry prebieha načítanie všetkých kariet v zásobníku. Tieto karty sa uložia do premennej `waste_list`. Ak sa v tejto premennej nájde karta z hodnotou jedna čiže eso. Hneď bude táto karta premiestená do vykladacej

hromádky a nebude sa s touto kartou ďalej v programe pracovať. To isté platí aj keď sa eso karta objaví v sekcii tableau. Taktiež bude táto karta premiestnená do vykladacej hromádky a nebude sa s ňou pracovať v tableau.

- **Karta o jedno menšia**
  - Prvým pravidlom, ktoré bolo treba uplatniť pri implementácii je, že keď chceme premiestňovať karty medzi sebou na tableau ploche alebo premiestniť kartu zo zásobníka na tableau, tak jedine karta, ktorá sa premiestňuje, musí byť presne o jedno číslo menšia ako karta, na ktorú sa bude premiestňovať.
- **Karta o jedno väčšia**
  - Pravidlo karta o jedno väčšia sa zasa bude využívať, keď chceme začať vykladať karty do vykladacej hromádky. Vo vykladacej hromádke musí taktiež byť karta presne o jedno nižšia ako karta, ktorá sa bude premiestňovať na túto kartu.
- **Rozdielna farba karty**
  - Rozdielna farba karty sa taktiež uplatňuje pri tom, keď chceme premiestňovať karty medzi sebou na tableau časti alebo premiestniť kartu zo zásobníka na tableaue plochu. Toto pravidlo sa musí uplatňovať s pravidlom karta o jedna menšia. Ale aj naopak, keď chceme využiť toto pravidlo, musíme využiť aj to druhé pravidlo.
- **Rovnaká farba karty**
  - Posledné základné pravidlo, ktoré algoritmus využíva, sa využíva ako pravidlo o karte o jedno väčšia. Do vykladacej hromádky môžu ísť karty len, ktoré sú rovnakej farby a zároveň sú o jedno väčšie ako karta, ktorá sa tam už nachádza.

Priebeh hry

Hra po spustení začína tým, že prejde postupne celý obsah zásobníka a detekované karty uloží do premennej `waste_list`. Následne, ak sa v tomto zásobníku bude nachádzať karta eso, tak túto kartu premiestni to vyhadzovacej hromádky. A program sa po tejto akcii dostane do `While cyklu`.

Vo while cykle program začína tým, že sa pozrie, aké karty sa nachádzajú v tableaue časti. Ak sa tam nachádza eso, tak hneď bude premiestnené do vykladacej hromádky. A hneď potom začne časť kódu, kde sa karty premiestňujú medzi sebou na tableaue ploche. Táto časť sa rozdeľuje na dve časti. Prvá časť zabezpečuje to, že karty, ktoré majú len jednu možnosť kam sa môžu presunúť, tak sa tam presunú. A potom druhá časť je to, čo zabezpečuje, že keď program už nevie ďalej, kde má pokračovať, tak využije túto možnosť. Viac o tomto je napísané v časti 7.2.

Následne program kontroluje, či má k dispozícii voľné miesto na tableaue, kde môže byť vložená karta kráľ. Ak takéto miesto nájde, pozrie sa, či má k dispozícii kartu kráľa, či sa karta nachádza v zásobníku alebo v tableaue, ale karta taká, ktorá sa ešte nenachádza úplne vpredu v tableaue riadku. Ak takúto kartu nájde, tak ju premiestni na voľné miesto.

A po skončení tejto funkcie príde na radu, že program kontroluje či má k dispozícii nejakú kartu, ktorú môže vyložiť na vykladaciu hromádku.

Ak program nemá k dispozícii žiadny možný ťah, tak využíva premennej `waste_list` a zo zásobníka vyloží kartu na tablaaue alebo do vykladacej hromádky. A tento cyklus pokračuje, dokým sa buď hra nedohrá alebo nebude potrebné sa vrátiť a pokračovať v inom preskúmavaní.

Prehľadávanie stavového priestoru

Pri hre Solitaire bolo treba využiť prehľadávanie stavového priestoru. Kvôli tomu, že pri väčšine hier sa stane, že sa hra zasakne a program, ale aj človek sa musí vrátiť a vyskúšať iný prístup. K tomu som využil prehľadávanie stavového priestoru pomocou `Depth First Search`.

A to tak, že sa v programe zaviedla globálna premenná, ktorá uchováva jednotlivé vzťahy:

tableu_global = [ ]

Táto premenná sa využíva, keď program príde do bodu, kedy sa má rozhodnúť z viacerých možnosti, kde môže premiestniť kartu. Takže do tejto premennej sa uloží `waste_list`, to sú karty, ktoré v tú chvíľu obsahuje zásobník. Ďalej sa tam uloži `tableu_list`, ktorý obsahuje zasa všetky karty, ktoré sa momentálne nachádzajú v tableaue. A uloží sa tam ešte aj to o akú kartu sa jedná, aj počet krokov, ktoré hra za celú dobu prešla dokým nedošla do tohto bodu. A vyberie jednu z možností a tam premiestni kartu.

Keď program príde o všetky možné kroky bez využitia vrátenia sa. Tak program zavolá funkciu `go_back_try_another`.

Táto funkcia využíva práve premennú `tablue_global` a to tak, že získa stav, ktorý bol ako posledne pridaný do tejto premennej. Z toho táto funckia získa informáciu o akú kartu sa jednalo. Aké karty sa nachádzali v zásobníku a aj v tableaue. A aj to, aký bol vtedy počet krokov. Tak vďaka tomuto sa hra pomocou `PyAutoGUI` a stlačenia crl + z, vráti o toľko, aký je rozdiel celkových krokov, mínus krokov koľko bolo uložených v `tableu_global`.

A karta si vyberie druhú možnosť, kde sa mohla v ten moment premiestniť a program zasa pokračuje znova do toho bodu, kde hra už nemá ďalší možný stav. Tak sa zasa zavolá táto funkcia.

Ukončenie hry

Ukončenie hry pri Solitaire môže nastať v troch prípadoch. Vtedy keď sa hra dohrá bez toho aby musela využiť prehľadávanie stavového priestoru a v premennej `tableu_p` a ani v premennej `waste_list` sa nenachádza už žiadna karta. Alebo sa dohrá s využitím stavového priestoru. Ale môže sa stať, že sa hra aj nedohrá a to vtedy, keď stále obsahuje karty v zásobníku alebo na tableaue, ale v premennej `tableu_global` sa už nenachádza žiadny ďalší stav určený na preskúmanie a nemá k dispozícii žiadne pravidlo, ktoré by mohol využiť.

Kapitola 8

Testovanie a záverečné štatistiky

V záverečnej kapitole si zhrnieme výsledky, s akými dopadli pragramy z oboch hier pri záverečnom testovaní. Ukážeme si, koľko sekúnd trvá dohrať hru Hľadanie Mín pri všetkých obtiažnostiach, aký bol najrýchlejšie nameraný čas, aký bol zasa najdlhší čas, kým bola hra úspešne dokončená a pri hre Solitaire si povieme, či je hru možné podľa výsledkov z testovania vždy dohrať a za aký čas môže táto hra byť dohraná. A či to algoritmu bude trvať dohrať hru len pár minút alebo až pár hodín.

8.1 Testovanie hry Hľadanie Mín

Hra Hľadanie Mín pri testovaní prešla 20 spusteniami pri každej z obtiažností, výsledky si môžeme pozrieť v tabuľke 8.1. Z tabuľky môžeme zistiť najrýchlejšie, najpomalšie a priemerné časy. Časy pri jednotlivých obtiažnostiach spustenia vyzerajú nasledovne:

- **Začiatočnícka obtiažnosť:**
  - Najrýchlejší čas: 4,46
  - Najpomalší čas: 12,29
  - Priemerný čas: 8,25
- **Pokročilá obtiažnosť:**
  - Najrýchlejší čas: 24,35
  - Najpomalší čas: 37,84
  - Priemerný čas: 30,03
- **Expert obtiažnosť:**
  - Najrýchlejší čas: 70,37
  - Najpomalší čas: 88,58
  - Priemerný čas: 77,53

Hľadanie Mín		
Začiatočník	Pokročilý	Expert
10,01	29,63	88,58
9,34	31,82	84,42
6,49	37,84	78,34
5,98	32,19	74,47
10,84	30,1	71,45
9,54	27,27	78,14
9,51	30,28	68,74
8,84	28,84	77,75
9,96	32,74	74,45
9,05	24,65	83,42
6,58	24,9	75,15
4,46	30,61	71,93
9,76	27,31	79,65
12,29	33,21	71,57
6,08	24,35	86,29
4,34	32,98	78,68
7,23	36,1	79,35
7,83	31,4	70,37
9,86	27,83	81,91
7,12	26,53	75,99
Priemerný čas		
8,25	30,03	77,53
Zrýchlenie		
50,40%	55,75%	55,67%

Tabuľka 8.1: Štatistika nameraných časov pri úspešných pokusoch v sekundách

Dôsledky zrýchlenia od minulej práce

Pri záverečnom testovaní sa mi podarilo zistiť, že sa hru Hľadanie Mín od minulej záverečnej práce na presne rovnakú tému podarilo zrýchliť v priemerných časoch pri začiatočníckej obtiažnosti z 16,63 sekúnd na 8,25 sekundy. Pri pokročilej obtiažnosti zo 67,85 sekundy na 30,03 sekúnd. A pri obtiažnosti expert sa priemerný čas skrátil o viacej ako 50% a to z času 174,87 sekúnd na 77,53 sekúnd. Tak som vyvodil dôsledky, ktoré viedli k zrýchleniu programu:

- **Využívanie rozdielnych algoritmov**
  - Jedným dôvodom zrýchlenia sú použité algoritmy. Okrem tých základných, ktoré využívame obaja. Ja som sa rozhodol využívať algoritmus založený na množinách. Pričom minulá práca využívala algoritmus, ktorý využíval princíp lineárnych algebraických rovníc, čo viedlo k dlhším výpočtom.
- **Detekcia hracích políčok**
  - Ďalším dôvodom, prečo sa časy zrýchlili, je detekcia farby pixelov skúmaných políčok. V minulej práci sa k detekcii farby pixelu využívala funkcia z knižnice

`PIL` s názvom `getpixel()`, ktorá patrí k tým pomalším možnostiam detekcie farby pixelu. Ja som v mojej práci využil k detekcii knižnicu `OpenCV`, ktorej detekcia farby naopak patrí k tým rýchlejším spôsobom detekcie farby pixelu.

8.2 Testovanie hry Solitaire

Tak isto ako testovania pri hre Hľadanie Mín, tak aj pri hre Solitaire som hru spustil 20 krát. Najväčšia nevýhoda tohto testovania bola, že keď som spustil hru, tak som sa už nemohol dotknúť myše a musel celý počítač nech hrá hru, ktorá mohla trvať aj pár hodín.

Z týchto 20 pokusov, ktoré som nameral, najrýchlejší čas, keď sa hra úspešne dohrala bolo 15 minút a 35 sekúnd. Vtedy hra nemusela využívať prehľadávanie stavového priestoru a nemusela sa vrátiť a tak nestrácala čas pri dokončení. Pri niektorých hrách sa čas pohyboval v rozmedzí od 20 minút do 30 minút. Ale sú tu aj kolá hier, kedy sa hra musela veľakrát vracať a dohratie takejto hry, teda najdlhšie nameranej hry bolo 2 hodiny a 49 mínút a 15 sekúnd. A podarilo sa mi pri týchto hrách nájsť takú hru, ktorá prešla všetky možné stavy v hre a nedokázala sa aj tak dohrať. Takýchto hier som pri 20 spusteniach zaznamenal šesť.

Čo ma viedlo k záverom, že pri implementácii pri prehľadávaní stavového priestoru som využil `Depth First Search`, ktorý sa ukázal ako nie úplne optimálny. Ale namiesto toho som mohol využiť prehľadávanie pomocou algoritmu A*. Ďalším dôvodom, prečo sa hry dohrávali tak dlho, môže byť nie najvhodnejšie zvolený návrh pri implementácii, ktoré karty sa majú kedy vyhadzovať a premiestňovať a z toho vzniklo to, že zásobník kariet sa pretočil viackrát niekedy ako by bolo treba.

Porovanie

Autor minulej bakalárksej práce uvádza, že dohranie niektorých hier jeho programu trvalo približne 20 minút. Toto sa podarilo dokázať aj mne, kde sa niektoré hry pohybovali pri dohraní v časovom rozmedzí 20 až 30 minút. Autor aj uvádza, že najdlhšie jeho program hral hru viacej ako 5 hodín. Mne sa podarilo namerať 2 hodiny a 49 minút. Ale možno keby program musel pracovať s väčším množstvom stavov, čas by sa zhoršil.

Spôsoby na zrýchlenie

- **prehľadávanie stavového priestoru**
  - Najvhodnejším spôsobom pre zrýchlenie je implementácia algoritmu A*. Tento algoritmus, keď je dobre navrhnutý a naimplementovaný, patrí medzi najrýchlejšie algoritmy pri takýchto problémoch. A patrí medzi algoritmy, ktoré sú úplné a aj optimálne. Čo by viedlo k tomu, že niektoré karty by neboli premiestňované zbytočne.

Kapitola 9

Záver

Cieľom tejto bakalárskej práce bolo v jazyku Python naimplementovať dva programy, ktoré budú schopné automaticky dohrať hry ako sú Hľadanie Mín a Solitaire.

Pre dosiahnutie cieľa práce bolo potrebné sa oboznámiť s pravidlami oboch hier, naštudovať si aké zákonitosti z pravidiel vyplývajú. Aké pravidlá bolo potrebné využiť pri tvorbe algoritmov pre hru Hľadanie Mín. Pri hre Solitaire preštudovať teóriu o prehľadávaní stavového priestoru. Ďalej bolo treba naštudovať si všetky potrebné knižnice pre detekciu hracích plôch pri oboch hrách alebo ako sa má myš automaticky pohybovať po obrazovke, ako má jednotlivé karty pri hre Solitaire chytiť a pretiahnuť na určené miesto.

Cieľ práce sa mi podarilo splniť. Naimplemnoval som dva plne funkčné programy, ktoré dokážu dohrať automaticky hry Winmine-XP a Solitaire.

Táto práca mi poskytla cenné a hodnotné poznatky v oblasti umelej inteligencie a počítačového videnia. Prekročil som hranice svojích doterajších vedomostí a objavil som široké spektrum nových techník, algoritmov a konceptov, ktoré nám tieto disciplíny ponúkajú. Vďaka tejto práci som získal bohaté skúsenosti, ktoré využijem vo svojej budúcej profesionálnej činnosti.

Výsledky práce

Pri implementácii algoritmu na dohranie hry Hľadanie Mín sa mi podarilo dosiahnuť výsledkov. Pri začiatočníckej obtiažnosti dohranie hry v priemere trvalo 8,25 sekúnd. Pri pokročilej obtiažnosti to je 30,03 sekúnd a pri obtiažnosti expert to je 77,53 sekundy. Podarilo sa mi od minulej práce zvýšiť efektivitu dohrania a celkového algoritmu až o 50%. Pri hre Solitaire sa mi podarilo dosiahnuť najrýchlejší čas dohrania 15 minút a 35 sekúnd.

Budúce vylepšenia

Nakoniec pár nápadov, na ktoré by sa v budúcnosti mohlo nadviazať. Pri hre Solitaire by sa mohla zaviesť heuristická funkcia, ktorá by ťahy ohodnotila efektívnejšie a to by viedlo k výrazne lepším časom pri dohraní hry. Ďalším vylepšením pri hre Solitaire môže byť to, že zatiaľ hra funguje len na jednu konkrétnu hru. Mohla by sa navrhnúť lepšia detekcia častí hry a detekcia kariet, ktorá by viedla k tomu, že by bol možný algoritmus použiť na viacero Solitaire hier, nie len na jednu konkrétnu. Pri hre Hľadanie Mín by sa mohol navrhnúť ďalší algoritmus, ktorý by dokázal oddialiť odhadovanie stlačenia políčka pomocou pravdepodobnosti, ktoré by viedlo k väčšej úspešnosti pri dohraní expert obtiažnosti.

Literatúra

[1] ANON.. *Solvitaire - The solitaire solver* [[online]]. [zobrazeno 2. Decembra 2023, 10:43]. Dostupné z: https://www.solvitaire.com/solvitaire/introduction.jsp.

[2] ASWEIGART. *PyGetWindow* [[online]]. 2021. [zobrazeno 29. Decembra 2023, 19:50]. Dostupné z: https://pygetwindow.readthedocs.io/en/latest/.

[3] CHOWDHARY, K. R. State Space Search. In: *Fundamentals of Artificial Intelligence*. India: Springer (India) Private Limited, 2020. ISBN 8132239709.

[4] COMMUNITY, O. *OpenCV* [[online]]. 2024. [zobrazeno 2. Januára 2023, 10:39]. Dostupné z: https://docs.opencv.org/4.x/d1/dfb/intro.html.

[5] GAMESCOMPUTERSPLAY. *Minesweeper-solver* [[online]]. 2023. [zobrazeno 10. Novembra 2023, 14:21]. Dostupné z: https://github.com/gamescomputersplay/minesweeper-solver.

[6] GIELIS, K., BRITO, F., TOURNOY, J. a VANDEN ABEELE, V. Can Card Games Be Used to Assess Mild Cognitive Impairment? A Study of Klondike Solitaire and Cognitive Functions. In:. New York, 2017. ISBN 1450351115.

[7] GOLAN, S. Minesweeper strategy for one mine. Elsevier Inc. 2014. ISSN 0096-3003.

[8] LIU, C., HUANG, S., NAYING, G., KHALID, M. N. A. a IIDA, H. A solver of single-agent stochastic puzzle: A case study with Minesweeper. Amsterdam: Elsevier B.V. 2022. ISSN 0950-7051.

[9] MAHATO, S., YADAV, D. K. a KHAN, D. A. A minesweeper game-based steganography scheme. Elsevier Ltd. 2017. ISSN 2214-2126.

[10] MASSAIOLI, R. *Solving Minesweeper with Matrices* [[online]]. 2013. [zobrazeno 15. Decembra 202š, 10:52]. Dostupné z: https://massaioli.wordpress.com/2013/01/12/solving-minesweeper-with-matricies/.

[11] MIKADOYELLOW. *Minesweeper Probability Calculator* [[online]]. 2022. [zobrazeno 4. Marec 2024, 13:20]. Dostupné z: https://s3.us-east-2.amazonaws.com/www.minesweepercalculated.com/Online+Minesweeper/index.html.

[12] MULROY, C. *Want to play Solitaire with real cards? Here's a guide (with pictures) to the solo game* [[online]]. 2021. [zobrazeno 9. Novembra 2023, 15:20]. Dostupné z: https://eu.usatoday.com/story/graphics/2022/12/18/how-to-play-solitaire-card-game-rules/10626190002/.

[13] O'Connor, T. *PyAutoGui and Retina Displays* [[online]]. 2016. [zobrazeno 29. Novembra 2023, 11:25]. Dostupné z: https://medium.com/@tracy_blog/pyautogui-and-retina-displays-2d5c37a5aa5e.

[14] Přikryl, S. *Program pro automatické hraní her Solitaire a Miny.* 2023. Bakalarská práca. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav informačních systémů.

[15] Russel, S. a Norvig, P. *Artificial Inteligence: A Modern Approach.* 3. vyd. New Jersey: Prentice Hall, 2010. ISBN 978-0-13-604259-7.

[16] Sweigart. *PyAutoGUI Documentation* [[online]]. 2021. [zobrazeno 20. Decembra 2023, 18:10]. Dostupné z: https://buildmedia.readthedocs.org/media/pdf/pyautogui/latest/pyautogui.pdf.

[17] Zbořil, F. Základy umělé inteligence : IZU. Brno: Fakulta informačních technologií. 2008.

Príloha A

Obsah priloženého pamäťového média

- **src** - Súbor obsahuje zdrojové kódy obidvoch programov
- **README.txt** - Súbor obsahuje pokyny pre rozbehnutie programov
- **sprava.pdf** - Technická správa v PDF
- **sprava_src** - Zložka obsahujúca zdrojové súbory technickej správy