VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MULTIAGENTNÍ A OPTIMALIZAČNÍ METODY PRO HRY TYPU STEALTH

MULTI-AGENT AND OPTIMALISATION METHODS FOR STEALTH GAMES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN LÁNCOŠ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Láncoš Jan**
Program: Informační technologie
Název: **Multiagentní a optimalizační metody pro hry typu Stealth**
Multi-Agent and Optimalisation Methods for Stealth Games
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s hrami typu Stealth a identifikujte možnosti zlepšení chování autonomních jednotek v těchto hrách vzhledem ke svým cílům (dopadení hráče). Zaměřte se hlavně na metody používaní v agentních a multiagentních systémech a také na optimalizační metody.
2. Definujte cíle, události, role a scénáře v těchto hrách. Navrhněte jejich rozšíření o výše doporučené metody, které by zvýšili racionalitu chování autonomních jednotek jako jednotlivců, ale i jako celku.
3. Pro jednotlivé role figur ve hře, to znamená jak pro figuru, která je hledána, tak pro figury, které ji hledají, implementujte algoritmy řízení a ověřte jejich schopnost plnit zadané cíle.
4. Vyhodnoťte úspěšnost obou stran hry pro různé volené skupinové strategie na obou stranách a diskutujte zjištěné výsledky.

Literatura:

- Russel, S., Norvig, P.: Artificial Intelligence, A Modern Approach, Pearson, 2009

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

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

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Tato bakalářská práce se zabývá implementací chování oponentů ve videohrách žánru „stealth“. Klade důraz na jeho věrohodnost v porovnání s chováním skutečných lidí a na kvalitní herní zážitek. Práce popisuje využití algoritmu A* pro dynamické hledání cest ve dvourozměrném prostředí. Dále se zabývá systémem patrolování oponentů, jejich schopností detekovat přítomnost hráče a týmovou spoluprací při snaze o hráčovo dopadení s využitím vzájemné komunikace. V rámci práce byla dále v jazyce C++ vytvořena jedna hratelná úroveň předvádějící popsané chování v praxi. Práce může sloužit jako inspirace pro tvorbu vlastních rozhodovacích systémů do počítačových her podobného typu.

Abstract

This bachelor thesis deals with the opponents' behaviour in stealth-based video games. It's main focus is the credibility of said behaviour in comparison to the opponents' real life counterparts and the overall immersiveness of the experience. The thesis describes the usage of the A* algorithm for dynamic pathfinding in a two-dimensional space. Furthermore it describes the opponents' patrolling system, their ability to detect the player's presence and also their ability to cooperate and communicate while trying to chase the player down. One playable level demonstrating the described behaviour has also been created as part of this thesis using the C++ language. The thesis can be used as an inspiration for anyone interested in making their own intelligent systems for computer games of a similar type.

Klíčová slova

plížení, tajnost, hra, videohra, počítačová hra, navigace, a*, a s hvězdičkou, systém rozhodování, inteligentní systém, umělá inteligence, ui, detekce, spolupráce, nadbíhání, hierarchická navigace, pronásledování, agentní, multiagentní

Keywords

stealth, sneaking, silent, game, video game, computer game, navigation, pathfinding, a star, artificial intelligence, intelligent system, ai, detection, cooperation, flanking, hierarchical pathfinding, chase, agent, multiagent

Citace

LÁNCOŠ, Jan. *Multiagentní a optimalizační metody pro hry typu Stealth.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Multiagentní a optimalizační metody pro hry typu Stealth

Prohlášení

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

.....
Jan Láncoš
11. června 2020

Poděkování

Rád bych poděkoval panu doc. Ing. Františkovi Zbořilovi, Ph.D. za zkušené vedení a pomoc při psaní této práce.

Obsah

Kapitola 1

Úvod

1.1 Motivace

Tato práce se zabývá videohrami žánru stealth a chováním počítačem řízených oponentů, kteří se v nich obvykle vyskytují. Motivy těchto videoher bývají špionáž, plížení a riziko odhalení. Hráč zpravidla čelí přesile, která má za úkol jej odhalit a eliminovat. Chování oponentů je proto třeba vhodně definovat. Přestože v poslední době mnoho her různých žánrů využívá stealth prvky a žánr nabírá na popularitě, často se vývojářům nedaří zachytit to, co činí stealth žánr pro hráče atraktivním.

Na videohrách mě osobně fascinují právě inteligentní systémy řídící chování postav a iluze realismu, kterou jsou vývojáři schopni těmito systémy navodit. Má práce by měla demonstrovat důležitost a soběstačnost dobrého systému rozhodování ve videohře a zároveň poskytnout čtenáři přehled současných metod používaných k tvorbě her tohoto typu. Chování oponentů je ve stealth žánru jedním z nejdůležitějších faktorů určujících kvalitu herního zážitku, a tím i kvalitu celé videohry. Žánr jsem si zvolil pro demonstraci právě z tohoto důvodu.

Práce řeší problematiku definice vhodného chování oponentů, popisuje herním průmyslem nejčastěji aplikovaná řešení a následně rozebírá i jejich nedostatky. Práce dále demonstruje implementaci tohoto chování na jednoduché videohře s využitím poznatků získaných při jejím psaní. Ve svém závěru práce obsahuje zhodnocení dosažených výsledků a diskutuje možná rozšíření.

1.2 Obsah kapitol

Kapitola 2 je úvodní a zabývá se žánrem stealth v kontextu počítačových her. Zaměřuje se především na jeho nejvýraznější prvky a definuje typickou roli oponentů v těchto hrách. Dále pojednává o nejčastějších algoritmech využívaných k popisu chování těchto oponentů.

Kapitola 3 tyto algoritmy a postupy hodnotí primárně z hlediska herního zážitku, ale i výpočetní náročnosti, a identifikuje některé problémy.

V kapitole 4 je dále definována referenční hra, ke které se bude zbytek práce vztahovat, a je navrhnut konkrétní systém chování oponentů s ohledem na poznatky z předchozích kapitol.

Kapitola 5 pak popisuje konkrétní řešení a implementaci chování hlídačů. Popisuje způsob navigace oponentů, detekci hráče, vzájemnou komunikaci a způsoby pronásledování a patrolování.

Konečně kapitoly 6 a 7 shrnují nejzajímavější poznatky a problémy zjištěné v průběhu implementace. Obsahují také celkové zhodnocení vypracovaného řešení včetně jeho nedostatků i předností.

Kapitola 2

Žánr stealth z pohledu vývojáře

Slovo stealth se dá s pomocí slovníku volně přeložit jako „tajné jednání“. Jedná se o podžánr hlavních herních žánrů, jako jsou například akční hry, simulátory, adventury či hry typu hraní na hrdiny. Hlavní myšlenkou videoher tohoto žánru je tedy před oponenty utajit své jednání a vyhýbat se jim. Oponentů je zpravidla více a konfrontace s nimi vede většinou k prohře či k horšímu bodovému ohodnocení. Hry obvykle poskytují hráči více strategií a přístupů ke splnění cíle.

2.1 Žánr v kontextu počítačových her

Žánr prošel značným vývojem. Za počátky žánru se dá považovat hra Pac-Man (1980). Principem hry bylo vyhnout se konfrontaci se silnějšími a početnějšími oponenty. Žánr definující titul je však až Castle Wolfenstein (1981). Hra se odehrávala za druhé světové války a úkolem hráče bylo z nacisty obývaného hradu Wolfenstein ukrást klíčové dokumenty. Hráč měl mnoho možností, jakým způsobem hrad infiltrovat, včetně převleků. Tento velmi úspěšný titul položil základ žánru stealth.

Po Castle Wolfenstein vyšlo mnoho titulů využívajících principů této hry. Nejvýznamnější z nich jsou: Beyond Castle Wolfenstein (1983), Metal Gear (1987) and Metal Gear 2: Solid Snake (1990). Stealth žánr skutečně prorazil v roce 1998, kdy vyšly tři hry postavené na stealth mechanikách. Byly to: Tenchu: Stealth Assassins, Metal Gear Solid a Thief: The Dark Project. Každá přistupovala k plíživosti jinak a všechny byly velmi kladně přijaty.

Za hru, která žánru stealth vydobyla na trhu trvalou pozici, se z této trojice považuje Metal Gear Solid. Hra nabízela napínavý příběh s důrazem na plně dabované postavy, krásnými lokalitami a inovativní přístup ke stealth mechanikám. Ve hře se hráč pokouší infiltrovat teroristickou základnu s nukleárními zbraněmi. Hra kladla velký důraz na příběh a vtahující herní zážitek, což přispělo k jejímu důležitému postavení ve svém žánru.

Vývojáři využívaných mechanik začalo přibývat a už se nejednalo pouze o vyhýbání se konfrontaci. Důrazem na velký repertoár možností splnění úkolu se proslavila například videohra Hitman: Codename 47 (2000). Ke klasickému skrývání se ve stínech a za překážkami se postupně přidalo využívání zbraní či nástrojů určených k sabotáži, tichá eliminace oponentů, převleky, využívání výškových rozdílů, přemisťování předmětů včetně mrtvých těl eliminovaných soupeřů, ničení zdrojů světla a kamer, či splynutí s davem. K zajištění dostatečné výzvy byli vývojáři nuceni též rozšířit možnosti oponentů. Kromě vylepšení chování oponentů jako jednotlivců se začal klást vyšší důraz na spolupráci. Je-li hráč odhalen

jedním nepřítelem, náročnost úrovně se zvýší a pravděpodobně se na scéně objeví nepřátel více. Dokončení úrovně bez spuštění poplachu zpravidla vede k určité odměně hráče.

Stealth prvky či úrovně se dnes objevují prakticky ve všech velkých akčních titulech. Pokud stealth není hlavním obsahem hry, většinou jdou úrovně překonat i jiným způsobem. Příkladem her, které využily stealth mechanik jsou například Assassin's Creed (2007), či The Elder Scrolls: Skyrim (2011). Ačkoliv jsou oba velmi kladně přijaté tituly a ve mnoha ohledech jsou považovány za průlomové, jejich stealth sekvence jsou častým terčem kritiky. Kritizována byla například nemožnost zotavení se z odhalení hráče nepřáteli a tím i vynucený restart úrovně a přílišná nevnímavost oponentů. Kritizována byla také nevěrohodná komunikace mezi nepřáteli, kteří kolektivně přecházeli mezi stavy pronásledování a klidu, jakoby byli jedinou bytostí. Výbornou týmovou komunikací se naproti tomu proslavila hra F.E.A.R. (2005). Oponenti se vzájemně kryli, vysílačkou hlásili své úmysly a hráčovu lokaci, využívali svého okolí a hráči nadbíhali, aby ho dostali do křížové palby [4].

Tato sekce čerpá převážně z článku „The history and meaning behind the 'Stealth genre'" Muhammada Al-Kaisyho. [1].

2.2 Nejčastěji využívané metody

Za dobu existence žánru se některé metody užívaly častěji a některé jsou dodnes nasazovány bez výrazných změn. Následující oddíl shrnuje několik nejčastěji využívaných metod užívaných k definici chování hlídačů ve stealth hrách. Oponenti s úkolem detekovat a eliminovat hráče budou pro účely práce bez ohledu na to, čím v rámci hry jsou, nazýváni hlídači.

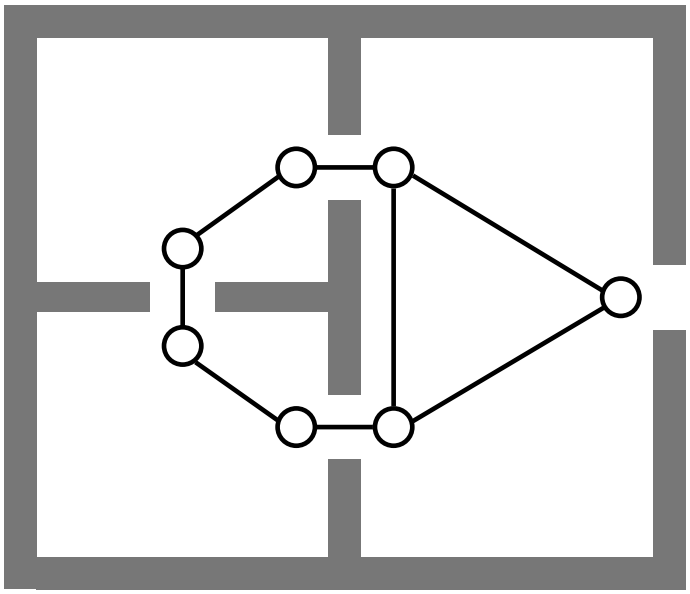
2.2.1 Navigace

Z hlediska pohybu mohou hlídači ve stealth hrách stát na místě, mohou přecházet mezi několika pevnými body, či se náhodně toulat po herní ploše. Většina stealth videoher při návrhu úrovní kombinuje tyto přístupy. Stojící hlídači slouží jako způsob odepření určitého prostoru hráči (tzv. „area denial"). Hlídači, kteří systematicky procházejí několik pozic, pak nutí hráče vytvořit si pozorováním z úkrytu určitou strategii a načasovat postup. Cyklický pohyb hlídačů je ve stealth žánru typický jev. Méně obvyklý, ale přesto využívaný, je náhodný pohyb hlídačů po mapě. Hlídačům je většinou také do jisté míry poskytnuta možnost hráče pronásledovat. Z toho důvodu musejí být schopni sejít z obvyklé trasy, mimo ni navigovat a následně se na ni i vrátit. Je proto téměř vždy potřeba implementovat schopnost vypočítat cestu, po které se mají vydat. K tomu lze využít více algoritmů, nejčastěji se však využívá metoda prohledávání stavového prostoru A*.

Metoda A*

Metoda A* je nejvyužívanější a nejznámější metoda pro řešení úloh prohledáváním stavového prostoru. Ve videohrách se využívá nejčastěji ke hledání cest mezi dvěma body, nejedná se však nutně o její jediné využití. Lze též využít k řešení logických úloh a ke hraní her celkově. Pokud má metoda A* sloužit pouze k navigaci v prostoru, je často vhodné herní mapu převést na vážený orientovaný graf (struktura skládající se z uzlů a hran o různých hodnotách, které uzly spojují). Způsobů reprezentace map grafem je více, ty nejvyužívanější jsou popsány níže v sekci 2.2.1 a vyobrazeny na obrázku 2.3.

Metoda směrem od zdroje systematicky prochází po hranách uzly grafu, dokud se nedostane k tomu cílovému, nebo dokud algoritmus není záměrně ukončen, například z důvodu

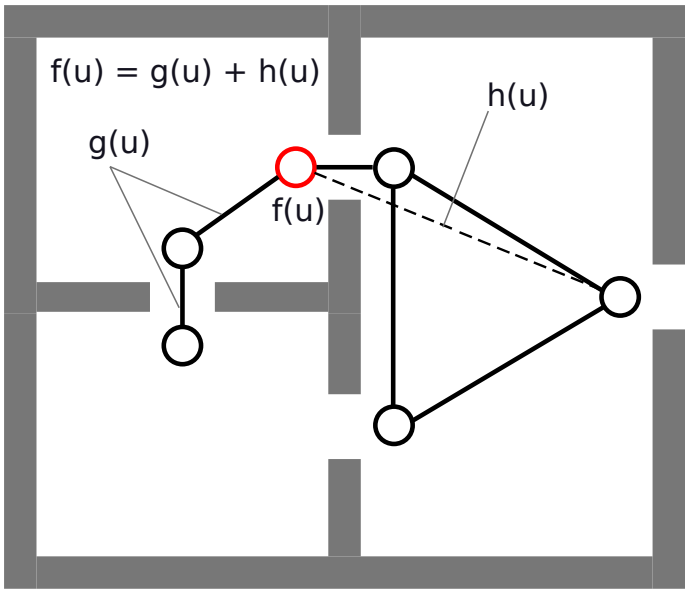


Obrázek 2.1: **Příklad reprezentace mapy grafem.** Zdi v tomto případě pro navigaci již nejsou relevantní a aktéři se mohou pohybovat po hranách mezi uzly.

přílišné výpočetní náročnosti. Výstupem metody je seznam uzlů vedoucích k cíli. Hrany v grafu reprezentují činy a uzly tím pádem stavy mezi nimi. Při navigaci pak hrany reprezentují přesun po přímé čáře o určité vzdálenosti a uzly pozice. Pořadí procházení uzlů se u různých metod prohledávání stavového prostoru liší, v našem případě je uzel k přednostnímu zpracování vybrán podle určitého ohodnocení. V případě metody A* jsou uzly ohodnoceny funkcí *f(u)* skládající se ze dvou dalších funkcí. Parametr *u* zde představuje zpracovávaný uzel. První z těchto funkcí je cena skutečně uražené cesty z původního uzlu do současného v jednotkách, které jsou při výpočtu relevantní. Tuto část představuje funkce *g(u)*. Při výpočtu cesty izomorfním prostředím se tedy jedná o součet vzdáleností mezi všemi procestovanými navazujícími uzly. Druhou částí je tzv. heuristická funkce, značená *h(u)*. Jedná se o odhad ceny cesty zbývající do cíle. Obě složky jsou vyobrazeny na obrázku 2.2.

A* je metoda úplná a optimální - garantuje tedy jednak nalezení cesty, a zároveň garantuje, že nalezená cesta je nejkratší možná. Aby metoda garantovala nalezení nejkratší možné cesty a tím pádem zůstala metodou A*, je potřeba použít takovou heuristickou funkci, která cenu cesty zbývající do cíle odhadne buď přesně, nebo ji podhodnotí. Při navigaci se jako heuristická funkce často používá výpočet vzdálenosti do cíle vzdušnou čarou - vzdálenost je tedy v nejlepším případě odhadnuta přesně, pokud k cíli lze dorazit po přímce, pravděpodobně je však podhodnocena. Jedná se o jednoduchou heuristickou funkci, která nevyužívá žádné další informace o prostoru. Lze navrhnout i jiné heuristické funkce, které zahrnují do výpočtu i prostředí, což poté vede k efektivnějšímu výpočtu. Cílem je taková heuristická funkce, která odhaduje cenu cesty zbývající do cíle co nejblíže té skutečné. Důvodem je zmenšení velikosti prostoru prohledávání a tím i doby výpočtu a paměťové náročnosti. Přesná heuristika by složitost algoritmu redukovala na lineární a prohledané uzly by se rovnaly výsledným. Mnohdy je však náročné vymyslet takovou heuristickou funkci, u které lze dokázat, že cestu nikdy nenadhodnotí.

Konkrétní implementace a přesný způsob využití algoritmu za účelem navigace jsou popsány v kapitole 5 v podsekci 5.3.1.



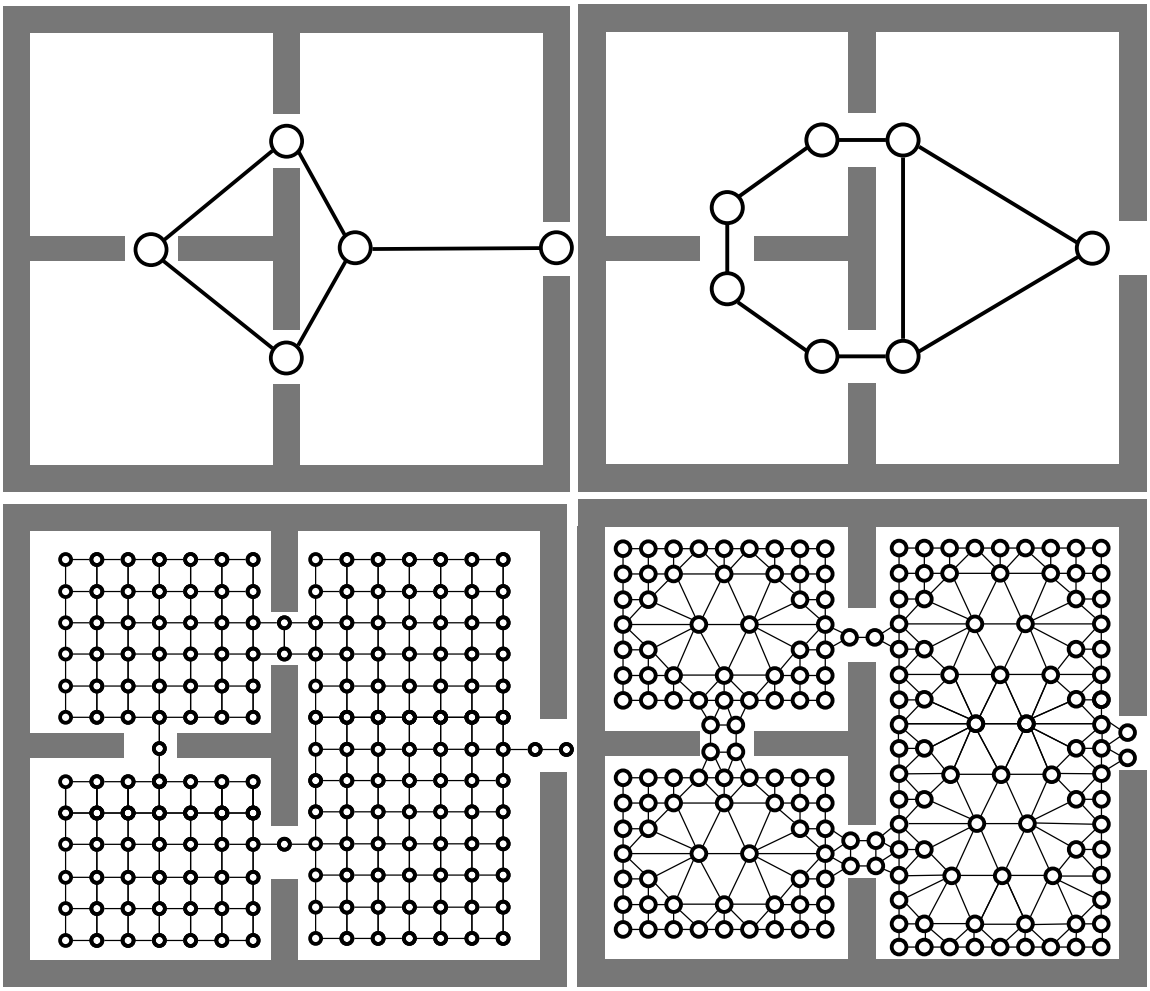
Obrázek 2.2: **Hodnotící složky uzlu.** Celkové ohodnocení uzlu funkce *f(u)* se skládá z délky doposud uražené cesty vyjádřené funkcí *g(u)*, a odhadu délky cesty zbývající do cíle, heuristické funkce značené *h(u)*, v tomto případě vzdálenosti do cíle přímou čarou. Patametr *u* všech funkcí představuje dotyčný uzel (zde vyznačen červeně). Uzel u dveří vpravo je cílovým uzlem.

Reprezentace herní plochy

Herní plochu lze převést na graf mnoha způsoby. Uzly lze určit na vhodných místech, jako jsou například dveře. Ty jdou reprezentovat uzly, ale i hranami. Veškerý přístupný prostor jde také vyplnit uzlovou mřížkou čtvercovou, trojúhelníkovou, či šestiúhelníkovou, případně lze použít jednu z variant fraktální mřížky. Fraktální mřížka má mnoho podob, jejich společným rysem je však vždy větší hustota uzlů poblíž překážek a na okrajích průchozího prostoru a naopak menší hustota uzlů ve volném prostranství. Příklady zmíněných reprezentací vyobrazuje ilustrace 2.3. Reprezentace mapy lze vytvořit manuálně i algoritmicky. Pro převod mapy na graf, který má přímo sloužit k navigaci, platí jediné pravidlo - je-li mezi dvěma uzly hrana, hlídač musí být schopen po hraně mezi nimi cestovat bez jakékoli kolize s překážkou.

Častým řešením navigace u větších herních projektů je tvorba tzv. navigační sítě (anglicky „navigation mesh“, často zkráceně jen „navmesh“) [6]. Hrací plocha je v takovém případě reprezentována nejčastěji vzájemně propojenými konvexními polygony, jak ukazuje obrázek 2.4. V rámci polygonu lze cestovat po přímých čarách, bez obav z kolize. K nalezení konkrétní cesty v rámci navigační sítě se využívá metoda „string-pulling“ [9] (volně přeloženo jako „tahání strun“). Navigační síť se málokdy tvoří manuálně, většinou je generována algoritmicky. Toto jsou však náročná řešení využívaná profesionálními vývojářskými studii[1] a proto se implementací těchto algoritmů práce dále nezabývá. Obvykle je navíc potřeba doplnit navigační systém o schopnost reagovat na dynamické překážky, což je další složitý separátní problém.

[1] Přednáška profesionálního vývojáře z Warhorse Studios popisující využití navmeshe, metodu string-pulling a řešení vyhýbání se dynamickým překážkám: https://www.youtube.com/watch?v=oux1p0R7Fm0



Obrázek 2.3: **Možnosti převodu mapy na graf.** Převod dveří na uzly, převod dveří na hrany, mřížkové vyplnění se čtvercovým základem, příklad fraktálního vyplnění. U fraktálního vyplnění si lze všimnout typicky větší hustoty uzlů poblíž překážek.

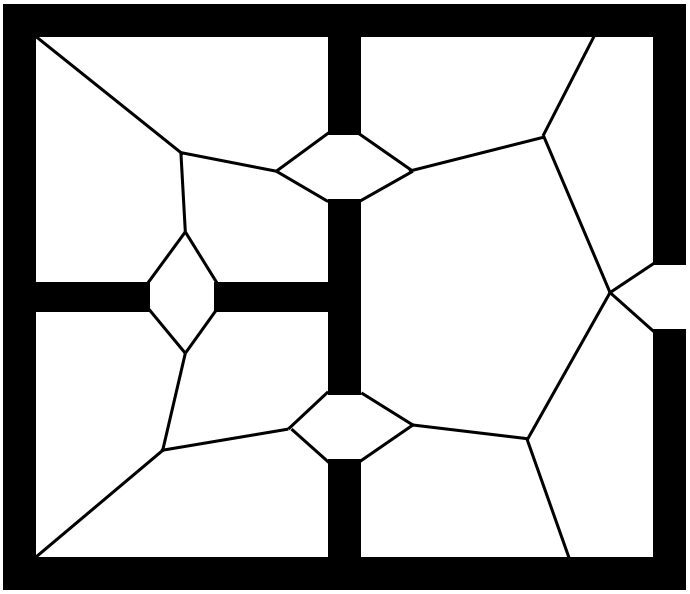
Primárním zdrojem pro celou sekci 2.2.1 je webová stránka Ajita Patela [5], která se v sérii „Amit's Thoughts on Pathfinding" dopodrobna zabývá těmito tématy.

2.2.2 Detekce

Hlídač schopen pohybu potřebuje hráče nějakým způsobem detekovat. Detekce hráče probíhá obvykle na třech úrovních. Hlídač může hráče vidět, slyšet, nebo se jej může dotýkat. Dále může reagovat v případě, že detekuje hráčovu činnost nepřímo, například zpozorováním přemístěných objektů či eliminovaných kolegů, nebo když mu je hráčova přítomnost kolegy nahlášena.

Vizuální detekce

Vizuální kontakt je zjištěn přepočítáváním přímé linie vidění (anglicky „line of sight", dále zkracováno LoS) [7]. Jedná se o pomyslnou úsečku mezi zdrojem vidění hlídače a jednoho či více bodů hráčova herního těla. Je-li úsečka nepřerušená, tzn. pokud neprotíná žádný



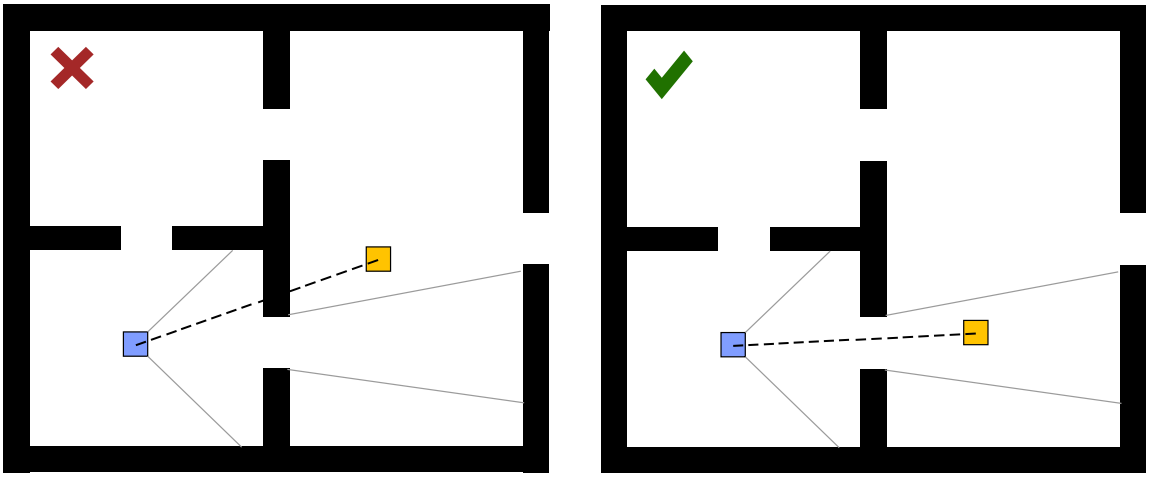
Obrázek 2.4: **Příklad jednoduché navigační sítě.** Všechny polygony musí být konvexní, čímž je zaručeno, že v nich lze cestovat po přímé čáře bez rizika kolize s překážkou.

neprůhledný objekt, hráč může být detekován. Většinou se popsaná metoda rozšiřuje definicí zorného úhlu (anglicky „field of vision“ nebo „field of view“, dále FoV) [8] hlídače. V tomto zorném úhlu se pozorovaný objekt musí nacházet, aby jej bylo možno testovat na LoS. Konkrétní rozměry a parametry FoV bývají velmi různorodé, obzvláště pak ve 3D hrách. Příklad využití principů LoS a FoV zobrazuje ilustrace 2.5.

Ke zmírnění dojmu bezchybného zraku a okamžitých reakcí hlídačů se většinou využívá určitá forma počítadla času, který hráč stráví pozorován - tedy kdy je současně v LoS a FoV. Jakmile počítadlo překročí určitou hranici, hráč je detekován a hlídač reaguje. Časová hranice detekce může záležet na mnoha různých faktorech jako je například osvětlení, s kolika body hráče lze navázat LoS a které to jsou, ve které části FoV se hráč nachází, vzdálenost mezi oběma aktéry, zda-li se hráč pohybuje nebo jestli je LoS v průběhu času opakovaně přerušována (například když se hráč pohybuje za řadou pravidelně rozmístěných objektů).

Zvuková detekce

Zvukovou detekci je zvykem počítat jako kompromis mezi vzdáleností hlídače od objektu a hlasitostí zvuku, který tento objekt vydal. Jen zřídka se implementují složitější simulace přenosu zvuku. Je možné šíření zvuku omezit pouze na místnost, kde se zdroj zvuku nachází, nebo intenzitu zvuku vhodně dělit za každý úsek zdi, který se nachází mezi zdrojem zvuku a hlídačem. Od povahy zvuku, jeho lokace, hlasitosti a vzdálenosti od hlídače se také odvíjí jeho reakce. Většinou se samotná reakce na zvuk nepočítá jako odhalení hráče, může však vést ke kontaktování ostatních hlídačů a celkovému zvýšení aktivity. V rámci zajímavého designu úrovní se většinou implementuje zvukový projev hlavně u pokusů o eliminaci oponentů, či jiných akcí s vysokým rizikem a zároveň vysokou potenciální odměnou. Obdobně se simulace sluchu často využívá při komunikaci mezi hlídači.



Obrázek 2.5: **LoS a FoV.** První ilustrace vyobrazuje situaci, kdy se hráč (žlutý čtverec) sice nachází v FoV hlídače (modrý čtverec), avšak nelze s ním navázat LoS, proto není vidět. Druhá ilustrace zobrazuje splnění obou podmínek a hráč je proto hlídačem spatřen.

Dotyk

Detekce dotyku je obvykle pouze zjištění potenciální kolize mezi dvěma objekty představující části hráče a hlídače. Je tedy součástí kolizního systému, který práce nepopisuje. V principu jde však o zachycení okamžiku, kdy kolizní systém zabránil jednomu aktérovi v pohybu na místo, kde se nachází jiný aktér nebo objekt. K překrytí aktérů tak nikdy nedojde, avšak příznak dotyku je nastaven.

2.2.3 Komunikace

Komunikace je v principu pouze výměna informací, které mají hlídači k dispozici. Určitá komunikace mezi hlídači a schopnost přivolat posily je standardní způsob, jak docílit dojmu přesily, a ve stealth hrách se vyskytuje velmi často.

Jedním z možných přístupů jak komunikaci simulovat, ale ve skutečnosti ji neimplementovat, je jednoduché přivolání posil formou tvorby nových, již alarmovaných nepřátel, kteří se na herním poli původně nenacházeli a kteří mezi sebou dále nekomunikují.

Skutečná komunikace pak spočívá ve schopnosti oponentů sdílet mezi sebou klíčové informace o hráčově pozici, jeho pravděodobných úmyslech či stavu. Tato schopnost je mnohdy omezena okruhem, v němž se hlídači navzájem slyší. V případně simulace komunikace vysílačkami nebo jinými dálkovými komunikátory vzdálenost nehraje roli.

Kapitola 3

Zhodnocení nejčastěji využívaných metod

3.1 Vliv umělé inteligence na hráčský zážitek

Inteligentní systémy jsou součástí hry jako jeden z hráčů. Počítač hrající hru má dva hlavní cíle - hrát ji racionálně, a hrát ji uvěřitelně [13]. Jeho schopností hrát jako nehráčský charakter (anglicky „non-player-character“, dále zkracováno NPC) lze upravovat dle potřeby obtížnost hry. Pokud je interakce s NPC představujícími lidské postavy klíčovou částí hry, schopnost hrát dobře a zároveň zachovávat dojem reality při vykazování chování podobného lidskému přispívá k lepšímu pohlcení hráče a lepšímu hernímu zážitku celkově. Chová-li se NPC uvěřitelně, hráč si k němu s větší pravděpodobností vytvoří lepší vztah.

Hráči téměř vždy preferují možnost hrát proti skutečným lidem, pokud hra nabízí tuto možnost. Dochází k tomu i přesto, že v mnoha případech je počítač schopen simulovat vyváženějšího nebo naopak silnějšího oponenta, pokud o to má hráč zájem. Důvodem jsou sociální vazby, které mezi sebou v průběhu hry protihráči navazují. Pozitivní pocity z překonání živého soupeře jsou mocnější, než překonání algoritmu, který prohru nijak nepociťuje, ani neprojevuje. Výjimkou jsou atmosferické hry a simulátory silně zaměřené na navození dojmu skutečného, žijícího světa, jako jsou hry typu hraní-na-hrdiny (anglicky „role-playing games“, častěji zkracováno jako RPG). Málokdy jsou totiž hráči schopni se ve hře chovat jako skuteční lidé a mají tendenci bořit realitu divokým a zbrklým chováním, za které ve hře na rozdíl od skutečného světa nehrozí vážné následky. Tyto hry jsou proto ve většině případů stále tvořeny pouze pro jednoho lidského hráče - o to více však musí vynahrazovat lidskou potřebu sociálních pout. Navození iluze žijícího světa obývaného skutečnými inteligentními bytostmi je pro ně tedy klíčové. RPG hry se obvykle zaměřují méně na projevy inteligence NPC pohybem a činy, zato více na rozsáhlé dialogy s hráčem a možné dynamické vztahy, které mezi sebou mohou navázat.

Není tedy jedinou starostí vývojáře vytvořit dobře hrajícího umělého hráče, ale také pokud možno vytvořit tomuto NPC osobitost formou emocionálních projevů. Hlídač, který si povídá pro sebe, píská si, či nahlas komunikuje se svými kolegy, bude vždy působit na hráče lidštěji a jeho překonání doprovázené patřičným emocionálním projevem bude pro něj mnohem cennější. Ani jeho schopnost hru hrát nesmí překonat hranici uvěřitelnosti. Proto je dobré, aby NPC nepodváděl, tzn. pokud možno využíval skutečně pouze těch informací a schopností, které by na jeho místě měl sám hráč.

Je vhodné vyhnout se i pouhému dojmu, že počítač podvádí. Pokud hlídač spatří hráče a nahlásí to kolegovi, který jej tiše obejde a následně zezadu zlikviduje, hráč z celé složité situace vidí pouze nečekané oznámení o prohře a jeho motivace ve hře pokračovat klesá. Hlasitá komunikace, které je hráč pasivně součástí, při které hlídači hlásí své záměry, tento problém řeší. Může sice vést k celkovému snížení obtížnosti hry, hráč se však méně často dostane do situace, v níž neví, proč prohrál. Obtížnost jde navíc vyvážit zase jiným způsobem. Hráč si také uvědomuje důvody stojící za rozhodnutími jeho oponentů a jejich schopnost hru hrát. Pokud hráč nemá možnost ocenit inteligenci svých umělých oponentů, nedává velký smysl na ní ztrácet čas.

Sekce 3.1 čerpá primárně z prací Making the Human Care: On Building Engaging Bots [16] a Towards Implementation of Social Interaction [15].

3.2 Problémy současných řešení

S poznatky z předchozí sekce se nyní dají zhodnotit metody popsané v kapitole 3.

3.2.1 Pohyb hlídačů

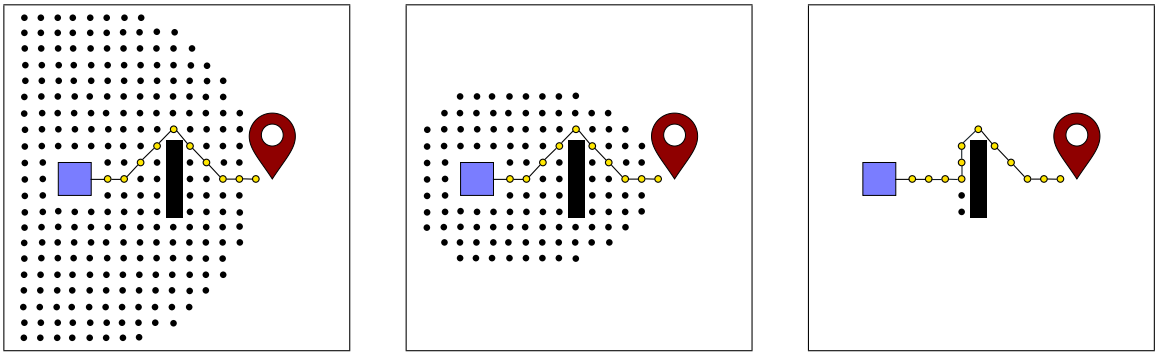
Pohyb umělého hráče by měl být takový, aby umožňoval lidskému protihráči tvorbu strategie. Čistě náhodný pohyb je tedy ve většině případů nevhodný. Hráč nemá možnost naplánovat postup, pokud neví, jak se hlídač zachová příště. Stealth hra v takovém případě ztrácí smysl a stává se z ní hra, v níž hrají nejvyšší roli náhoda a postřeh. Při definici patrolování hlídačů je vhodné pokrytí hlídané plochy ideálně takovým způsobem, aby hráč měl možnost pohybové vzorce jednotlivých hlídačů pochopit. Cyklický pohyb mezi několika klíčovými body tento cíl splňuje. Čistě statický hlídač hru příliš neobohatí a slouží tedy pouze jako nepřekonatelná překážka. Cyklický pohyb však lze s přestávkami, v nichž se hlídač na moment mění ve statického, snadno zkombinovat. Hlídač navíc působí lidštěji, když nezůstává neustále jen v pohybu. Zastávky mohou být provedeny na místech, na kterých to z hlediska herního světa dává smysl, případně se hlídač může rozhlížet, nebo komunikovat.

Alternativy k metodě A*

Metoda A* využívaná k nalezení cesty mezi dvěma body je doposud nepřekonaným kompromisem mezi snahou minimalizovat výpočetní náročnost a zároveň vypočítat skutečně nejkratší dostupnou cestu. Nalezení nejkratší cesty však také nemusí být pro herního vývojáře primární podmínkou. Pokud je NPC schopen přesunout se do cíle dostatečně rychle a po cestě, která dává z hlediska herního světa smysl, je většinou daleko větším problémem výpočetní náročnost. Je tedy vhodné zamyslet se nad složitějšími heuristikami, než je pouze vzdálenost vzdušnou čarou, případně lze algoritmus A* upravit změnou poměru, v němž jsou obě složky ohodnocení jednotlivých uzlů. Přidáním větší váhy výsledku heuristické funkce se algoritmus přiblíží variantě metody hladový algoritmus (anglicky „Greedy Search“ [14]). Již negarantuje nejkratší možnou cestu, výpočetní náročnost je však vždy nižší. Vhodný poměr složek pro hru lze pak zjistit experimentálně. Přiřazením větší váhy složce představující uraženou cestu nelze získat kratší cesta, než kterou garantuje algoritmus A* při poměru 1:1 a pouze tím pádem narůstá paměťová náročnost a doba výpočtu [11]. Algoritmus tímto způsobem degenerujeme na šířkové prohledávání s respektováním cen přechodů (anglicky „uniform cost search“, zkracováno UCS [14]). Manipulace vah tímto způsobem je tedy zbytečná. Demonstrace rozdílů v množství stavů při různých formách

manipulace se složkami vyobrazuje ilustrace 3.1. Pokud je rychlost výpočtu pro vývojáře důležitější, než garance nejkratší cesty, a nechce vymýšlet lepší heuristickou funkci, může se uchýlit k tomuto řešení. Vzhledem k charakteru trasy, kterou vrací hladový algoritmus, lze tímto způsobem například vhodně simulovat pohyb hlídačů v neznámém prostředí.

Na výpočetní náročnost však může mít mnohem větší dopad způsob grafové reprezentace herní plochy.

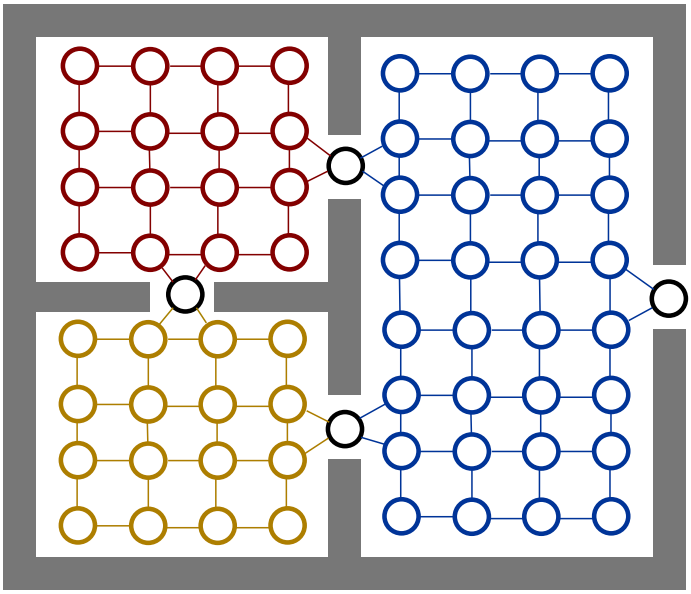


Obrázek 3.1: **A* při extrémní manipulaci vah jednotlivých složek.** První obrázek zobrazuje stavovou explozi při hodnocení uzlů pouze cenou uražené cesty (vzniká algoritmus nazývaný Uniform Cost Search). Druhý obrázek zobrazuje složky A* ve správném poměru 1:1. Třetí zobrazuje využití pouze heuristické funkce (Greedy Search). Je vidět, že v tomto případě již nalezená cesta není nejkratší. V obou předchozích případech nalezená cesta nejkratší je, algoritmus A* ve správném poměru však k nalezení cesty dospěl podstatně rychleji.

Počet uzlů grafu není při převodu mapy na graf nijak omezen, hustá mřížková reprezentace se proto jeví jako nejvhodnější, protože mapu reprezentuje nejdetailněji. Výpočetní doba algoritmu A* však při velkém počtu uzlů začne velmi rychle narůstat, a je-li mřížkou vyplněn prázdný prostor, dochází k velkému počtu výpočtů zcela zbytečně. Jako lepší řešení se tedy nabízí spíše fraktální mřížka. Stísněné prostory, kde je potřeba pokrýt více okrajových bodů, jsou uzly vyplněny více, větší a volné prostory naopak méně.

Manuální definice grafu je časově náročný proces a zvlášť u větších map se tedy nepoužívá. Aplikovat lze však hierarchický přístup [6]. Mapa se v takovém případě rozdělí na úseky, které spojuje pokud možno co nejmenší množství hran. Při výpočtu cesty se pak využije metoda A* nejprve na nejvyšší úrovni, kde vybere sekvenci úseků, která vede k cíli, a poté se nad každým úsekem spustí metoda znovu, aby nalezla cestu v jeho rámci. Tento přístup lze kombinovat s libovolným dalším přístupem na nižších úrovních. Pro dobrou účinnost tohoto přístupu je potřeba minimalizovat počet hran spojujících úseky. Vhodná implementace hierarchického hledání cest může dobu výpočtu výrazně zrychlit [12]. Příklad hierarchického rozdělení mapy zobrazuje ilustrace 3.2.

V případě více pohyblivých charakterů na mapě, kteří jsou schopni vzájemné kolize, je zároveň potřeba počítat se schopností vyhýbání se dynamickým překážkám. Tohle je však náročný problém a pro účely práce není příliš důležitý, proto se jím bude při implementaci zabývat pouze okrajově. V praxi však může ovlivnit volbu způsobu reprezentace mapy i způsob navigace postav celkově.



Obrázek 3.2: **Hierarchické rozdělení mapy pro dvouúrovňové nasazení algoritmu A*.** Černé uzly umístěné ve dveřích představují vyšší navigační úroveň. Barevně odlišeny jsou uzly náležící do jednotlivých místností. Výpočet cesty pak probíhá po částech a černé uzly působí jako mosty mezi místnostmi.

3.2.2 Chování hlídačů

V otázce vizuální detekce je zajímavým příkladem hra Half-Life 2 (2004). K typickému řešení s využitím LoS a FoV byla přidána podmínka, že hráč není detekován až do okamžiku, kdy se v jeho vlastním FoV nachází sám nepřítel [2]. V praxi tím byly eliminovány potenciálně nežádoucí útoky ze zálohy, kdy si hráč nepřítele nemohl všimnout, a současně tím byl snížen počet okamžiků, kdy hráče hra trestala za něco, čemu on nerozuměl. Dalším častým problémem při detekci je pak přílišná ostrozrakost, nebo naopak nevnímavost hlídačů. Obojí záleží pouze na balančních schopnostech vývojáře a testování vhodného nastavení úhlů, dosahu a detekčních časovačů.

Risk je důležitou součástí videoher a měl by vyvažovat jakoukoli akci s potenciálně velkou odměnou. Stealth hra potřebuje hráče odradit od přehnaného riskování. Přinejmenším v rámci úrovně je proto vhodné, aby hlídači disponovali nějakou pamětí. Pokud se okamžitě po ztracení stopy hráče vrátí do klidového stavu přesto, že míjí své zlikvidované kolegy, riskantní akce ztratí svoji váhu.

Častým problémem NPC charakterů jsou také okamžité robotické reakce. Zpracování informace člověku chvíli trvá a stejně tak její předání. Pokud při detekci jedním hlídačem hráč skrze něj instantně upozorní na svoji pozici i všechny ostatní, pravděpodobně si bude připadat podvedený. Stejně tak, pokud po hráči hlídač vypálí bez jakéhokoli zpoždění v okamžiku, kdy jej spatří. Při simulaci lidských oponentů je očekáváno lidské chování a komunikace se všemi nedostatky, chceme-li navodit sociální dynamiku popsanou v sekci 3.1.

Kapitola 4

Návrh inteligentního systému jednoduché stealth hry

V rámci práce bude vyvinuta jednoduchá stealth hra demonstrující pokud možno co nejvhodnější kombinaci výše zmíněných metod odpovídajících rozsahu projektu. Důraz bude kladen v první řadě na kvalitní herní zážitek a schopnost hlídačů efektivně jednat a polapit hráče. Druhotně bude hodnocena i výpočetní náročnost implementovaného řešení.

Hra bude napsána v jazyce C++17[1] s využitím knihovny SDL 2.0[2] pro snadnější přístup k periferiím. Kapitola 5 se bude zabývat pouze implementací algoritmů souvisejících s chováním hlídačů. Prostředí hry popsané v této kapitole se tedy v implementační kapitole 5 považuje za hotové, stejně jako kolizní systém aktérů a schopnost cestovat po libovolně nakloněné úsečce.

4.1 Podmínky vítězství a ovládání

Tři ze čtyř charakterů budou NPC hlídači, čtvrtý bude ovládaný hráčem. Ke splnění úrovně se hráč musí dostat na určitou pozici na vzdáleném okraji herní plochy, stiskem klávesy `T` na této pozici splnit podmínku pro návrat a následně se dostat zpět na výchozí pozici. Prohrou hra skončí v okamžiku, kdy hráč stráví příliš dlouhou dobu v blízkosti a v LoS/FoV kteréhokoli z hlídačů.

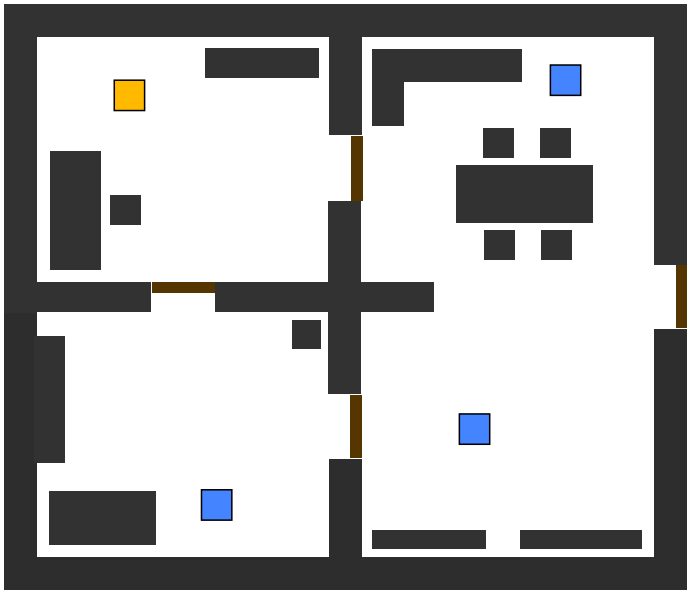
Hráč i hlídači jsou reprezentováni každý svým pohyblivým čtvercem s možností přesunu po libovolně nakloněné úsečce konstantní rychlostí na zadaný bod. Hráč je kvůli ovládání klávesami omezen pouze na pohyb osmi standardními směry. Pohybovat se může stiskem kurzorových kláves. Má možnost se skrčit podržením klávesy `CONTROL` a sprintovat podržením klávesy `SHIFT` při pohybu. Při skrčení je do výpočtu LoS zahrnuto více překážek, hráč má tedy větší možnost se skrýt. Rychlost pohybu je při skrčení snížena.

4.2 Popis prostředí

Prostředí hry bude dvourozměrné, pozorované z ptačí perspektivy. Herní plochou bude uzavřený půdorys jednoho patra budovy, spolu s nábytkem a dveřmi. Všechny předměty včetně zdí a charakterů budou reprezentovány pevnými, neprostupnými čtyřúhelníky. Příklad zjednodušené herní plochy je vidět na obrázku 4.1.

[1] Multiparadigmatický programovací jazyk, vznikl jako rozšíření jazyka C. Více na: https://isocpp.org/

[2] Simple DirectMedia Layer pro snadný přístup k periferiím. Více na: https://www.libsdl.org/



Obrázek 4.1: **Příklad herní plochy o třech místnostech.** Modré čtverce symbolizují hlídače, žlutý hráče. Hnědé obdélníky symbolizují dveřní křídla a jsou průchozí. Je-li blízko nich hlídač s úmyslem projít skrz, dveřní křídla dočasně zmizí, hráč je může otevřít stiskem příslušné klávesy. Vše ostatní jsou nepohyblivé, neprůchozí překážky.

4.2.1 Místnosti

Herní prostředí bude jako skutečný půdorys budovy rozdělen na místnosti, ty sousedící mohou být vzájemně propojeny dveřmi. Místnost je definována jako obdélník, na jehož ploše se nachází libovolný počet neprostupných obdélníků simulujících nábytek. Obvod místnosti zevnitř lemují též neprostupné obdélníky simulující zdi. Jednu zeď může tvořit více obdélníků, mezery mezi nimi mohou tvořit dveře. Zdi místnosti mají pouze poloviční tloušťku výsledných zdí. Ty vznikají, pokud spolu sousedí dvě místnosti.

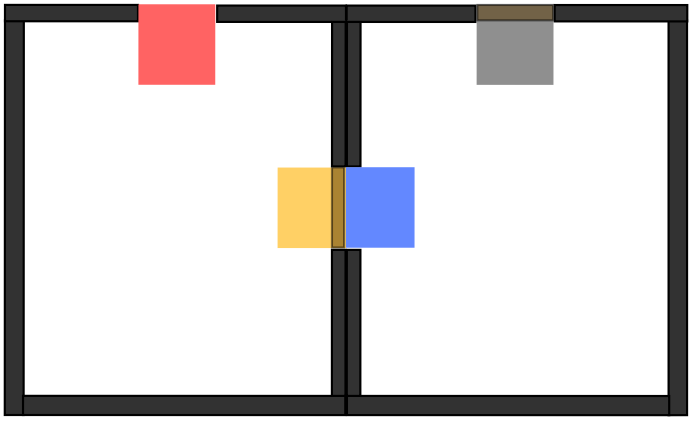
Jednotlivé objekty v místnosti obsahují údaj o své výšce. Jedná se o desetinné číslo v rozsahu $0,0 - 1,0$. Předměty s větší výškou než je $0,7$ skryjí hráče v každém případě, předměty s výškou nižší než $0,3$ jej neskryjí nikdy. Předměty s výškou ležící v intervalu $0,3 - 0,7$ skryjí hráče pouze v případě, že je skrčen.

4.2.2 Dveře

Dveře tvoří dvě vzájemně provázané instance struktury představující polovinu dveřního koridoru. Každá tato polovina náleží jedné ze dvou místností, které dveře spojují. Struktura obsahuje:

- souřadnice navigačního uzlu této poloviny dveří,
- zda tato polovina dveří obsahuje obdélník symbolizující dveřní křídlo,
- zda jsou dveře otevřené, či zavřené, tedy jestli je dveřní křídlo viditelné a průchozí, nebo naopak (tato část se vždy u obou instancí shoduje),
- ukazatel na druhou polovinu dveří.

Tato polovina dveřního koridoru musí být umístěna v mezeře mezi dvěma obdélníky představující části jedné zdi. Musí mít současně korespondující polovinu koridoru v sousední místnosti, a musejí stát přesně proti sobě. Vždy pouze jedny z této dvojice koridorů obsahují dveřní křídlo. Jeden dveřní koridor je tedy tvořen dvěma vzájemně propojenými instancemi struktury reprezentující dveře. Příklad dveřního koridoru zobrazuje ilustrace 4.2.



Obrázek 4.2: **Dveřní koridory dvou místností.** Dveřní koridory místností musejí stát proti sobě a jsou vzájemně provázány. Vždy pouze jeden ze dvojice provázaných koridorů obsahuje dveřní křídlo. Na obrázku jsou barevně odlišeny čtyři dveřní koridory, žlutý a modrý společně tvořící jeden průchod, kdy žlutý obsahuje dveřní křídlo. Aby červený a šedý byly validní, musely by mít korespondující sousedy.

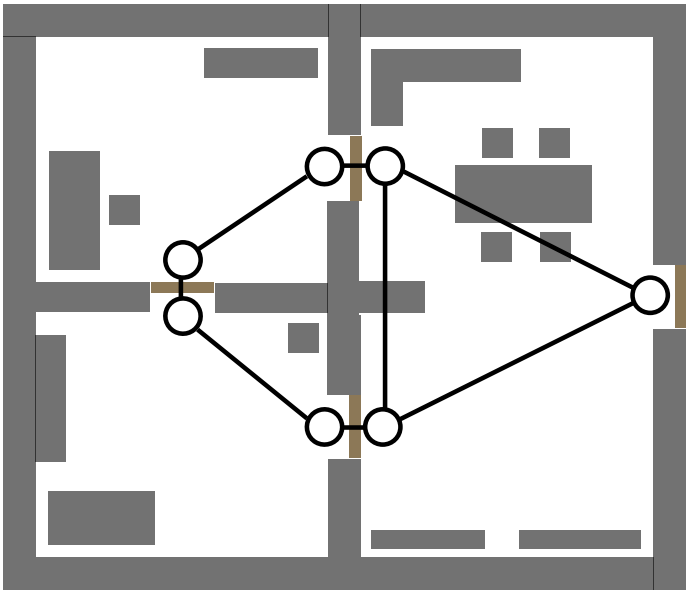
4.3 Navigace hlídačů

Hlídači jsou schopni přesunu po libovolně nakloněné úsečce a mají definovaný kolizní systém zajišťující, že pokud se při přesunu nachází v cestě překážka bránící jim v posunu po jedné z os, tento posun se neuskuteční. Posun po druhé ose je však neovlivněn. Pokud překážka brání v pohybu po obou osách, hlídač se nepohybuje. Vlastní navigace hlídačů bude probíhat na dvou úrovních, v obou dvou však bude nasazena metoda A* (dříve popsaný hierarchický přístup). Vyšší úroveň má pouze za úkol zjistit, která sekvence dveří vede od hlídače do místnosti, ve které se nachází jeho cíl.

4.3.1 Reprezentace mapy

Mapa bude pro tento účel převedena na graf následujícím způsobem: všechny dveře budou reprezentovány jako dva samostatné uzly. Každý z této dvojice uzlů se nachází v jedné z místností, které dveře spojují, a oba jsou propojeny hranou. Vzdálenost mezi nimi bude zanedbatelná a nebude mezi nimi přípustná jiná překážka, než samotný obdélník reprezentující dveřní křídlo. Bude se tedy vycházet z toho, že mezi nimi není třeba cestu hledat a vzdálenost stačí přímou čarou překonat. Všechny uzly dveří nacházející se v jedné místnosti jsou v rámci vyšší úrovně A* také propojeny hranou, jak ukazuje obrázek 4.3.

K navigaci mezi dvěma body, které se nacházejí v jedné místnosti, bude využit mřížkový přístup s tím rozdílem, že mřížka nebude stálá vůči mapě, ale vůči hlídači. Při výpočtu cesty v kterémkoli okamžiku je tedy hlídač zdrojem mřížky, a uzly se teoreticky mohou v průběhu hry nacházet ve všech možných validních pozicích dané místnosti. Popsaný jev



Obrázek 4.3: **Reprezentace vyšší úrovně mapy.** Pro určení sekvence dveří vedoucích do místnosti obsahující hlídačův cíl bude mapa převedena na graf konvertováním dveřních koridorů na hrany. Pokud vzniklé uzly náleží do jedné místnosti, jsou též vzájemně propojeny hranami.

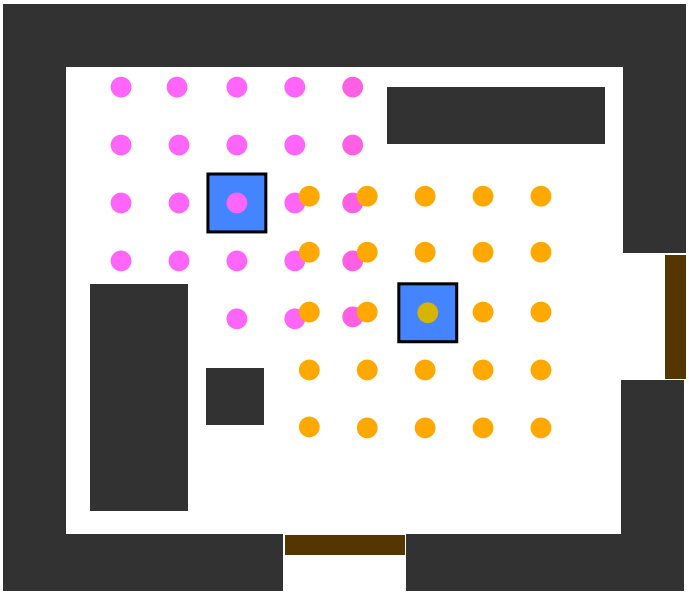
znázorňuje ilustrace 4.4, která zobrazuje dva hlídače a část jejich relativních navigačních mřížek. Vhodná velikost kroku bude zvolena experimentálně. Dosáhne-li prohledávání uzlu, jehož vzdálenost k cíli bude menší, než daný krok, byla cesta nalezena. Cíl je poté zařazen na konec výsledného seznamu uzlů. Generování uzlů a detaily algoritmu jsou popsány v kapitole 5. Výše popsaný způsob ukotvení mřížky je autorské řešení. Měl by poskytovat kvalitní pokrytí mapy za cenu přijatelné výpočetní náročnosti. Toto tvrzení bude v rámci práce ověřeno.

4.4 Chování hlídačů

Hlídači budou tři s možností dálkové komunikace, které se pasivně účastní i hráč. Hra tím simuluje hráčem odchytávanou rádiovou komunikaci mezi hlídači. Herní plocha bude manuálně rozdělena na tři shluky místností, z nichž každý bude mít na starost jeden z hlídačů.

4.4.1 Patrolování

Každý z hlídačů bude mít za úkol cyklicky patrolovat v jednom ze tří shluků. Budou tedy v pevném pořadí procházet svůj seznam místností a v každé z těchto místností s výjimkou místností předem označených jako chodba se zastaví a rozhlédnou. Pořadí místností bude též určeno manuálně. Způsob patrolování se liší v okamžiku, kdy jsou hlídači v klidovém stavu, oproti stavu pohotovosti po zpozorování hráče. Není-li důvod pro podezření, hlídači se pohybují pomaleji a do místností, které nejsou průchozí, pouze nahlížejí, tzn. nikdy neopustí dveřní prostor a dveře za sebou nezavřou. V případě stavu pohotovosti, kdy hráč není v dohledu a zároveň se nepodařilo jej nalézt na očekávaných místech, procházejí hlídači



Obrázek 4.4: **Uzlová mřížka s ukotvením v centru hlídače.** Na obrázku jsou vyobrazeni dva hlídači, každý s příkladem své relativní mřížky. Mřížka vůči svému okolí není stálá a záleží pouze na přesné lokaci hlídače v okamžiku generování. Uzly tedy mohou teoreticky nabýt všech možných pozic v dostupném prostoru.

místnost důsledně. V praxi to znamená, že vejdou hlouběji do místnosti, zavřou za sebou dveře a prohlédnou místnost z více úhlů tak, aby zkontrolovali pokud možno maximum dostupného prostoru. V průběhu procesu patrolování jsou hlídači připraveni reagovat na případné nahlášení hráčovy polohy svými kolegy. Jsou také připraveni reagovat na hráče přímo.

4.4.2 Detekce

Hlídači mohou hráče s jistotou detekovat pohledem a dotykem. Zvuk bude ze hry vynechán z důvodu nedostatku zkušeností s implementací zvukových efektů, kvůli čemuž by hráč neměl zpětnou vazbu ohledně hlasitosti svých akcí. K vizuální detekci bude využita obvyklá metoda LoS v kombinaci s FoV. LoS bude definována jako překážkou nepřerušená linka vedená ze středu hlídačova kolizního čtverce do středu hráčova kolizního čtverce. Překážky mají různé výšky a hráč má možnost se za ně skrčit. K vizuální detekci dochází až po určité době strávené nepřerušeně v LoS a FoV. Dotyk, tedy potenciální kolize mezi hráčem a hlídačem, vede k okamžité detekci.

4.4.3 Pronásledování

Po detekci hráče hlídač přechází do stavu pronásledování, ve kterém s určitým zpožděním nahlásí kolegům jeho pozici. Upozornění hlídači v závislosti na své poloze a cílové místnosti reagují různým způsobem: nachází-li se hráč v průchozí místnosti, pokoušejí se mu vhodně nadběhnout. Pokud se hráč nachází v místnosti s jedinými dveřmi, zaujmou pozici přede dveřmi řečené místnosti a tyto dveře hlídají. V případě, že hráče vidí, přidávají se k pronásledování. Cílem hlídačů není hráčova pozice přímo, snaží se pouze zaujmout pozici dostatečně blízko na to, aby spustili odpočet konce hry.

Ztratí-li hlídač hráče z dohledu, jde na jeho poslední známou pozici. V případě, že tato pozice je ve dveřích, projde jimi. Místnost, v níž se nyní nachází, důkladně prohledá a pokud v ní existují dveře vedoucí do slepé místnosti, prohledá ji také. Najde-li hráče, pokračuje v pronásledování. V případě, že jej nenajde, vrací se k patrolování ve svém sektoru, tentokrát však již ve stavu pohotovosti. Příloha A obsahuje tři diagramy, které společně definují hlídačovo chování v souladu s předchozím popisem. Každý z diagramů popisuje jeden z hlídačových záměrů a způsob, kterým se snaží dosáhnout svého současného cíl. Společně popisují vyšší úroveň jeho rozhodování, tedy způsob tvorby nových cílů.

O komunikaci budou hlídači informovat hráče vizuálně, zobrazením symbolu příslušné plánované akce v blízkosti dotyčného hlídače. Mohou hráče informovat o detekci zobrazením slova „Hey!“ a o přivolání posil zobrazením symbolu vykřičníku.

Kapitola 5

Implementace specifikovaného inteligentního systému

Hlídač se z hlediska svého chování projevuje prakticky výhradně svým pohybem. Je to jediný způsob, kterým může přímo ohrozit hráče a tím splnit svoji roli. S okolím jinak interaguje pouze, když komunikuje s kolegy, nebo když otevírá či zavírá dveře. Hlídač má tedy zjednodušeně pouze jeden hlavní úkol - najít si situačně vhodný cíl, ke kterému se má pohybovat, a jakmile je nalezen, dopravit se k němu. Hlavní funkce, která hlídačovo chování obstarává, je k tomuto účelu rozdělena do tří logických bloků. V každém jednotlivém herním cyklu[1] je tato funkce volána vždy postupně pro každého hlídače.

Úkolem prvního bloku hlavní funkce je aktualizace informací, se kterými bude funkce následně pracovat. Součástí tohoto bloku je zjištění aktuálně relevantních překážek a nastavení rozhodovacích a stavových příznaků. Následně má za úkol výměnu informací s ostatními hlídači, například zda-li někdo může poskytnout hráčovu aktuální pozici.

Na základě těchto již aktuálních informací funkce v druhém bloku s pomocí jednoduchého větvení vyhodnotí, zda-li je potřeba měnit současný hlídačův cíl. Pokud ano, podle nastavených příznaků zvolí vhodnou metodu nalezení nového cíle a provede ji. Každá z těchto metod má za úkol hlídači najít cíl, avšak za jiných okolností a jiným způsobem.

Jakmile je cíl nalezen, popř. nezměněn, je ve třetím bloku vždy volána funkce pro výpočet cesty do cíle, která sama vyhodnotí, je-li v tomto okamžiku potřeba cestu přepočítat. Jakmile je cesta aktuální, je proveden jeden dílčí krok směrem k následující ze seznamu vypočítaných pozic.

Funkce je algoritmicky popsána ve výpisu 5.1 a její bloky budou následně probrány ve větším detailu.

[1] anglicky tzv. „game loop“[3], probíhá v ideálním případě 60x za vteřinu

```
jednej():
{
    nacti_prekazky()
    aktualizuj_udalosti()
    komunikuj_s_kolegy()

    najdi_cil()

    prepocitej_cestu()
    proved_krok()
}
```

Výpis 5.1: **Hlavní funkce hlídače.** Tato funkce se stará o veškerou činnost hlídače a je rozdělena do tří logických bloků: aktualizace informací, změna cíle a navigace.

5.1 Aktualizace informací

Úkolem prvního logického bloku je aktualizovat data, která hlídač využije k rozhodování v daném momentu.

5.1.1 Zjištění relevantních překážek

Nejprve je třeba aktualizovat seznamy překážek, s nimiž se v tomto okamžiku může hlídač setkat. Toho je dosaženo určením místnosti, či více místností, v nichž se hlídač v tento moment nachází. Přes překážky v těchto místnostech se dále iteruje a odkazy na ně jsou umístěny do několika seznamů.

Seznamy překážek, které hlídač uchovává, jsou tři. Některé překážky mezi sebou tyto seznamy sdílejí, jiné ne. Každý herní cyklus jsou seznamy vyprázdněny a znovu obnoveny.

Fyzické překážky

První ze seznamů představuje všechny překážky, se kterými hlídač při pohybu v daný moment může přijít do kolize a za žádných okolností se tedy nemůže stát, aby se v jakémkoliv okamžiku hlídačův kolizní čtverec s jedním z těchto objektů překrýval. Do tohoto seznamu jsou umístěny ukazatele na všechny kusy nábytku, zdi a dveřní křídla všech místností, do nichž hlídač svým kolizním čtvercem zasahuje. Dále jsou do něj umístěny kolizní čtverce představující všechny ostatní postavy na herní ploše.

Navigační překážky

Druhý seznam představuje překážky, které hlídač pouze považuje za neprostupné. Dveřmi projít může, proto dveřní křídla do tohoto seznamu na rozdíl od prvního umístěny nejsou. Jako neprostupné však považuje dveře, ve kterých někdo stojí[2]. Do seznamu jsou tak přidány i celé dveřní prostory, které momentálně mají příznak signalizující, že jsou někým obsazené.

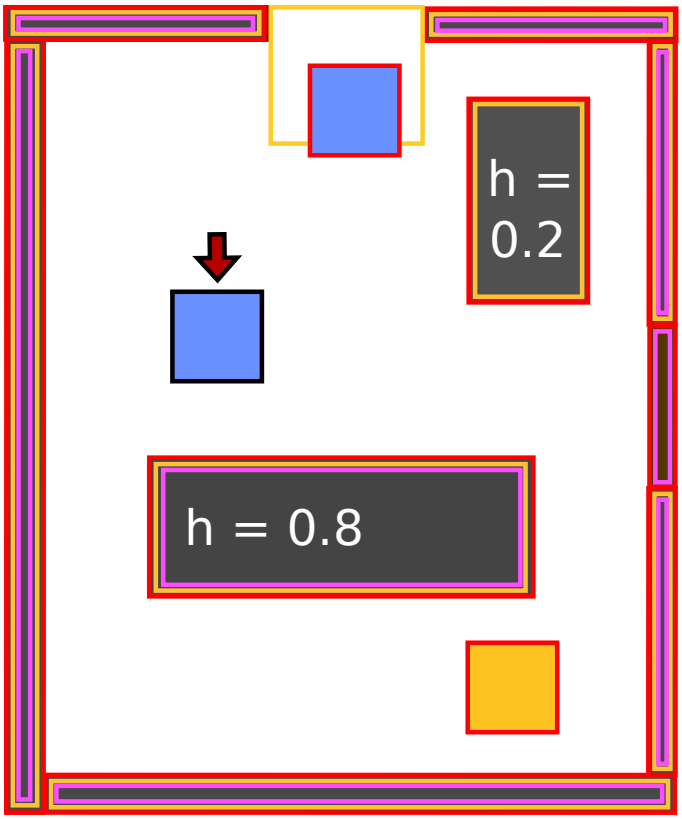
Do seznamu jsou rovněž přidány kolizní čtverce všech aktérů, kteří jsou současně ve stejné místnosti a zároveň jsou dostatečně blízko. Agent tím při výpočtu cesty získává povědomí o překážejících hlídačích v blízkém okolí. Je zbytečné zahrnovat i hlídače ve větší vzdálenosti, než je délka strany samotného hlídače v součtu s jeho současnou rychlostí,

[2]Interakce s dveřmi je detailně popsána v sekci 5.3.3

protože jejich poloha se mění dynamicky a neaktuální výpočet by vedl k vyhýbání se překážkám, které se již na daném místě v době příchodu nenacházejí. Tento efekt by byl obzvlášť výrazný, kdyby se hlídači pohybovali proti sobě a hrozila by kolize. Z tohoto důvodu si hlídač současně drží přehled o počtu těchto potenciálně překážejících aktérů. Jakmile se jejich počet změní, tedy někdo se buď dostal do blízkosti hlídače, nebo naopak tento prostor opustil, nastaví se příslušný příznak nutící hlídače k přepočtu cesty.

Vizuální překážky

Třetí seznam představuje překážky, za které se hráč potenciálně může schovat. Překážky přidané do tohoto seznamu závisí na tom, zda se hráč momentálně krčí, na jejich výšce a na místnosti, v níž se hráč nachází. Pokud se hráč nachází v místnosti, která nesousedí s tou místností, ve které se nachází dotyčný hlídač, seznam se vůbec nevyplňuje. Pokud se nachází v sousedící místnosti, jsou do tohoto seznamu zahrnuty i její překážky. Příslušnost překážek do různých seznamů zobrazuje ilustrace 5.1.



Obrázek 5.1: **Obrázek překážek v různých seznamech.** Modré čtverce reprezentují hlídače, žlutý hráče. Překážky jsou barevnými obrysy odlišeny podle náležitosti k určitým seznamům. Červený seznam představuje neprostupné objekty, žlutý představuje ty objekty, které hlídač označený šipkou za neprostupné považuje. Růžový seznam představuje objekty, za které je hráč momentálně schopen se schovat. h zde vyjadřuje výšku objektu. U výšek 0,8 a 0,2 nezáleží na tom, zda je hráč skrčen.

5.1.2 Aktualizace příznaků

Po aktualizaci seznamů je potřeba aktualizovat i příznaky, podle kterých se odvíjí další hlídačova činnost. Primárně je nutné zjistit, jestli hlídač vidí hráče. Toho je dosaženo otestováním, jestli kterákoli z překážek v příslušném seznamu koliduje s úsečkou vedenou ze středu hráčova kolizního čtverce do středu hlídačova kolizního čtverce. Kolize mezi čtyřúhelníkem reprezentujícím překážku a úsečkou je prakticky průnik této úsečky a kterékoliv strany čtyřúhelníka. Algoritmus výpočtu je předveden ve výpisu 5.2.

```
kolize_usecky_s_useckou(x1, y1, x2, y2, x3, y3, x4, y4):
{
    a = ((x4-x3) * (y1-y3) - (y4-y3) * (x1-x3)) / ((y4-y3) * (x2-x1) - (x4-x3) * (y2-y1))

    b = ((x2-x1) * (y1-y3) - (y2-y1) * (x1-x3)) / ((y4-y3) * (x2-x1) - (x4-x3) * (y2-y1))

    IF (a >= 0 AND a <= 1 AND b >= 0 AND b <= 1):
        RETURN 1
    ELSE:
        RETURN 0
}
```

Výpis 5.2: Funkce určující, zda se dvě úsečky protínají [10]. Úsečky jsou určeny dvěma body, každý o dvou souřadnicích a s vlastním číslem.

Pro určení LoS tedy stačí dva vnořené cykly. Vnější iteruje přes všechny relevantní překážky, vnitřní přes všechny úsečky tvořící čtverec překážky a s každou z nich počítá průsečík se spojnicí středů hlídače a hráče. Jediná kolize stačí k vyloučení LoS.

Pro úplné potvrzení očního kontaktu je ještě potřeba ověřit, že hráč leží v hlídačově FoV. Toho je dosaženo testováním podmínky, že úhel, který svírá směr agentova pohledu s přímkou mezi hráčem a hlídačem je menší, než polovina námi definovaného FoV.

```
lezi_bod_v_FoV(x, y)
{
    uhel

    IF (x  >= 0.0 AND y < 0.0):
        uhel = (180 / pi) * abs(atan(x / y))

    ELSE IF (x >= 0.0 AND y >= 0.0):
        uhel = 90.0 + ((180 / pi) * abs(atan(y / x)))

    ELSE IF (x < 0.0 AND y < 0.0):
        uhel = 270.0 + ((180 / pi) * abs(atan(y / x)))

    ELSE:
        uhel = 180.0 + ((180 / pi) * abs(atan(x / y)))

    IF ((abs(hlidac->uhel - uhel) < ([fov] / 2))
        OR (hlidac->uhel > 240 AND (uhel + 360 - hlidac->uhel) < ([fov] / 2))
        OR (uhel > 240 AND (hlidac->uhel + 360 - uhel) < ([fov] / 2))
       ):
       RETURN 1

    RETURN 0
}
```

Výpis 5.3: **Algoritmus určující, zda bod leží v FoV hlídače.** Konstanta `fov` určuje zorný úhel hlídače, v našem případě nastaveno na 180°. Proměnná `uhel` představuje úhel, který svírá linie pohledu hlídače se spojnicí jeho středu a dotyčného bodu.

Aby však mohl být příznak vizuální detekce nastaven, musejí být obě výše popsané podmínky splněny po určitou dobu. Jak dlouhá tato doba je určeno vzdáleností hlídače od hráče. Třída hlídače obsahuje čítač reálných hodnot od 0 do 110, kde 0 reprezentuje klidový stav a hodnoty vyšší než 100 reprezentují odhalení. Stojí-li hráč tak blízko hlídači, že se jej téměř dotýká, k hodnotě by měla každý herní cyklus být přičtena taková hodnota, aby odhalení z klidového stavu netrvalo déle, než třetinu vteřiny, což je často citovaná reakční doba člověka. Naopak stojí-li relativně daleko, přičítaná hodnota by neměla být příliš velká. Doba odhalení bude v našem případě fixně nastavena na dvě vteřiny, pokud je hráč dále, než polovinu délky úhlopříčky herní mapy. Mezi touto vzdáleností a polohou hlídače klesá doba odhalení lineárně. V případě, že LoS/FoV nelze s hráčem navázat a současně čítač neobsahuje hodnotu 0, je dekrementován. Odečítaná hodnota by měla být nižší, než nejnižší možná hodnota inkrementu při odhalování, kterou hra dovoluje, tedy v našem případě taková, aby odečítání ze 100 do 0 trvalo déle, než dvě vteřiny. Tato hodnota bude tedy nastavena na čtyři vteřiny. V takovém případě, bude-li hráč v libovolné vzdálenosti rychle měnit pozici a bude střídavě viděn a neviděn přesně v poměru 1:1, bude časem nevyhnutelně odhalen. Algoritmus vizuální detekce popisuje výpis 5.4.

```
je_hrac_detekovan()
{
    min = (100.0 / (2.0 * 60.0))
    max = (100.0 / ((1.0/3.0) * 60.0))

    IF (los AND fov):
        vzdalenost = spocitej_vzdalenost(hrac->pozice, hlidac->pozice)

        IF (vzdalenost > [max_vzdalenost]):
            detekce += min

        ELSE:
            c = vzdalenost / [max_vzdalenost]
            detekce += max - (c * (max - min))

            IF (detekce >= 110.0):
                detekce = 110.0

    ELSE:
        detekce -= (min / 2)
        IF (detekce <= 0.0) detekce = 0.0

    IF (detekce >= 100.0):
        RETURN 1

    ELSE:
        RETURN 0
}
```

Výpis 5.4: **Algoritmus rozhodující, zda došlo k detekci.** Proměnná `min` zde symbolizuje minimální přírůstek čítače `detekce`, dosažení hodnoty 100 trvá dvě sekundy, `max` reprezentuje maximální přírůstek a čítač s ním dosáhne hodnoty 100 za třetinu sekundy. Konstanta `max_vzdalenost` je vzdálenost s minimálním přírůstkem, za kterou již přírůstek neklesá a zůstává roven `min`.

Dalšími příznaky, které je třeba nastavit, jsou mimo jiné příznak o docestování do cíle a příznak úkroku. Příznak o doražení do cíle je nastaven, pokud hlídač momentálně stojí na pozici, která je současně jeho cílová. Později je využit k signalizaci potřeby nalezení nového cíle.

Příznak úkroku vyjadřuje nastání situace, kdy hlídač sice má nastaven nějaký cíl, ale momentálně narazil na nečekanou překážku, která cestu blokuje. Aby se jí nějakým způsobem vyhnul, tento důležitý cíl si přesune do paměti a přepíše jej nějakým náhradním, určeným vhodnou vyhýbací metodou, ke kterému se v současnosti dostat může. Jakmile se hlídač s pomocí těchto náhradních cílů překážce úspěšně vyhne, tento příznak mu řekne, že není třeba generovat hlavní cíl znovu, ale místo toho lze obnovit z paměti původní, na který se kvůli překážce nezdařilo dorazit[3].

5.1.3 Komunikace s kolegy

Poslední fázi aktualizace informací je komunikace mezi hlídači. Třída hlídač obsahuje strukturu reprezentující jeho paměť obsahující informace relevantní pro předání kolegům, jako je místnost, v níž je hráč nachází. Obsahuje současně ukazatel na mysli ostatních hlídačů.

[3]Generace alternativních cílů je detailně popsána v sekci 5.2.3 a a její systém popisuje diagram 5.3

V případě, že je nastaven příznak vizuálního kontaktu s hráčem, hlídač aktualizuje pozici hráče ve své paměti a nastaví příznak značící aktivní komunikaci. Rovněž inkrementuje svůj stupeň rozrušení.

V případě, že hlídač hráče nevidí, ve funkci pouze iteruje přes paměti svých kolegů. Jestli některý z nich má nastavený příznak komunikace, hlídač si od něj okopíruje stupeň rozrušení a místnost, v níž se hráč nachází a na základě své a hráčovy zjištěné polohy nastaví další příznaky týkající se způsobu pronásledování. Tím je simulována dálková komunikace mezi hlídači, při které si vzájemně sdělují místnost, v níž se hráč nachází. Hlídač nesmí aktivně komunikovat příliš často, proto je možnost komunikovat omezena časovým intervalem pěti sekund. Každých pět sekund tedy při pronásledování nahlásí aktuální pozici hráče svým kolegům. Algoritmus popisuje výpis 5.5.

```
komunikace()
{
    IF (hrac_detekovan):
        casovac_reci->zapnout()

    cas = casovac_reci->uplynuly_cas()

    IF (cas > 5s AND hrac_je_v_jine_mistnosti_nez_naposledy):
        hlidac->mluvi = 1
        casovac_reci->vypnout()

    IF (los):
        RETURN

    FOR (ostatni_hlidaci, jiny_hlidac = iterator):
        IF (!jiny_hlidac->mluvi):
            CONTINUE

        hlidac->hracova_pozice = jiny_hlidac->hracova_pozice
        hlidac->uroven_rozruseni = jiny_hlidac->uroven_rozruseni
        hlidac->preruseni = 1

        IF (sdilis_s_hracem_mistnost):
            pronasleduj_hrace()

        ELSE IF (hracova_mistnost_ma_jedny_dvere):
            hlidej_tyto_dvere()

        ELSE:
            nadbehni_hraci()
}
```

Výpis 5.5: **Komunikace hlídače s kolegy.** Přerušení u hlídače signalizuje dlouhodobějším funkcím, jako je například rozhlížení, aby se přerušily, a hlídač se ihned mohl věnovat pronásledování.

5.2 Systém rozhodování

Na základě nyní již aktuálních informací a nastavených příznaků je hlídač schopen rozhodnout, jakou proceduru zvolí pro výběr cíle, a je-li vlastně potřeba jej měnit. Jednoduchým

větvením na základě nastavených příznaků je vyhodnocen současný stav a vybrána vhodná procedura. Jednotlivé stavy obvykle trvají nějakou dobu a procedury jsou tím pádem volány při každém jednotlivém herním cyklu po dobu trvání těchto stavů. Je proto důležité s tím při jejích psaní počítat. Pokud je cílem procedury pouze pouze nalézt nový cíl (o vlastní přesun k cíli se stará až třetí blok), musí být tato funkce ošetřena tak, aby skutečně byl cíl vygenerován pouze jednou za dobu trvání stavu, jinak dojde k opakovanému přepisování cíle a ve výsledku tedy k žádnému pohybu.

5.2.1 Generátor pozic

V následujících funkcích se hojně využívá generátoru pozic. Generátor je v principu cyklus, který nejprve s pomocí funkce `rand()` vytvoří pseudo-náhodnou pozici v určitém vymezeném prostoru. Následně na tuto pozici umístí tzv. odhad - kolizní čtverec, který bude sloužit k testování použitelnosti pozice. Jeho rozměry mohou být shodné s rozměry kolizního čtverce hlídače, který simuluje, popř. mohou být větší, pokud je kolem hlídače potřeba určitý volný prostor. Tento odhad je následně testován na libovolné podmínky, jako je například kolize s jinými objekty, vzdálenost od určitých klíčových bodů, zda nezasahuje mimo místnost, apod. Projde-li všemi testy, pozice je přijata a generátor vrací získanou pozici. Pokud porušuje jednu nebo více podmínek, pozice je neplatná a cyklus se opakuje, dokud není nalezena taková, který podmínky splňuje. Zde hrozí riziko uváznutí v nekonečném cyklu, je proto vhodné použít nějakou bezpečnostní zarážku. Lze například omezit počet vygenerovaných pozic takovým číslem, aby generace netrvala příliš dlouho a nezpomalila chod hry. V jednom herním cyklu se vše včetně vykreslování musí stihnout za $16,\overline{6}$ms pro zachování 60 snímků za sekundu. Princip generátoru popisuje výpis 5.6.

```
generator_pozic()
{
    Ctverec odhad
    odhad->delka_strany = hlidac->delka_strany
    pozice_prijata = 1

    FOR (maximalne_stokrat):
        pozice_prijata = 1

        odhad->pozice->x = hlidac->pozice->x - [posun] + rand() % [rozsah]
        odhad->pozice->y = hlidac->pozice->y - [posun] + rand() % [rozsah]

        IF (kolize(odhad, zdi)):
            ok = 0

        IF (jakakoli_jina_podminka):
            ok = 0

        IF (pozice_prijata):
            BREAK

    IF (pozice_prijata):
        hlidac->cil = odhad->pozice

    ELSE:
        hlidac->cil = hlidac->pozice
}
```

Výpis 5.6: **Generátor pozic v hráčově okolí s bezpečnostní zarážkou.** Konstanty `posun` a `rozsah` určují rozměry hráčova okolí, kde mají pozice být generovány. Pokud nebyla přijata ani jedna ze sta vygenerovaných pozic, hlídač přijme za cíl svoji současnou pozici a v tomto okamžiku se nehýbe.

5.2.2 Časovače

V moha okamžicích je též potřeba využít časování, například při simulaci zpoždění reakcí. Knihovna SDL 2.0 k tomuto účelu nabízí funkci SDLGetTicks()[4], která vrací počet milisekund, které uběhly od inicializace modulu SDL. Je vhodné vytvořit si třídu časovače[5], která s pomocí odečítání těchto časů přesně určí uběhnutí určitého časového okamžiku. S pomocí časovačů lze pak vytvořit funkce, které vracejí booleovskou jedničku v okamžiku, kdy uběhne určitý čas, ve kterém může hlídač provádět jinou činnost, například se rozhlížet. Obdobně lze postupovat při zpoždění reakcí při pronásledování nebo při zpoždění výměny komunikace. Stejný časovač je využit i ve třídě reprezentující místnost, kde se využívá k určení, zda byla nedávno zkontrolována.

5.2.3 Rozhodovací větvení

Hlídač při rozhodování prioritně řeší krizové stavy a dynamické vyhýbání, až následně pronásledování, hledání hráče a nakonec patrolování. Rozhodování má podobu algoritmického

[4] Více informací lze nalézt na: https://wiki.libsdl.org/SDL_GetTicks

[5] Příklad časovače a jiných využití modulu SDL2.0 nabízí tutoriál Lazy Foo' Productions: https://lazyfoo.net/tutorials/SDL/index.php

větvení, jak ukazuje výpis 5.7. Ke generování nových cílů většina funkcí využívá určitou formu výše popsaného generátoru pozic.

```
najdi_cil()
{
    IF (jsi_dlouhodobe_nemobilni):
        nahodne_ukroc()

    ELSE IF (jsi_blokovan AND jsi_ve_dverich AND na_tyto_dvere_nekdo_ceka):
        uvolni_dvere()

    ELSE IF (jsi_blokovan):
        ustup_prekazce()

    ELSE IF (vidis_hrace OR dotykas_se_hrace):
        pronasleduj_hrace()

    ELSE IF (nadbihas):
        nadbehni_hraci()

    ELSE IF (dohanis):
        dozen_hrace()

    ELSE IF (prohledavas):
        prohledej_mistnosti()

    ELSE IF (dorazil_jsi_do_cile AND nebyl_to_hlavni_cil):
        obnov_cil()

    ELSE IF (dorazil_jsi_do_cile AND rozhlizeni()):
        najdi_cil()
}
```

Výpis 5.7: **Rozhodování hlídače na základě nastavených příznaků.** Funkce `rozhlizeni()` v poslední podmínce provádí rozhlížení a vrací kladnou hodnotu pouze v případě, že rozhlížení dokončeno. Není-li žádný z příznaků nastaven, cíl se nemění a hlídač pouze pokračuje v pohybu.

Patové situace

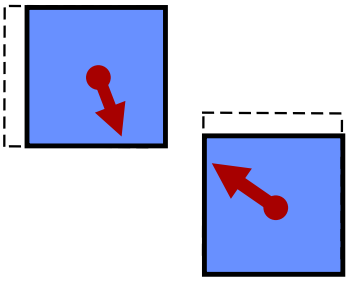
Pokud se hlídač příliš dlouho nepohybuje přesto, že by v tomto okamžiku podle příznaků měl být v pohybu, je volána procedura řešící patové situace. Tohle je velmi nežádoucí situace a pokud nastane, je řešena jako první. Dochází k ní v případě, že ostatní způsoby vyhnutí se dynamickým překážkám selhaly. Hlídač v tomto okamžiku začne s pomocí generátoru pozic tvořit pozice v jeho okolí do vzdálenosti jedné délky své strany. Pokud je některá z nich validní, tedy nekoliduje s žádnou překážkou, které hlídač považuje za neprostupné, využije ji jako cíl k úkroku. V tomto okamžiku je též nastaven příznak úkroku zmíněný výše. Jakmile hlídač do přechodného cíle dorazí, obnoví původní cíl a zkusí do něj cestovat znovu (viz obr. 5.3). Není-li to možné, opět se některá z procedur pro řešení blokace pokusí situaci vyřešit vhodným úkrokem.

Uvolnění dveří

Pokud hlídač pouze není schopen najít cestu k cíli a současně stojí ve dveřích, je to další prioritní situace. Jeho prioritou je v tomto případě opustit dveřní prostor, aby neomezil ostatní hlídače. Generuje tedy cíle v okolí dveřního prostoru a opět provádí úkrok. Vzhledem k tomu, že dveřní koridor se skládá ze dvou dveřních prostorů, snaží se hlídač generovat nejprve pozice kolem té poloviny, která leží dále na jeho plánované trase. Až v okamžiku, kdy se generace nezdaří více než stokrát, zahrne do výpočtu i druhou polovinu dveří. Hlídači se díky popsanému přístupu méně vracejí po vlastních stopách, což prospívá celkové plynulosti jejich pohybu a vzájemnému vyhýbání.

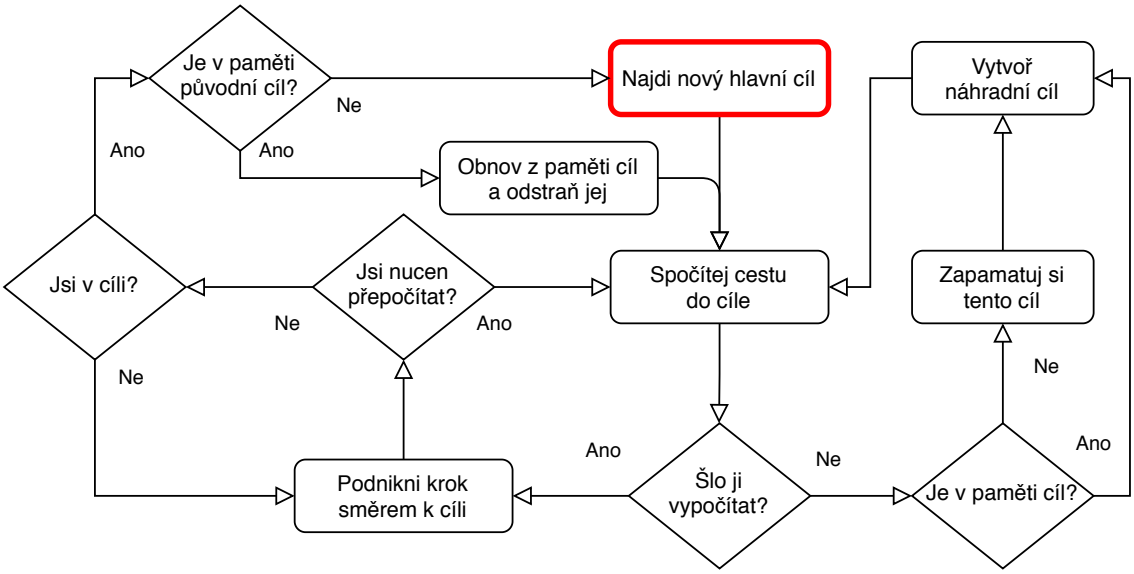
Úkrok stranou

Pokud ve dveřích nestojí, ale cestu stále najít nemůže, je volána procedura pro drobný úkrok stranou. Překáží-li si hlídači mimo dveřní prostor, s největší pravděpodobností budou mít dostatek prostoru na vyhnutí se při minimálním uzpůsobení. Hlídač se vždy pokusí vyřešit situaci úkrokem o jeden pixel, a to směrem doprava. Toho je dosaženo určením, kterým směrem hlídač směřuje, tento směr je zjednodušen na základní čtveřici směrů nahoru/dolů/doleva/doprava, a poté hlídač ukročí tímto směrem + 90°. Situaci vyobrazuje ilustrace 5.2. Hlídači jdoucí proti sobě se tedy zdánlivě pokusí vzájemně vyhnout, každý jedním, nebo více úkroky. Pohyb při úkroku je vždy pouze o jeden pixel a než hlídač ukročí znovu, vždy se pokusí vypočítat cestu do cíle. V tomto okamžiku je naplno využita výhoda relativní uzlové mřížky používané k navigaci v rámci místnosti, která se tedy vždy posune o pixel.



Obrázek 5.2: **Vyhýbání hlídačů úkrokem stranou.** Hlídači, kteří si vzájemně překážejí v cestě, se pokusí o vyhnutí provedením úkroku stranou. Směr, kterým se hlídač dívá, je uchováván ve stupních, např. hlídač směřující vzhůru obsahuje údaj o svém úhlu 0°. Při určování směru úkroku je směr pohledu zaokrouhlen na celý násobek 90° a následně je k němu dalších 90° přičteno. Tímto směrem se hlídač pokusí ukročit o vzdálenost jednoho pixelu. Červené šipky na obrázku reprezentují směr hlídačova pohledu, přerušované čáry pak pozici, kterou hlídači provedením úkroku zaujmou.

Tři výše popsané techniky řeší problémy s dosažením cíle kvůli dynamickým překážkám a ve výjimečných případech s navigací v těsných prostorách. Obrázek 5.3 znázorňuje princip kontinuální tvorby cílů a současně vyhýbání se dynamickým překážkám. Popisuje princip navigace na nižší úrovni a vlastní pohyb mezi cíli, na rozdíl od diagramů v příloze A, které popisují vyšší rozhodování.



Obrázek 5.3: **Princip navigace nižší úrovně.** Červeně je zvýrazněna výchozí činnost. Náhradní cíle tvoří tři výše popsané metody. Příčiny potřeby přepočítat cestu popisuje sekce 5.3. Způsob hledání hlavních cílů záleží na současném záměru vyšší úrovně.

Pronásledování

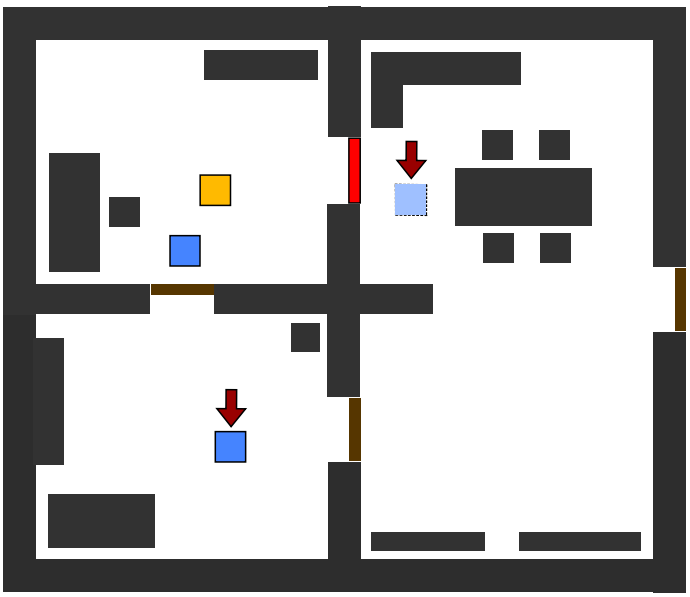
V žebříčku priorit následuje reakce na hráče. Pokud platí, že se hlídač hráče dotýká nebo jej vidí, začíná jej pronásledovat. Při pronásledování hlídač mění rychlost podle vzdálenosti. Pokud je hráč daleko, hlídač se pohybuje rychleji, jakmile se však přiblíží, postupuje obezřetněji a tudíž pomaleji. Hlídač v tomto okamžiku zapomíná svůj původní cíl a soustředí se pouze na pronásledování. Vygeneruje si tedy nový cíl. Prvním kandidátem na cíl je hráčova přesná pozice. Generátor zkontroluje, jestli má kolem sebe hráč dostatek místa, v případě že ne, generuje pozice v hráčově blízkém okolí, vždy však takové, ze kterých lze navázat s hráčem LoS. Tím však hrozí situace, kdy pokud nelze využít hráčova pozice, hlídač vygeneruje každý herní cyklus novou pozici v jeho okolí a ve výsledku pak nejde nikam. Proto po otestování hráčovy přesné pozice předtím, než se začnou generovat pozice nové, otestuje se znovu validita cíle vypočítaného již v minulém cyklu. Pokud je stále validní, použije se.

Nadbíhání a číhání

Další možností hlídače, pokud neprovádí úkrok nebo nepronásleduje, je nadbíhání. Je nutné nejprve zjistit vhodnou místnost k obklíčení. Toto rozhodnutí probíhá již dříve, při komunikaci s agenty, kdy se nastavuje příznak obklíčení. Hlídač se rozhodne na základě počtu dveří, které má místnost, v níž se hráč podle dostupných informací nachází. Jsou-li pouze jedny, hráč nemá možnost úniku a hlídač si vybírá k nadběhnutí sousední místnost a bude v ní číhat. Pokud je dveří v místnosti více, hlídač se pokusí nadběhnout přímo do ní. Hlídač dále po dobu nadbíhání dostává bonus k rychlosti.

Pokud se jedná o slepou místnost, hlídač číhá vygenerováním s ničím nekolidující pozice v sousední místnosti s možností navázat LoS s dveřmi. Do této pozice se přesune a hlídá z ní dveře, dokud není vyrušen hráčem, nebo informací kolegů o aktuálnější poloze. V takovém případě může pokračovat v pronásledování, nebo se znovu pokusit o nadběhnutí.

Pokud se jedná o místnost s více dveřmi, proběhne v ní pokus o nalezení dveří, ke kterým je hráč blíž, než přítomný pronásledující hlídač. Hráč se pravděpodobně bude vždy snažit o pohyb směrem od hlídačů, proto takové dveře dávají k nadběhnutí největší smysl. Je-li takových dveří dostupných více, vybere hlídač ty, které jsou od hráče nejdále. V úvahu však přichází pouze takové dveře, které mají nastavený příznak znamenající, že jsou dosažitelné z obou stran ze všech částí mapy. Tyto příznaky je potřeba nastavit již při tvorbě mapy, manuálně, nebo algoritmicky. Pokud se tyto dveře podaří najít, vybere si hlídač místnost, kam tyto dveře z pohledu hráče vedou a v ní vygeneruje náhodnou pozici s dostatkem prostoru v okolí a která s ničím nekoliduje. Při pozdějším hledání cesty jsou z výpočtu právě tyto dveře vyřazeny, čímž je hlídač nucen se vyhnout místnosti, v níž se hráč nachází, a tím mu tedy efektivně nadběhnout. Situace je naznačena na obrázku 5.4. Pokud se průchozí dveře ve správné pozici nalézt nepodaří, generuje hlídač tuto pozici jednoduše přímo někde v hráčově současné místnosti a o nadbíhání se nepokouší.



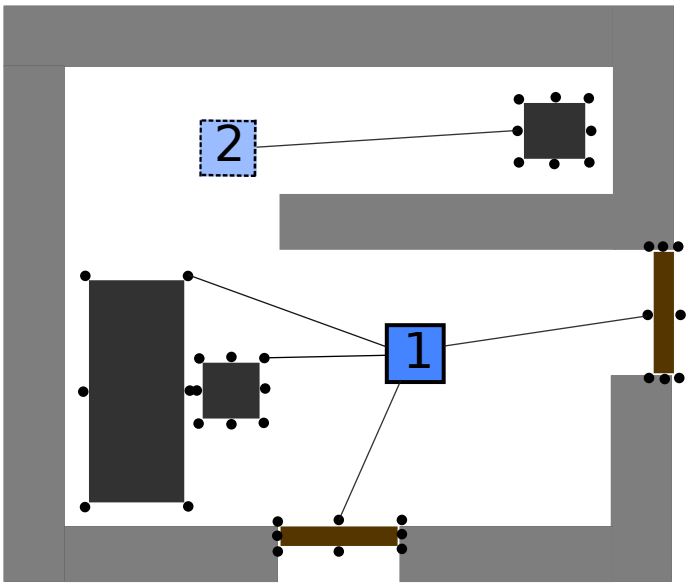
Obrázek 5.4: **Situace umožňující nadbíhání.** Červené dveře jsou označeny jako dosažitelné ze všech bodů mapy z obou stran. Současně jsou v takové pozici, že žlutý hráč leží mezi nimi a přítomným pronásledujícím modrým hlídačem. Jsou proto použity k nadběhnutí. Hlídač označený šipkou se k nim musí dostat ze strany, kde se nenachází hráč a zároveň je po cestě nesmí využít. Je proto nucen jít okolo a nadběhnout.

Hledání hráče

Pronásledující hlídač nevzdává pronásledování v okamžiku, kdy s hráčem ztratí LoS, ale pokračuje na poslední známou hráčovu pozici. Pokud tato pozice zasahovala do nějakého dveřního prostoru, hlídač těmito dveřmi projde do sousední místnosti předpokládaje, že hráč těmito dveřmi také prošel. Pokud se mu do tohoto okamžiku nepodaří znovu navázat oční kontakt, přechází do stavu prohledávání místností.

V tomto stavu hlídač do nově vytvořeného seznamu umístí místnost, v níž se momentálně nachází a současně všechny místnosti, které s ní sousedí a mají pouze jeden vstup. Pokud má sousedící místnost vstupy dva, ale ten druhý vede do místnosti s jedním, přidá

hlídač celé slepé rameno do seznamu. Tyto místnosti má nyní hlídač za úkol prohledat. Důsledné prohledání místnosti spočívá v náhodném vygenerování cílů tak, aby měl hlídač možnost alespoň jednou navázat LoS s každým ze všech kusů nábytku, které se v místnosti nacházejí. LoS stačí navázat s kterýmkoliv ze čtyř rohů objektu, nebo s kterýmkoliv ze čtyř středů jeho stran. Jakmile je vygenerován cíl, jsou z něj počítány LoS pro všechny tyto body všech objektů v místnosti. Pokud je u objektu jednou navázána LoS, hlídač považuje objekt za zkontrolovaný a pokračuje dalším. Hlídač si pamatuje, které kusy nábytku zatím nezkontroloval, a přenáší si tento poznatek do okamžiku, kdy dorazí na další kontrolní pozici. V tomto okamžiku generuje další cíl a pokračuje stejným způsobem. Jakmile je seznam zatím nezkontrolovaných kusů nábytku prázdný, teprve v tomto okamžiku je místnost označena jako prohledaná a hlídač vybere ze seznamu další. Způsob prohledávání místnosti je demonstrován na obrázku 5.5.



Obrázek 5.5: **Kontrola objektů místnosti.** Číslem 1 je označena první kontrolní pozice hlídače. Ten je z ní schopen zkontrolovat všechny objekty v místnosti s výjimkou jednoho. Z příští vygenerované pozice musí být viditelný alespoň jeden z objektů, které doposud nezkontroloval. V tomto případě je takový objekt jen jeden a ten je viditelný z pozice označené číslem 2.

Takto postupuje, dokud nenalezne hráče a nezačne opět pronásledovat, nebo dokud nevyprázdní seznam místností k prohledání. Pokud jej vyprázdní, pokračuje patrolováním podle původního plánu, tentokrát však se zvýšenou ostražitostí a vyšší rychlostí, a již všechny místnosti prohledává výše popsaným způsobem. Předtím, než hlídač začne místnost prohledávat, zkontroluje, jestli nebyla nedávno prohledána kolegou. Třída místnost za tímto účelem obsahuje časovač. Pokud byla prohledána před méně, než třiceti sekundami, přeskakuje ji.

Patrolování

Poslední z možností, na kterou může hlídač reagovat, je nastavený příznak docestování do cíle. Tohoto příznaku hojně využívají i dříve zmíněné metody, ale pokud se vyskytuje samostatně a žádný z příznaků předchozích metod nebyl současně nastaven, znamená to,

že hlídač potřebuje vygenerovat novou pozici při obyčejném patrolování. V klidovém stavu generuje hlídač v každé místnosti ze seznamu vždy pouze jednu pozici. Pokud se jedná o místnost s jedinými dveřmi, cíl je vygenerován ve vnitřní polovině dveřního prostoru. Dokud hlídač dveřní prostor neopustí, dveře zůstanou otevřené. Tímto je simulováno tzv. nahlédnutí do místnosti a laxní přístup hlídačů neočekávajících komplikace. V případě průchozích místností je generována pozice kdekoli v rámci místnosti, kde je dostatek místa. Pokud je hlídač ve stavu pozornosti z důvodu předchozího setkání s hráčem, prohledává místnosti způsobem popsaným v sekci prohledávání místností. Místnosti k procházení jsou hlídači přiděleny manuálně a ten je prochází postupně a cyklicky. Pokud je jeho prohledávání přerušeno, naváže po uklidnění situace tam, kde přestal.

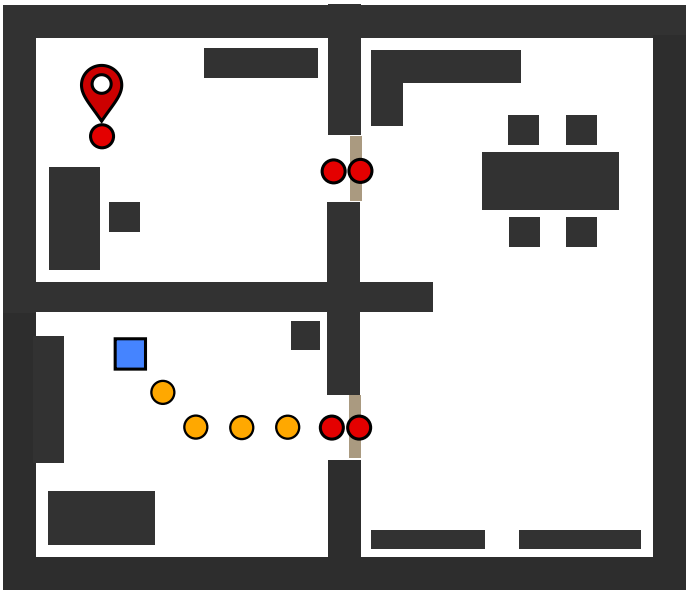
Tato logická větev je též v podmínce omezena funkcí obstarávající rozhlížení se po místnosti. Funkce je volána vždy při testování podmínky a dokud nevrátí kladnou hodnotu, generování nové pozice se neprovede. Místo toho funkce provádí otáčení simulující zkoumání místnosti pohledem a dokud tento pohyb není kompletně proveden, vrací nulovou hodnotu. Hlídač díky nasazení této funkce do podmínky provede rozhlédnutí vždy, když dorazí na novou pozici v místnosti, kterou má prozkoumat. Algoritmus funkce popisuje výpis 5.8.

```
rozhledni_se()
{
    IF (priznak_preruseni):
        RETURN 1

    IF (!casovac_rozhlizeni_zapnut()):
        casovac_rozhlizeni->zapnout()

    soucasny_cas = casovac_rozhlizeni->uplynuly_cas()
    celkovy_cas = spocitat_cas_k_rozhlizeni()

    IF (soucasny_cas < (celkovy_cas / 4)):
        hlidac->aktualizuj_smer_pohledu(mistnost->stred)

    ELSE IF (soucasny_cas < (celkovy_cas / 2)):
        hlidac->uhel -= [inkrement]

    ELSE:
        hlidac->uhel += [inkrement]

    IF (cas > celkovy_cas):
        casovac_rozhlizeni->vypnout()
        RETURN 1

    RETURN 0
}
```

Výpis 5.8: **Funkce obstarávající rozhlížení.** K určení vhodné délky procesu rozhlížení je využita separátní funkce, která rozhodne na základě rozměrů dotyčné místnosti a hlídačovy polohy vůči jejímu středu. Hlídač se rozhlíží tím déle, čím blíže je ke středu místnosti a čím je místnost rozlohou větší. Delší doba rozhlížení konstantní rychlostí znamená též větší pokrytý úhel. V první čtvrtině tohoto vygenerovaného času se hráč natočí do středu místnosti. Ve druhé čtvrtině se námi zvolenou rychlostí otáčí doleva a zbylou polovinu času se otáčí doprava. Kladnou hodnotu funkce vrací v případě, že bylo rozhlédnutí dokončeno, nebo jiná část kódu vyvolala přerušení procedury. Inkrement je v v kódu nastaven na 1.0, tedy odpovídající rychlosti otáčení 60°/s, může však být nastaven libovolně.

5.3 Výpočet cesty

Třída hlídače obsahuje dva seznamy souřadnic, které společně určují cestu. Jeden ze seznamů obsahuje souřadnice dveří vedoucích do místnosti, kde se nachází hlídačův cíl, a cíl samotný. Druhý, dále nazývaný „nižší“, obsahuje souřadnice pozic vedoucích k prvním z těchto dveří v rámci místnosti. Obsah seznamů je naznačen v ilustraci 5.6. Pokud po sobě dvě pozice v nižším seznamu následují, musí být možné mezi nimi cestovat po přímce a to bez rizika kolize s nehybnou překážkou. Pokud je seznam dveří prázdný s výjimkou cíle, neobsahuje nižší seznam cestu ke dveřím, nýbrž přímo k cíli.



Obrázek 5.6: **Seznamy pozic odlišeny barvami.** Červené uzly označují pozice patřící do vyššího seznamu. S výjimkou cíle se jedná vždy o dveře. Žluté uzly jsou nižší cesta, ta vede vždy pouze po první červený uzel. Jakmile je seznam žlutých uzlů vyprázdněn, je vypočítán nový.

Ke kompletnímu přepočtu cesty dochází pouze v případě, že:

- došlo ke kolizi hlídače s jiným objektem,
- jiná funkce nastavením patřičného příznaku přepočet vyžaduje,
- seznam obsahující sekvenci dveří vedoucích k cíli je prázdný.

Ten totiž mimo dveře obsahuje jako poslední pozici i vlastní cíl, takže dokud k němu hráč nedorazí, prázdný být nesmí.

Jakmile je cesta přepočítána, případně pokud k přepočtu vůbec nedošlo, hlídač vezme první z pozic v nižším seznamu a provede krok jejím směrem. Jakmile na pozici v průběhu opakovaného volání funkce dorazí, odstraní ji ze seznamu a provede to samé s další pozicí. Jakmile je prázdný nižší seznam, znamená to příchod k prvním dveřím z vyššího seznamu, a proběhne výpočet cesty spojující tyto dveře s následujícími, případně se samotným cílem, jedná-li se o poslední místnost.

Přepočet cesty probíhá voláním dvou A* procedur. Algoritmus funguje v obou případech stejně, mění se pouze způsob generování uzlů. Procedura pro dveře se volá pouze v případě,

že je seznam dveří prázdný. Procedura pro nižší seznam se volá naopak pouze, pokud seznam dveří prázdný není, neboť počítá cestu k jeho první položce.

5.3.1 Implementace algoritmu A*

Nejprve je potřeba sestrojit seznam OPEN, který bude obsahovat uzly určené k expanzi a seznam CLOSED, který bude obsahovat seznam expandovaných uzlů. Do seznamu OPEN je následně umístěn počáteční uzel. Uzel v našem případě reprezentuje struktura, která obsahuje souřadnice pozice, která uzel identifikuje, celkové ohodnocení uzlu skládající se z doposud uražené cesty a heuristiky a nakonec ukazatel na rodičovský uzel, který v případě prvního uzlu neukazuje nikam a funguje jako zarážka. Jeho ohodnocení je pouze jeho heuristická složka, protože zatím žádná cesta uražena nebyla. Jako heuristická funkce je nasazen výpočet vzdálenosti do cíle vzdušnou čarou.

Následuje hlavní cyklus algoritmu, kdy je nejprve ověřeno, že seznam OPEN není prázdný. Pokud je, úloha nemá řešení, a vyhledávání je ukončeno jako neúspěšné. Pokud není, ze seznamu je vybrán uzel s nejnižším ohodnocením - nejnižším součtem již uražené cesty a vzdálenosti do cíle.

Pokud je tento uzel uzlem cílovým, prohledávání je úspěšné a procedura se v cyklu vrátí po ukazatelích na svému rodičovskému uzlu zpět k původnímu. Všechny tyto uzly průběžně ukládá do prázdného seznamu PATH, který následně vrací jako výslednou cestu. Ukládá je v opačném pořadí a originální pozici vynechává, protože na ní už hlídač stojí.

Pokud uzel cílovým není, expanduje se. V tomto bodě figurují dva různé generátory uzlů. Výběr záleží na tom, zda v současnosti probíhá výpočet vyšší, či nižší cesty. Detaily obou generátorů budou popsány později. Všechny nově vygenerované uzly, které nejsou v seznamu CLOSED se umístí do seznamu OPEN. Seznam CLOSED je proto nutné pro každý nově vygenerovaný uzel prohledat. Expandovaný uzel se poté umístí do seznamu CLOSED. Pokud se uzel se stejnou pozicí v seznamu OPEN už vyskytuje, v seznamu se ponechá ten s nižším ohodnocením. Opět je proto nutné seznam postupně projít. Je též nutné zařadit uzel do seznamu OPEN na správnou pozici s ohledem na jeho ohodnocení (seznam musí být seřazen vzestupně). Následuje návrat na začátek smyčky.

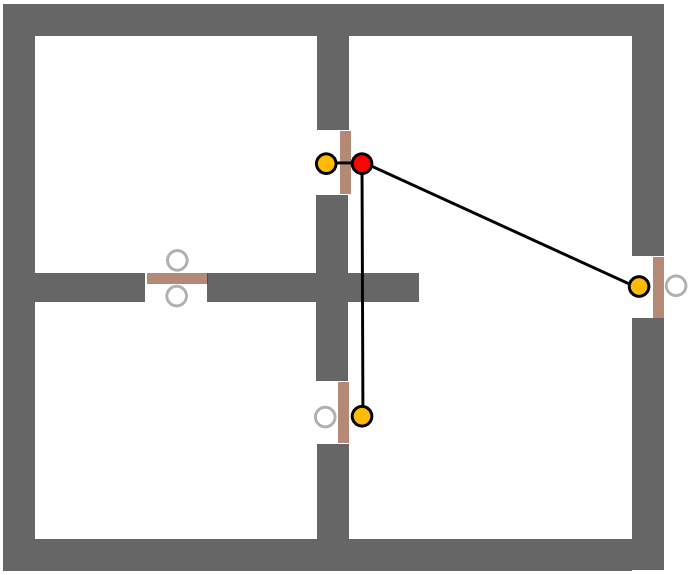
Implementaci algoritmu popisuje studijní opora Základy umělé inteligence IZU: Studijní opora [14].

Generování vyšší úrovně

V případě hledání cesty mezi místnostmi do cíle je třeba použít generátor vyšší úrovně. Nejprve určí místnost, v níž se expandovaný uzel nachází a následně prochází všechny její dveře. Za každé z nich vytvoří a ohodnotí uzel s jejich pozicí. Pokud se pozice některých dveří kryjí se současným expandovaným uzlem, uzel je vytvořen z jejich souseda, se kterým jsou dveře ukazateli provázané. Tím vznikne možnost projít do vedlejší místnosti, jak ukazuje obrázek 5.7. Generátor též počítá s možností nadbíhání – pokud má hlídač v paměti některé dveře označené jako obkličovací a současně se pokouší nadbíhat, z takových dveří se uzel netvoří. Potenciálních uzlů není mnoho a výpočet vyšší úrovně tedy není nijak výrazně časově náročný.

Generování nižší úrovně

Generátor nižší úrovně pracuje jiným způsobem. Postupně se pokusí vytvořit osm nových uzlů v okolí rodičovského uzlu, na obou osách a obou diagonálách. Uzly na osách se gene-



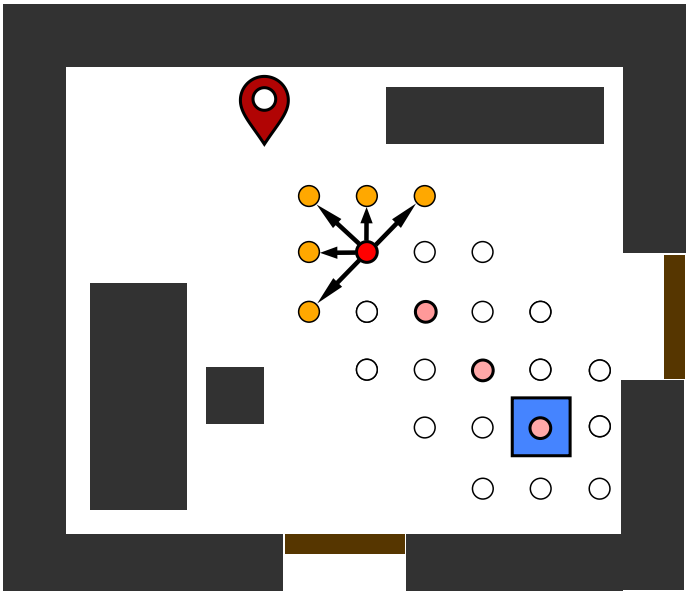
Obrázek 5.7: **Vyšší generace.** Pokud je expandovaný uzel současně uzel představující dveře, jsou jeho potomky všechny dveřní uzly nacházející se ve stejní místnosti a korespondující uzel na druhé straně dveří.

rují ve vzdálenosti dané programátorem jako experimentálně zjištěný vhodný krok pro A* algoritmus, uzly na diagonálách musejí být ve vzdálenosti $\sqrt{2}\times$ větší, aby byla zachována mřížka. V opačném případě by síť uzlů v průběhu výpočtu houstla do nekonečna. Pro každý z uzlů je dále zkonstruován tzv. „odhad“, čtverec na pozici uzlu, který má velikost hlídače, popř. je větší, pokud kolem pozice hlídače chceme při cestování nějaký prostor. S tímto odhadem jdou následně provádět integritní kontroly. Otestováním kolize s objekty, které má hlídač v seznamu překážek, které považuje za neprostupné, lze například ověřit, jestli je tato pozice validní. V případě že ne, není tento uzel generován. Kromě kolizí se stěnami a nábytkem se také testuje, zda se pozice nachází v prohledávané místnosti. Při ohodnocení nových uzlů je třeba pamatovat, že uzly na diagonálách neleží od rodičovského ve stejné vzdálenosti, jako uzly na osách, a složka uražené cesty je proto potřeba navýšit o skutečnou vzdálenost mezi nimi. Nižší generace je naznačena na ilustraci 5.8.

Výpočet je omezen časovačem, který jej omezuje na 10 milisekund. Je to příliš dlouho na to, aby se v tomto herním cyklu dala garantovat frekvence 60 snímků za vteřinu, k výpočtu cesty však nedochází příliš často, proto je riziko okem postřehnutelného propadu snímkování relativně nízké. Pokud metoda do té doby cestu v nižší úrovni nenalezla, nebo již dříve vyhodnotila, že nalézt nelze, ukončí se s neúspěchem. Nižší seznam v tomto případě zůstává prázdný a je nastaven příznak značící blokaci cesty, se kterým pracují dříve zmíněné krizové procedury. V praxi to znamená nalezení nového cíle, ze kterého znovu proběhne pokus o výpočet cesty k současnému nedosažitelnému cíli.

5.3.2 Vyhlazení cesty

Výsledná nižší cesta může kvůli své mřížkové povaze působit nepřirozeně. Hlídač pohybující se pouze v osmi směrech a ve schodovitých vzorech nenavozuje pocit pohybujícího se člověka. Proto je výsledná cesta ještě upravena algoritmem, který ji systematicky projde a schodovité útvary odstraní.

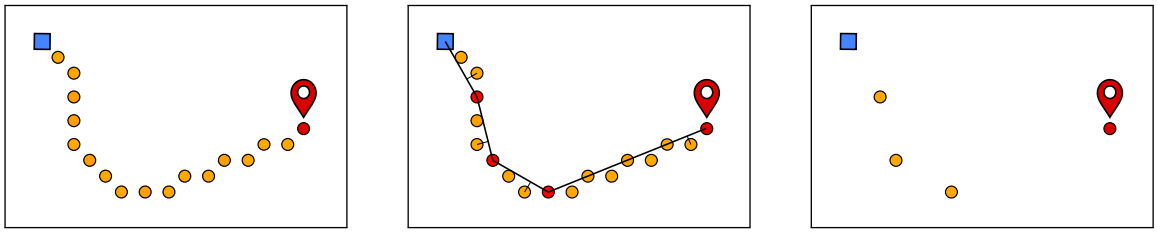


Obrázek 5.8: **Nižší generace.** Bílé uzly představují vygenerované, avšak zatím neexpandované uzly. Růžové uzly jsou již expandované uzly. Červený uzel je expandovaný v tomto okamžiku a žluté jsou jeho potomci. Dva bílé uzly v okolí červeného nejsou přepsány žlutými, protože mají výhodnější ohodnocení. To stejné platí pro nejbližší růžový uzel.

Algoritmus pracuje vždy se dvěma body cesty, které prokládá přímkou. Platí, že pokud je odchylka všech bodů, které na cestě leží mezi nimi, menší, než stanovená hranice, všechny tyto body mohou být z cesty vynechány, a hlídač tak tento úsek projde po přímce. Mějme A reprezentující první z bodů a B reprezentující druhý. Algoritmus nejprve označí první bod cesty jako A i B. Žádné body mezi nimi neexistují, algoritmus tedy pokračuje a jako B označí následující pozici. Při třetím kroku již mezi nimi leží bod a lze tedy kontrolovat podmínka nejvyšší možné odchylky. Dokud podmínka platí pro všechny body, algoritmus pokračuje s kontrolami a posunuje B. Jakmile však podmínka platit přestane, tj. jeden z bodů ležících mezi A a B je od přímky vedené mezi nimi příliš daleko, B se posune o jeden krok zpět, tedy na pozici, při které podmínka ještě platila. Všechny body mezi A a B jsou následně odstraněny, tento úsek se považuje za vyřešený, současný bod B se stává novým A a algoritmus pokračuje novým úsekem. Pokud B nelze posunout dále z důvodu konce cesty a podmínka stále platí, tento úsek je též proložen stejným způsobem. Výsledkem algoritmu je zjednodušená cesta o menším počtu pozic, po které hlídač přechází již bez schodovitých vzorů a s využitím více směrů. Výchozí a výsledný stav algoritmu spolu s jeho průběhem jsou naznačeny na obrázku 5.9, algoritmus samotný je popsán výpisem 5.9.

```
vyhlad_cestu()
{
    A~= B = cesta->zacatek
    vysledek->pridej_do_seznamu(A)

    WHILE (1):
        chyba = 0

        FOR (od A~do konce cesty, B = iterator):
            x = B->pozice->x - A->pozice->x
            y = A->pozice->y - B->pozice->y

            c = 0

            IF (x == 0):
                a = 1
                b = 0

            ELSE:
                a = y / x
                b = -1

            FOR (od A~do B, IT = iterator):
                a1 = IT->pozice->x - A->pozice->x
                a2 = A->pozice->y - IT->pozice->y

                IF (!sqrt((a * a) + (b * b))):
                    CONTINUE

                v~= abs((a * a1) + (b * a2)) / sqrt((a * a) + (b * b))

                IF (v~> ([vyhlazovani] * [a_s_hvezdou])):
                    chyba = 1

            IF (chyba):
                B = B - 1
                BREAK

        IF (chyba):
            vysledek->pridej_do_seznamu(B)
            A~= B

        ELSE:
            BREAK

    IF (vysledek->konec != cesta->konec):
        vysledek->pridej_do_seznamu(cesta->konec)
}
```

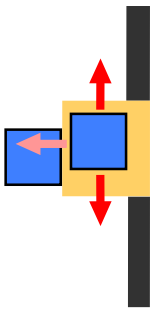
Výpis 5.9: **Funkce pro vyhlazení cesty.** Výpočet vzdálenosti bodu od přímky je v praxi výpočet výšky trojúhelníka. Konstanta `a_s_hvezdou` je velikost kroku algoritmu A* v pixelech, konstanta `vyhlazovani` je experimentálně určená konstanta určující míru vyhlazení. V našem případě je nastavena na 0,65. Vyšší číslo způsobuje, že hlídač naráží do překážek, nižší neodstraní schodovité útvary v cestě.



Obrázek 5.9: **Algoritmus k vyhlazování cest.** První z obrázků zobrazuje původní cestu. Druhý demonstruje běh algoritmu nad touto cestou. Červeně jsou označeny uzly, v jejichž následovnících podmínka o maximální odchylce přestala platit a jsou proto označeny jako body, kde se cesta láme. Třetí obrázek zobrazuje již zjednodušenou výslednou cestu.

5.3.3 Procházení dveřmi

Dveřní koridory jsou přirozeně zúženými úseky a jako takové působí hlídačům problémy při vzájemném vyhýbání. Vyhýbání ve dveřích je tedy řešeno vzájemným vyloučením. První hlídač, který překryje jeden ze dvou prostorů, které tvoří dveřní koridor, je oba obsadí. Dokud tento koridor neopustí, jiný hlídač do těchto prostor nesmí vstoupit a zahrnuje je do svého seznamu překážek, které považuje na neprostupné. Pokud se jiný hlídač do těchto dveří vstoupit pokouší, u dveří je nastaven příznak signalizující, že na dveře někdo čeká. Pokud je nyní hlídači ve dveřích cesta blokována překážkou, je jeho prioritou při vyhýbání opustit dveřní prostor a dveře odblokovat použitím funkce s adekvátním generátorem pozic. Prostory mají takové rozměry, aby je šlo opustit pohybem třemi různými směry, a byla proto nižší pravděpodobnost patové situace. Situace je naznačena na obrázku 5.10.



Obrázek 5.10: **Vzájemná přednost hlídačů ve dveřích.** Modrý hlídač ve žlutě vyznačeném dveřním prostoru do něj vstoupil první. Druhý hlídač se o vstup pokusil, byl mu však odepřen. U dveří je nastaven příznak, že někdo čeká na možnost vstoupit. Pokud druhý hlídač současně prvnímu blokuje cestu a ten je nucen k přepočtu, je vybrána procedura pro odblokování dveří. Dveřní prostor má záměrně rozměry větší, než hlídač, aby ten měl vždy možnost prostor opustit. Přestože druhý hlídač jeden z možných směrů k opuštění prostoru blokuje, dva další jsou volné.

Kapitola 6

Zhodnocení vypracování

Tvorba hry se neobešla bez komplikací. Mnoho problému vzešlo z faktu, že byla hra implementována prakticky od základů, bez využití externího herního enginu. Mnoho času bylo tím pádem věnováno tvorbě kolizního systému, prostředí a herní úrovně. Problémy při tvorbě těchto částí mnohdy vedly k nasazení nekompletních řešení, které v pozdějších fázích komplikovaly vývoj rozhodovacího systému a navigace. Vývoj ztěžovala také nedostatečná zkušenost s programovacím jazykem C++.

Výsledkem je i přesto jedna hratelná úroveň hry s možností zvítězit i prohrát proti třem počítačem řízeným oponentům, kteří jsou schopni při nahánění hráče spolupracovat jako tým a kteří jsou schopni se bez větších potíží pohybovat po herní ploše. Jak výsledná hra vypadá ukazuje snímek obrazovky 6.1.

6.1 Schopnost navigovat

Hierarchické nasazení algoritmu A* svůj účel splnilo. V raných verzích hry proběhly pokusy o nasazení pouze jedné úrovně algoritmu s pohyblivou čtvercovou mřížkou, počet procházených stavů při výpočtu však podle očekávání nebyl zvládnutelný a hra nestíhala. Při omezení mřížky na jedinou místnost a přesunu hledání cesty mezi místnostmi na vyšší úrovni se již situace stala řešitelnou. Bylo však nutné určit vhodnou velikost kroku pro generaci nových navigačních uzlů. Aby byl hlídač schopen ze všech výchozích pozic vejít do prostoru širokého jako je on sám, musí být krok být roven 1px. Takto drobný krok však nebylo možné nasadit ani v místnosti o velikosti 100x100px, výpočet trval neúnosně dlouho.

6.1.1 Použitelnost pohyblivé mřížky

Relativní mřížka však nabízí možnost posunu sama sebe, pokud její umístění nevyhovuje. To umožňuje nasazení většího kroku. Pokud se nezdaří nalézt cestu do úzkého prostoru, hlídač se může pokusit o výpočet znovu příští herní cyklus z lehce upravené výchozí pozice. Dokud pozici mění konzistentně o 1px po obou osách, časem se dostane do prostoru širokého, jako je on sám. Nelze tím však řešit situaci, kdy je tento úzký prostor složitějšího než čtyřúhelníkového tvaru. Pokud se při tvorbě úrovně podaří takovým prostorám vyhnout, algoritmu postačí k efektivnímu hledání cest mnohem větší krok. Teoreticky nemusí být menší, než délka strany hlídače, v našem případě 15px. Bylo experimentálně ověřeno, že procestování všech uzlů v mřížce s takto velkým krokem vyplňující místnost o velikosti 200x200px netrvá déle než jednotky milisekund, což lze do herního cyklu zakomponovat bez větších obtíží a zachovat frekvenci šedesáti snímků za sekundu.

Obrázek 6.1: **Herní plocha.** Implementovaná hra podle specifikace v předchozích kapitolách. Hlídač nalevo spatřil hráče v černém a upozorňuje na jeho pozici své kolegy v pravé části mapy. Hráčova počáteční pozice je v levé horní místnosti a dokumenty k ukradení leží na stole v pravé dolní místnosti. Rozměry mapy jsou mimo oba balkony 525x350px, rozměry postav jsou 15x15px.

Hierarchický systém s posuvnou mřížkou na nižší úrovni je tedy v praxi použitelný. Na složitosti prostoru uvnitř místností však záleží, jak velký je potřeba krok a jestli se výpočet stíhá během jednoho herního cyklu. Jednou z možností vylepšení algoritmu je rozložení výpočtu do více herních cyklů, případě lze k výpočtu využít separátní proces.

6.1.2 Dynamické vyhýbání

Vzájemné vyhýbání hlídačů je klíčové hlavně ve dveřích a v úzkých prostorách, a oba přístupy se ukazují jako funkční. Nejužší místností na mapě je chodba, která na šířku měří 35 pixelů. Pojme tedy dva hlídače vedle sebe. Ti však většinou jdou proti sobě prostředkem místnosti, vyhýbání je proto téměř vždy nutné. Vzhledem k tomu, že oba uhýbají směrem doprava, vyhnou se rychle a efektivně způsobem, který připomíná lidské chování.

Obdobně probíhá vyhýbání ve dveřích, kdy hlídač před vstupem do dveří počká, dokud je druhý neopustí. Z perspektivy hráče to vypadá, jako by si dávali přednost. Systém přesto není dokonalý a obzvláště v úzkých místech, kde se současně kříží dveřní prostory, může dojít k chaotickému vyhýbání či déle trvající vzájemné blokaci. Nestalo se však, že by hlídači takovou situaci po čase nevyřešili.

6.2 Dodržování role a schopnost zvítězit

Hlídači procházejí prostor pro hráče stravitelným způsobem v cyklech, které jdou vypozorovat, ale zároveň nejsou zjevné. Pohybují se po přirozených drahách, do místností nahlížejí a rozvážně se po nich rozhlížejí. Navozují dojem sice svědomitých, ale nespěchajících hlídačů.

Při zpozorování hráče dělí zapojení ostatních hlídačů realistická prodleva, při které má hráč možnost rychle promyslet plán úniku. Hlídači nejsou rychlí, jako běžící hráč, jsou však přesto schopni jej relativně dlouho stíhat. Pokud se tedy hráč chce ztratit z dohledu, je ve většině případů nucen využít chodeb. Okruh, který je však tvořen právě těmito třemi chodbami, které jsou na snímku obrazovky 6.1 vyplněny oranžovou barvou, a dvěma dalšími místnostmi, slouží výborně k nadbíhání. Právě zde se schopnost hlídačů komunikovat a nadbíhat projeví nejvíce. Pro hráče je také velmi nežádoucí být viděn, když vchází do slepé místnosti, protože hlídači si na něj před ní počkají.

Úroveň není náročná, jakmile člověk zná pohybové vzorce hlídačů; lze ale vyhrát i bez toho. Hra tím však svádí k trpělivějšímu přístupu, což byl účel. Návrh obtížnějších úrovní by spočíval primárně v použití více hlídačů, složitějších prostor, či úkolů skládajících se z více částí, nutících hráče strávit v úrovni více času. Úpravou chování hlídačů obtížnost hry příliš měnit nelze, pokud je snaha udržet jejich projevy na hranici uvěřitelnosti.

6.3 Názor hráčů

Hru kromě autora testovali čtyři další hráči s různými zkušenostmi s videohrami. Každý zúčastněný dostal prostor seznámit se s ovládáním v úrovni bez oponentů a současně mu bylo sděleno, co je jeho úkolem. Následně byli do hry přidání soupeři.

Tři ze čtyř hráčů byli schopni úroveň po třech až pěti neúspěšných pokusech splnit. V průběhu hraní si začali uvědomovat zvyklosti hlídačů a jejich vnímavost. Zpočátku se při detekci nepokoušeli o únik a úroveň vzdali. Později při úniku dva hráče zaskočilo, že je hlídači obklíčili, protože spolupráci mezi nimi nečekali. Překvapivá byla též schopnost hlídačů prohledávat místnosti důsledným způsobem a nalézt hráče skrčeného za překážkou. Hru jeden z testujících popsal slovem „stresující“, nicméně odmítl testování ukončit, dokud nevyhraje.

Kapitola 7

Závěr

Cílem práce bylo vyvinout stealth videohru s oponenty, kteří se v rámci možností co nejvíce chovají jako spolupracující tříčlenný tým hlídačů. Práce měla za úkol shrnout obvykle využívané metody při tvorbě tohoto typu her a zhodnotit je z hlediska kvality herního zážitku. Dále bylo jejím cílem vybrat nejvhodnější metody pro tyto účely a popsat jejich implementaci. Vytvořená hra by měla demonstrovat důležitost a soběstačnost dobrého systému rozhodování ve stealth hře.

Pozitivní ohlasy testujících na překvapivé taktiky hlídačů společně s faktem, že úroveň pro ně nebyla ani nesplnitelná, ani vyloženě snadná, mě vedou k závěru, že cíl práce byl splněn. Jejím výsledkem je minimalistická, ale přesto zábavná hra, v níž se hráč postaví třem počítačem řízeným oponentům, kteří jsou společnými silami schopni jej porazit. Dostatečně tím dle mého názoru demonstruje důležitost a funkčnost inteligentního systému ve stealth hrách.

Přínos práce je v první řadě osobní. Umožnila mi projít si procesem tvorby hry od základu, a tím mi poskytla mnoho cenných zkušeností. Práce může sloužit jako dobrý soubor užitečných zdrojů a jako shrnutí implementačních detailů některých často používaných metod.

Praktická část splnila všechna důležitá kritéria a je v ní potenciál pro další rozšíření. Kód v současném stavu není dostatečně modulární na to, aby šel využít v jiných projektech, může tedy sloužit spíše jako inspirace. Hra samotná není ve stavu hodném distribuce. Některé části kódu kvůli úspoře času využívají dočasná a nedokonalá řešení. Jejich nedostatečnost se projevila až v průběhu práce a provedení úprav nebylo prioritní. Nezdařilo se také hru optimalizovat natolik, aby na mém vlastním stroji garantovala za všech okolností plynulý běh. Jako možné rozšíření práce se tedy nabízí:

- náhrada nedostatečných či nekompletních řešení trvalými a modularizace kódu,
- optimalizace hry s využitím lepších algoritmů nebo rozložení výpočtů,
- implementace většího počtu herních mechanik
- náležité zvukové a grafické zpracování společně s uživatelským rozhraním.

Výše jmenované kroky by dle mého názoru mohly vést k úspěšnému dokončení hry, která by po vytvoření více úrovní byla připravena k distribuci.

Literatura

[1] Al Kaisy, M. *The history and meaning behind the'Stealth genre'* [online]. Gamasutra, červen 2011 [cit. 2020-03-14]. Dostupné z: https://www.gamasutra.com/blogs/MuhammadAlkaisy/20110610/7764/The_history_and_meaning_behind_the_Stealth_genre.php.

[2] Leonard, T. *Building an AI Sensory System: Examining The Design of Thief: The Dark Project* [online]. Gamasutra, 2003 [cit. 2020-03-28]. Dostupné z: https://www.gamasutra.com/view/feature/131297/building/_an/_ai/_sensory/_system/_.php.

[3] Madhav, S. *Game Programming Algorithms and Techniques: A Platform-Agnostic Approach*. Addison-Wesley Professional, 2013 [cit. 2020-05-12]. Dostupné z: https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/Game%20Programming%20Algorithms%20and%20Techniques_%20A%20Platform-Agnostic%20Approach%20%5BMadhav%202013-12-29%5D.pdf. ISBN 9780321940155.

[4] Orkin, J. Three States and a Plan: The A.I. of F.E.A.R. In: Game Developer Conference. [online]. 2006 [cit. 2020-03-16]. Dostupné z: http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf.

[5] Patel, A. *Amit's Thoughts on Pathfinding* [online]. Red Blob Games, 2014 [cit. 2020-04-10]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/.

[6] Patel, A. *Map representations* [online]. Red Blob Games, 2020 [cit. 2020-04-10]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html.

[7] Persson, M. Development of three AI techniques for 2D platform games. In: Karlstad University. [online]. 2005 [cit. 2020-04-16]. Dostupné z: https://www.diva-portal.org/smash/get/diva2:4762/FULLTEXT01.pdf.

[8] RogueBasin. *Discussion:Field of Vision* [online]. RogueBasin, červen 2014 [cit. 2020-04-16]. Dostupné z: http://www.roguebasin.com/index.php?title=Discussion:Field_of_Vision.

[9] Smrček, P. *Warhorse Studios: Problémy otevřeného světa v Kingdom Come: Deliverance* [online]. 2018 [cit. 2020-02-20]. Dostupné z: http://www.seminar.fjfi.cvut.cz/prezentace/prezentace_smrcek.pdf.

[10] Thompson, J. *Collision Detection: Line/Rectangle* [online]. [cit. 2020-05-10]. Dostupné z: http://www.jeffreythompson.org/collision-detection/line-rect.php.

[11] Vinther, A. S.-H. a Vinther, M. S.-H. *Pathfinding in Two-dimensional Worlds: A survey of modern pathfinding algorithms, and a description of a new algorithm for pathfinding in dynamic two-dimensional polygonal worlds.* Diplomová práce. [cit. 2020-04-25]. Dostupné z: https://www.cs.au.dk/~gerth/advising/thesis/anders-strand-holm-vinther_magnus-strand-holm-vinther.pdf.

[12] Wube Software. *Friday Facts #317 - New pathfinding algorithm* [online]. Říjen 2019 [cit. 2020-05-10]. Dostupné z: https://factorio.com/blog/post/fff-317.

[13] Yannakakis, G. N. a Togelius, J. *Artificial Intelligence and Games.* 1st. Springer Publishing Company, Incorporated, 2018 [cit. 2020-04-21]. Dostupné z: http://gameaibook.org/book.pdf. ISBN 3319635182.

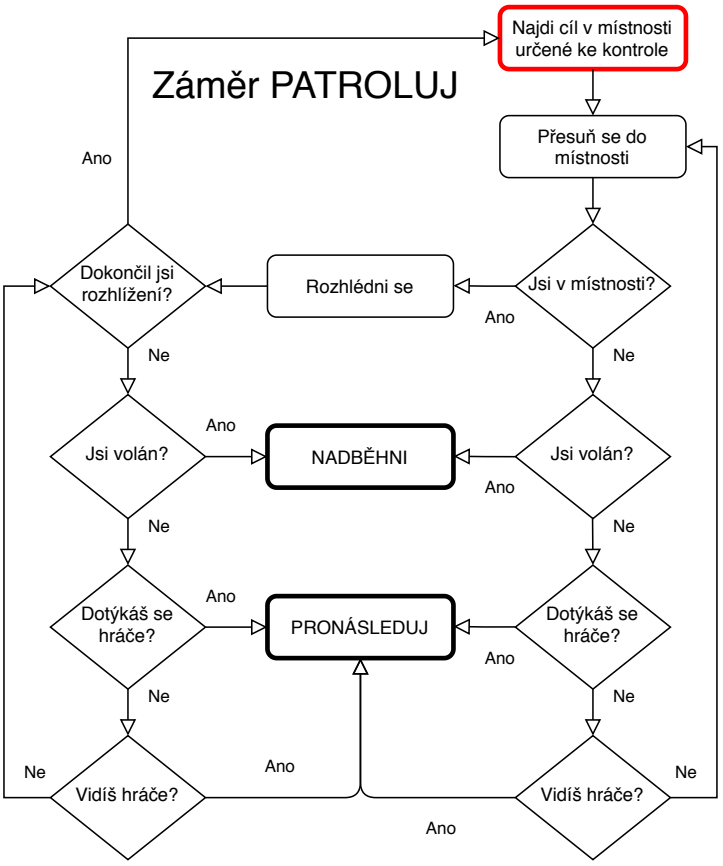
[14] Zbořil, F. a Zbořil, F. *Základy umělé inteligence IZU: Studijní opora* [online]. 2013 [cit. 2020-02-25]. Dostupné z: https://www.fit.vutbr.cz/study/courses/IZU/private/oporaizu-esf-5a.pdf.

[15] Zubek, R. Towards Implementation of Social Interaction. In: Northwestern University Computer Science. [online]. 2003 [cit. 2020-04-21]. Dostupné z: http://robert.zubek.net/publications/towards-social-interaction.pdf.

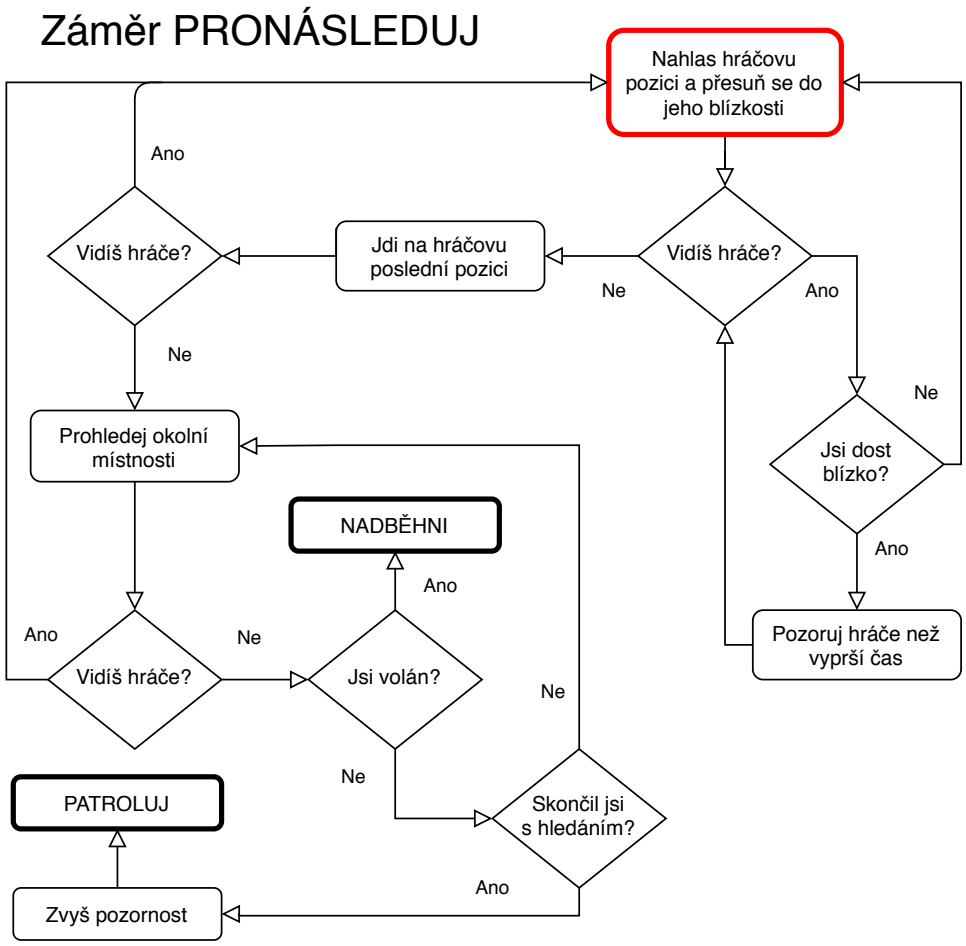
[16] Zubek, R. a Khoo, A. Making the Human Care: On Building Engaging Bots. In: Northwestern University Computer Science. [online]. 2003 [cit. 2020-04-21]. Dostupné z: https://users.cs.northwestern.edu/~khoo/downloads/papers/AAAI-SpringSymp02-MakingHumanCare.pdf.

Příloha A

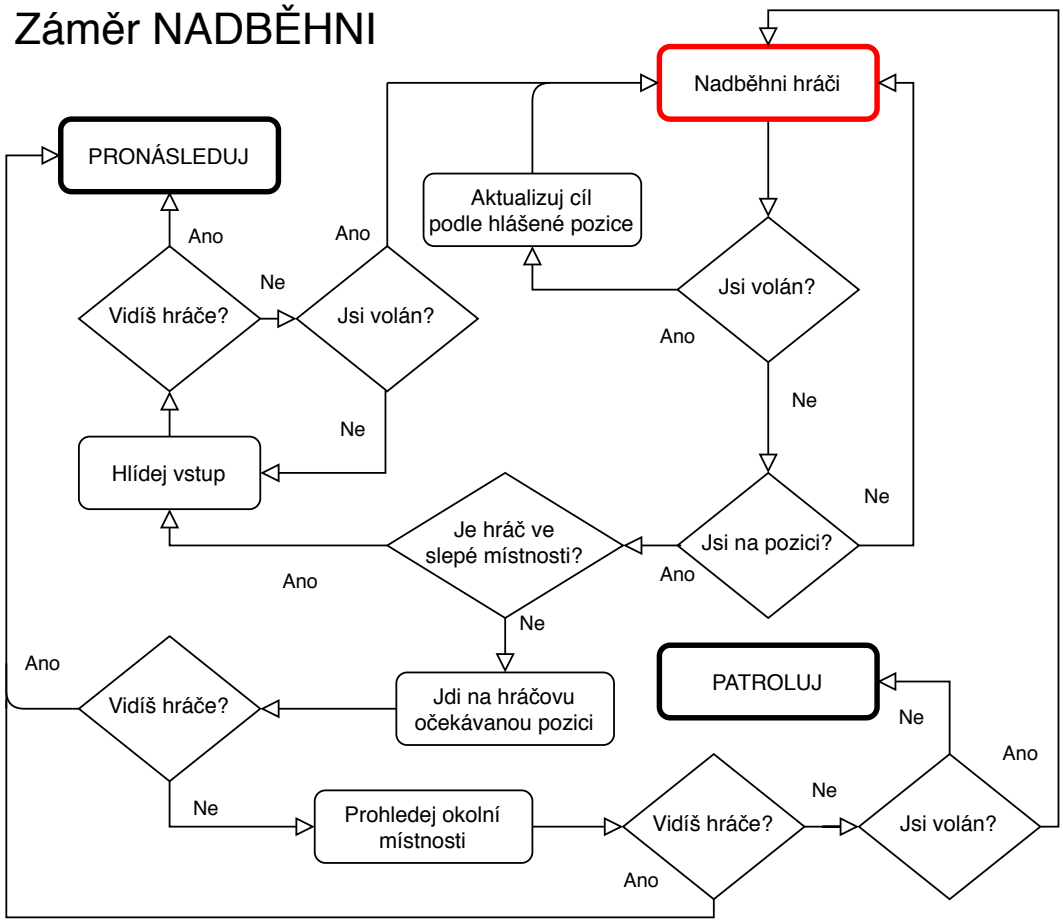
Diagramy plánů jednotlivých záměrů hlídače



Obrázek A.1: **Diagram plánu pro záměr patrolování.** Červeně je označena výchozí činnost. Tučně jsou vyznačeny přechody do stavů s jiným záměrem. Procházení místností v tomto případě znamená cyklické generování a procházení cílů v místnostech, kde má hlídač za úkol patrolovat. Volán je hlídač v případě, že jeden z jeh kolegů spatří hráče a nahlásí jeho pozici.



Obrázek A.2: **Diagram plánu pro záměr pronásledování.** Červeně je označena výchozí činnost. Tučně jsou vyznačeny přechody do stavů s jiným záměrem. V případě, že hlídač pozoruje hráče z dostatečné blízkosti, je spuštěn odpočet konce hry. Zvýšení pozornosti v tomto případě znamená, že pokud doposud hlídač nebyl ve stavu pohotovosti, kdy prohledává místnosti důkladně, měl by do něj přejít.



Obrázek A.3: **Diagram plánu pro záměr nadbíhání.** Červeně je označena výchozí činnost. Tučně jsou vyznačeny přechody do stavů s jiným záměrem. V případě, že hlídač hráči nadběhl ke slepé místnosti, hlídá její dveře. Pokud místnost slepá není, hlídač do ní vejde a pokouší se pronásledovat.

Příloha B

Obsah přiloženého CD-R

CD-R obsahuje:

- `/`
  - `game` ..... Adresář obsahující spustitelný soubor hry a vše ke spuštění potřebné
  - `lib` ..... Adresář obsahující knihovny potřebné k překladu projektu
  - `project` ..... Adresář obsahující zdrojové soubory textu práce
  - `source` ..... Adresář obsahující zdrojové soubory hry
  - `projekt.pdf` ..... Text diplomové práce
  - `readme.md` ..... Uživatelská příručka