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 **STANISLAV PŘIKRYL**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

147740

Ústav: Ústav inteligentních systémů (UITS)
Student: **Přikryl Stanislav**
Program: Informační technologie
Specializace: Informační technologie

Název: **Program pro automatické hraní her Solitaire a Miny**
Kategorie: Umělá inteligence
Akademický rok: 2022/23

Zadání:

1. Nastudujte pravidla her Solitaire a Miny. U obou her se zamyslete nad zákonitostmi, které z pravidel vyplývají.
2. Navrhněte program v jazyce Python, který umožní obě hry hrát automaticky. Zamyslete se nad možnostmi spuštění programu, detekce okna programu a rozpoznání políček a čísel v Minách a karet a jejich hodnot v Solitaire. Vámi navržený program musí v Minách umět vybrat ta políčka, kde víme, že mina není, musí umět odsimulovat možné pozice min a z toho vybrat správnou možnost a v případě, že se dostane do stavu, kdy není možné určit, kam bezpečně kliknout, musí umět vypočítat pravděpodobnosti jednotlivých políček a vybrat to nejvhodnější. U Solitaire musí být schopen prohledávat stavový prostor hry tak, aby byl schopen hru úspěšně dokončit.
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.

Literatura:

- 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: 1.11.2022
Termín pro odevzdání: 17.5.2023
Datum schválení: 3.11.2022

Abstrakt

Cílem této práce je navrhnout, následně naimplementovat, a nakonec otestovat program, který bude automaticky hrát hry Solitaire Klondike a Hledání min. Tohoto cíle bylo dosaženo s využitím programovacího jazyka Python. Podařilo se vytvořit program, který je schopen odehrát hru Hledání min s 91% úspěšností dokončení hry bez šlápnutí na minu a program, který je schopný hrát hru Solitaire Klondike, rozhodovat se o tom, který tah použít jako další a má-li hra řešení, hru úspěšně dohrát. Výsledky této práce umožňují čtenáři lépe pochopit problematiku obou her a zákonitosti, které je nutné řešit během implementace programu, který má sloužit k automatickému hraní her. Tato práce taktéž čtenáři nastíní různé postupy, které lze při implementaci podobného programu použít.

Abstract

The goal of this work is to design, then implement, and finally test a program that will automatically play the games Solitaire Klondike and Minesweeper. This goal was achieved using the Python programming language. We were able to create a program that is able to play the game of Minesweeper with a 91% success rate of completing the game without stepping on a mine, and a program that is able to play the game of Solitaire Klondike, decide which move to use next, and if the game has a solution, successfully complete the game. The results of this work allow the reader to better understand the issues of both games and the problems that must be addressed during the implementation of a program that is intended to be used to play the games automatically. This thesis also outlines for the reader the various techniques that can be used when implementing a similar program.

Klíčová slova

Python, Hledání min, Solitaire Klondike, Umělá inteligence, Prohledávání stavového prostoru, DFS

Keywords

Python, Minesweeper, Solitaire Klondike, Artificial intelligence, State space search, DFS

Citace

PŘIKRYL, Stanislav. *Program pro automatické hraní her Solitaire a Miny*. Brno, 2023. 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

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Stanislav Přikryl
17. května 2023

Poděkování

Děkuji panu Ing. Jaroslavu Rozmanovi, Ph.D. za odborné vedení této bakalářské práce, vstřícnost při konzultování a za položené otázky, které dokázaly navézt ke zdárnému řešení. Dále bych chtěl poděkovat mému štěněcímu parťákovi Cornymu, který mě dokázal rozveselit v těch nejtěžších chvílích i přes to, že jsem se mu nemohl naplno věnovat.

Obsah

Kapitola 1

Úvod

V dnešním světě informačních technologií se počítačové hry staly nedílnou součástí každodenního života mnoha lidí. Solitaire a Hledání min patří k těm nejznámějším, které si získaly srdce mnoha generací. Tyto hry představují nejen zdroj zábavy, ale také účinný prostředek pro rozvoj logického myšlení a strategického plánování.

Tato práce se zaměřuje na vývoj programů pro automatické hraní her Solitaire Klondike a Hledání min, které jsou kompatibilní s již existujícími hrami GNOME mines[1] a AisleRiot[2] na platformě Linux. Mým hlavním cílem je navrhnout a implementovat algoritmy v jazyce Python, které umožní automatické hraní těchto klasických her a zhodnocení jejich účinnosti. Tento projekt má pro mě osobní význam, neboť mi tyto hry připomínají dobu, kdy jsem neměl přístup k internetu a málokdy se mi je podařilo úspěšně dohrát, což mě motivovalo prozkoumat je na hlubší úrovni. Navíc tato práce může sloužit jako dobrý zdroj informací pro ty, kteří se zajímají o algoritmy vhodné pro hraní těchto her. Budoucí výzkumníci či studenti se mohou inspirovat tímto textem při výběru algoritmů pro podobné projekty a zjistit, jak jsou efektivní pro dané hry.

V průběhu práce budu řešit detekci okna hry, detekci políček a čísel ve hře Hledání min, výpočet pravděpodobnosti výskytu min, rozpoznání karet a jejich hodnot, prohledávání stavového prostoru hry Solitaire Klondike a návrh způsobů pro zrychlení hraní obou her.

Práce je rozdělena do několika kapitol. Ve 2. kapitole se čtenář seznámí s problematikou her Solitaire Klondike a Hledání min a vysvětlením jejich pravidel. Kapitola 3 se věnuje návrhu a implementace algoritmů pro automatické hraní těchto her. Kapitola 4 popisuje samotnou implementaci algoritmů a použité technologie. V 5. kapitole se zaměřuje na testování a hodnocení výsledků. Kapitola 6 shrnuje dosažené výsledky a diskutuje možnosti dalšího výzkumu v této oblasti.

[1] GNOME je svobodné a otevřené desktopové prostředí pro operační systémy Linux a Unix. Nabízí uživatelům širokou škálu aplikací, včetně her. `https://www.gnome.org/`

[2] AisleRiot je kolekce karetních her Solitaire, která je součástí balíčku GNOME Games. AisleRiot nabízí několik desítek variant Solitaire, které lze hrát na platformě Linux. `https://wiki.gnome.org/Apps/Aisleriot`

Kapitola 2

Teoretický základ

V této kapitole se seznámíme s teoretickými základy, které jsou nezbytné pro porozumění problematiky her Solitaire Klondike a Hledání min. Nejprve si představíme stručnou historii a význam obou her, abychom pochopili jejich kontext a důležitost. Poté se podrobně seznámíme s pravidly her, což je nezbytné pro návrh a implementaci algoritmů pro automatické hraní. Nakonec stručně prozkoumáme některá, již existující, řešení pro automatické hraní těchto her.

2.1 Hra Solitaire Klondike

Solitaire, jinými slovy Pasiáns, je rodina karetních her, které hraje zpravidla pouze jeden hráč. Jejich vznik sahá až na konec 18. století, kdy byly tyto hry používány jako forma věštění, když se hráči podařilo hru dohrát, měl pak mít štěstí, v opačném případě smůlu [7]. Jednou z nejpopulárnějších verzí těchto pasiánsových her se stala varianta, která je přezdívána Klondike[1]. Tato verze byla masivně zpopularizována s příchodem systému Microsoft Windows verze 3.0 roku 1990. Microsoft tou dobou hru využíval k učení uživatelů základní manipulace s myší, nicméně se stala nejpoužívanější aplikací tohoto systému na dlouhých 15 let [3]. Existují desítky různých variant této karetní hry, mezi nejznámější z nich patří např. Spider, FreeCell, Pyramid, BlackHole, atd.

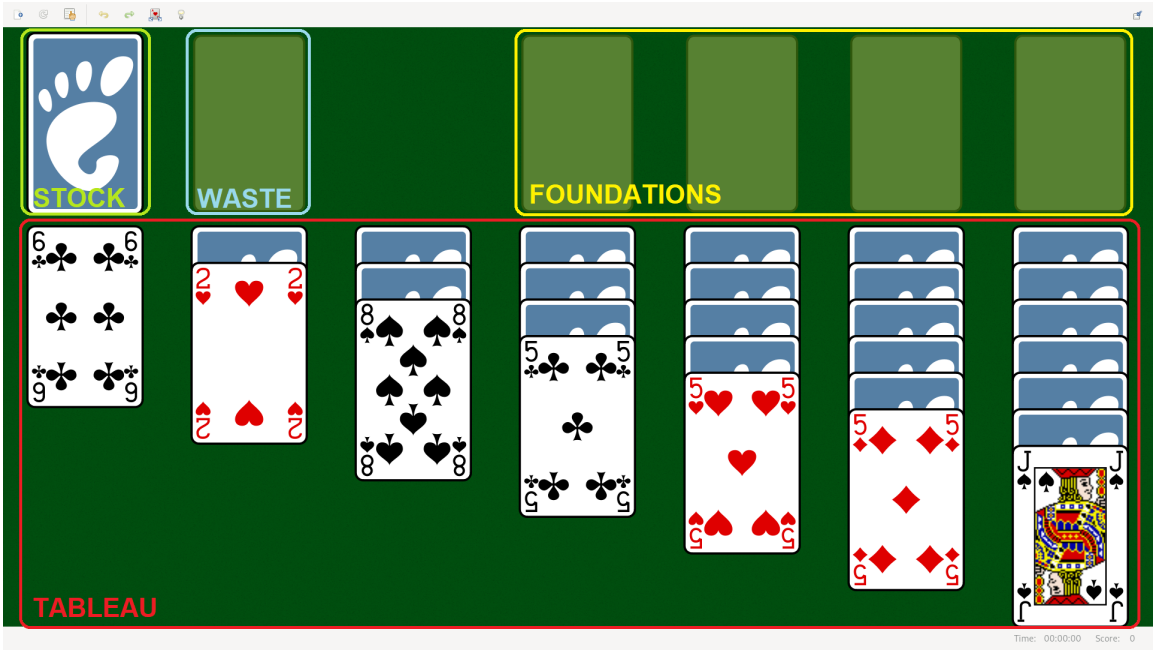
2.1.1 Pravidla hry Solitaire Klondike

Solitaire Klondike se hraje s běžným balíčkem 52 karet, které jsou označeny čtveřicí symbolů a jsou rozděleny do dvou barev (červená a černá). Pro jednotlivé symboly se používají různá pojmenování, která vycházejí z různých karetních systémů, v této práci se budeme držet následujícího pojmenování: červená srdce, černé kříže, červené káry a černé piky. Toto pojmenování odpovídá symbolům z obrázku 2.3. Každý ze symbolů je zobrazen na sérii 13 karet, které začínají esem, pokračují čísly 2 až 10 a končí kartami J (Janek), Q(Královna) a K (Král).

Karty se během hry nacházejí na herním poli, které tvoří celkem 4 části, a to jsou:

- **Sloupce karet** (anglicky tableau)
  - Sedm sloupců karet, které tvoří hlavní část herní plochy. Karty jsou rozloženy tak, že první sloupec obsahuje jednu kartu, druhý sloupec obsahuje dvě karty

[1] Tato verze hry je v některých státech známa pod jménem Canfield.



Obrázek 2.1: Herní plocha hry Solitaire po spuštění se zvýrazněním jednotlivých částí.

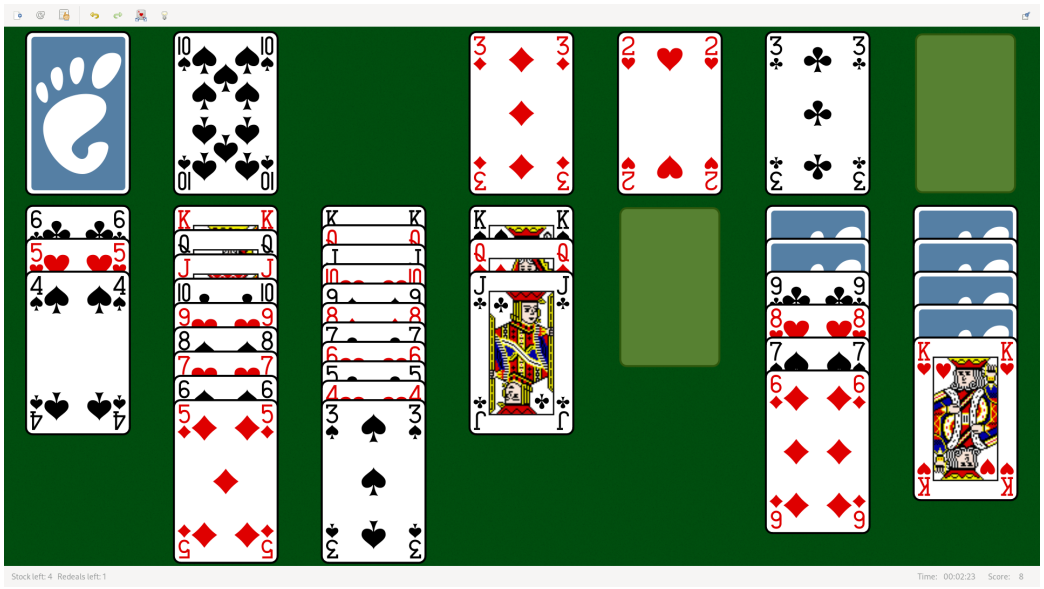
a tak dále, až sedmý sloupec obsahuje sedm karet. Horní karta každého sloupce je otočená barvou nahoru, zatímco zbývající karty jsou otočeny barvou dolů.

- **Zásobník zbylých karet** (anglicky stock)
  - Zásobník zbylých karet, které nejsou součástí tableau. Hráč může během hry procházet zásobník a používat karty z něj, pokud nenajde vhodný tah na tableau.
- **Odkládací hromádka** (anglicky waste)
  - Odkládací hromádka, kam jsou odkládány karty z vrcholu zásobníku stock. Karty z waste mohou být také zařazeny do hry, pokud se pro ně objeví vhodný tah.
- **Vykládací hromádky** (anglicky foundations)
  - Čtyři vykládací hromádky, kam hráč uspořádá karty podle barev a hodnot, od nejnižšího (eso) po nejvyšší (král).

Přesné umístění jednotlivých části herního pole je vyobrazeno na obrázku 2.1.

Hra začíná s rozložením karet na herní ploše do sedmi sloupců na tableau a zbylými 24 kartami v zásobníku stock. Během hry je hráči umožněno přemisťování karet mezi sloupci, které je omezeno pravidly, jaké karty lze položit na sebe. Kartu lze přemístit na jinou kartu, pouze tehdy, má-li přemisťovaná karta hodnotu právě o 1 nižší, než karta na kterou se přesouvá. Karty se také musejí lišit svou barvou (červenou nebo černou). Například červenou devítku lze přesunout pouze na černou desítku. Karta krále může být přemístěna pouze na prázdný sloupec. Hráč může také přemisťovat skupiny karet, které splňují tato pravidla, jako celek.

V případě, že hráč narazí na situaci, kdy nelze žádnou kartu přemístit v rámci tableau, může použít zásobník stock, který obsahuje zbylé karty. Hráč může sejmout kartu z vrcholu



Obrázek 2.2: Herní plocha hry Soliterie během hry.

Obrázek 2.3: Symboly hracích karet (zleva): srdce, kříže, káry, piky.

zásobníku stock a použít ji na tableau. Pravidla pro prohlížení karet ze zásobníku se mohou pro různé verze hry lišit, ale běžně může hráč procházet zásobník po jedné, dvou nebo třech kartách zároveň. Tento proces prohlížení celého zásobníku zbylých karet (anglicky redeals) se může několikrát opakovat, v některých verzích hry není tento proces nikterak omezen, v jiných verzích hry může hráč prohlížet zásobník nanejvýše jedenkrát, až třikrát. V této práci budeme pracovat s tou verzí, kde může hráč procházet zásobník pouze po jedné kartě a je mu umožněno prohlédnout celý zásobník třikrát.

Hra končí právě tehdy, kdy hráč úspěšně uspořádá všechny karty na čtyři vykládací hromádky foundations podle barev a hodnot vzestupně od esa až po krále [5].

Cílem hry je potom zaplnit každou ze 4 vykládacích pozic stejnou barvou uspořádaně vzestupně od esa až po krále [10].

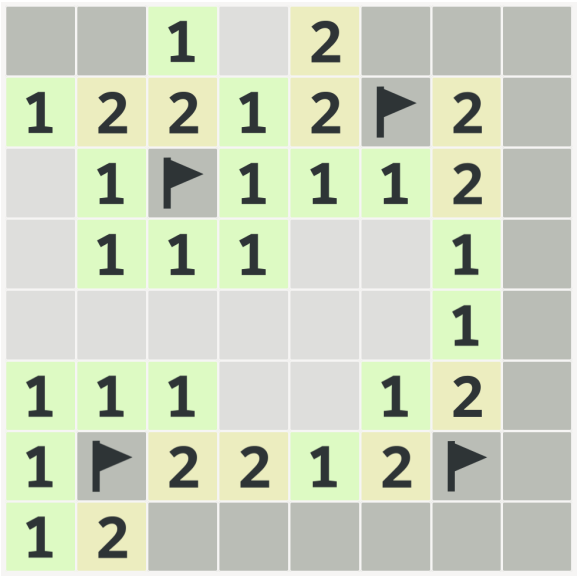
2.2 Hra Hledání Min

Hledání min (anglicky minesweeper) je klasická logická hra, která byla masivně zpopularizována operačním systémem Microsoft Windows, jehož součástí se stala již roku 1991 [6].

Tato hra je založena na jednoduchém konceptu označování min a odhalování bezpečných políček. Herní pole se sestává zpravidla ze čtvercovité mřížky políček[2], pod kterými se nacházejí miny, nebo bezpečná místa.

Herní pole může mít různé velikosti a obtížnosti, což umožňuje hráčům přizpůsobit si hru podle svých dovedností a zkušeností. Nejčastěji se setkáváme se třemi základními obtížnostmi, a to jsou: Začátečník, Pokročilý a Expert. Každá obtížnost se liší počtem min, které jsou na ploše rovnoměrně náhodně rozmístěny, a rozměry herního pole, což přímo ovlivňuje komplexnost hry a hlavně dobu potřebnou k jejímu dokončení.

Problematika hledání min je zajímavá nejen z hlediska zábavy, ale také z hlediska teoretické informatiky. Tato hra je považována za NP úplný problém[3] [4].



Obrázek 2.4: Herní pole hry Hledání Min během hry na úrovni Začátečník.

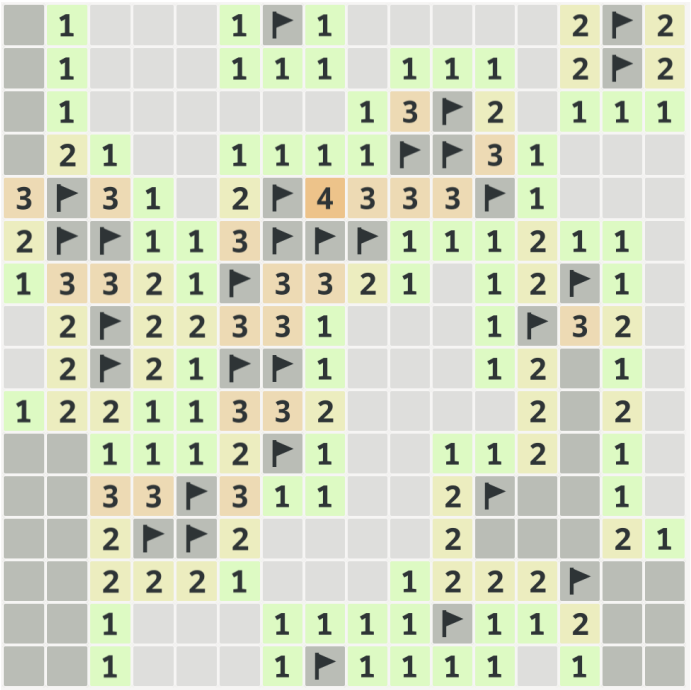
2.2.1 Pravidla hry Hledání min

Hledání min je založeno na několika jednoduchých pravidlech a postupech, které hráč musí dodržovat, aby úspěšně dokončil hru. Níže jsou popsaná pravidla a mechaniky hry.

- **Herní pole**: Hra se odehrává na čtvercovém, či obdélníkovém herním poli. Velikost a tvar pole se liší na základě zvolené obtížnosti. Obtížnost Začátečník má herní pole o rozměrech 8x8 s 10 minami, Pokročilý má 16x16 se 40 minami a Expert má pole o velikosti 30x16 s 99 minami.

[2]Existují různé verze hry, kde je herní plocha zasazena do různých obrazců, které hráči usnadňují, nebo naopak ztěžují průchod hrou.

[3]NP úplnost znamená, že problém patří do třídy problémů NP (nedeterministicky polynomiální) a že je možné všechny ostatní problémy z této třídy na něj redukovat v polynomiálním čase.



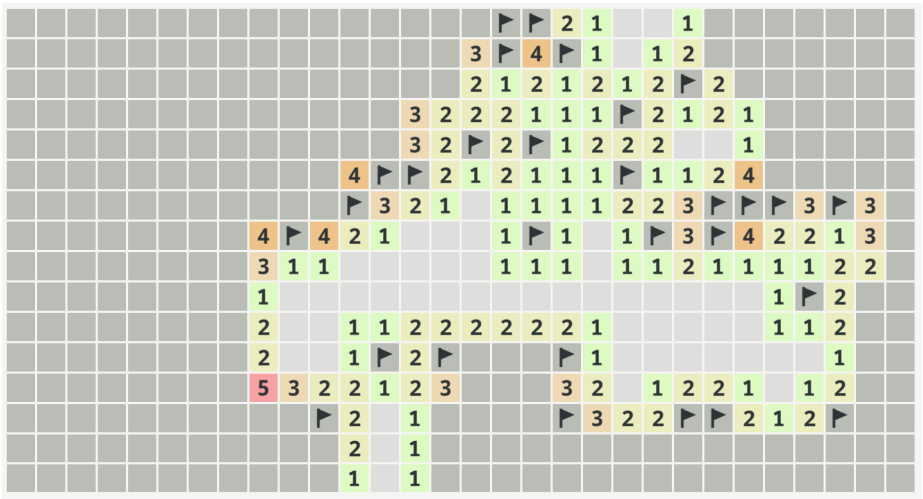
Obrázek 2.5: Herní pole hry Hledání Min během hry na úrovni Pokročilý.

- **Rozmístění min**: Na začátku hry jsou miny rovnoměrně náhodně rozmístěny na herním poli. Hráč neví, kde jsou miny umístěny a jeho úkolem je je odhalit a označit je symbolem vlajky.
- **Odkrývání políček**: Hráč odkrývá políčka na herním poli stisknutím levého tlačítka myši. Pokud se pod odkrytým políčkem nenachází mina, zobrazí se na něm číslo, které udává počet sousedních políček obsahujících miny. Pokud odkryté políčko obsahuje minu, odhalí se všechny miny na herním poli a hra končí neúspěchem.
- **Odhalování prázdných oblastí**: Jestliže hráč odkryje políčko, pod jehož sousedními políčky se nenachází ani jedna mina (tedy číslo 0), hra automaticky odkryje všechna sousední prázdná políčka a políčka s čísly, dokud nenarazí na políčka sousedící s minami. Tato mechanika značně zrychluje hru a umožňuje tak hráči rychleji získávat informace o rozložení min na herním poli.
- **Označování min**: Hráč může označit políčko, o kterém si myslí, že obsahuje minu, stisknutím pravého tlačítka myši. Takto označené políčko je pak zakryto symbolem vlaječky. Toto označení může hráč kdykoliv během hry zvrátit opětovným stlačením pravého tlačítka myši na označeném políčku[4].
- **Konec hry**: Hráč vyhrává hru, pokud odkryje všechna políčka neobsahující miny a správně označí všechna políčka obsahující miny. Hráč hru prohrává v ten moment, když se mu podaří odkrýt políčko obsahující minu, což vede k explozi miny a ukončení

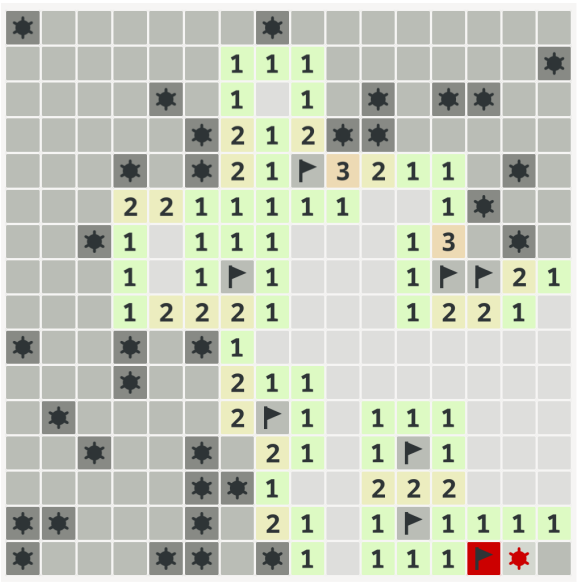
[4]Některé verze hry podporují, po opětovném stisknutí pravého tlačítka myši na políčku označeném vlaječkou, označit políčko symbolem vlaječky s otazníkem. Tato mechanika pomáhá hráčům vizuálně odlišit políčka, kde jsou si jistí, že se pod políčkem nachází mina, a políčka, kde si zatím jisti nejsou.

hry. Cílem hry je tedy odhalit a označit všechny miny na herním poli co nejrychleji a bez chyb.

- **Ostatní**: Součástí hry mohou být také další prvky, jako jsou například časový limit, možnost nápovědy, nebo další herní prvky jiných variant hry.



Obrázek 2.6: Herní pole hry Hledání Min během hry na úrovni Expert.



Obrázek 2.7: Herní pole hry Hledání Min během hry na úrovni Pokročilý, ukazující stav hry po odhalení políčka s minou.

Pravidla hry jsou jednoduchá a velmi snadno pochopitelná, což činí hru přístupnou hráčům různých věkových skupin a zkušeností. Zároveň hra poskytuje dostatečnou výzvu pro ty, kteří se chtějí ve hře zdokonalovat a dosáhnout tak co nejlepších výsledků.

2.3 Existující řešení

V průběhu let se vyvinulo mnoho řešení a algoritmů, které by byly schopny automatického hraní her Solitaire Klondike a Hledání min. Tyto algoritmy zahrnují různé přístupy, od jednoduchých heuristik, až po složitější metody, jako jsou vyhledávací algoritmy nebo strojové učení.

2.3.1 Metody pro prohledávání stavového prostoru

Prohledávání stavového prostoru je jednou z hlavních metod používaných pro řešení úloh v umělé inteligenci. Tyto metody využívají toho, že lze různá řešení úlohy namapovat na abstraktní datovou strukturu orientovaného grafu, nejčastěji se můžeme setkat s datovou strukturou strom. Jednotlivé uzly grafu představují možné stavy, ve kterých se může úloha nacházet, přechody mezi těmito stavy jsou potom reprezentovány hranami grafu. Tyto metody se dále dělí na metody neinformované (slepé) a na metody informované. V kontextu her Solitaire Klondike a Hledání min můžeme zahrnou algoritmy, jako jsou:

- **Slepé prohledávání do hloubky** (DFS[5]): DFS je algoritmus založený na prohledávání grafu, který systematicky prozkoumává stavový prostor hry tak, že se nejprve prohledává strom do hloubky. Tento algoritmus může být vhodný pro řešení her s malým stavovým prostorem, ale může se hodit i pro problémy s většími a komplexnějšími prostory [9].
- **Slepé prohledávání do šířky** (BFS[6]): BFS podobným způsobem, jako algoritmus DFS, prohledává stavový prostor, nicméně oproti DFS se zaměřuje na prohledávání grafu do šířky.
- **A* algoritmus**: A* patří mezi nejznámější a nejpoužívanější informované metody prohledávání stavového prostoru. Tento algoritmus používá tzv. heuristickou funkci, což je nějaký princip ohodnocení přechodu mezi stavy, určující který z přechodů by měl vést blíže k řešení úlohy. Tento algoritmus může být efektivní pro hledání optimálního řešení[7] ve hře Solitaire Klondike.

2.3.2 Strojové učení

Strojové učení (anglicky Machine Learning) je podoblast umělé inteligence zabývající se vývojem algoritmů a matematických modelů, které umožňují počítačovým systémům učit se a zlepšovat své výkony na základě získaných dat, aniž by byly explicitně naprogramovány. Cílem strojového učení je vytvářet modely, které dokáží generalizovat a předpovídat výsledky na základě nových, dříve neznámých, dat. Strojové učení lze dále rozdělit do několika kategorií podle způsobu učení:

- **Učení s učitelem** (anglicky Supervised Learning): U této metody jsou modely trénovány na základě trénovacích dat, která obsahují jak vstupní proměnné, tak odpovídající cílové výstupy. Úkolem modelu je pak naučit se závislost mezi vstupními a výstupními proměnnými tak, aby mohl předpovídat výstupy pro doposud neznámá data.

[5]DFS - Depth-First Search.
[6]BFS - Breadth-First Search.
[7]Optimálním řešením je myšleno nejlepší možné řešení.

- **Učení bez učitele** (anglicky Unsupervised Learning): V tomto přístupu nejsou k dispozici žádné cílové hodnoty a model se učí najít strukturu nebo vzory v datech bez cky Reinforcement Learning): Tato metoda se zaměřuje na učení agenta[8], jak se chovat v prostředí tak, aby maximalizoval své odměny či zisky na základě interakcí s prostředím a získávání zpětné vazby ve formě odměn [9].

2.3.3 Další přístupy

Existuje řada dalších přístupů, které mohou být použity pro automatické hraní her. Jako další přístup můžeme zmínit například expertní systémy, které jsou založeny na simulaci činnosti lidských expertů [9].

[8] Agentem je považován jakýkoliv systém, který je schopen pozorovat okolní prostředí, a interagovat s ním.

Kapitola 3

Analýza a návrh

Tato kapitola se zaměřuje na analýzu a návrh řešení pro automatické hraní her Solitaire Klondike a Hledání min. Cílem této práce je navrhnout program v jazyce Python, který bude schopen spustit a hrát obě hry automaticky, s důrazem na detekci herních oken, rozpoznání herních prvků a interakci s hrami.

Nejprve je třeba prozkoumat různé aspekty těchto her, jako jsou jejich pravidla, herní mechanismy a výzvy, které představují pro automatizaci. Dále je třeba analyzovat existující řešení a přístupy, které byly popsány v předchozí kapitole 2.3, a zvážit jejich výhody a nevýhody v kontextu této práce.

Následně se zaměříme na návrh vlastního řešení jak pro automatické hraní hry Solitaire Klondike, tak pro hru Hledání min. Tento návrh bude zahrnovat popis architektury systému, volbu vhodných algoritmů a postupů, které budou použity pro řešení různých problémů a úkolů, které tyto hry představují. Důležitou součástí návrhu bude také zohlednění požadavků a omezení, která mohou ovlivnit výběr a implementaci řešení.

Cílem této kapitoly je připravit pevný základ pro následující implementaci a testování navrženého programu. V této kapitole bude také kladen důraz na integraci navrženého programu pro automatické hraní Solitaire Klondike a Hledání min s existujícími hrami GNOME Mines[1] a AisleRiot, které jsou dostupné na Linuxovém prostředí.

3.1 Požadavky na program

Tato část práce se zaměřuje na požadavky, které jsou stejné, jak pro hru Solitaire Klondike, tak pro hru Hledání min. Jedná se o způsoby, jak lze hry pustit z programu psaného v jazyce Python, jak následně při úspěšném spuštění hry detekovat okno hry a jak zajistit interakci s prostředím her.

3.1.1 Spuštění hry

V Linuxovém prostředí je možné spouštět hry hned několika způsoby. Možnosti pro spuštění her z programu psaného v jazyce Python mohou být následující:

1. Použití knihovny `os`[2]: Knihovna `os` poskytuje základní funkce pro práci s operačním systémem, včetně spouštění externích programů. Hry Solitaire Klondike a Hle-

[1] Informace o hře GNOME-Mines jsou dostupné na: `https://help.gnome.org/users/gnome-mines/stable/`. Tato hra je licencovaná pod licencí `CC BY-SA 3.0`.

[2] Více informací o knihovně `os` najdete na: `https://docs.python.org/3/library/os.html`

dání min mohou být spuštěny pomocí funkce `os.system()` nebo `os.popen()`, které umožní spustit příkaz v shellovém[3] prostředí. Níže jsou uvedené dva řádky kódu ukazují, jak je snadné pomocí této knihovny spustit hru Hledání min, oběma zmíněnými způsoby.

```
os.system("gnome-mines")
os.popen("gnome-mines")
```

2. Použití knihovny `subprocess`[4]: Knihovna `subprocess` je modernější alternativou ke knihovně `os` pro spouštění externích programů. Tato knihovna poskytuje větší kontrolu nad spouštěním procesů a umožňuje lépe zachytit výstup. Hry Solitaire Klondike a Hledání min mohou být spuštěny pomocí funkce `subprocess.Popen()`, která spustí příkaz v novém procesu. Řádek kódu níže předvádí, jak toto spuštění externího programu provést, jedná se o obdobnou formu, jako při použití knihovny `os`.

```
subprocess.Popen(["gnome-mines"])
```

3.1.2 Detekce okna hry

Hned po úspěšném spuštění hry, bude nutno provést detekci okna, aby bylo možné se hrou interagovat. V Linuxovém prostředí je hned několik možností, jak detekovat okno spuštěné hry:

1. Použití knihovny `python-xlib`[5]: Knihovna `python-xlib` umožňuje interakci s X Window System[6], což je základní systém pro grafické rozhraní v Linuxových distribucích. Tato knihovna umožňuje získat seznam otevřených oken a zjistit o nich informace, jako jsou názvy oken, jejich rozměry a souřadnice. Tyto informace pak lze použít k identifikaci oken obou her.

2. Použití knihovny `wnck`[7]: Knihovna `wnck` (Window Navigation Construction Kit) může být další z možností pro detekci oken aplikací v Linuxovém prostředí. Tato knihovna je navržena pro práci s otevřenými okny na ploše, umožňuje získat informace jako jsou název, rozměry, poloha a stav aktivního okna. Získání informací o aktivním okně je výhodné v případě, že jsme v předešlém kroku hru spustili.

```
screen = Wnck.Screen.get_default()
screen = force_update()
game_window = screen.get_active_window()
geometry_of_window = game_window.get_geometry()
```

[3]Shell je textové uživatelské rozhraní určené pro komunikaci s operačním systémem, které umožňuje provádět příkazy, manipulovat se soubory a adresáři, spouštět aplikace a další funkce. V Linuxových distribucích jsou běžně používány různé shelly, jako například Bash, Zsh nebo Fish.

[4]Více informací o knihovně `subprocess` na: `https://docs.python.org/3/library/subprocess.html`

[5]Více informací o knihovně `xlib` najdete na: `https://python-xlib.github.io/`

[6]X Window System, je systém pro správu grafického uživatelského rozhraní, tzv. GUI, pro Unixové systémy a jejich varianty. Více o X Window System najdete na: `https://www.cs.odu.edu/~zeil/cs252/latest/Public/xwinlaunch/index.html`

[7]Více informací o knihovně `wnck` najdete na: `https://lazka.github.io/pgi-docs/Wnck-3.0/`

Výše uvedený kousek kódu ukazuje, jak díky čtyřem řádkům kódu mohu získat potřebné informace o okně spuštěné hry, proměnná `geometry_of_window`, po provedení tohoto kódu, obsahuje x a y souřadnice levého horního rohu okna, jeho šířku a jeho výšku. Tyto informace jsou klíčové pro následující práci se hrou.

3. Použití nástrojů příkazové řádky: Nástroje jako `wmctrl`[8] mohou být taktéž použity pro detekci oken aplikací v Linuxovém prostředí. Tyto nástroje lze spustit pomocí výše zmíněné knihovny `subprocess` a zpracovávat jejich výstup. Níže je uveden kousek kódu, který provede získání ID okna spuštěné hry, s pomocí knihovny `subprocess` a nástroje `wmctrl`.

```
output = subprocess.check_output(["wmctrl", "-l"]).decode("utf-8")
for line in output.split("\n"):
    if game_name in line:
        window_id, _, _, window_title = line.split(maxplit = 3)
```

Použití nástroje jako je `wmctrl`, může vyžadovat hlubší znalosti systému X Window System.

3.1.3 Skenování a rozpoznávání prvků herní plochy

Po úspěšném spuštění hry a detekci okna hry je nezbytné zaměřit se na způsoby skenování a rozpoznávání prvků herní plochy. Tento proces je klíčový pro následující interakci s hrami. Skenování herní plochy spuštěné hry může být provedeno dvěma základními způsoby:

1. **Přímé čtení dat z paměti aplikace**: Tento způsob je technicky náročný, ale může poskytnout přesnější a rychlejší způsob získávání informací o herní ploše. V Linuxovém prostředí je možné využít různé nástroje a knihovny pro přístup k paměti procesu, jako je `ptrace`[9] nebo `gdb`[10]. Tato metoda může vyžadovat hlubokou znalost paměťové struktury zkoumaného programu.

2. Použití knihovny `os`[11]: Knihovna `os` poskytuje základní funkce pro práci s operačním systémem, včetně spouštění externích programů. Hry Solitaire Klondike a Hledání min mohou být spuštěny pomocí funkce `os.system()` nebo `os.popen()`, které umožní spustit příkaz v shellovém[12] prostředí. Níže jsou uvedené dva řádky kódu ukazují, jakje snadné pomocí této knihovny spustit hru Hledání min, oběma zmíněnými způsoby.

3. **Zpracování snímku obrazovky**: Tento přístup je velmi univerzální a snazší naimplementaci. Pro tuto metodu lze využít knihovny, které umožňují získat snímek obrazovky a jeho následné zpracování, jako jsou knihovny `pyautogui`[13], `PIL`[14] (Python Image Library) nebo populární `OpenCV`[15].

[8] Více informací o nástroji `wmctrl` najdete na: `https://linux.die.net/man/1/wmctrl`
[9] Více informací o nástroji `ptrace` najdete na: `https://man7.org/linux/man-pages/man2/ptrace.2.html`
[10] Více informací o nástroji `gdb` najdete na: `https://man7.org/linux/man-pages/man1/gdb.1.html`
[11] Více informací o knihovně `os` najdete na: `https://docs.python.org/3/library/os.html`
[12] Shell je textové uživatelské rozhraní určené pro komunikaci s operačním systémem, které umožňuje provádět příkazy, manipulovat se soubory a adresáři, spouštět aplikace a další funkce. V Linuxových distribucích jsou běžně používány různé shelly, jako například Bash, Zsh nebo Fish.
[13] Více informací o knihovně `pyautogui` na: `https://pyautogui.readthedocs.io/en/latest/index.html`
[14] Více informací o knihovně `PIL` najdete na: `https://pillow.readthedocs.io/en/stable/`
[15] Více informací o `OpenCV` najdete na: `https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html`

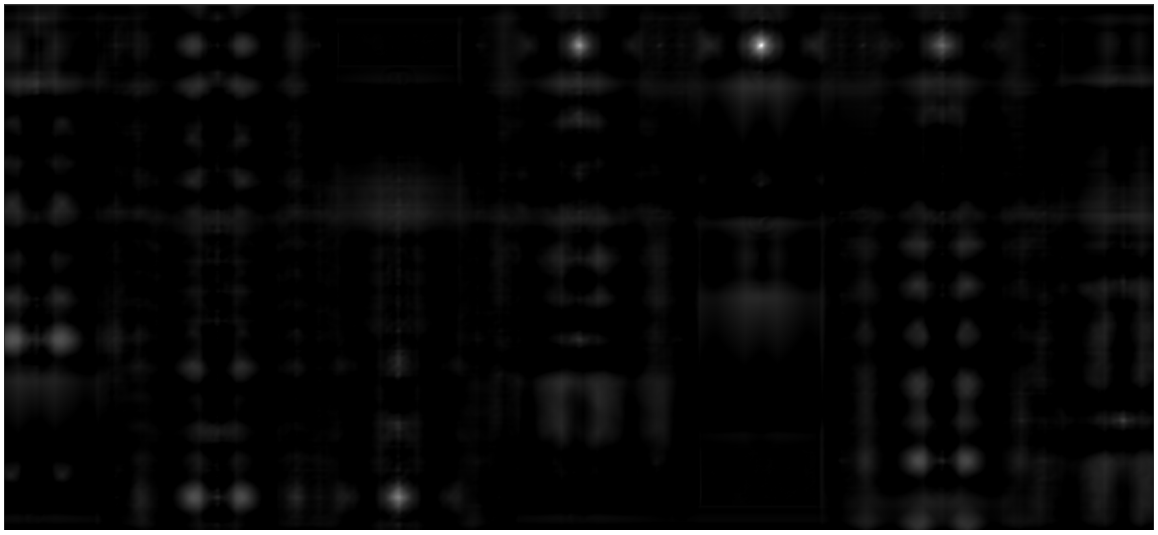
Pro zpracování snímku obrazovky a následné rozpoznávání herních prvků je vhodné využít techniky počítačového vidění a zpracování obrazu. Knihovna `OpenCV` je jednou z nejvhodnějších knihoven pro tento účel. Pomocí knihovny `OpenCV` lze provádět různé operace, jako je detekce hran, rozpoznávání tvarů, extrakce barev, nebo template matching[16], což může pomoci rozpoznat jednotlivé prvky hry, jako jsou karty v Solitaire Klondike, čísla v Hledání min, nebo hrany minového pole.

Použití techniky template matching, bude zřejmě nejjednodušším způsobem, jak naimplementovat skenování aktuálního stavu herní plochy. Níže uvedený kód ukazuje, jak by takový template matching mohl probíhat ve hře Solitaire Klondike při vyhledávání konkrétní karty na snímku obrazovky herní plochy.

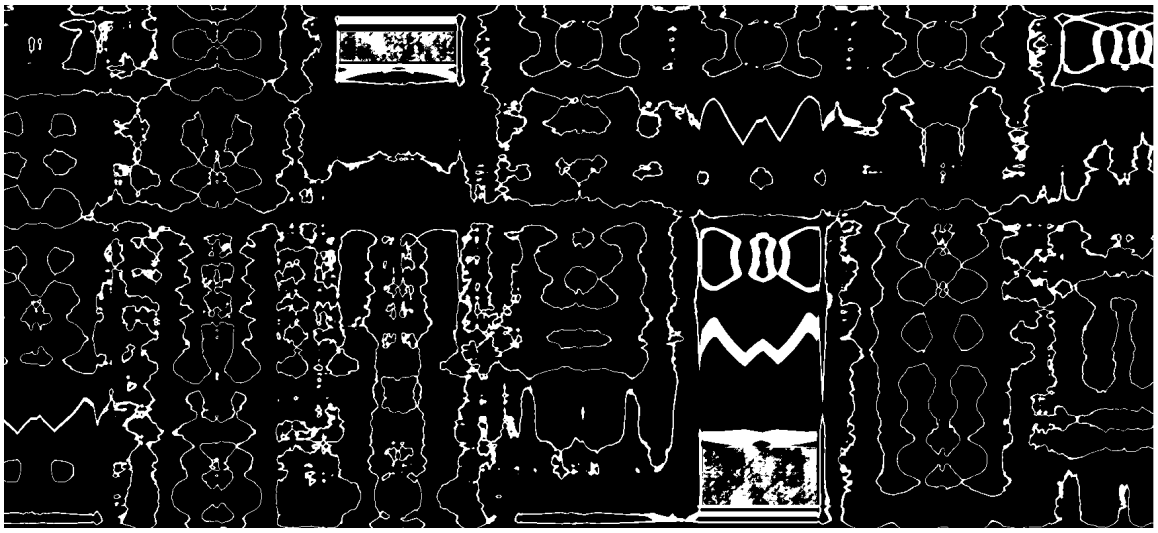
```
card_template = cv2.imread("card.png")
screenshot = cv2.imread("screenshot_of_game.png")
result = cv2.matchTemplate(screenshot, card_template, cv2.TM_CCOEFF_NORMED)
```

Tento kód nejprve načte obrázek vyhledávané karty s využitím funkce `cv2.imread()`, následně využije tu stejnou funkci pro načtení snímku obrazovky, obsahujícím herní plochu. Tato funkce umožňuje již při načítání obrázku, změnit jeho barevné schéma. S přidáním argumentu `cv2.IMREAD_GRAYSCALE` můžeme změnit barevné schéma na černobílé, což může urychlit vyhledávání karty na herní ploše. Poslední řádek kódu potom provádí samotný template matching pomocí funkce `cv2.matchTemplate()`. Jak vypadá výsledek takového vyhledávání karty na herní ploše se můžete podívat na obrázku 3.1, který reprezentuje výsledek vyhledávání přesné polohy srdcové dvojky na herní ploše z obrázku 2.2. Na obrázku si lze všimnout světlých a tmavých míst, světlá místa jsou ta, kde byla zjištěna jistá míra podobnosti, naopak tmavá místa reprezentují, že zde k žádné podobnosti nedochází. Při vyhledávání souřadnice karty, je nutno volit práh podobnosti. Ne vždy lze nalézt 100% shodu, proto je nutné tento práh nastavit na nejvyšší přípustnou hodnotu, kdy jsou vyhledávané prvky nalezeny, obvykle se tento práh podobnosti může nacházet v rozmezí 85% až 95%. Čím vyšší je práh podobnosti, tím přesnější získáme souřadnice hledané karty. V posledním řádku kódu si však lze ještě všimnout posledního argumentu funkce `cv2.matchingTemplate()`, a to je `cv2.TM_CCOEFF_NORMED`. Tímto argumentem uživatel volí matematickou metodu, která bude využita pro porovnávání obrázku karty s obrázkem herní plochy. Knihovna `OpenCV` implementuje hned 6 matematických metod, kde každá z metod se může hodit pro různé situace. Použitá metoda `cv2.TM_CCOEFF_NORMED` se zdá být ideální pro vyhledávání karet na snímku herní plochy, za to například metoda `cv2.TM_CCOEFF` je pro tento konkrétní typ úlohy nepoužitelná, výstup funkce `cv2.matchTemplate()` s využitím této metody si můžete prohlédnout na obrázku 3.2.

[16]Template matching je technika počítačového vidění, která umožňuje nalezení části obrazu, které se shodují se zadanou šablonou nebo vzorem.



Obrázek 3.1: Výsledek vyhledávání funkce `cv2.matchingTemplate()` používající metodu `cv2.TM_CCOEFF_NORMED`.



Obrázek 3.2: Výsledek vyhledávání funkce `cv2.matchingTemplate()` používající metodu `cv2.TM_CCOEFF`.

3.1.4 Interakce s herní plochou

Další klíčovou částí programu pro automatické hraní her Solitaire a Miny je schopnost interagovat s herním prostředím. Pro tuto interakci můžeme zvolit některou z následujících možností.

1. Jednou z možností, pro ovládání myši a klávesnice z programu psaného v jazyce Python, je využití nativních API[17] operačního systému. V případě Linuxového prostředí

[17] API (Application Programming Interface) je sada funkcí, procedur, protokolů a nástrojů, která tvoří rozhraní pro programování aplikací.

by to mohlo zahrnovat práci s knihovnami, jako je `Xlib` nebo `XBC`, které poskytují rozhraní pro komunikaci s X Window System, grafickým subsystémem používaném ve většině distribucí Linuxu. Tyto knihovny umožňují programátorům ovládat myš, klávesnici a další vstupní zařízení přímo na úrovni X Window System, což poskytuje velkou flexibilitu a kontrolu. Nicméně, práce s těmito knihovnami může být složitá a vyžaduje hlubší znalosti technických detailů tohoto subsystému.

2. Alternativní možností nativního přístupu je použití knihoven, které zjednodušují práci s myší a klávesnicí v jazyce Python. Takové knihovny poskytují jednoduché rozhraní pro běžné úkoly, jako je pohyb kurzoru myši, klikání na tlačítka nebo třeba stisk klávesových zkratek. Navíc jsou často navrženy tak, aby byly nezávislé na platformě a fungovaly stejně na různých operačních systémech. Jedna z takových knihoven pro jazyk Python je `pyautogui`. Tato knihovna poskytuje jednoduché a velmi přehledné rozhraní pro práci s myší a klávesnicí. Knihovna `pyautogui` nabízí všechny základní operace s myší a klávesnicí, zmíněné v předešlém odstavci.

Pro interakci s prostředím hry bude nutné zabezpečit především ovládání myši. Nicméně se může hodit i ovládání klávesnice, obě hry podporují použití klávesových zkratek, které lze během hraní aktivně využívat. Ve hře Solitaire Klondike je umožněno použití zkratky 'ctrl'+'z' pro návrat o krok zpět, a zkratky 'ctrl'+'d' pro automatické kliknutí na zásobník stock. Hra Hledání min neumožňuje využívat klávesové zkratky během hry samotné, ale před výběrem obtížnosti nám klávesové zkratky 'ctrl'+'1' až 'ctrl'+'3' umožní vybrat požadovanou obtížnost hry. Je však nutné dávat pozor na to, v jakém rozložení je klávesnice. Knihovna `pyautogui`, při použití funkce `pyautogui.hotkey()`, simuluje stisknutí kláves v řádku nad klávesami 'Q', 'W', 'E', atd. Při používání anglické klávesnice nedochází k problému, nicméně knihovna `pyautogui` nepočítá s českým rozložením klávesnice a při nutnosti stisknout nějaké číslo, se provede stisknutí znaku specifického pro českou klávesnici.

3.2 Návrh programu pro hraní Solitaire Klondike

V této kapitole se podrobně zaměříme na návrh algoritmu pro automatické hraní hry Solitaire Klondike. Tato část práce se bude dělit na několik částí. Nejprve se budeme zabývat programovou reprezentací herní plochy a ukládáním jednotlivých herních prvků. Dále prozkoumáme možnosti hledání možných tahů pro aktuální konfiguraci hry. Na závěr se zaměříme na algoritmy a strategie, které by mohly být použity pro automatické hraní této hry.

3.2.1 Struktura pro ukládání herní plochy Solitaire Klondike

Pro efektivní automatické hraní hry Solitaire Klondike, je nezbytné navrhnout vhodnou datovou strukturu pro reprezentaci herní plochy a jednotlivých herních prvků. Takováto struktura musí umožňovat rychlé a snadné přístupy k informacím o kartách, jejich umístění na herní ploše a možných tazích. Lze vymyslet několik různých datových struktur, které by mohly být vhodné pro reprezentaci herní plochy hry Solitaire Klondike.

1. **Vnořené seznamy**:

   Jedním z možných způsobů jak v programu reprezentovat herní plochu této hry, je použití vnořených seznamů nebo n-tic. Pro každý z herních prvků, které jsou blíže popsány v kapitole 2.1.1, můžeme použít samostatný seznam nebo n-tici, který by

obsahoval informace o kartách a o aktuálním stavu herního prvku. Například, vykládací hromádky foundations mohou být reprezentovány jako čtyři seznamy nebo n-tice, obsahující karty seřazené podle symbolu a hodnoty karet. Sloupce na tableau mohou být reprezentovány jako dalších sedm seznamů, které obsahují karty ve sloupcích, s informací o tom, které karty jsou odkryté a které jsou doposud neodkryté. Stejným způsobem mohou být reprezentovány i zbylé prvky herní plochy stock a waste. Níže je uvedený kousek kódu, který ukazuje možnou inicializaci všech prvků herní plochy (tableau, stock, waste, foundations) před prvním skenováním hry, následně demonstruje vkládání karet a nakonec získávání karet.

```
tableau = [[] for _ in range(7)]
foundation = [[] for _ in range(4)]
stock = []
waste = []

tableau[0].append(('K', 'H'))
tableau[1].extend([('Q', 'L'), ('J', 'D')])
foundation[0].extend([('A', 'C'), ('2', 'C'), ('3', 'C')])
stock.extend([('10', 'H'), ('9', 'C')])

first_tableau_card = tableau[0][0]
second_foundation_card = foundation[0][1]
top_stock_card = stock[-1]
```

Použití vnořených seznamů se může na první pohled jevit jako kvalitní a efektivní řešení. Nicméně v tomto konkrétním příkladě fungují jednotlivé prvky herní plochy hry jako jednotlivci. V případě, že bychom chtěli celou herní plochu předávat argumentem nějaké funkci, bylo by vhodné tyto čtyři prvky vnořit do ještě jednoho seznamu, a to by vedlo ke zhoršení přehlednosti kódu, kdybychom pro přístup ke konkrétní kartě potřebovali mnoho indexů.

2. **Slovníky**:

   Alternativní možností reprezentace herní plochy této hry je použití slovníků, kde klíče reprezentují jednotlivé herní prvky a hodnoty jsou seznamy nebo n-tice obsahující informace o kartách. Tento přístup může usnadnit přístup k informacím, o kartách a jejich pozicích, protože slovníky umožňují rychlé vyhledávání podle klíčů. Níže je uveden kód, který nainicializuje herní plochu obdobným způsobem, jako kód výše, ale s využitím datové struktury slovníku.

```
game_board = {
    'tableau': [[] for _ in range(7)],
    'foundation': [[] for _ in range(4)],
    'stock': [],
    'waste': []
}

game_board['tableau'][0].append(('K', 'H'))
```

```
game_board['tableau'][1].extend([('Q', 'L'), ('J', 'D')])
game_board['foundation'][0].extend([('A', 'C'), ('2', 'C')])
game_board['stock'].extend([('10', 'H'), ('9', 'C')])

first_tableau_card = game_board['tableau'][0][0]
second_foundation_card = game_board['foundation'][0][1]
top_stock_card = game_board['stock'][-1]
```

Pro reprezentaci herní plochy hry Solitaire Klondike je datová struktura slovníku o něco vhodnější možností, než vnořené seznamy. Slovník uchovává herní plochu jako jeden celek, k dílčím částem herní plochy se přistupuje s využitím klíčů, ze kterých jde na první pohled vyčíst, o jakou část herní plochy se jedná. Nicméně datová struktura slovníku může vykazovat jiné potíže, které vycházejí ze způsobu předávání slovníku do jiné proměnné. Výše uvedený kód vytváří slovník `game_board`, v případě, kdybychom chtěli slovník `game_board` předat do několika různých proměnných, předal by se do proměnných pouze odkaz na původní slovník `game_board`. Tato realita vede k tomu, že se při pokusu o modifikaci kterékoliv ze slovníků uloženého v několika proměnných, modifikují všechny slovníky v proměnných, protože proměnné obsahují pouze odkaz na původní slovník. Této komplikaci se však dá vyvarovat vytvářením kopie slovníku při vkládání do dalších proměnných pomocí metody `copy()`, nebo funkce `dict()`.

3. **Třída**:

   Další z možností reprezentace herní plochy hry Solitaire Klondike je vytvoření třídy, která bude obsahovat všechny potřebné atributy a metody pro práci s herní plochou. Tato třída by mohla mít atributy pro reprezentaci všech 4 základních prvků herní plochy. Použitím třídy můžeme získat lepší abstrakci a snazší manipulaci s herní plochou, jelikož metody třídy mohou skrývat implementační detaily a umožňují jednoduchý přístup k funkcím, jako by mohlo být přidávání, odebírání nebo pohyb karet. Navíc, s použitím třídy můžeme jednoduše rozšířit nebo upravit funkcionalitu, pokud by bylo třeba implementovat další verze nebo variace hry Solitaire. Taková třída by mohla poskytovat metody pro manipulaci s kartami a herní plochou, například pro přesun karet mezi sloupci nebo na vykládací hromádky, otočení karet v zásobníku nebo odklad karty na odkládací hromádku. Výhodami využití abstraktní datové struktury třída, jsou lepší modularita kódu, která usnadňuje údržbu a rozšiřitelnost programu. Také to umožňuje jednodušší testování jednotlivých částí programu, protože metody třídy mohou být testovány nezávisle na celku. Níže je uvedena část kódu vytvářející strukturu třídy pro reprezentaci herní plochy této hry.

```
class GameBoard:
    def __init__(self):
        self.tableau = [[] for _ in range(7)]
        self.foundation = [[] for _ in range(4)]
        self.stock = []
        self.waste = []

    def add_card(self, card, location):
        if location.startswith("tableau"):
```

```
            column = int(location[-1]) - 1
            self.tableau[column].append(card)
        elif location == "stock":
            self.stock.append(card)
        elif location == "waste":
            self.waste.append(card)
        elif location.startswith("foundation"):
            suit = int(location[-1]) - 1
            self.foundation[suit].append(card)

game_board = GameBoard()

game_board.add_card(('K', 'H'), 'tableau1')
game_board.add_card(('A', 'C'), 'foundation1')
game_board.add_card(('10', 'H'), 'stock')

first_tableau_card = game_board.tableau[0][0]
second_foundation_card = game_board.foundation[0][1]
top_stock_card = game_board.stock[-1]
```

Výše uvedená varianta vytvoření struktury pro reprezentaci herní plochy hry Solitaire Klondike s použitím struktury třídy se zdá být efektivní z pohledu rozšířitelnosti a možnosti vytváření vnitřních metod pro práci s herní plochou. Metoda `add_card` třídy `GameBoard` vytváří rozhraní pro pohodlné vkládání jednotlivých karet na konkrétní části. Přistupování k jednotlivým kartám je obdobné, jako tomu bylo u datové struktury slovníku. Třída nám umožňuje vytvářet široké spektrum metod usnadňující práci s herní plochou.

3.2.2 Hledání dostupných tahů

Vyhledání všech možných dostupných tahů pro aktuální konfiguraci hry je dalším důležitým krokem, kterým se budeme zabývat v této části práce.

Hledání dostupných tahů zahrnuje prozkoumávání herní plochy a identifikaci karet, které lze přesunout. K tomu je třeba brát v úvahu pravidla hry, která specifikují omezení na přesuny karet mezi sloupci na tableau, prohledávání karet v zásobníku stock, nebo přesuny karet na vykládací hromádky foundations.

Pro identifikaci dostupných tahů můžeme implementovat algoritmus, který zkoumá aktuální stav herní plochy a kontroluje, zda je možné provést určité tahy na základě pravidel hry. Při prozkoumávání aktuálního stavu herní plochy je důležité detekovat všechny možné dostupné tahy. Definování všech existujících tahů může vést k usnadnění vyhledávání konkrétních tahů. Tahy můžeme zadefinovat na základě pravidel hry.

1. Přesun karet mezi sloupci na tableau.
2. Odhalení karty ze zásobníku stock.
3. Přesun karty z odkládací hromádky waste na některý ze sloupců na tableau, nebo na některou z vykládacích hromádek z foundations.

4. Přesun karty z vrcholu sloupce na tableou, na některou z vykládacích hromádek z foundations.

Pravidla hry nám ještě k tomu diktují podmínky, které musí splňovat, jak přesouvaná karta, nebo hromádka karet, tak karta na kterou je přesouváno. Při provádění tahů mezi sloupci na tableau je nutné dodržet pravidlo, že přesouvaná karta musí mít hodnotu o jedna menší, než karta na kterou je přesouváno. Zároveň se musejí ve sloupci střídat barvy karet (červená, černá). Při provádění tahu, kdy přesouváme jednu kartu na některou z hromádek z foundations, se karty na hromádce nesmí lišit barvou, dokonce musejí mít všechny stejný symbol a karty jsou na takovéto hromádce řazeny podle hodnot opačně, než je tomu na sloupcích na tableau. Pro kartu s hodnotou K (král) platí ještě další pravidlo. Karta krále může být přesunuta pouze na sloupec, který neobsahuje žádnou kartu. Vzhledem k těmto realitám můžeme tahy zadefinovat podrobněji, a tím tak umožnit komplexnější analýzu dostupných tahů.

1. Přesun karet mezi sloupci na tableau, který odkryje doposud neodkrytou kartu.
2. Přesun karet mezi sloupci na tableau, který neodkryje žádnou novou kartu.
3. Přesun karty krále ze sloupce na tableau, na prázdný sloupec.
4. Přesun karty z vrcholu sloupce na tableau, na příslušnou vykládací hromádku z foundations. Varianta, kdy je odkryta doposud neodkrytá karta.
5. Přesun karty z vrcholu sloupce na tableau, na příslušnou vykládací hromádku z foundations. Varianta, kdy není odkryta žádná nová karta.
6. Otočení nové karty ze zásobníku stock.
7. Otočení již známé karty ze zásobníku stock.
8. Přesun karty z vrcholu hromádky waste na vrchol sloupce na tableau.
9. Přesun karty krále z vrcholu hromádky waste, na prázdný sloupec na tableau.
10. Přesun karty z vrcholu hromádky waste, na příslušnou vykládací hromádku z foundations.
11. Přesun karty z vrcholu některé z vykládacích hromádek na vrchol sloupce na tableau.
12. Přesun všech karet z waste na zásobník stock.
13. A mnoho dalších tahů.

Jelikož vytváříme algoritmus pro počítačový program, není problémem si zapamatovat, na jaké pozici v zásobníku stock, nebo na odkládací hromádce waste, se nachází která karta. Tento přístup nám umožňuje vytvářet tahy s kartami, které se přímo nenacházejí na vrcholu zásobníku karet stock, ale někde hlouběji. Abychom si tyto karty zpřístupnili, bude potřeba využít výše zmíněného tahu číslo 7 tolikrát, aby se dostala potřebná karta na vrchol odkládací hromádky waste. Musíme si však hlídat zbývající počet možných průchodů zásobníku (redeals). Pokud bychom se tomuto omezení nevěnovali, mohl by algoritmus vyhledat tahy s kartami, které nejsou v aktuálním stavu hry získatelné tímto způsobem.

3.2.3 Strategie pro úspěšné dokončení hry

Volba strategie, kterou při hraní hry Solitaire Klondike zvolit, je naprosto klíčová k úspěšnému a efektivnímu vítězství v této hře. Zde si představíme několik klíčových aspektů, které je třeba zvážit při návrhu a implementaci automatizovaného řešení pro tuto hru, jako jsou pravděpodobnost dohrání hry, velikost stavového prostoru, prioritizace dostupných tahů a techniky prohledávání stavového prostoru.

Jeden z důležitých aspektů tohoto problému je fakt, že ne všechny konfigurace hry Solitaire Klondike je možno dohrát, procentuální pravděpodobnost toho, kolik konfigurací počátečního stavu hry je dohratelných, se odvíjí od toho, jaká jsou pravidla pro danou verzi hry. Tato pravděpodobnost se bude lišit u hry, kdy prohlížíme zásobník stock po jedné kartě, nebo když ho prohlížíme po třech kartách zároveň. Záleží také na tom, jaký je maximální počet průchodů zásobníku stock (redeals), pravděpodobnost se bude lišit pro variantu, když je tento počet omezen, a když není. V poslední řadě záleží na tom, zda nám hra povoluje vracet se ve hře zpět, a jak moc. Některé verze hry Solitarie Klondike hráči omezují možnost vrátit se o krok zpět, zavedením maximálního počtu návratů na předešlý krok. Každý z těchto aspektů přímo ovlivňuje šanci na dohrání hry a některé konfigurace tak činí přímo nedohratelnými [2].

Stavový prostor hry Solitaire Klondike může být extrémně rozsáhlý, během jednoho průchodu hrou může být prozkoumáno až 5000 konfigurací herní plochy [1]. Vzhledem k tomuto obrovskému číslu je důležité efektivně pracovat s množinou dostupných tahů pro aktuální stav herní plochy. Při prohledávání stavového prostoru lze využít různých technik zmíněných v kapitole 2.3.

Práci se stavovým prostorem nám také může usnadnit efektivní metrika pro volbu tahu, který se má provést jako další.

1. **Prioritizace tahů**: Jednou ze strategií, které lze zvolit pro výběr nejlepšího dostupného tahu, je dávání přednosti některým tahům, před jinými. Například můžeme upřednostňovat tahy které odkrývají doposud neodkrytou kartu, nebo třeba tahy, které uvolní sloupec pro kartu krále. Prioritizování tahů se nejvíce podobá přemýšlení běžného hráče.

2. **Heuristicky a hodnotící funkce**: Další z možných strategií, jak volit nejlepší možné tahy, je využití hodnotící funkce, která dokáže rychle odhadnout kvalitu zvoleného tahu. Dokáže také zohledňovat vybrané metriky pro určování výhodnosti tahu. Taková metrika může například zohledňovat snižování počtu neznámých karet, snižování počtu karet na tableau, zvyšování počtu karet ve vykládacích hromádkách foundations, a mnoho dalších.

3. **Vyvažování sloupců**: Při volbě tahu může být důležité udržovat rovnoměrné rozložení karet mezi sloupci na tableau. Toto řešení zabezpečuje dostatek prostoru pro manipulaci s kartami a přesouvání karet mezi sloupci.

Jak již bylo zmíněno výše, ne všechny počáteční konfigurace hry lze dohrát. Když k tomuto faktu přimyslíme to, že píšeme program, který bude automaticky hrát již existující hru, jehož ovládání, skenování, rozhodování se jaký tah zvolit, zabere určité množství času, zjistíme že některé konfigurace hry budou moci zabrat hodiny, než program dojde do cílového stavu.

3.3 Návrh programu pro hraní Hledání min

V této části se zaměříme na návrh algoritmu pro automatické hraní hry Hledání min. Návrh bude rozdělen na 4 základní aspekty tohoto algoritmu, které spolu úzce souvisí a společně pak umožní efektivní řešení hry. Tyto části zahrnují strukturu pro ukládání herního pole, výběr políček, kde se nacházejí miny, výběr bezpečných políček, a výběr nejvhodnějšího políčka v případě, že nelze jednoznačně určit pozici miny, nebo bezpečného pole, zkoumáním okolí jednoho políčka.

3.3.1 Struktura pro ukládání herního pole Hledání min

Pro efektivní řešení hry Hledání min je klíčové navrhnout vhodnou strukturu pro ukládání informací o herním poli. Taková struktura by měla umožňovat rychlý přístup k informacím o jednotlivých políčkách, jako jsou jejich souřadnice, aktuální stav (odkryté, neodkryté, označené vlaječkou) a počet sousedních políček obsahující miny. Jelikož se herní pole Hledání min může odehrávat ve 3 rozměrech, popsaných v kapitole 2.2.1, které jsou čtvercového nebo obdélníkového charakteru, bude nejvhodnější zvolit datovou strukturu dvourozměrného pole (matici) o velikosti odpovídající rozměrům aktuální obtížnosti hry. Každý prvek této matice bude reprezentovat jedno políčko herního pole a bude obsahovat potřebné informace o stavu políčka, jeho souřadnice a počet sousedních políček, obsahující miny. V nejsložitějším případě, při zvolení obtížnosti Expert, má herní pole rozměr 30×16, což tvoří celkem 480 jednotlivých políček, kde každé z nich uchovává nejednu informaci.

V jazyce Python existuje hned několik způsobů, jak reprezentovat matici:

1. Seznam seznamů: Tento způsob reprezentace matice spočívá v použití datové struktury seznamu, která obsahuje další datové struktury seznamu. Každý vnitřní seznam pak může reprezentovat jeden řádek, nebo jeden sloupec matice. Přístup k jednotlivým políčkům herního pole je pak velmi jednoduchý a rychlý pomocí indexace. Níže uvedený kus kódu předvádí vytvoření takového způsobu reprezentace matice.

```
minesweeper_board = []
for _ in range(16):
    row = []
    for _ in range(30):
        row.append(0)
    board.append(row)

minesweeper_board[0][0] = -1
minesweeper_board[1][1] = 1
```

Výše uvedený kód vytváří reprezentaci matice o rozměrech obtížnosti Expert, tj. 16 řádků a 30 sloupců. Každé z políček naplní hodnotou 0, která v tomto okamžiku reprezentuje neodkryté políčko. Využití seznamu seznamů není vůbec špatné pro reprezentaci matice, nicméně existují knihovny, které se přímo zabývají tvorbou a prací s maticemi.

2. `Numpy`[18]: `Numpy` je knihovna pro práci s numerickými daty. Tato knihovna obsahuje třídu `numpy.array`, jež může být použita pro reprezentaci matice. `Numpy` navíc poskytuje mnoho funkcí pro manipulaci s maticemi a výpočty s nimi spojené.

   ```
   minesweeper_board = numpy.zeroes((16, 30), dtype=int)

   minesweeper_board_3d[0, 0] = -1
   minesweeper_board_3d[0, 0] = 1
   ```

   Výše uvedený kus kódu provádí obdobnou inicializaci herního pole, jako to bylo u seznamu seznamů, nicméně využívá knihovnu `numpy`. Tato inicializace matice je v kódu přehlednější, programátorovi se zúží místa, kde by mohl udělat chybu. Navíc není problém matici rozšířit o další rozměry, v případě potřeby.

Jak bylo zmíněno na začátku této kapitoly, každý prvek matice bude reprezentovat jedno políčko herní plochy, pro které potřebujeme uchovávat více než jednu informaci. Toto může být zajištěno několika způsoby:

1. `Rozšíření matice o další rozměr`: Místo původně zamýšlené dvojrozměrné matice můžeme vytvořit vícerozměrnou matici. Přidáním hloubky do matice prakticky vytvoříme nové vrstvy původní dvojrozměrné matice, kde každá z vrstev může obsahovat jeden specifický typ informace o herním poli. Tento způsob může být snadno vytvořen pomocí již zmíněné knihovny `numpy`.

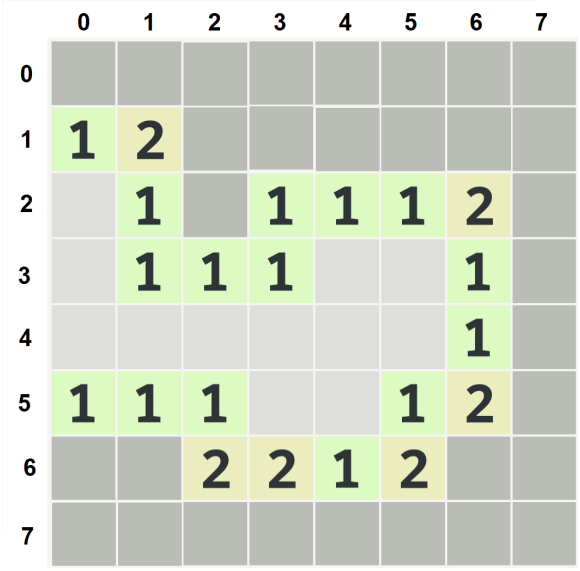
2. `Speciální datová struktura`: Můžeme vytvořit vlastní datovou strukturu, která bude reprezentovat jedno políčko herního pole a ukládat v sobě informace o něm. Taková datová struktura může být reprezentována například třídou `Class`.

3.3.2 Výběr políček s minami

Snad nejjednodušší způsob, jak vybrat políčka, pod kterými se nachází mina, je založit výběr takových políček na základě informací, které poskytují čísla sousedních odkrytých políček. Číslo na odkrytém políčku udává počet min pod sousedními políčky. Pokud je pro konkrétní políčko počet neodkrytých sousedních políček roven hodnotě na odkrytém políčku, můžeme s jistotou říci, že se pod každým ze sousedních neodkrytých políček nachází mina. Tato sousední políčka pak můžeme označit vlaječkou.

Uvažujme herní plochu, po odhalení prvního náhodného políčka, hry Hledání min s aktuálním stavem herní plochy znázorněném na obrázku 3.3. Budeme-li se držet metody odhalování políček s minami popsané výše, zjistíme postupným průchodem všech odkrytých políček, obsahující číslo, zjistíme, že můžeme jednoznačně označit právě 3 miny vlaječkou. Jedná se o políčka na souřadnicích (2, 2), (6, 1) a (6, 6), kde první hodnota v závorce udává číslo řádku a druhá hodnota udává číslo sloupce. Po označení těchto políček vlaječkami se dostaneme do stavu, který znázorňuje obrázek 3.4. Takto jsme byli schopni velmi jednoduše označit políčka, kde se 100% nachází mina, dalším krokem bude zjistit, kde se nachází bezpečná políčka pro aktuální stav herního pole.

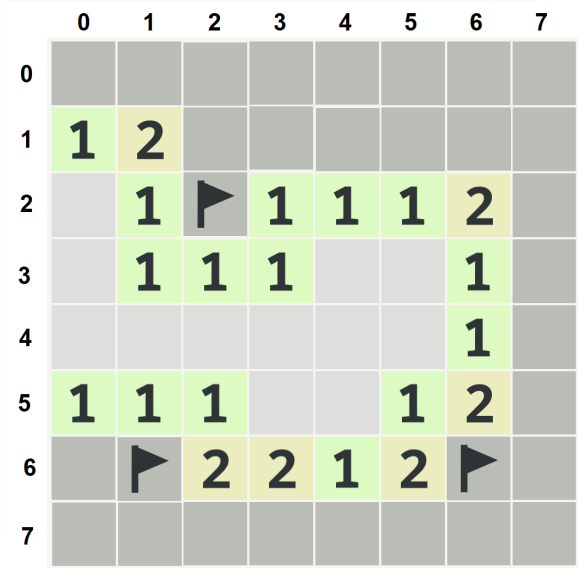
[18] Více informací o knihovně `numpy` najdete na: `https://numpy.org/`



Obrázek 3.3: Příklad herní plochy po rozkliknutí náhodného políčka.

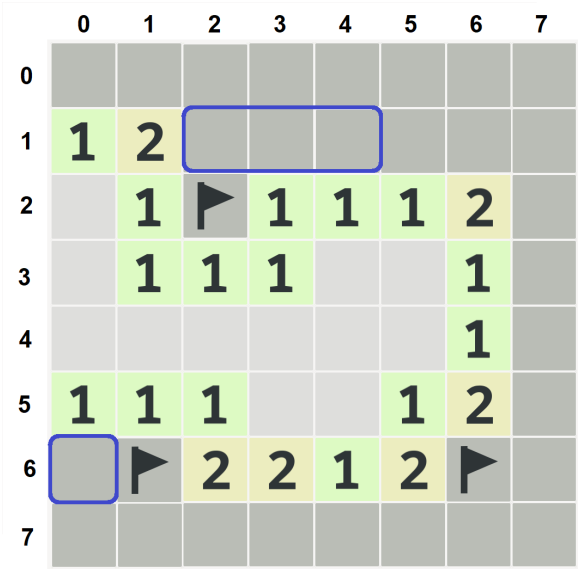
3.3.3 Výběr bezpečných políček

Výběr políčka, pod kterým se nenachází mina je prvním krokem, který musí každý hráč hry Hledání min udělat hned na začátku hry, proto aby hru započal. Jak již bylo výše zmíněno, číslo na odkrytém políčku udává počet min pod sousedními políčky. Pozorujeme-li odkryté políčko, jehož číslo se rovná počtu označených min vlaječkou pod sousedními políčky, a mezi jeho sousedními políčky zbývají ještě další neodkrytá políčka, můžeme je s jistotou odkrýt. Z pravidel hry vychází, že se pod takovými zbylými políčky mina vyskytovat nesmí, pokud samozřejmě nebyla políčka označena vlaječkou chybně.



Obrázek 3.4: Herní plocha po označení políček s minami vlaječkami.

Budeme-li pokračovat v příkladu, který jsme započali v předešlé podsekci 3.3.3, čeká nás průchod přes odkrytá políčka, obsahující číselnou hodnotu, říkající nám kolik se mezi sousedními buňkami nachází min. Dodržováním jednoduchého postupu vysvětleného výše, můžeme s jistotou tvrdit, že se bezpečná políčka nacházejí na souřadnicích (1, 2), (1, 3), (1, 4) a (6, 0). Pro lepší vizualizaci se můžete podívat na obrázek 3.6. Berme zjišťování políček s minami s následujícím odhalováním bezpečných políček, jako jednu iteraci, pak lze takto algoritmicky pokračovat v iterování, dokud úspěšně nedohrajeme hru. Nicméně se tímto postupem můžeme dostat do situace kdy s použitím těchto pravidel nebudeme schopni odhalit pozici min ani polohu bezpečných políček, příklad takové konfigurace je uveden na obrázku 3.5. V takové konfiguraci můžeme prozkoumat každé odkryté políčko, obsahující číslo, vedle něhož se nachází nějaké doposud neodkryté políčko, a nebudeme schopni těmito snadnými pravidly zjistit, jaký provést následující tah. Jak řešit takový problém probereme hned v následující části.



Obrázek 3.5: Herní plocha po označení políček s minami vlaječkami, s modře vyznačenými políčky, které na 100% neskrývají minu.

3.3.4 Výběr nejvhodnějšího políčka

Jak jednoznačně určit, kde se nachází mina a kde je bezpečné políčko jsme si uvedli výše. Hra Hledání min se však může dostat do takové konfigurace, kdy není po prozkoumání jednotlivých políček jasné říci, kde se nachází další mina, popřípadě kde se nachází bezpečné políčko. V takových případech je nutné zvolit některou z následujících strategií:

1. **Rozšíření zkoumaného políčka o další sousední políčko**: Pro zjištění dalšího nejvhodnějšího tahu, můžeme rozšířit zkoumání o jedno sousední políčko a na základě dvou pozorovaných hodnot zvolit efektivní strategii pro získání dalšího možného kroku. Důležité je, aby mělo sousední políčko ve svém okolí nějaká neodkrytá políčka, a aby se tato neodkrytá políčka obou pozorovaných políček překrývala. Tímto způsobem lze rozšiřovat zkoumaná políčka o další sousední políčka, ale komplexnost strategií se bude zvyšovat.

Obrázek 3.6: Konfigurace herní plochy, kde nelze určit poloha miny, ani bezpečného pole, na základě pozorování jednoho políčka.

```
1   nfn_a := non_flagged_neighbours(tile_a)
2   nfn_b := non_flagged_neighbours(tile_b)
3   a_value := value(tile_a) - |flagged_neighbours(tile_a)|
4   b_value := value(tile_b) - |flagged_neighbours(tile_b)|
5   if a_value - b_value == |nfn_a \ nfn_b|:
6       flag_all(nfn_a \ nfn_b)
7       step_all(nfn_b \ nfn_a)
```

Výše napsaný pseudokód jsem převzal z volně dostupného git repozitáře od uživatele **ChasJC23**[19], který naimplementoval agenta využívající strategii, kterou si teď blíže popíšeme, pro automatické hraní hry Hledání min.

Tato strategie využívá dvojici sousedních buněk, kde každá z nich má ve svém okolí nějaká neodkrytá políčka, která se vzájemně překrývají. Na řádku 1 a 2 probíhá inicializace množin `nfn_a` a `nfn_b`, které jsou naplněny sousedními políčky pro každé políčko z prozkoumávané dvojice, které doposud nebyly rozkliknuty ani označeny. Na následujících dvou řádcích probíhá korekce číselné hodnoty každého z prozkoumávaných políček tak, že se od tohoto čísla odečte počet políček, označených vlaječkou, v okolí zkoumaného políčka. Následně se provede porovnání, kde se odečte nová hodnota políčka `b` od nové hodnoty políčka `a`. Tato hodnota se následně porovnává s velikostí množiny rozdílu množin `nfn_a` a `nfn_b`. Jsou-li tyto hodnoty stejné, můžeme označit všechna políčka množiny `nfn_a \ nfn_b` vlaječkou a všechna políčka množiny `nfn_b \ nfn_a` můžeme rozkliknout. Tento postup je spolehlivý, neplatí však pro všechny sousední dvojice. Jedná se o jednu z mnoha metod, které lze použít pro komplexnější rozhodování se o tom, jaký tah zvolit.

[19] `https://github.com/ChasJC23/SetBasedMinesweeperAgent`

2. **Simulace možných pozic zbývajících min**: Tento přístup spočívá v provedení velkého počtu simulací, které by mohly modelovat všechna možná rozložení min na neodkrytých políčkách a vyhodnocovat tak pravděpodobnost, že jednotlivá políčka obsahují minu. Na základě těchto pravděpodobností může být vybráno bezpečné políčko s nejnižší pravděpodobností výskytu miny

3. **Soustava lineárních rovnic**: Na základě odkrytých čísel políček na herním poli, můžeme vytvořit systém rovnic, který popisuje možné konfigurace min na neodkrytých políčkách. Řešení této soustavy rovnic nám pak umožní odhadnout pravděpodobnosti pro jednotlivá neodkrytá políčka. [6]

4. **Neuronové sítě**: Další z možností je použít neuronové sítě[20], které mohou být natrénovány na rozpoznávání složitých vzorů a odhadování pravděpodobnosti výskytu min na základě informací dostupných z herního pole [8]. Tento přístup může být velmi efektivní zejména v situacích, kdy je konfigurace herní plochy velmi komplexní.

Při implementaci algoritmu pro výběr nejvhodnějšího políčka je důležité dbát na efektivitu a přesnost, aby byl algoritmus schopen rychle a spolehlivě identifikovat výskyt min a minimalizovat tak riziko chybného výběru. Pro dosažení kvalitnějších výsledků, lze implementovat více výše zmíněných řešení dohromady. Kromě toho je také důležité průběžně aktualizovat informace o herním poli a pravděpodobnostech jejich políček, je-li nutno tyto pravděpodobnosti počítat.

Na konec této části je nutné se zmínit, že i přes volbu nejefektivnější možné metody pro získání pravděpodobnosti, se může hra dostat do takové konfigurace, kde zbývá označit poslední minu a je na výběr pouze ze dvou možných polí. A přesně to je pravděpodobnost 50% na 50%.

[20] Neuronová síť je typem umělé inteligence, která se snaží napodobit způsob fungování lidského mozku.

Kapitola 4

Implementace

Tato kapitola se věnuje vlastní realizaci automatického hraní her Solitaire Klondike a Hledání min, která se bude opírat o prvky, které jsou diskutovány v předešlé kapitole o analýze a návrhu. Budou zde popsány části kódu, které vykonávají stěžejní aspekty obou her, na které bylo taktéž apelováno v předešlé kapitole. Také zde bude popsána implementace neobvyklých, či náročných problémů, které se při implementaci vyskytly.

První část této kapitoly se bude věnovat implementaci společných aspektů, které se musely implementovat pro automatické hraní obou her a nijak zásadně se od sebe neliší. Do této části hry se řadí zejména implementace spouštění her a detekce okna hry. Další část se zaměřuje na implementaci programu pro automatické hraní hry Solitaire Klondike. V poslední části této kapitoly se budeme věnovat implementačním detailům programu pro automatické hraní hry Hledání min.

4.1 Společné části

Tato část popisuje implementaci společných částí obou programů, které mají stejný charakter, jež byly diskutovány v kapitole 3.1. Popisuje se zde implementace spouštění her a detekce okna her.

4.1.1 Implementace spuštění her

Obě hry jsou spouštěny s využitím knihovny `subprocess`, která byla diskutována v části 3.1.1, následovně:

```
#Pro hru Hledání min
minesweeper = subprocess.Popen(['gnome-mines'], shell=True)

#Pro hru Solitaire
solitaire = subprocess.Popen(['sol'], shell=True)
```

Hned po provedení tohoto kódu je nutno, před prováděním dalšího kódu, proces alespoň na jednu vteřinu uspat pomocí funkce `time.sleep()`. Je to důležité proto, aby se hra stihla včas zapnout před prováděním jakýchkoliv dalších akcí.

Hra Solitaire Klondike nám navíc umožňuje využití přepnutí hry do režimu celé obrazovky pomocí klávesy "F11", ta se stiskne pomocí funkce `pyatuogui.press("f11")`. Hra Hledání min nám neumožňuje přepnout hru do režimu celé obrazovky s použitím klávesové zkratky, nicméně to u této hry nebude tak důležité, jako u hry Solitaire Klondike.

4.1.2 Implementace detekce okna hry

Detekce okna her je řešena s využitím knihovny `wnck` pomocí mnou naimplementovaných pomocných funkcí: `get_location_of_solitaire_window()` pro hru Solitaire Klondike a `get_location_of_mines_window()` pro hru Hledání min, které se nacházejí v modulech s názvy `solitaire_detection.py` a `mines_game_detection.py`. Více o těchto modulech si můžete přečíst níže, v částech 4.2.1 a 4.3.1. Tyto funkce inicializují globální proměnnou `window_size`, která je datovou strukturou tuple, jež obsahuje 4 prvky. První dva prvky obsahují `x` a `y` souřadnici levého horního rohu okna hry a poslední 2 prvky obsahují šířku a výšku okna v pixelech. Tyto informace, o rozměrech a poloze okna hry na ploše, jsou důležité pro následující skenování. Díky této proměnné je nám umožněno vytvářet snímky obrazovky pouze v regionu okna hry, a tím tak omezit skenovanou plochu, což vede ke zrychlení a zefektivnění. Tento princip skenování pouze určitého regionu se bude dále hodit převážně při skenování jednotlivých sloupců herní plochy hry Solitaire Klondike, nebo nových karet na zásobníku stock, protože bude zapotřebí skenovat pouze nově odhalené karty.

4.2 Program pro hraní hry Solitaire Klondike

Tato část čtenáře blíže seznámí s implementačními detaily jednotlivých částí programu pro automatické hraní hry Solitaire Klondike. Budou zde popsány moduly programu a jejich obecný obsah, datová struktura pro ukládání herní plochy, princip skenování herních prvků a problémy s tím související, vyhledávání dostupných tahů a použitá strategie řešení hry.

4.2.1 Struktura programu

Program se skládá z 5 modulů:

- **solitaire.py**: `solitaire.py` je hlavním modulem. Tento modul neobsahuje žádné vlastní funkce, používá funkce níže zmíněných modulů `solitaire_detection.py` a `solitaire_solve.py`. V tomto modulu probíhá spuštění hry, detekce okna, inicializace poměrů rozlišení, první skenování a inicializce herní plochy. Nakonec se odtud přechází do modulu `solitaire_solve.py`, kde se děje veškerá logika automatického hraní hry.

- **solitaire_detection.py**: `solitaire_detection.py` je modulem obsahujícím funkce pro spouštění hry, detekci okna hry, zabezpečování poměrů rozlišení, inicializace struktury pro herní plochu a její následné inicializace, skenování karet a další pomocné méně zajímavé funkce.

- **solitaire_printing.py**: Tento modul slouží čistě pro zabezpečení přehledných výpisů. Obsahuje především funkce, které tisknou aktuální stav herní plochy a množinu možných tahů nad aktuální herní plochou. Jelikož mohou výpisy obsahovat tisíce řádků, tak jsou výpisy přesměrovány do souboru `game.log`.

- **solitaire_get_moves.py**: Tento modul obsahuje funkce pro vytvoření množiny dostupných tahů ze struktury, která zrcadlí aktuální stav herní plochy. Jsou zde navíc obsaženy funkce, které kontrolují, zda-li není vytvářený tah v rozporu z pravidly hry Solitaire Klondike. Také se zde nachází funkce, které zajišťují ohodnocení jednotlivých tahů, sloužící pro následující výběr nejvhodnějšího dostupného tahu.

- **solitaire_solve.py**: Asi nejpodstatnějším z modulů je modul `solitaire_solve.py`, který v sobě obsahuje veškeré funkce pro aktualizování jednotlivých prvků herní plochy po provedení tahu, funkce pro vykonávání všech definovaných typů tahů, a v neposlední řadě funkci řídící mechaniku hry a prohledávání stavového prostoru.

4.2.2 Datová struktura pro herní plochu

Pro programovou reprezentaci jsem zvolil strukturu slovník, která se při spuštění hry nainicializuje funkcí `get_board_init()`, implementovanou v modulu **solitaire_detection.py**, jež vrací vytvořenou strukturu `game_board`. Níže je předvedena již zmíněná struktura herní plochy:

```
game_board = {
    'tableau': [
        {
            'cards': [UNKNOWN_CARD] * i + [],
            'region': (reg[X], reg[Y], column_width, column_height),
        }
        for i, reg in enumerate(column_positions)
    ],
    'foundations': {
        'H': [],
        'C': [],
        'D': [],
        'L': []
    },
    'stock': {
            'cards': [("ret", "ret", "ret")] + [UNKNOWN_CARD] * 24
    },
    'waste': {
            'cards': [],
            'coords': (column_width + column_width // 2, top_margin //2),
            'region': (column_width, 0, column_width, top_margin)
    },
}
```

Všimněte si, že u herních prvků tableau, stock a waste ukládám jak karty, tak i souřadnice a rozměry regionu, kde se prvky ve hře nachází. Jak již bylo zmíněno v části 4.1.2, tyto regiony potřebuji proto, abych zmenšil plochu, na které se bude vyhledávat nová karta. U prvku tableau se tento region počítá pro každý sloupec zvlášť. Dále si také všimněte, že struktura obsahuje ve své inicializační fázi mnoho proměnných, jako jsou `column_width`, `top_margin` a další. Tyto proměnné obsahují číselné hodnoty, které reprezentují počet pixelů, jež reprezentují nějakou vzdálenost na herní ploše. Proměnné tohoto typu se odrážejí o předdefinované konstanty, které obsahují konkrétní číselné hodnoty pro herní plochu na obrazovce s rozlišením 1920x1080. Aby byly tyto konstanty přenositelné i na obrazovky s jiným rozlišením, je zapotřebí zajistit přeškálování těchto konstant na základě získaných poměrů mezi referenčním rozlišením a aktuálním rozlišením. Tyto poměry jsou získávány s pomocí funkce `get_resolution_ratios()` implementované v modulu `solitaire_detection.py`.

Jednotlivé karty, které se dále v kódu vkládají do struktury `game_board`, jsou reprezentovány datovou strukturou tuple, která obsahuje hodnotu karty, barvu karty a symbol

karty. V případě neznámých karet, se do struktury vkládá tuple se všemi prvky s hodnotou `None`. Až po odhalení, o jakou kartu se jedná, se tato neznámá karta nahrazuje kartou nově získanou.

4.2.3 Skenování herních prvků

Veškeré skenování je zabezpečeno s pomocí knihovny `OpenCV`. První skenování herní plochy je nejdůležitější a také nejdéle trvá, protože je potřeba vyhledat sedm karet nacházejících se na vrcholcích sloupců na tableau, a k dispozici máme 52 možných karet v balíčku.

Nejprve se vytvoří snímek herní plochy a následně se cyklicky načítají jednotlivé snímky karet, a pomocí funkce `cv2.matchTemplate()` se zjišťuje, zda-li se karta nachází na aktuální herní ploše. Jednotlivé snímky karet byly pořízeny na obrazovce s rozlišením 1920x1080, pokud bychom tyto snímky porovnávaly se snímkem herní plochy pořízeném na obrazovce s odlišným rozlišením, funkce `v2.matchTemplate()` by žádnou z karet neodhalila. Proto je zapotřebí karty přeškálovat podle poměru mezi referenčním a aktuálním rozlišením. To se provádí pomocí funkce `get_card()` v modulu `solitaire_detection.py`, která načte snímek karty do proměnné a následně provede přeškálování a vrací proměnnou obsahující přeškálovanou kartu.

Na konci tohoto skenování se množina sedmi nalezených karet převede na formát (hodnota, barva, symbol) a seřadí se podle `x` souřadnice, od nejlevější po nejpravější kartu, a následně jsou v tomto pořadí vloženy do struktury `game_board` na vrcholy odpovídajících sloupců na tableau.

4.2.4 Hledání dostupných tahů

Prohledávání aktuálního stavu herní plochy, za účelem získání všech dostupných tahů se provádí v modulu **solitaire_get_moves.py** pomocí funkce `get_possible_moves()`. Tato funkce se uvnitř větví na funkce:

- `get_moves_on_tableau()`
- `get_moves_on_foundation()`
- `get_moves_from_foundation()`
- `get_moves_on_stock()`
- `get_moves_on_waste()`

Každá z výše zmíněných funkcí vyhledává možné tahy na příslušné části herní plochy. Při každém získání dostupného tahu se počítá hodnota tohoto tahu, která bude určovat jeho prioritu. Pro získání této hodnoty se využívá funkce `get_heuristic_value()`, která tvoří ohodnocení tahu na základě počtu neznámých karet po provedení tahu, počtu karet v zásobníku stock a na hromádce waste po provedení tahu a počtu karet na vykládacích hromádkách foundations. Každý z aspektů této heuristiky je jinak důležitý a tato důležitost je definována váhou pro každý z těchto aspektů. Po nalezení všech dostupných tahů a jejich ohodnocení jsou tahy vkládány do seznamu `possible_moves`, který se na konci funkce `get_possible_moves()` předává funkci `sort_possible_moves()`, která vrací seřazený seznam tahů na základě hodnoty heuristické funkce od nejvýhodnějšího tahu po nejméně výhodný tah.

4.2.5 Strategie hry

Strategie hry přímo závisí na zvoleném algoritmu pro průchod stavovým prostorem. Ve své implementaci jsem zvolil modifikovanou verzi algoritmu DFS, který je podrobněji popsán v části 2.3.1. Funkce `dfs_iter()`, implementovaná v modulu `solitaire_solve.py`, funguje na principu iterativní modifikace algoritmu DFS. Od algoritmu DFS se liší tím způsobem, že provádí první dostupný tah ze seznamu `possible_moves`, který však byl seřazen na základě hodnotící funkce.

Na začátku je ve funkci `play_game()` nainicializován počáteční uzel `init_node`, každý uzel je pak reprezentován dvojicí (herní plocha aktuálního uzlu, seřazený seznam doposud neprovedených tahů). Pokračuje se funkcí `dfs_iter()`, která vezme počáteční uzel, vloží jej na zásobník `open_list` a započne cyklus obsluhující odehrání hry. Tento cyklus poběží tak dlouho, dokud není zásobník `open_list` prázdný, což se může stát pouze tehdy, je-li prozkoumán celý stavový prostor hry, bez nalezení řešení. Jinými slovy, hra nemá řešení. V každé z iterací se nejprve získá aktuální stav herní plochy a možné tahy, z vrcholu zásobníku. Následně se provede ověření, zda-li není aktuální stav stavem cílovým, pokud by tomu tak bylo, funkce končí a vrací počet tahů, které vedly k nalezení řešení. Není-li aktuální stav stavem cílovým, provede se kontrola, jestli ještě existují nějaké tahy, které nad touto konfigurací nebyly doposud vyzkoušeny. Pokud neexistují další proveditelné tahy nad tímto stavem, je nutné se vrátit o tah nazpátek, což prakticky znamená to, že je tento stav odstraněn z vrcholu zásobníku `open_list` a na novém vrcholu zásobníku se nachází stav, který byl aktuální před provedením posledního tahu. Existují-li nad aktuálním stavem herní plochy doposud neprovedené tahy, je vytvořena charakteristika herní plochy, jinými slovy heš (anglicky hash). Tento heš se vloží do množiny `closed_list`, pokud se v ní už jednou nenachází. V dalším kroku se získá nejlépe ohodnocený tah ze začátku seznamu dostupných tahů, ze kterého je následně odstraněn. Tento tah se předá funkci `do_move()`, která jej obslouží. Ve své implementaci uvažuji i takové tahy, které si vyžadují více než jeden přesun karet, nebo jedno kliknutí na zásobník stock, typicky se jedná o přesun karty, která se nachází někde uvnitř zásobníku zbylých karet. Takový tah vyžaduje několik kliknutí na zásobník stock a následný přesun karty na příslušný cíl. Když takový tah nikam nevede je zapotřebí se vrátit o tah zpět, ale tento konkrétní tah si vyžaduje stisknutí vrácení se zpět více než jedenkrát. Proto, aby program věděl, kolikrát se musí vracet, aby se vrátil na konfiguraci před provedením posledního tahu, využívá zásobník `undo_count`. Do tohoto zásobníku je po každém provedení jakéhokoliv tahu vloženo číslo, reprezentující počet přesunů nebo kliknutí, které byly zapotřebí pro provedení tahu. Je-li potřeba se vracet na předchozí tah, program zjistí hodnotu z vrcholu zásobníku `undo_count` a provede návrat tolikrát, kolikrát je definováno získaným číslem. Po provedení každého tahu, se nejprve zjišťuje, zda-li jsem se v této konfiguraci již jednou nenacházel, k tomuto ověřování slouží právě množina `closed_list`. Během průchodu stavového prostoru se také může stát, že pro aktuální stav nejsem schopen vytvořit žádný další tah, v tomto případě se provede návrat na stav před provedením posledního tahu.

4.2.6 Známé nedostatky

Při spouštění programu na méně výkonném počítači, dochází k tomu, že program stíhá vykonávat úkony myší rychleji, než se stačí aktualizovat informace na obrazovce. Toto vede k chybnému skenování, vzniklém na základě vytváření snímků herní plochy dříve, než se stihne provést tah aktualizovat. Tento problém lze řešit tak, že se bude před funkci, pořizující snímek herní plochy, zavádět čekání, to však povede k celkovému zpomalení programu.

4.3 Program pro hraní hry Hledání min

Tato část se zaměřuje na implementaci jednotlivých částí programu pro automatické hraní hry Hledání min. Budou zde popsány moduly programu a jejich obecný obsah, datová struktura pro ukládání herní plochy, princip skenování herního pole, výběr políček, která mohou být odkryta, či označena vlaječkou, popis implementace získávání pravděpodobností políček obsahující minu.

4.3.1 Struktura programu

Program se skládá ze 3 modulů:

- **mines.py**: `mines.py` je hlavní modul. Tento modul neobsahuje žádné vlastní funkce, používá funkce níže zmíněných modulů `mines_game_detection.py` a `mines_play.py`. V tomto modulu probíhá spuštění hry, získání informací z argumentů příkazové řádky, detekce okna hry, a v neposlední řadě inicializace herního pole. Nakonec se odtud přechází do modulu `mines_play.py`, kde se děje veškerá logika automatického hraní hry.
- **mines_game_detection.py**: Tento modul obsahuje funkce obsluhující získání informací z argumentů příkazové řádky, spuštění hry, detekci okna hry, vytváření snímků herního pole, získání a inicializace souřadnic pro jednotlivá políčka herního pole, získávání informací z herního pole, ověřování zda-li byla hra úspěšně, či neúspěšně dohraná, a taky další podpůrné funkce.
- **mines_play.py**: `mines_play.py` je nejpodstatnějším z modulů, obsahuje funkce pro interakci s herním polem, hledání jednoznačných bezpečných políček, hledání jednoznačných políček obsahující miny, počítání pravděpodobností výskytu min a další méně podstatné funkce.

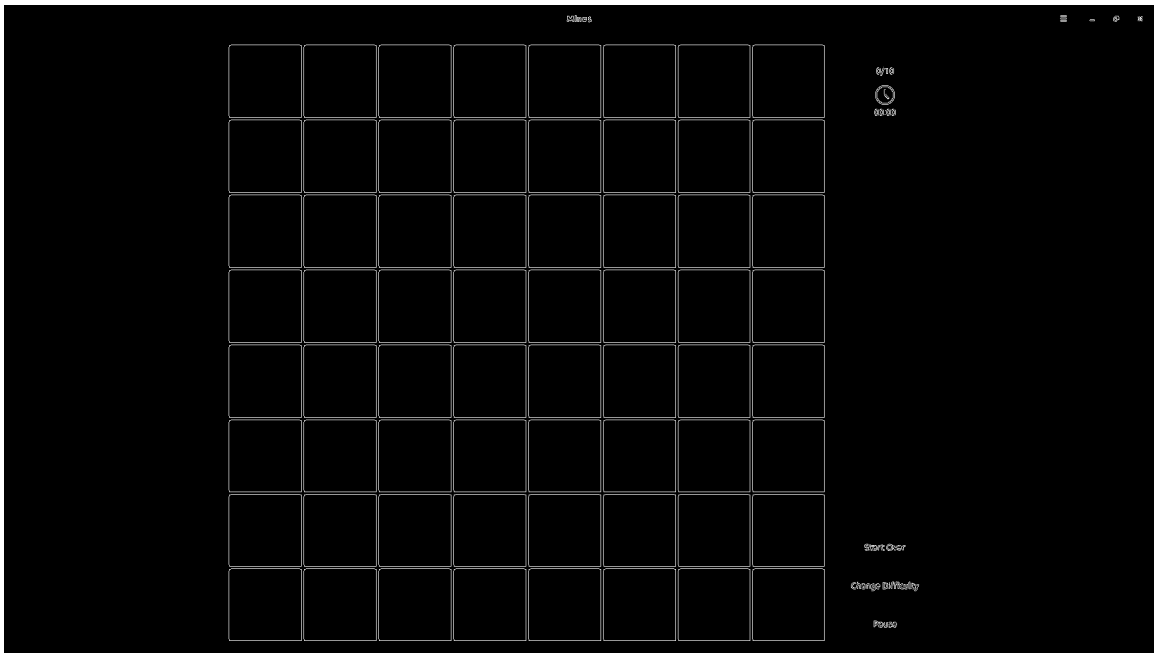
4.3.2 Datová struktura pro herní pole

Pro herní pole jsem vytvořil datovou strukturu pomocí funkce `numpy.zeroes()`. Vytvářím trojrozměrnou matici, která má rozměry odpovídající zvolené obtížnosti, a k tomu má třetí rozměr o velikosti 4. První dvě vrstvy používám pro uchování `x` a `y` souřadnic, třetí vrstvu využívám pro uchování informace o stavu políčka. Poslední vrstva obsahuje informaci o tom, zda-li je ještě potřeba konkrétní políčko prozkoumávat, nebo zda-li ho můžu při dalším skenováním ignorovat.

4.3.3 Skenování herního pole

Pro skenování herního pole je nejprve zapotřebí získat souřadnice a rozměry herní matice, to je čtvercová, popřípadě obdélníková část herní plochy, obsahující pouze políčka. Tyto souřadnice a rozměry potřebuji získat za tím účelem, abych mohl nainicializovat souřadnice každého z políček a pak k nim velmi snadně přistupovat. Tyto souřadnice mi již nepomůže získat knihovna `pyautogui`, poněvadž se rozměry herní plochy mohou lišit, vzhledem k různým distribucím Linuxu. Tohle vyvolá stejný problém, jako potřeba přeškálovávání snímků karet, zmíněná v části 4.2.3. Proto je potřeba využít jiného způsobu získání těchto informací, a to provádím pomocí knihovny `OpenCV`, konkrétně s využitím funkcí `cv2.Canny()` a `cv2.HoughLinesP()`. Funkce `cv2.Canny()` mi na snímku herní plochy detekuje všechny

hrany, graficky znázorněný výstup této funkce si můžete prohlédnout na obrázku 4.1. Z tohoto výstupu jsem schopen pomocí funkce `cv2.HoughLinesP()` detekovat úsečky o určité minimální délce. Tato funkce vrací seznam nalezených úseček, detekované zvýrazněné úsečky si můžete prohlédnout na obrázku 4.2. Po získání seznamu úseček, tento seznam předám funkci `get_edges_of_matrix()`, která mi vrátí souřadnice nejspodnější horizontální úsečky a souřadnice nejlevější vertikální úsečky. Pomocí těchto souřadnic jsem pak schopen si získat souřadnice a rozměry herní matice.

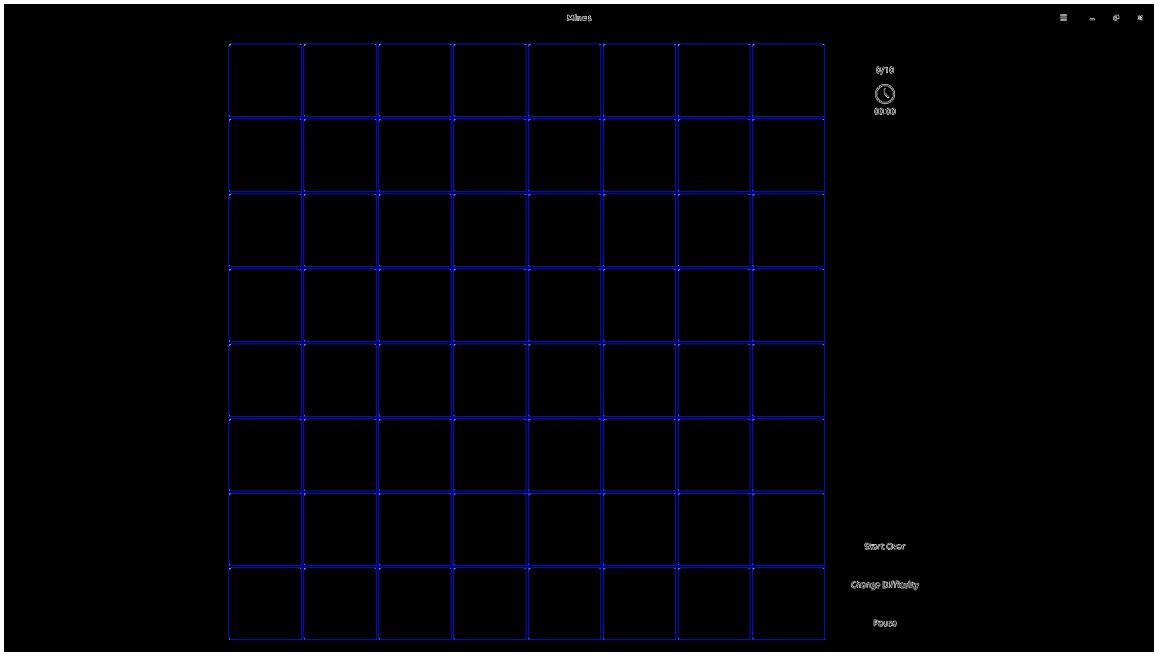


Obrázek 4.1: Graficky znázorněný výstup funkce `cv2.Canny()`.

Skenování herní matice probíhá na základě získávání informací o barvě políčka. Při skenování herní matice získávám barvu pixelu konkrétního políčka, a díky této informaci a faktu, že každý možný stav políčka je zvýrazněn jinou barvou, vím v jakém stavu se konkrétní políčko nachází, popřípadě jaké se na políčku nachází číslo. Skenují se pouze ta políčka, která mají v matici nastavenou informaci o tom, že ještě nebyly zcela prozkoumány, toto mi zajistí vyšší rychlost a efektivitu při skenování.

4.3.4 Výběr jednoznačných políček

Jednoznačná políčka jsou taková políčka, o kterých mohu sebejistě tvrdit, že se pod nimi nachází nebo naopak nenachází mina. Princip zjišťování jednoznačných políček je přímo popsán v části 3.3. Implementace získávání takovýchto jednoznačných políček obsluhují funkce `find_safe_fields()` a `find_easiest_mines()`. Tyto funkce pracují na podobném principu, kdy je prozkoumáváno konkrétní políčko obsahující číslo a minimálně jedno neodkryté sousední políčko. Prochází se všechna sousední políčka prozkoumávaného políčka a pokud je sousední políčko neodhalené, inkrementuje se čítač `unknown_count`, pokud je sousední políčko označeno vlaječkou, inkrementuje se čítač `flags_count`. Je-li po prozkoumání všech sousedních políček hodnota součtu obou čítačů rovna číslu prozkoumávaného políčka, označí se všechna neodkrytá sousední políčka vlaječkou. Je-li pak hodnota čítače



Obrázek 4.2: Modře znázorněné úsečky, detekované pomocí funkce `cv2.HoughLinesP()`.

počítající počet vlaječek rovna číslu prozkoumávaného políčka, odhalí se všechna neodkrytá sousední políčka.

4.3.5 Výběr nejednoznačných políček

Pro výběr nejednoznačných políček jsem se rozhodl využít metody rozšíření zkoumaného políčka o další sousední políčko, v kombinaci s řešením soustavy lineárních rovnic. Obě tyto metody naimplementoval, uživatel **PedroKKr**, a jeho řešení je volně dostupné z jeho repozitáře[1]. Po bližším prozkoumání jeho řešení jsem se rozhodl o využití jeho implementace, jak metody rozšíření zkoumaného políčka o další sousední políčko, tak i výpočtu pravděpodobností na základě řešení soustavy lineárních rovnic.

Princip fungování první z metod je podrobně popsán v části 3.3.4. Implementace této části přímo odráží její popis, v již zmíněné části. Převzatý kus kódu bylo zapotřebí upravit tak, aby byl kompatibilní s reprezentací herní plochy řešené v této práci, bylo také zapotřebí napsat pomocné funkce, které tuto kompatibilitu zajistí.

Metoda řešení soustavy lineárních rovnic funguje na tom principu, že se pro každé políčko obsahující číslo a alespoň jedno neodkryté sousední políčko, vytvoří lineární rovnici popisující toto políčko. Například pokud políčko obsahuje číslo 3 a má mezi svými sousedy 5 neodkrytých políček, vytvoří následující rovnici $a1+a2+a3+a4+a5 = 3$. Pojmenování $a1$, $a2$, $a3$, atd. je pro každé hraniční neodkryté políčko unikátní (hraničním políčkem je myšleno neodkryté políčko, které má alespoň jedno sousední políčko odkryté), to zajistí závislost mezi jednotlivými rovnicemi a vzniká tak soustava lineárních rovnic. Taková soustava lineárních rovnic se potom dá vyřešit s použitím knihovny `sympy`[2] pomocí funkce `sympy.linsolve()`.

Strategie pro úspěšné dohrání hry potom spočívá v opakovaném odhalování jednoznačných políček. V případě, že nelze určit žádné jednoznačné políčko, jsme nuceni odhalit

[1] `https://github.com/PedroKKr/Minesweeper-probability-calculator`

[2] Více informací o knihovně `sympy` si můžete přečíst na: `https://www.sympy.org/en/index.html`

některá nejednoznačná políčka. Určitý počet her lze úspěšně dohrát pouze na základě odhalování jednoznačných políček.

Kapitola 5

Testování

V této kapitole se zaměříme na testování implementovaných programů. V první části se podíváme na způsob testování programu pro automatické hraní hry Hledání min. V této části se také podíváme na získaná data a vysvětlíme si proč existují případy, že program neúspěšně odkryje políčko s minou. V další části se zaměříme na testování automatického hraní hry Solitaire Klondike. Také se v této části blíže se seznámíme s důvody, proč jisté konfigurace hry trvají vyřešit i několik hodin.

5.1 Testování automatického hraní hry Hledání min

Při testování hry Hledání min byla každá z obtížností spuštěna 30 krát, výsledky testování si můžete prohlédnout v tabulce 5.1. Z této tabulky lze vyčíst základní informace, jako jsou čas nejrychlejší hry, čas nejpomalejší hry, počet neúspěšných her, nebo průměrný čas na hru. Tyto informace jsou pro jednotlivé úrovně následující:

- **Začátečník:**
  - Čas nejrychlejší úspěšné hry: 7,609s
  - Čas nejpomalejší úspěšné hry: 27,757s
  - Počet neúspěšných her: 1 ze 30
  - Průměrná doba trvání hry: 16,63s
- **Pokročilý:**
  - Čas nejrychlejší úspěšné hry: 49,243s
  - Čas nejpomalejší úspěšné hry: 91,227s
  - Počet neúspěšných her: 2 ze 30
  - Průměrná doba trvání hry: 67,85s
- **Expert:**
  - Čas nejrychlejší úspěšné hry: 148,521s
  - Čas nejpomalejší úspěšné hry: 225,796
  - Počet neúspěšných her: 5 ze 30
  - Průměrná doba trvání hry: 174,87s

Neúspěšné hry značí ty hry, kdy se program rozhodoval, které políčko s nejnižší pravděpodobností výskytu miny odhalit, a šlápl přitom na minu. Bylo vypozorováno, že veškeré neúspěšné hry jsou způsobeny právě zmíněným rozhodováním. Ve většině případech, je pro několik políček vypočítána stejná nejnižší pravděpodobnost. Program v tomto případě odhaluje první nalezené políčko s nejnižší pravděpodobností. Změna tohoto způsobu volby políčka s nejnižší pravděpodobností výskytu miny, by velmi pravděpodobně vedla ke změnám v úspěšnosti dohrání hry.

Hledání min					
Začátečník		Pokročilý		Expert	
čas (s)	✓/X	čas (s)	✓/X	čas (s)	✓/X
13,226	✓	54,358	✓	174,55	X
12,934	✓	68,246	✓	201,255	✓
14,174	✓	78,374	✓	156,655	✓
24,27	✓	72,13	✓	219,54	✓
21,746	✓	61,885	✓	162,218	✓
21,39	✓	70,649	✓	147,042	X
12,181	✓	49,243	✓	183,7	✓
17,412	✓	67,911	✓	160,47	✓
12,659	✓	63,057	✓	167,769	✓
17,791	✓	54,542	✓	155,55	X
17,916	✓	82,648	✓	160,02	✓
12,712	✓	64,463	✓	159,05	✓
19,908	✓	85,624	✓	159,279	✓
27,147	X	55,023	✓	162,618	✓
15,213	✓	63,84	X	202,276	X
14,396	✓	87,776	✓	186,193	✓
20,373	✓	63,205	✓	150,283	✓
16,003	✓	91,277	✓	173,221	✓
27,096	✓	71,76	✓	199,564	✓
11,412	✓	65,997	✓	190,156	✓
7,609	✓	70,662	X	188,469	✓
16,353	✓	80,894	✓	183,555	✓
22,158	✓	64,813	✓	225,796	✓
15,858	✓	66,101	✓	157,681	✓
8,067	✓	58,867	✓	148,521	✓
12,321	✓	56,455	✓	156,727	✓
11,354	✓	55,182	✓	168,24	✓
27,757	✓	58,939	✓	161,695	X
18,361	✓	77,822	✓	172,564	✓
9,173	✓	73,732	✓	198,248	✓
Průměr 16,63		Průměr 67,85		Průměr 174,87	

Tabulka 5.1: Statistika testování automatického hraní hry Hledání min.

5.2 Testování automatického hraní hry Solitaire Klondike

Testování automatického hraní hry Solitaire Klondike je poněkud náročnější, zejména časově náročnější. Při testování funkčnosti tohoto programu se v několika málo případech podařilo vyřešit hru do 20ti minut. Nejrychlejší naměřený čas byl 12 minut a 33 sekund. Problémem je, že při běhu programu uživatel nemůže používat počítač k jinému účelu než je pozorování hry, protože program aktivně využívá jak myš s klávesnicí, tak prostor celé obrazovky. S touto znalostí byl napsán skript, který bude spouštět hru 30 krát za sebou a sbírat statistiky, jež by prováděl testování přes noc, kdy není potřeba počítač využívat. Při takto prováděném testování přes noc byla naměřena nejdéle trvající úspěšná hra, která trvala 3 hodiny, 34 minut a 38 sekund, a teprve druhá iterace stále běžela a čas hry ukazoval více, než 5 hodin.

Důvod, proč program hraje hru tak dlouho je prostý. Stavový prostor této hry dokáže nabývat obrovských rozměrů. Uvažujme počáteční konfiguraci hry takovou, která bude umožňovat okamžité vykládání všech sloupců na vykládací hromádky foundations a po vyčerpání všech karet z tableu, bude možné postupně vykládat každou z karet kterou odhalíme z vrcholu zásobníku stock. Jedná se tedy snad o nejrychleji řešitelnou konfiguraci této hry, k jejíž řešení je zapotřebí provést 76 tahů (52 pro každou z vykládaných karet + 24 krát odhalení karty z vrcholu zásobníku). Takováto nejrychleji řešitelná konfigurace dokáže při dobře zvolené strategii vytvořit stavový prostor se zanořením 76ti uzlů + počáteční uzel, což je minimum prozkoumaných uzlů, které vedou k řešení této hry. Při nejdelším naměřeném úspěšném řešení, bylo prozkoumáno celkem 2945 uzlů, to je obrovský stavový prostor.

Dalším důvodem, proč program hraje hru tak dlouho je, že ne pro všechny konfigurace hry Solitaire Klondike existuje řešení.

ne pro všechny konfigurace hry Solitaire Klondike existuje řešení

Kapitola 6

Závěr

Cílem této práce bylo vytvořit program, který bude schopen automatického hraní her Solitaire Klondike a Hledání min.

Podařilo se mi vytvořit dva samostatné programy, přičemž jeden z nich je schopen hrát hru GNOME-Mines a druhý z nich je schopen hrát hru AisleRiot Solitaire. Existuje-li pro konkrétní konfiguraci hry řešení, programy jsou schopny toto řešení najít a hru úspěšně dohrát.

Před samotnou implementací programů pro automatické hraní obou her bylo potřeba podrobně nastudovat pravidla obou her, zaměřit se na zákonitosti vyplývající z těchto pravidel. Tato pravidla a zákonitosti byly podrobněji popsány v kapitole 2. Dalším zásadním krokem bylo provést analýzu a návrh budoucí implementace. V kapitole 3 byly popsány různé technologie a postupy, které mohou být vhodné pro implementaci jednotlivých aspektů obou her. Následně bylo v kapitole 4 popsáno několik zásadních částí vlastní implementace. V kapitole 5 byly diskutovány výsledky testování obou programů, bylo zde popsáno, jaké kroky vedly k neúspěchu ve hře Hledání min, také zde bylo odůvodněno množství času potřebné pro dohrání hry Solitaire Klondike.

Program pro automatické hraní hry Hledání min se podařilo vytvořit tak, že je schopen odehrát hru na úrovni Začátečníka průměrně za 17 sekund, na úrovni Pokročilého průměrně za 69 sekund a na úrovni Experta průměrně za 175 sekund. Na základě naměřených statistik lze tvrdit, že má program 91% úspěšnost. Nejkratší úspěšné dohrání hry Solitaire Klondike trvalo 12 minut a 33 sekund, zato nejdelší úspěšné dohrání hry trvalo 3 hodiny, 34 minut a 38 sekund.

Tato práce mi umožnila získat cenné zkušenosti napříč různými obory, jako jsou například umělá inteligence, nebo počítačové vidění. Práce na těchto programech a zejména analyzování zákonitostí, které vyplývají z pravidel her, mě upevnilo v rozhodování, kterým oblastem informačních technologií se chci v budoucnu blíže věnovat.

Závěrem mám hned několik nápadů, jak bych mohl na programech v budoucnu pokračovat. Každý z programů aktuálně funguje pro jednu verzi již existující hry. Mohlo by být zajímavé hlouběji zabřednout do oblasti počítačového vidění a naimplementovat dokonalejší metody skenování herních ploch, které by umožňovaly programům odehrát i jiné existující verze her. Další z možností, jak v práci pokračovat, by mohlo být využití strojového učení k tomu, aby byla analyzována aktuální strategie hraní hry Solitaire klondike. Na základě této analýzy by pak mohla dojít ke zjemňování heuristické funkce pro ohodnocování tahů, které by vedlo k efektivnějšímu řešení hry.

Literatura

[1] Bjarnason, R., Tadepalli, P. a Fern, A. Searching Solitaire in Real Time. *J. Int. Comput. Games Assoc.* 2007, sv. 30, s. 131–142. Dostupné z: https://web.engr.oregonstate.edu/~afern/papers/solitaire.pdf.

[2] Blake, C. a Gent, I. P. The Winnavility of Klondike Solitaire and Many Other Patience Games. *ArXiv preprint arXiv:1906.12314*. Červen 2019. Dostupné z: https://arxiv.org/abs/1906.12314v4.

[3] Kasík, P. Už 25 let mrhá celý svět časem: Microsoft nás naučil tahat karty myší. *IDNES.cz*. Květen 2015, [cit. 1.11.2022]. Dostupné z: https://www.idnes.cz/technet/technika/solitaire-hra-microsoft-windows-1990.A150521_152426_tec_technika_pka.

[4] Kayne, R. Minesweeper is NP-complete. *The Mathematical Inthelligencer*. 2000, sv. 22, č. 2, s. 9–15. Dostupné z: https://link.springer.com/content/pdf/10.1007/BF03025367.pdf.

[5] Longpré, L. a McKenzie, P. The Complexity of Solitaire. In: *Mathematical Foundations of Computer Science 2007*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 182–193. ISBN 978-3-540-74456-6.

[6] Nakov, P. a Wei, Z. Minesweeper, #Minesweeper. Duben 2003. Dostupné z: https://minesweepergame.com/math/minesweeper-minesweeper-2003.pdf.

[7] Parlett, D. Solitaire. *Encyclopedia Britannica*. Červenec 1998, [cit. 2.11.2022]. Dostupné z: https://www.britannica.com/topic/solitaire-card-game.

[8] Peña Castillo, L. a Wrobel, S. Learning Minesweeper with Multirelational Learning. In:. Leden 2003, s. 533–540. Dostupné z: https://www.researchgate.net/publication/220812024_Learning_Minesweeper_with_Multirelational_Learning.

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

[10] Yan, X., Diaconis, P., Rusmevichientong, P. a Roy, B. Solitaire: Man Versus Machine. MIT Press. 2004, sv. 17. Dostupné z: https://proceedings.neurips.cc/paper/2004/file/48c3ec5c3a93a9e294a8a6392ccedeb4-Paper.pdf.

Příloha A

Obsah přiloženého paměťového média

Na paměťovém médiu, které je součástí této práce, se nacházejí následující složky a soubory:

- **README.txt** - Obsahující pokyny pro zprovoznění programů
- **src** - Obsahuje zdrojové kódy
- **video** - Obsahuje krátké ukázky běhu obou programů
- **zprava.pdf** - Technická zpráva v PDF
- **zprava_src** - Složka obsahující zdrojové soubory technické zprávy