SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta informatiky a informačných technológií

Evidenčné číslo: FIIT-16768-120896

Generátor plošinových hier

Bakalárska práca

Študijný program: informatika
Študijný odbor: informatika
Školiace pracovisko: Ústav informatiky, informačných systémov a softvérového inžinierstva
Vedúci záverečnej práce: Mgr. Matej Pecháč, PhD.

Bratislava 2025 **Branislav Trstenský**

Slovenská technická univerzita v Bratislave
Ústav informatiky, informačných systémov a softvérového inžinierstva

Fakulta informatiky a informačných technológií
2024/2025

ZADANIE BAKALÁRSKEJ PRÁCE

Autor práce:	Branislav Trstenský
Študijný program:	informatika
Študijný odbor:	informatika
Evidenčné číslo:	FIIT-16768-120896
ID študenta:	120896
Vedúci práce:	Mgr. Matej Pecháč, PhD.
Vedúci pracoviska:	doc. Ing. Ján Lang, PhD.
Názov práce:	**Generátor plošinových hier**
Jazyk, v ktorom sa práca vypracuje:	slovenský jazyk
Špecifikácia zadania:	V oblasti učenia posilňovaním - oblasť umelej inteligencie, ktorá sa zaoberá rozhodovaním agentov - sa používajú na tréning a hodnotenie staré ATARI hry. Tieto hry bežia v emulátore, co značne spomaľuje tréning. Zároveň nemáme plný prístup k ich stavu, co nám neumožňuje analyzovať agentov na celom stavovom priestore. Analyzujte existujúce ATARI hry Montezuma's Revenge a Pitfall!, ktoré patria medzi najzložitejšie a zároveň najzaujímavejšie z pohľadu výskumu. Navrhnite generátor, ktorý by dokázal vytvárať úrovne podobné týmto hrám a skopíroval ich mechaniky. Užívateľ by si mal vedieť sám generovať úrovne na základe niekoľkých parametrov, ktorými nastavuje ich zložitosť. Implementujte takýto generátor spolu s mechanikami a grafickým rozhraním v C++ alebo Python (odporúča sa). Grafika bude dostupná v rámci zadania. Sústreďte sa na to, aby efektívne a rýchlo bežalo paralelne viacero inštancií hry (napr. 128). Na záver spravte manuál, ako generátor používať a pripravte niekoľko ukážkových úrovní.
Rozsah práce:	40
Termín odovzdania práce:	12. 05. 2025

Čestne vyhlasujem, že som túto prácu vypracoval(a) samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave, 11. mája 2025.

Branislav Trstenský

Anotácia

Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Študijný program:	informatika
Autor:	Branislav Trstenský
Bakalárska práca:	Generátor plošinových hier
Vedúci diplomovej práce:	Mgr. Matej Pecháč, PhD.

Január 2025

Pri spätnoväzobovom učení, v oblasti umelej inteligencie zaoberajúcej sa rozhodovaním agentov, sa používajú na tréning a hodnotenie staré ATARI hry. Tieto hry bežia v emulátore, čo značne spomaľuje tréning. Zároveň nemáme plný prístup k ich stavu, čo nám neumožnuje analyzovať agentov na celom stavovom priestore.

Táto práca sa zaoberá vytvorením generátora, ktorý je schopný vytvoriť úrovne podobné ATARI hrám, ako sú Pitfall! alebo Montezuma's Revenge, ktoré patria medzi najzložitejšie a zároveň najzaujímavejšie z pohľadu výskumu.

Generátor umožňuje používateľovi jednoducho vytvoriť nové úrovne nastavením malého počtu parametrov. Tiež je jeho súčasťou replikácia mechaník týchto hier, čo umožňuje tieto hry spustiť pre použitie s agentom umelej inteligencie.

Annotation

Slovak University of Technology Bratislava

FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Degree course:	informatika
Autor:	Branislav Trstenský
Bachelor's Thesis:	Platform game generator
Supervisor:	Mgr. Matej Pecháč, PhD.

Január 2025

For reinforcement learning, in the field of artificial intelligence dealing with agent decision-making, old ATARI games are used for training and evaluation. These games run in an emulator, which significantly slows down training. At the same time, we do not have full access to their state, which does not allow us to analyze agents in the entire state space.

This thesis deals with the creation of a generator capable of generating levels similar to ATARI games such as Pitfall! or Montezuma's Revenge, which are among the most complex and at the same time most interesting from a research point of view.

The generator allows users to easily create new levels by setting a small number of parameters. It also includes a replica of the mechanics of these games, which allows these games to be run for use with an artificial intelligence agent.

Obsah

1 Úvod

Pri učení posilňovaním, v oblasti umelej inteligencie zaoberajúcej sa rozhodovaním agentov, sa používajú na tréning a jeho hodnotenie staré ATARI hry. Tieto hry bežia v emulátore, čo značne spomaľuje tréning. Zároveň nie je možný plný prístup k ich stavu, čoneumožňujeanalyzovať agentov na celom stavovom priestore.

Táto práca sa zaoberávytvorenímgenerátora, ktorý jeschopný vytvoriť úrovne podobné ATARI hrám, ako sú Pitfall! alebo Montezuma's Revenge, ktoré patria medzi najzložitejšie a zároveň najzaujímavejšie z pohľadu výskumu.

Generátor umožňujepoužívateľovi jednoducho vytvoriť nové úrovne nastavením malého počtu parametrov. Tiež je jeho súčasťou replikácia mechaník týchto hier, čo umožňuje tieto hry spustiť pre použitie s agentom umelej inteligencie.

Využitie aplikácie, ktorá je schopná prevádzky na moderných zariadeniach, odstraňuje značnú réžiu výkonu, v porovnaní s používaním emulátora. To umožňuje spustenie väčšieho množstva inštancií hry paralene na jednom zariadení.

Súčasťou tohto projektu je aj implementácia prostredia pre knižnicu Gymnasium, ktorá sa používa na sprostredkovanie komunikácie medzi UI agentmi a prostrediami. [5]

2 Stav existujúcich riešení

Strojové učenie agentov z vysokodimenzionálnych senzorických vstupov ako je zrak a reč, je jedným z dlhodobých cieľov učenia posilňovaním [13]. Trénovanie na situáciách z reálneho sveta, však vyžaduje veľké množstvo ručne označkovaných trénovacích dát. Algoritmy učenia posilňovaním však vyžadujú odmenu, ktorá je priamospojenás predošlými akciami agenta, takže sa vyžaduje, aby vstup pre učenie bol tvorený sekvenciou spojitých dát, ktoré sa dynamicky menia so správaním agenta.

Použitie hier umožňuje priamu reakciu na akcie agenta a konštantné generovanie senzorických vstupov priamo z grafického výstupu hry. Hry tiež obsahujú indikátory úspechu agenta, t.j. hodnoty skóre, ktoré je možné použiť ako spätnú väzbu.

Hry pre hernú konzolu ATARI sa používajú pretože ponúkajú množstvo komplexných úloh, ktoré boli navrhnuté tak, aby boli náročné pre ľudských hráčov [1]. Pre interakciu agenta, bežiaceho na modernom hardvéri a hry navrhnutej pre hernú konzolu, je použitýemulátor. Ten umožňuje vloženie akcií vybraných agentom a následné prečítanie stavu hry, pre ich vyhodnotenie prostredníctvom skóre hry. Agent samotný nemá prístup k internému stavu hry, ale len k surovým obrazovým dátam. [9, 10]

Spustenie hry na emulátore však vytvára komplikácie v podobe značne zvýšenej potreby na výpočtové prostriedky oproti pôvodnej hre. Kvôli rozdielnej architektúre medzimodernýmizariadeniami, nie je možné priamo spustiť pôvodný strojový kód hry na zariadení, ale je potrebné simulovať virtuálnu konzolu, čo spôsobuje značnú réžiu, keďže pre každú strojovú inštrukciu hry, je potrebné vykonať väčšie množstvo strojových inštrukcií na cieľovom zariadení. Existujú projekty, kde sú optimalizované výpočtové náklady pre emuláciu, napriklad použitím GPU prostredníctvom architektúry CUDA [3], ale aj v týchto prípadoch stále existuje réžia emulácie. Jedným z riešení tohto problému je re-implementácia hry pre moderné systémy alebo tvorba novej hry, ktorá je prispôsobená pre moderné systémy.

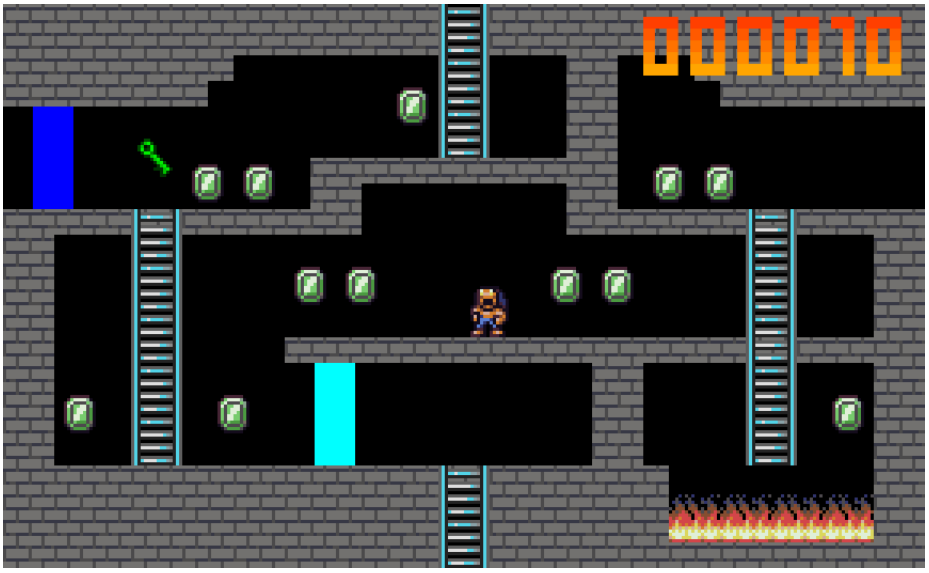
Pre strojové učenie je dôležité, aby bol model generalizovaný, t.j. zabezpečenie toho, aby model dokázal vytvoriť dobré predpovede na doposiaľ neevidovaných dátach. V kontexte učenia posilňovaním, je to schopnosť reagovať na situácie, ktoré nenastali v procese učenia. Inak môže nastať pretrénovanie, čo pre učenie posliňovaním znamená zapamätanie si konkrétneho postupu krokov pre získanie odmeny, namiesto reakcií na aktuálny stav hry.

Napríklad bolo dokázané, že hoci UI agent na riadenie simulovaného auta po konkrétnej pretekárskej trati má dobré výsledky, výsledný model sa naučil riadiť iba túto konkrétnu trať. Postupným zahrnutím náročnejších úrovní do hodnotenia zdatnosti však možno vyvinúť model, ktorý dokáže dobre riadiť mnoho tratí, dokonca aj také, ktoré by sa model nebol schopný sa naučiť od začiatku. [14]

Použitie procedurálne generovaných prostredí umožňuje trénovanie modelu nakonštatnenových prostrediach, čo predchádza zapamätaniu si iba jedného prostredia a núti model naučiť sa reagovať na dopredu neznáme situácie [2]. Pre strojové učenie je vhodné postupne zvyšovať obtiažnosť úrovní počas tréningu. Procedurálne generovanie umožňuje zmenu prostredia na základe viacerých parametrov, čo zabezpečí vhodné prispôsobenie obtiažnosti. [8]

3 Voľba spôsobu riešenia

Výsledkom tejto práce je generátor a herná aplikácia, ktorá replikuje herné mechaniky hry Montezuma's Revenge. Tvorba novej hry umožňuje spustenie kódu na moderných zariadeniach a priamy prístup k internému stavu hry. Hra Montezuma's Revenge bola vybratá ako cieľ replikácie, pre jej špecifickú štruktúru, t.j. každá úroveň je tvorená viacerými miestnosťami, čo vytvára vhodný cieľ pre procedurálnu generáciu jednotlivých úrovní.



Obr. 1: Snímka obrazovky hernej aplikácie

Hra obsahuje zberateľné odmeny vo forme drahokamov, nepriateľov rozmiestnených po miestnostiach, ako aj výzvu spočívajúcu v skokoch medzi platformami. Ich rozmiestnenie a počet je vhodným cieľom pre parametrizáciu.

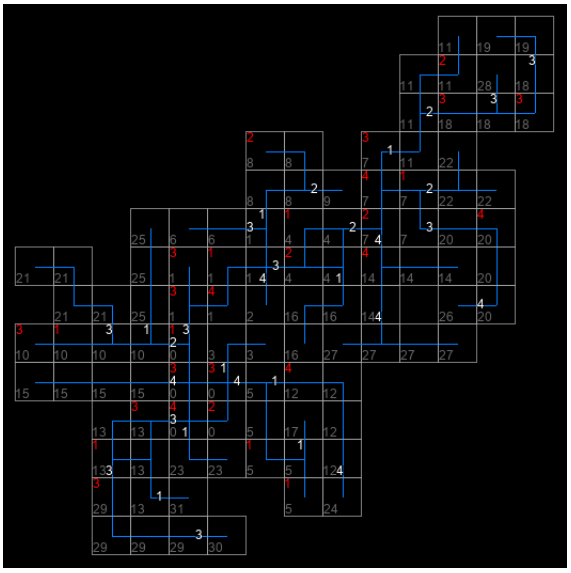
Tiež je tu výzva existujúca medzi miestnosťami, t.j. zbieranie kľúčov na odomknutie dverí pre ďalší postup cez úroveň.

Aplikácia je implementovaná v jazyku Python [12]pre jednoduchú inter-operáciu s knižnicou Gymnasium [5], ktorá sa používa na sprostredkovanie komunikácie medzi UI agentmi a prostrediami. Grafický výstup je generovaný cez knižnicu Pygame [11]. Aplikácia nemá žiadny zvukový výstup, keďže tento nie je potrebný pre tréning agenta.

4 Opis riešenia

Proces generovania úrovne spočíva vo vytvorení viacerých zón, ktoré sú tvorené miestnosťami. Zóny sú navzájom prepojené zamknutými dverami a v niektorých miestnostiach sú rozmiestnené kľúče potrebné pre ich otvorenie.

Rozmiestnenie miestností a ich prepojenia sú generované pomocou algoritmu na generovaniebludísk: prehľadávanie do hĺbky s náhodným spätným návratom [6]. Na rozdiel od bludiskových generátorov, tento proces pracuje na nekonečnej rovine, kde je obmedzený iba počet miestností (celkovo aj v jednotlivých zónach). Generovaná miestnosť môže tvoriť začiatok novej zóny, kde vstup do tejto miestnosti je zamknutý a zároveň je jediným vstupom do tejto zóny. Postupným tvorením zón od začiatočného bodu, sa definuje hierarchia, kde podradené zóny sú ďalej od začiatočného bodu. Je zabezpečené, aby každý zamknutý prechod do podradenej zóny, mal priradený kľúč, umiestnený niekde v jednej z nadradených miestností.



Obr. 2: Vygenerovaná úroveň
Spojenie miestností je zobrazené modrou čiarou. Bielym textom je zobrazené, aký kľúč je potrebný na otvorenie prechodu a červeným textom je zobrazené, aký kľúč sa v miestnosti nachádza.

Obsah miestností je tvorený náhodným generátorom a kombinovaním dopredu vytvorených súčastí, podľa špecifikovaných parametrov. Každá súčasť obsahuje herné objekty, ktoré tvoria prostredie pre interakciu shráčskoupostavou, a zástupné objekty, ktoré sú rekurzívne nahradzované ďalšími súčasťami podľa daných parametrov.

Výsledkom tejto práce je aj program editora, ktorý umožňuje prostredníctvom grafického rozhrania definovať tieto súčasti, ich objekty a špecifikovať parametre pre ich generovanie.

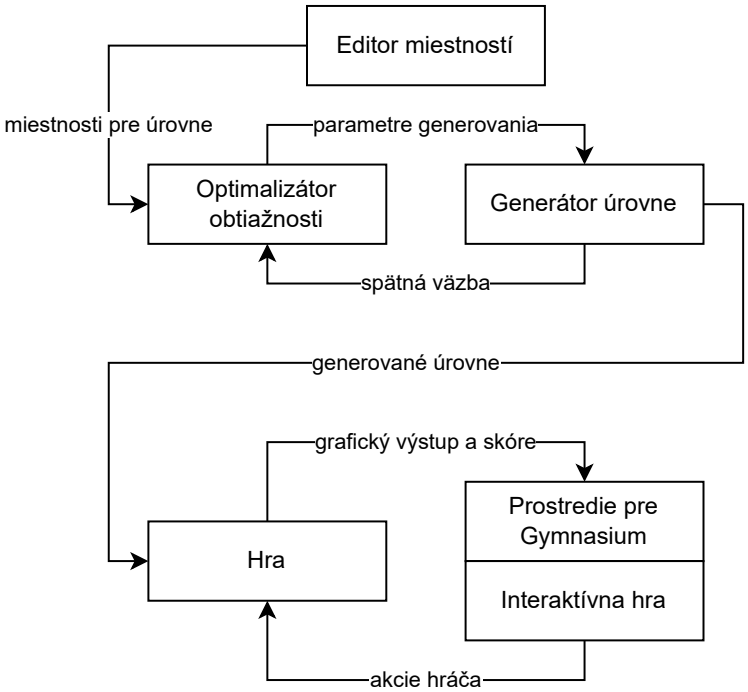
Vygenerované úrovne sú hrateľné prostredníctvom hernej aplikácie. Aplikácia obsahuje grafický výstup, ako aj jednoduchú fyzikálnu simuláciu. Hra poskytuje štyri možné akcie: pohyb vľavo a vpravo, skok a lezenie hore a dole.

4.1 Špecifikácia požiadaviek

Projekt je rozdelený na viacero komponentov, z ktorých každý tvorí jednu vrstvu jeho funkcionality:

- Editor miestností
- Optimalizátor obtiažnosti
- Generátor úrovne
- Hra
- Prostredie pre Gymnasium

V editore vývojár definuje miestnosti, a ich súčasti, ktoré budú použité pri generovaní úrovne. Generátor úrovne následne tieto miestnosti použije pre generovanie úrovne a výsledné úrovne sú Optimalizátorom obtiažnosti ohodnotené a na základe ich obtiažnosti sú parametre generovania upravené tak, aby sa dosiahla obtiažnosť zvolenápoužívateľom.



Obr. 3: Komponenty projektu

Finálna úroveň je následne vložená do Hry, ktorá generuje grafický výstup na základe používateľského vstupu. S hrou interaguje buď reálny človek, pri interaktívnom použití, alebo umelá inteligencia, pri použití prostredia pre Gymnasium.

4.1.1 Funkčné požiadavky

Kód	Názov	Popis
F_K	**Kompatibilita**	**Použitie s modelom prostredníctvom knižnice Gymnasium**
F_K_01	Vytvorenie prostredia pre Gymnasium	Súčasťou projektu je vytvorenie prostredia pre použitie s modelom prostredníctvom knižnice Gymnasium.

F_K_02	Vstup do prostredia v Gymnasium	Prostredie prijíma ako vstup akcie equivalentné vstupu od používateľa.
F_K_03	Výstup z prostredia v Gymnasium.	Prostredie ako výstup ponúka aktuálne skóre a grafický výstup obdobný výstupu na obrazovku.
F_INT	**Interakcia s používateľom**	**Vstup a spätná väzba programu**
F_INT_01	Program prijíma používateľský vstup	Program prijíma používateľský vstup ako akcie vyjadrené stlačením tlačidla na klávesnici. Program prijíma nasledovné akcie: ísť hore, ísť dole, ísť doprava, ísť doľava a skok.
F_INT_02	Grafické znázornenie herných objektov.	Každý herný objekt musí byť reprezentovaný individuálnym grafickým prvkom.
F_INT_03	Program indikuje skóre	Na obrazovke je graficky znázornené aktuálne skóre pomocou textu.
F_INT_04	Program indikuje obsah inventáru	Na obrazovke sú graficky znázornené predmety, obsiahnuté v inventári, použitím grafických objektov sidentickýmobsahom ako herné objekty, ktoré sú zdrojom týchto predmetov.
F_PC	**Hráčska postava**	
F_PC_01	Horizontálny pohyb	Program umožňuje pohyb hráčovej postavy na zemi.
F_PC_02	Program umožňuje skok zo zeme	Program umožňuje skok hráčovej postavy, ak je v danom momente na zemi.
F_PC_03	Kolízia s miestnosťou	Program rieši kolíziu hráčovej postavy a geometrie miestnosti tak, aby nebolo možné prejsť cez steny, preskočiť cez strop alebo prepadnúť cez podlahu.
F_PC_04	Program umožňuje lezenie po rebríku	Program umožní hráčovi vstúpiť na rebrík stlačením tlačidla skoku, keď je hráčska postava v kontakte s rebríkom. Hráčovi je umožnené liezť hore a dole a tlačidlom skoku skočiť z rebríka. Stav lezenia je potrebné udržať pri prechode medzi miestnosťami.
F_PC_05	Program umožňuje zbieranie predmetov do inventára	Pri kolízii hráčskej postavy a herného objektu, ktorý reprezentuje zbierateľný predmet, je tento herný objekt odstránený a zároveň je predmet vložený do inventára. Inventár je obmedzený; v prípade že je inventár plný, je táto interakcia zablokovaná.

F_PC_06	Program umožňuje použitie predmetov z inventára	Interakcia je spustená automaticky pri kolízií s interaktívnym herným objektom. Pre každý herný objekt je aplikovateľný iba jeden predmet, takto nemôže vzniknúť nejasnosť, ktorý predmet má byť použitý. Použitý predmet je odobratý z inventára, ak inventár obsahuje viacero aplikovateľných predmetov, je použitý iba jeden.
F_PC_07	Program reaguje na kontakt z nebezpečnými objektami.	Pri kolízií hráčskej postavy a nebezpečného objektu, je pozícia hráča nastavená na vstup do aktuálnej miestnosti, ktorý bol naposledy hráčom použitý.
F_GW	**Herný svet**	
F_GW_01	Herný svet je rozdelený na oddelené miestnosti	Miestnosti ani objekty v nich neinteragujú s inými miestnosťami.
F_GW_02	Stav objektov relevantných pre postup hry je zachovaný, aj keď hráč opustí miestnosť	Daný stav je zachovaný: stav dverí, prítomnosť kľúčov, prítomnosť drahokamov a výsledok výberu zástupných objektov.
F_GW_03	Miestnosti sú umiestnené v mriežke	Miestnosti v susedných bunkách sú nazývané susedné. Susedné miestnosti môžu, ale nemusia mať spojenie. Miestnosť má 0 až 4 susedov, maximálne jeden sused pre každý zo štyroch smerov.
F_GW_04	Úroveň udáva zoznam miestností a ich pozíciu	
F_GW_05	Pre každú miestnosť je uložený stav spojenia pre všetky susedné miestnosti	Existuje práve jedno spojenie pre každú susednú miestnosť. Možné stavy sú: zatvorený prechod, otvorený prechod, prechod na kľúč (špecifikovaný druh kľúča). Ak neexistuje sused pre daný smer, je stav nastavený ako zatvorený.
F_GW_06	Je umožnený prechod medzi miestnosťami	Ak je spojenie definované ako zatvorené, prechod je blokovaný stenou. Ak je spojenie otvorené, prechod je voľný. Ak je spojenie na kľúč, sú postavené do cesty dvere. Prechod je realizovaný herným objektom, ktorý načíta spojenú miestnosť pri kontakte s hráčom a prenesie hráča do nej. Tento objekt je vytvorený, iba ak prechod nie je zatvorený.
F_GW_07	Každá miestnosť môže, ale nemusí obsahovať kľúč	V každej miestnosti môže existovať maximálne jeden kľúč.

F_GW_08	Každá miestnosť obsahuje herné objekty	Výber objektov je obmedzený tak, aby obsah spĺňal požiadavky kompatibility so stavmi vstupov miestnosti a prítomnosť kľúča (ak ho miestnosť má).
F_CH	**Výzva**	
F_CH_01	Miestnosti poskytujú výzvu na zručnosti pohybu	Výzva je tvorená prostredníctvom potreby skokov medzi platformami a nebezpečenstvom pádu do ohňa.
F_CH_02	Miestnosti poskytujú výzvu na vyhýbanie sa nepriateľom	
F_CH_03	Celková úroveň poskytuje výzvu na orientáciu	Výzva je tvorená potrebou nájdenia kľúčov, nutných na pokrok a nájdenie cieľa úrovne.
F_CH_04	Optimalizácia obtiažnosti úrovne	Program má možnosť upraviť parametre generovania úrovne tak, aby mala požadovanú obtiažnosť.
F_PG	**Pokrok**	
F_PG_01	Zbieranie kľúčov na prechod cez zamknuté dvere	Každá miestnosť môže, ale nemusí obsahovať kľúč. Dvere aj kľúčemajúšpecifikovanú farbu a na dvere je aplikovateľný iba kľúč rovnakej farby. Nemôže vzniknúť situácia, kedy hráč nemá možnosť nájsť kľúč ku každým zamknutým dverám.
F_PG_02	Zbieranie drahokamov na zvýšenie skóre	
F_PG_03	Zbieranie špeciálnych kľúčov na otvorenie cieľa úrovne	Namiesto farebných kľúčov môže miestnosť obsahovať špeciálny kľúč na otvorenie cieľa úrovne. V úrovni je rozmiestnený presný počet týchto kľúčov na otvorenie cieľa. Tieto kľúče sú graficky odlíšené od farebných kľúčov.
F_PG_04	Cieľ úrovne	V úrovni je jedna miestnosť zvolená ako cieľová. V danej miestnosti je umiestnený herný objekt, reprezentujúci cieľ úrovne. Táto miestnosť neobsahuje žiadne nebezpečenstvo.
F_PG_05	Otvorenie cieľa úrovne	Cieľ úrovne je zamknutý. Herný objekt cieľa indikuje počet špeciálnych kľúčov potrebných pre jeho otvorenie. Keď sa hráč priblíži k cieľu a má v inventári špeciálne kľúče, sú kľúče odobraté a počet potrebných kľúčov je znížený. Ak je počet potrebných kľúčov nulový, cieľ úrovne sa odomkne.
F_PG_06	Výhra	Pri kontakte hráča s odomknutým cieľom je hra ukončená.

F_T	**Testovanie**	**Program obsahuje funkcie, ktoré uľahčujú vývoj programu. Tieto funkcie nie sú určené pre hráča.**
F_T_01	Program umožňuje zobrazenie mapy celkovej úrovne	Mapa je otvorená stlačením tlačidla, ktoré nie je súčasťou štandardných vstupov hry. Na mape sú zobrazené všetky miestnosti spolu so stavom prechodov, aktuálna pozícia hráča, poloha kľúčov a cieľa.
F_T_02	Program umožňuje voľný prechod medzi miestnosťami	Stlačením tlačidiel, ktoré nie sú súčasťou štandardných vstupov hry, je možná teleportácia hráča do každej zo susedných miestností.
F_ED	**Editor**	**Program obsahuje editor, ktorý umožňuje návrh miestností.**
F_ED_01	Editor umožňuje úpravu miestností	Miestnosť je zvolená vložením cesty k súboru, ktorý definuje miestnosť, ako parameter pri spustení editora. Ak súbor neexistuje, je vytvorená prázdna miestnosť.
F_ED_02	Editor je spustený ako separátny program	
F_ED_03	Editor umožňuje vyskúšať vytvorenú miestnosť bez potreby spustenia hry	Stlačením klávesovej skratky je práve editovaná miestnosť duplikovaná a je do nej pridaná hráčska postava. Funkcionalita editora je počas testu zastavená. Opätovným stlačením klávesovej skratky, je test ukončený.
F_ED_04	Editor automaticky ukladá zmeny	Po každej zmene sú dáta miestnosti uložené do súboru.
F_ED_05	Editor umožňuje vytvoriť inštanciu herných objektov z dopred daného zoznamu	Stlačením šípky hore alebo dole na klávesnici, je možné zo zoznamu objektov vybrať želaný objekt. Písaním na klávesnici je možné zoznam filtrovať. Stlačením tlačidla myši, je zvolený objekt vytvorený na polohe kurzora.
F_ED_06	Editor umožnuje presun a zmenu veľkosti objektov	Kliknutím na objekt je objekt zvolený. Ťahaním objektu kurzorom je možné objekt presunúť. NA stranách zvoleného objektu sú zvýraznené 4 oblasti, ktoré je možné ťahať myšou a tak zmeniť jeho veľkosť.
F_ED_07	Editor umožňuje odstránenie objektov	Ak je zvolený objekt, stlačením klávesovej skratky je zvolený objekt vymazaný.
F_ED_08	Editor umožňuje zmenu parametrov ovplyvňujúcu nahradenie zástupných objektov	Ak je zvolený objekt, ktorý podporuje konfiguráciu, písaním na klávesnici je možné nastaviť jeho parametre. V tomto stave je filtrovanie obsahu zoznamu objektov pozastavené.

F_ED_09	Editor umožňuje vrátiť zmeny	Stlačením klávesovej skratky je posledná zmena vrátená. Ak nebola od vrátenia vykonaná žiadna zmena, je možné inou klávesovou skratkou zmenu znovu aplikovať.

4.1.2 Nefunkčné požiadavky

Kód	Názov	Popis
NF_01	Načítanie objektov miestností	V pamäti sú načítané herné objekty iba aktuálnej miestnosti, v ktorej je hráč prítomný.
NF_02	Súbory miestností	Miestnosti sú uložené vo formáte `json`. Súbory sú prečítané raz pri spustení programu.
NF_03	Program	Program je napísaný v jazyku Python.
NF_04	Grafické rozhranie	Pri použití programu používateľom je vytvorené okno, presne také veľké, aby obsahovalo jednu miestnosť.
NF_05	Grafické zdroje	Všetky grafické prvky sú obsiahnuté v jednom súbore, formátovanom ako atlas, ktorý je načítaný pri spustení programu.

4.2 Návrh

Použitie programu je rozdelené na dve štádiá:

1. Návrh a testovanie miestností
2. Použitie programu na trénovanie agenta umelej inteligencie

Súčasťou projektu je vytvorenie počiatočnej kolekcie miestností a ich súčastí, ktoré budú distribuované spoločne s programom, a sú dostatočné pre úspešné vykonanie štádia 2. Koncový používateľ teda nemá potrebu používať editor, ale vždy má možnosť upraviť alebo vytvoriť miestnosti, či ich súčasti, pre svoje potreby.

Pre štádium 2, používateľ musí vykonať tieto kroky:

1. Špecifikácie požadovanej obtiažnosti
2. Spustenie optimalizátora obtiažnosti a výber najlepšej úrovne
3. Inicializácia hry
4. Spustenie interaktívnej hry alebo prostredia pre Gymnasium

4.2.1 Generátor úrovne

Generovanie úrovne je rozdelené na viacero etáp:

1. Rozloženie miestností úrovne
2. Položenie špeciálnych kľúčov a cieľa úrovne
3. Rozloženie kľúčov pre odomknutie dverí
4. Výber obsahu miestností

V prvej etape sú vytvorené miestnosti, ktoré ešte nemajú žiadny obsah, a sú vytvorené spojenia medzi nimi. Generovanie začína koreňovou miestnosťou, z ktorej sa postupne úroveň rozrastá o ďalšie miestnosti.

V druhej etape sú rozmiestnené špeciálne kľúče a cieľ úrovne. Ich polohy sú dané parametrami generátora, ktoré sú dané optimalizátorom obtiažnosti. Ich poloha sa teda mení tak, aby sa dosiahla požadovaná obtiažnosť.

V tretej etape sú rozmiestnené kľúče na odomknutie dverí. Je vytvorený jeden kľúč pre každé zamknuté dvere a kľúč je umiestnený tak, aby ho hráč mohol získať bez odomknutia dverí, ktoré odomyká.

Vo štvrtej etape sú vybraté obsahy miestností. Z množiny všetkých koreňových súčastí sú vybrané také, ktoré majú prechody zhodné s prechodmi miestnosti a súčasne, ak je v miestnosti vytvorený kľúč, tak musí mať koreňová súčasť stanovenú polohu pre tento objekt.Špeciálne prípady sú miestnosti so špeciálnym kľúčom alebo cieľom úrovne. Tieto majú separátne množiny. Z výberu je následne náhodne vybraná koreňová súčasť.

4.2.2 Optimalizátor obtiažnosti

Optimalizátor používa jednoduchý genetický algoritmus pre výber parametrov pre generátor úrovne [15]. Pre každý set parametrov je vygenerovaná úroveň a je vyhodnotená jej zhoda s vybranou obtiažnosťou. Sety parametrov s najlepšou zhodou, sú potom použité pre vytvorenie následnej generácie.

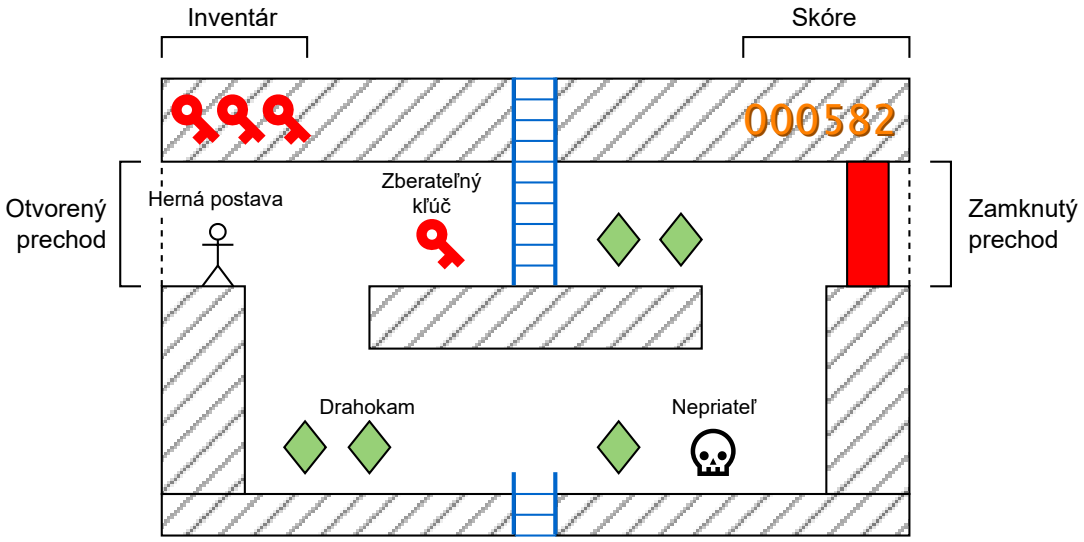
Pre vyhodnotenie zhody je pre danú úroveň nájdená optimálna cesta pre zber všetkých špeciálnych kľúčov a dosiahnutie cieľa úrovne.

4.2.3 Hra

V hernej časti hráč prechádza cez miestnosti s cieľom postupne zbierať všetky špeciálne kľúče a dosiahnuť cieľ úrovne. Obrazovka hry vždy zobrazuje presne jednu miestnosť. Obsah miestností je reprezentovaný hernými objektami. Na obrazovke sa tiež zobrazuje obsah inventára a aktuálne skóre.

Hlavné herné objekty sú:

- Herná postava
- Steny miestnosti ⇒ predstavujú bariéru, ktorá zabraňuje hráčovi opustiť miestnosť mimo daných prechodov, ale aj platformy, ktoré sú súčasťou výzvy skokom
- Rebrík ⇒ umožňuje vertikálny pohyb
- Drahokamy ⇒ keď ich hráč zbiera, zvyšujú jeho skóre o konštantnú hodnotu
- Nepriatelia ⇒ pri kontakte s hráčom je vrátený na vstup do miestnosti, cez ktorý vošiel



Obr. 4: Herná obrazovka

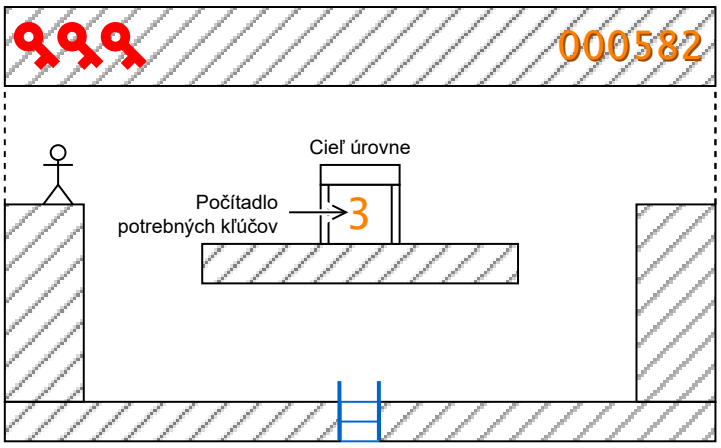
- Zberateľný kľúč ⇒ pri kontakte s hráčom sa presunie do inventára, môže to byť farebný alebo špeciálny kľúč
- Otvorený prechod ⇒ umožňuje hráčovi prejsť do susednej miestnosti
- Zamknutý prechod ⇒ ak má hráč v inventári kľúč zhodnej farby, tento kľúč je odstránený a prechod sa zmení na otvorený
- Cieľ úrovne ⇒ pri kontakte s hráčom, ak má hráč v inventári špeciálny kľúč, je tento kľúč odstránený a zníži sa počítadlo potrebných kľúčov; keď sa počítadlo dostane na nulu, cieľ sa otvorí a pri kontakte s hráčom je hra ukončená

Súčasťou hernej časti je aj fyzikálny systém, ktorý simuluje gravitáciu, umožňuje detekciu kolízií hernej postavy s interaktívnymi objektmi a nepriateľmi a taktiež zabraňuje prechádzanie cez steny.

4.2.4 Prostredie pre Gymnasium

Prostredie pre Gymnasium [5]berie ako vstup 5 členný vektor, kde každý prvok je buď 1 alebo 0 a reprezentuje jeden z možných vstupov hráča:

- Skok
- Pohyb hore
- Pohyb vpravo
- Pohyb vľavo
- Pohyb dole



Obr. 5: Miestnosť s cieľom úrovne

Ako výstup, prostredie ponúka grafický výstup hry, aktuálne skóre a hodnotu odmeny. Odmena sa počíta ako zmena skóre, takže keď sa napriklad skóre zvýši o 10, tak odmena v danom kroku bude 10.

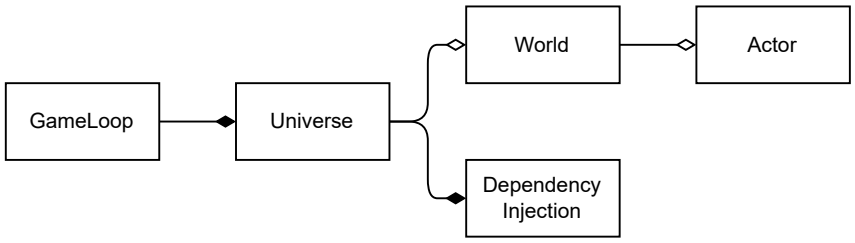
4.3 Implementácia

Aplikácia je implementovaná v jazyku Python [12]pre jednoduchú inter-operáciu s knižnicou Gymnasium [5], ktorá sa používa na sprostredkovanie komunikácie medzi UI agentmi a prostrediami. Grafický výstup je generovaný cez knižnicu Pygame [11].

Prístup ku grafickému výstupu a používateľskému vstupu je abstrahovaný tak, aby sa kód hry nemusel modifikovať medzi interaktívnym riadením človeka alebo riadením agenta umelej inteligencie.

Hra sa skladá z herných objektov, ktoré sú umiestnené do sveta, ktorý je následne umiestnený do vesmíru. Táto architektúra je implementovaná týmito triedami:

- `Actor` ⇒ Nadtrieda všetkých herných objektov
- `World` ⇒ Reprezentuje herný svet
- `Universe` ⇒ Kontajner pre objekty, s ktorými herné objekty interagujú
- `DependencyInjection` ⇒ Umožňujú herným objektom získať referencie na podporné objekty
- `GameLoop` ⇒ Umožňuje interakciu používateľa s hrou



Obr. 6: Štruktúra hry

4.3.1 Herné objekty

Herné objekty sú inštanciami tried odvodených z triedy `Actor`. Prepísaním metód a modifikáciou polí nadtriedy, majú herné objekty definovanú svoju interakciu s fyzikálnym systémom, grafickú reprezentáciu a ich správanie pri generovaní miestností.

Aby boli herné objekty funkčné, musia byť pridané do sveta. Svet je trieda, ktorá riadi životnosť herných objektov, ich vykresľovanie a interakciu medzi nimi. V tom istom čase môže existovať viacero svetov, ale iba jeden je aktívny - čo znamená, že v ňom prebieha čas, je vykresľovaný na obrazovku a hráč s ním môže interagovať.

Aktívny svet je uložený v objekte triedy `Universe`. Trieda `Universe` taktiež umožňuje herným objektom prístup k inštancii tried, prostredníctvom ktorých zakresľujú svoju grafickú reprezentáciu na obrazovku a prijímajú vstup odpoužívateľa.

Všetky herné objekty obsahujú nasledovné polia:

- Pozícia ⇒ dvojrozmerný vektor, ktorý určuje polohu objektu vo svete
- Veľkosť ⇒ dvojrozmerný vektor, ktorý určuje veľkosť objektu
- Referencia na objekt sveta, v ktorom sa objekt nachádza
- Referencia na objekt vesmíru, v ktorom sa objekt nachádza
- Vrstva ⇒ enumerácia, ktorá udáva prioritu pre vykresľovanie na obrazovku
- Kolízne príznaky ⇒ definujú spôsob interakcie s objektom vo fyzikálnom systéme

Herný objekt môže patriť do jednej z nasledovných vrstiev, pričom objekt v neskoršej vrstve je vždy vykreslený nad objektom v predchádzajúcej vrstve.

- Pozadie
- Normálny
- Grafické rozhranie

Herný objekt môže mať nasledovné kolízne príznaky:

- Trigger ⇒ umožňuje reagovať na prekryt s iným objektom
- Statický ⇒ fyzikálny systém predchádza kolízii s daným objektom

Triedy definujúce herné objekty môžu reagovať na rôzne udalosti buď v hernom svete alebo v ich životnom cykle, prostredníctvom prepísania metód. Udalosti, na ktoré triedy môžu reagovať, sú nasledovné:

- Vykreslenie
- Posun času
- Vytvorenie objektu
- Pridanie objektu do sveta
- Odstránenie objektu zo sveta
- Prekryt s objektom typu trigger

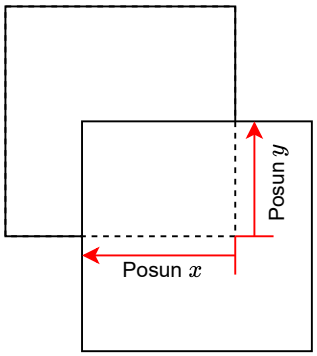
4.3.2 Fyzikálny systém

Pre umožnenie pohybu hráča a jeho interakciu s objektami v miestnosti, aplikácia implementuje jednoduchý fyzikálny systém. Je zavedená iba jednoduchá detekcia kolízií a statické riešenie kolízií.

Pre detekciu kolízií sú objekty redukované na jednoduchý obdĺžnik, odvodený od ich pozície a veľkosti. Objekty, ktoré majú nastavené kolízne príznaky trigger alebo statické, sú uložené do polí. Keď chce objekt zistiť či je v kolízii s jedným z týchto objektov, je jeho obdĺžnik porovnaný s obdĺžnikmi týchto objektov.

V prípade detekcie objektov typu trigger, je pri detekcii prekrytu spustená na obidvoch kolíznych objektoch metóda obsluhujúca túto udalosť.

V prípade detekcie statických objektov, je kolízia vyriešená posunom kolízneho objektu - konkrétne toho, ktorý nie je statický. Porovnaním rozdielu hrán obdĺžnikov reprezentujúcich obidva objekty v osi x a y je vypočítaný najmenší možný posun, ktorý je následne aplikovaný na kolízny objekt.



Obr. 7: Riešenie kolízií

Kolízie sú spracované len pre objekty, ktoré si to explicitne vyžiadajú. Konkrétne sa tento systém používa v týchto prípadoch:

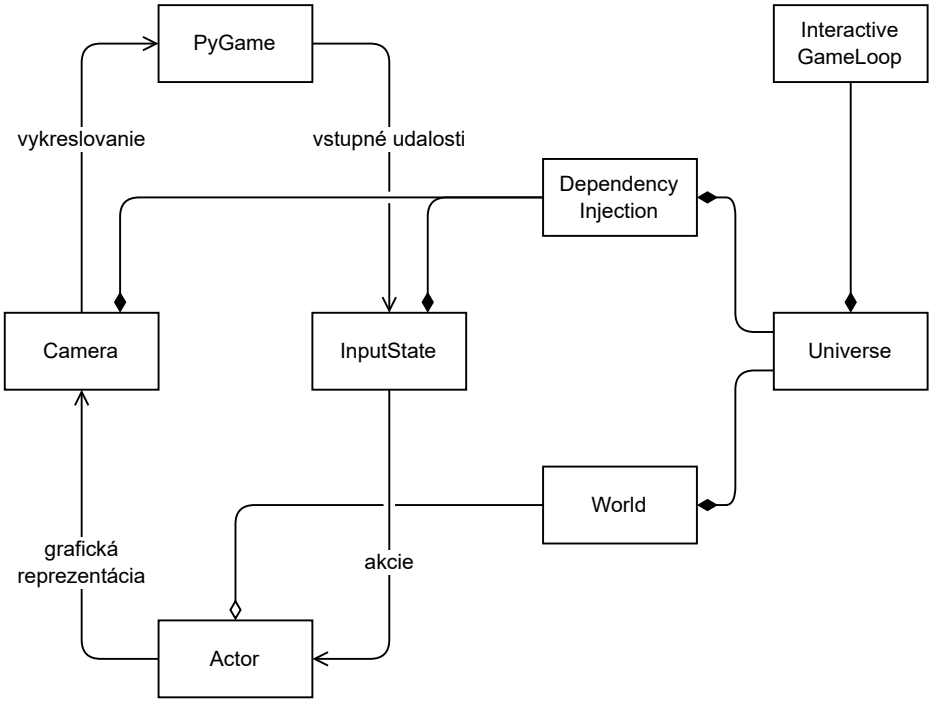
- Kolízia hráča so stenami
- Kolízia nepriateľov so stenami
- Interakcia hráča s interaktívnymi objektmi

- Interakcia hráča s nepriateľmi

4.3.3 Herná slučka

Interakciu herných objektov a hráča sprostredkúvajú podporné objekty, ktoré sú vytvorené triedou `GameLoop` a registrované v registri `DependencyInjection`. Keď chcú herné objekty tvoriť grafický výstup alebo konzumovať vstup od používateľa, vyžiadajú si inštanciu týchto objektov z registra. Podporné objekty pre tento účel sú nasledovné:

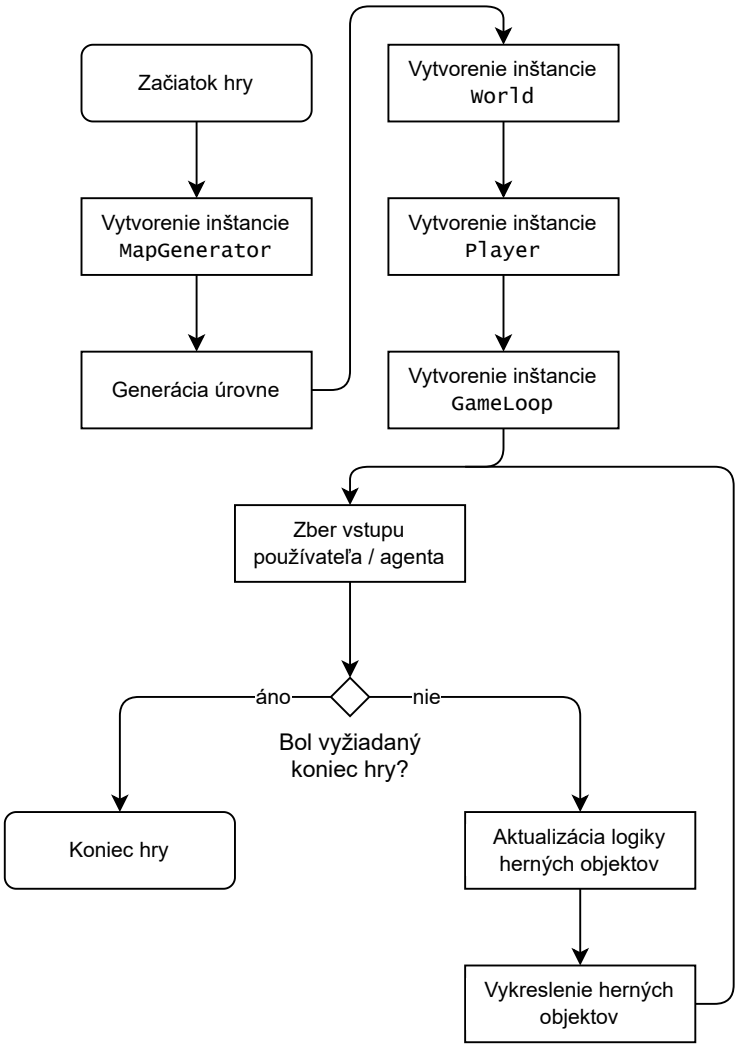
- `Camera` ⇒ Generuje grafickú reprezentáciu herných objektov
- `InputState` ⇒ Poskytuje prístup k vstupným akciám hráča



Obr. 8: Funkcia podporných objektov pre interakciu s používateľom

Pri použití hry človekom sa používa podtrieda `InteractiveGameLoop`, ktorá vytvorí herné okno a zbiera vstup z klávesnice. Pri použití hry agentom umelej inteligencie sa modifikuje stav v `InputState` podľa rozhodnutí agenta a grafický výstup hry sa uloží do poľa pixelov, ktorý tvorí vstup pre agenta.

Pred spustením hry je vygenerovaná úroveň prostredníctvom triedy `MapGenerator`. Následne je vytvorený `World` pre prvú miestnosť a je do nej umiestnený hráč. Nakoniec je vytvorená inštancia `GameLoop` a herná slučka sa vykonáva, až kým nie je zatvorené herné okno človekom, resp. kým nie je ukončené trénovanie agenta. Proces hernej slučky je ilustrovaný v diagrame 9.



Obr. 9: Herná slučka

4.3.4 Generovanie

Úroveň je tvorená množinou miestností umiestnených na mriežke. Počas generovania si program udržiava: zoznam existujúcich miestností, zoznam čakajúcich miestností a ukazovateľ na aktuálnu miestnosť a na aktuálnu zónu.

Každá miestnosť má definované pole spojení. Pole obsahuje štyri prvky, kde každý prvok reprezentuje jeden smer. Každý prvok je celé číslo s jednou z nasledujúcich hodnôt:

- $-1 \Rightarrow$ žiadne spojenie
- $0 \Rightarrow$ spojenie, ktoré nie je zamknuté
- $\geq 1 \Rightarrow$ typ kľúča, ktorým je spojenie zamknuté

Typy kľúčov sú rozlišované na základe farby, kde ľubovoľný kľúč istej farby otvára dvere prislúchajúce tejto farbe. Všetky možné kľúče sú definované jednoduchým poľom farieb, takže je možné ich počet zmeniť alebo prispôsobiť odtiene farby, ak má agent problém ich rozlíšiť.

Taktiež každá miestnosť obsahuje jej pozíciu na mriežke úrovne a číslo zóny, v ktorej sa nachádza. Pri generovaní sa pre každú miestnosť udržiava pole smerov, ktoré ešte neboli preskúmané.

Každá zóna je označená poradovým číslom. Pre každú zónu je udržiavané pole všetkých miestností, ktoré obsahuje.

Prvým krokom generátora je vytvoriť rozloženie miestností. Program najprv vytvorí miestnosť na pozícií $[0, 0]$ a vloží ju do zoznamu existujúcich a čakajúcich miestností. Následne podľa parametrov generuje nové miestnosti až pokiaľ nedosiahne rozloženie spĺňajúce požiadavky definované parametrami.

Parametre generátora sú:

- Maximálny počet miestností
- Maximálna šírka úrovne
- Maximálna výška úrovne
- Minimálny počet miestností pre zónu
- Maximálny počet miestností pre zónu
- Veľkosť štartovacej zóny
- Šanca vytvoriť uzamknuté spojenie
- Šanca vytvoriť spätný návrat
- Poloha špeciálnych kľúčov
- Poloha cieľa úrovne

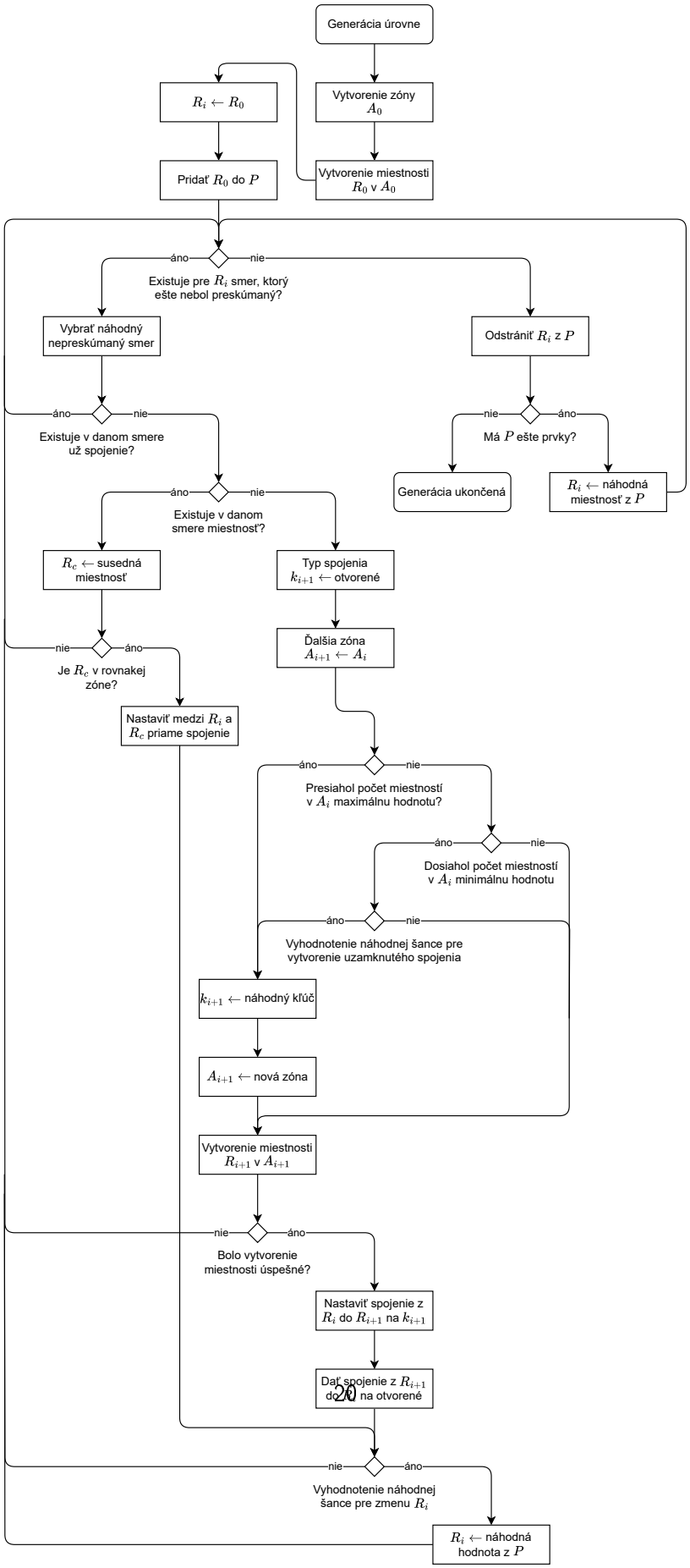
Stav generátora je nasledovný:

- $A_i \Rightarrow$ aktuálna zóna
- $R_i \Rightarrow$ aktuálna miestnosť
- $A \Rightarrow$ zoznam všetkých zón
- $R \Rightarrow$ zoznam všetkých miestností
- $P \Rightarrow$ zoznam čakajúcich miestností
- $x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}} \Rightarrow$ minimálne a maximálne dosiahnuté hodnoty pozície miestností
- $K \Rightarrow$ zoznam potrebných kľúčov

Priebeh generovania rozloženia miestností je ilustrovaný v diagrame 20. Generovanie sa odvoláva na nasledovné procedúry:

- **Procedúra** `pridať_miestnosť`
  - **Vstup:** pozícia miestnosti L, číslo zóny a
  - **Výstup:** vytvorená miestnosť
  1. **Ak** je pozícia miestnosti mimo rozsahu definovaného $x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}$
     1.1. Je vypočítaný nový rozsah
     1.2. **Ak** je definovaná maximálna šírka alebo výška a nový rozsah ju prekračuje
        1.2.1. Funkcia je prerušená
     1.3. Rozsah je aktualizovaný na nový rozsah
  2. Vytvorená miestnosť R_{i+1} z pozíciou L a číslom zóny a
  3. Miestnosť R_{i+1} pridaná do zoznamu miestností R
  4. Miestnosť R_{i+1} pridaná do zoznamu miestností v zóne A_a
  5. **Výstup** R_{i+1}
- **Procedúra** `pridať_zónu`
  - **Vstup:** nadradená zóna A_i, ak existuje
  - **Výstup:** vytvorená zóna
  1. $i \leftarrow$ poradové číslo zóny
  2. Vytvorená zóna A_{i+1}, s poradovým číslom i a nadradenou zónou A_i
  3. Zóna A_{i+1} pridaná do zoznamu zón A
  4. **Výstup** A_{i+1}

Po vytvorení rozloženia miestností, program rozmiestni kľúče potrebné na otvorenie vytvorených uzamknutých spojení. Najprv sú umiestnené špeciálne kľúče a cieľ úrovne. Program iteruje cez všetky zamknuté prechody v poradí od tých v zónach najbližších k začiatku úrovne, až k tým najvzdialenejším. Toto je vykonané pomocou porovnávania poradového čísla zóny. Program potom nájde miestnosť, kde je možné vložiť kľúč (t. j. miestnosti, kde nie je kľúč, špeciálny kľúč alebo cieľ úrovne) tak, aby nevznikla situácia, že hráč môže daný kľúč získať iba po otvorení prechodu, ktorý odomyká.



Obr. 10: Generovanie úrovne

Generovanie mapy má na starosti trieda `MapGenerator`, ktorá vytvorí viacero objektov `RoomInfo`, ktoré obsahujú informácie o miestnostiach v úrovni a o spojeniach medzi nimi.

Súčasti miestností sú definované objektami `RoomPrefab`. Každý objekt `RoomInfo` má referenciu na `RoomPrefab`, ktorý tvorí prvú koreňovú súčasť pre generáciu objektov v miestnosti.

4.3.5 Generovanie objektov miestnosti

Za účelom šetrenia výpočtových prostriedkov, sú objekty miestností načítané vždy iba pre miestnosť, v ktorej sa hráč aktuálne nachádza. Všetky objekty v miestnosti sú vygenerované práve v bode, kedy hráč do miestnosti vstúpi a zároveň sú všetky objekty vymazané hneď ako z miestnosti odíde.

Každá miestnosť je reprezentovaná separátnym svetom. Tento svet je vytvorený pri vstupe používateľa a je v ňom spustený proces generovania miestnosti. Obsah miestností je tvorený náhodným generátorom a kombinovaním dopredu vytvorených súčastí, podľa špecifikovaných parametrov. Tieto súčasti sú tvorené zoznamom herných a zástupných objektov.

Zástupný objekt je podtrieda herného objektu, ktorá obsahuje konfiguráciu, podľa ktorej je vybraný herný objekt alebo ďalšia súčasť, ktorej objekty sú rekurzívne spracované. Konfigurácie pre tento výber obsahuje podmienky, ktoré používateľ vyberie tak, aby sa generácia miestnosti vyvíjala podľa parametrov a náhodného generátora. Proces generovania objektov je ilustrovaný v diagrame 11.

Parametre môžu byť globálne - tie ktoré určujú obtiažnosť celej úrovne, alebo lokálne parametre, t.j. typy spojení, ktoré miestnosť má.

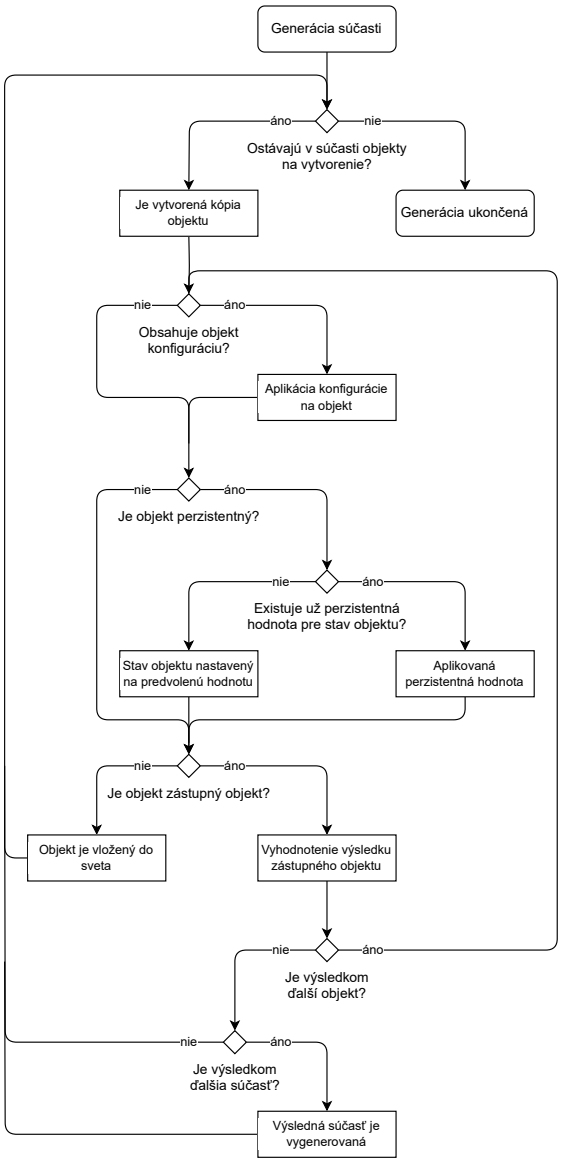
Globálne parametre sú nasledovné:

- Obtiažnosť nepriateľov
- Obtiažnosť skokov
- Počet odmien

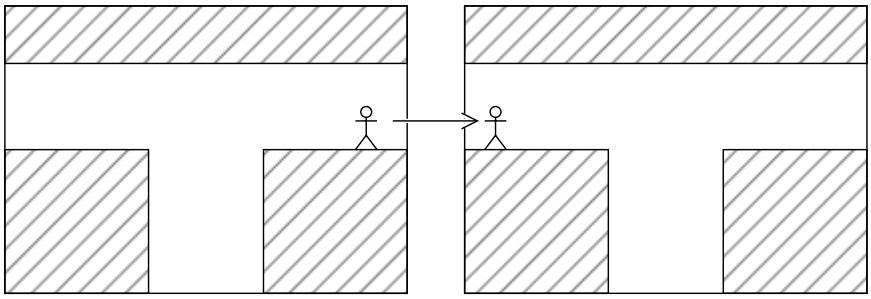
Jedinou výnimkou je koreňová súčasť, ktorá je vybraná pri generácií celej úrovne. Táto má v sebe definované, ktoré spojenia podporuje a či má možnosť obsahovať v sebe kľúč. Pre každú miestnosť v úrovni je náhodne vybraná koreňová súčasť z množiny tých, ktoré podporujú požiadavky miestnosti. Toto umožňuje dopredu zistiť, či existuje miestnosť, ktorej požiadavky nie je možné naplniť a generovanie prerušiť.

Keďže sú všetky objekty miestnosti odstránené pri odchode hráča, dôsledky jeho interakcie s objektami ako je napríklad zbieranie odmien alebo odomykanie prechodov, by boli stratené. Preto každá miestnosť obsahuje pole perzistentných hodnôt, do ktorého objekty nastavené ako perzistentné ukladajú svoj stav. Výsledok vyhodnotenia zástupných objektov je tiež uložený, aby sa tieto výpočty nemuseli opakovať.

Generovanie miestnosti má na starosti objekt triedy `RoomController`, ktorý vytvorí objekt `World` a spustí generovanie objektov miestnosti. Následne umiesti hernú postavu na správne miesto - a to tak, aby stál v logickom spojení s miestnosťou, z ktorej do novej miestnosti prišiel.



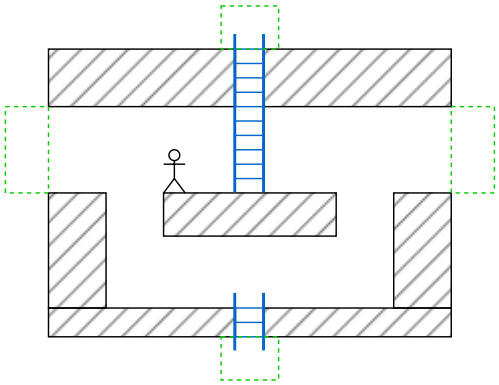
Obr. 11: Generácia objektov miestnosti



Obr. 12: Umiestnenie hernej postavy pri prechode medzi miestnosťami

Pri generovaní miestnosti je vytvorený objekt `RoomInstantiationContext`, ktorý obsahuje všetky parametre pre jej generovanie. Načítanie objektov v súčasti, má na starosti trieda `LevelDeserializer`, ktorá pri každom načítaní objektu, spustí spätné volanie do `RoomInstantiationContext` pre spracovanie objektu podľa diagramu 11.

Na stranách miestnosti sú v prípade prítomnosti spojenia s inou miestnosťou vytvorené interaktívne objekty triedy `RoomTrigger`, ktoré pri kontakte s hráčom, ho presunú do spojenej miestnosti.



Obr. 13: Herné objekty, ktoré presunú hráča do spojenej miestnosti

Pre diagnostiku generovania pri vývoji, alebo pre možnosť získania viac informácií pre vyhodnocovanie tréningu agenta, je vytvorený podporný objekt `MapView`, ktorý pri stlačení tlačidla `M` zobrazí mapu úrovne a umožňuje stlačením kláves `I`, `J`, `K` a `L` premiestňovať hráčsku postavu medzi miestnosťami.

4.3.6 Optimalizácia parametrov

Pre používateľa môže byť náročné modifikovať individuálne parametre generátora. Preto program obsahuje súčasť, ktorá je schopná na základe stanovenia parametrov optimálnej cesty cez úroveň nastaviť parametre tak, aby výsledná úroveň splnila tieto parametre.

Používateľ nastavuje štyri druhy obtiažnosť.

- `JUMP` ⇒ počet skokov v ceste
- `REWARD` ⇒ počet drahokamov v ceste
- `ENEMY` ⇒ počet nepriateľov v ceste (obtiažnejší nepriatelia sa rátajú ako väčší počet nepriateľov)
- `SPRAWL` ⇒ dĺžka cesty

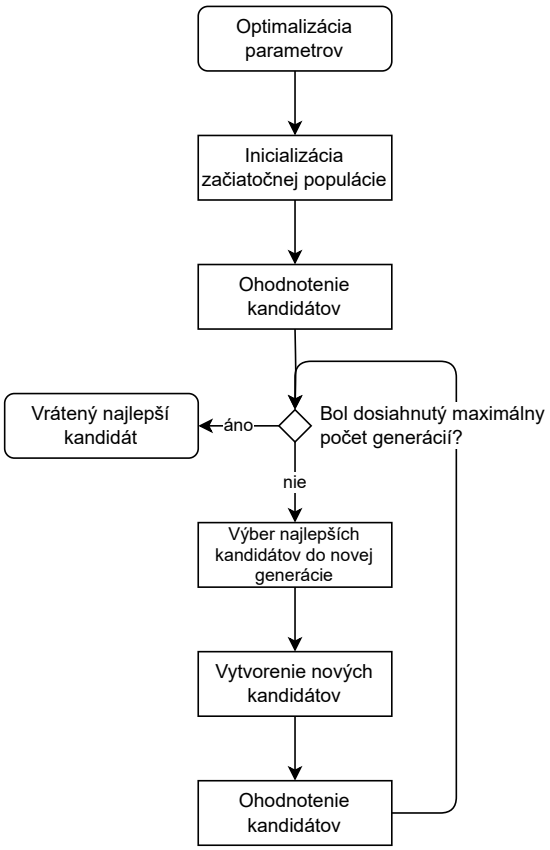
Optimálna cesta obsahuje zbieranie všetkých špeciálnych kľúčov v optimálnom poradí, pre minimalizáciu dĺžky, a následné otvorenie cieľa úrovne. Tiež je sledovaný počet kľúčov potrebných na otvorenie dverí a v prípade, že nejaký kľúč chýba, tak je cesta modifikovaná pre zber potrebného kľúča.

Obr. 14: Optimálna cesta

*Optimálna cesta je tvorená viacerými segmentami, kde každý je označený inou farbou a očíslovaný. Cesta začína segmentom 0. Začiatok segmentu je indikovaný znakom ^ a koniec *. Na obrázku je možné vidieť, ako cesta prejde cez všetky miestnosti obsahujúce špeciálny kľúč (označené -2) a skončí v miestnosti s cieľom úrovne (označená -3).*

Parametre cesty sú merané sčítaním parametrov každej miestnosti v ceste. Parametre sú pripočítané vždy, keď cesta prejde cez danú miestnosť, okrem parametru drahokamov, pretože drahokamy sa dajú získať len raz. Program iteratívne mení parametre z pôvodných náhodných hodnôt až kým sa nedosiahnú želané parametre cesty.

Optimalizáciu parametrov má na starosti trieda `DifficultyOptimizer`. Program používa genetický algoritmus, kde každý jedinec je kandidát na finálnu úroveň a parametre generátora úrovne tvoria gény jedincov. Najlepšie jedince sa podieľajú na tvorbe jedincov pre ďalšiu generáciu. Proces genetického algoritmu je ilustrovaný v diagrame 4.3.6.



Obr. 15: Proces genetického algoritmu

Nové jedince sa tvoria z n najlepších kandidátov predošlej generácie (východisková hodnota je 30%, ale je konfigurovateľná používateľom). Tvorba nových jedincov má viacero možností. Možnosti sú posudzované v poradí. Každá možnosť má určitú pravdepodobnosť, a ak nie je vybratá, je posúdená ďalšia možnosť. Možnosti sú nasledovné:

1. 30% Zmena pozícií špeciálnych kľúčov v jednom kandidátovi
2. 50% Náhodné pozmenenie parametrov v jednom kandidátovi
3. 100% Náhodná kombinácia parametrov z dvoch kandidátov a následná mutácia jedného parametru

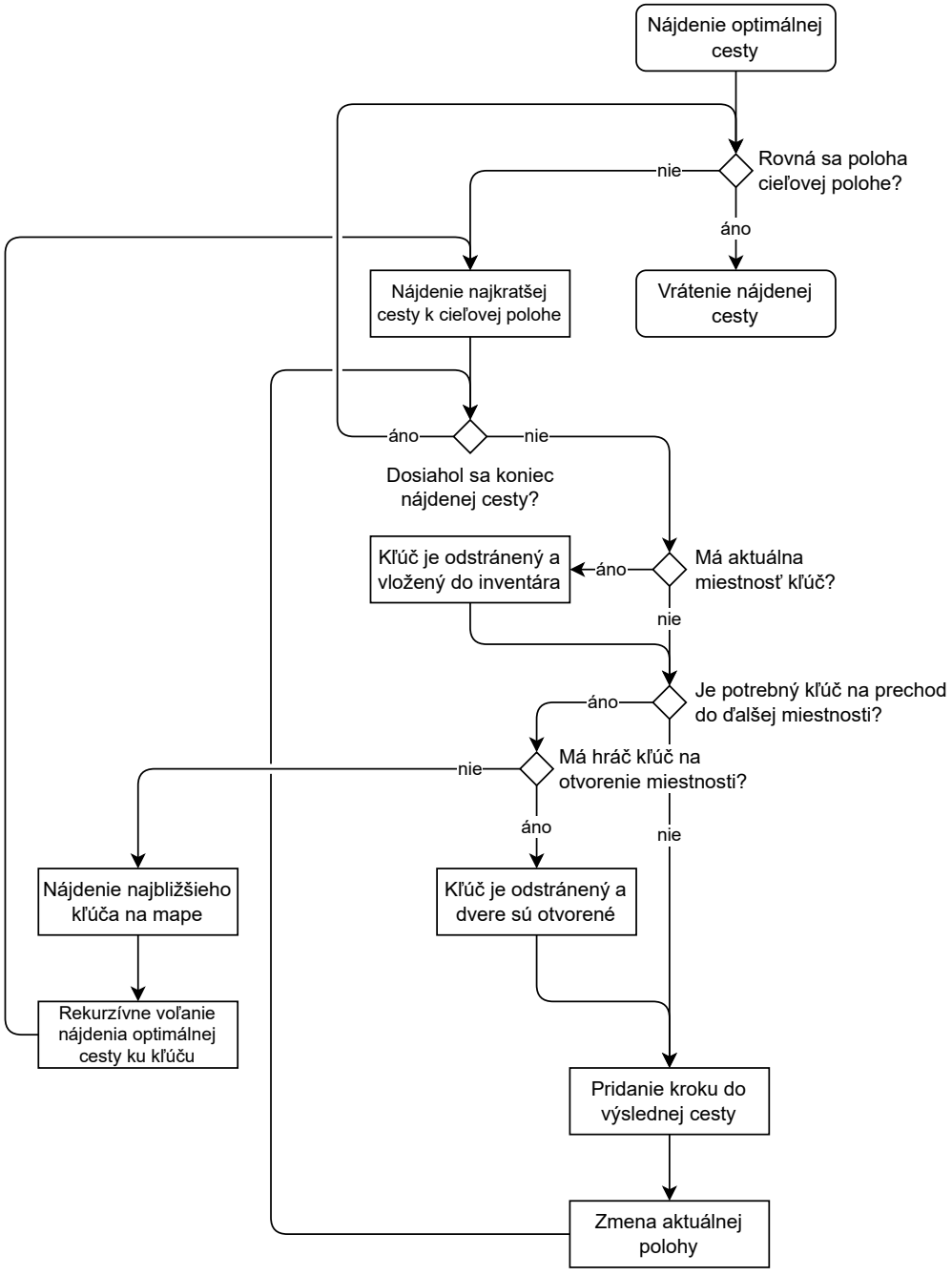
Taktiež sa určitý počet kandidátov presunie do novej generácie bez zmeny, aby sa predišlo regresii kvality (východisková hodnota je 20%, ale je konfigurovateľná používateľom).

Kandidáti sú ohodnotení podľa rozdielu parametrov ich optimálnej cesty s vybranými parametrami. Optimálna cesta je nájdená triedou `LevelSolver`. Táto otestuje každé možné poradie zbierania špeciálnych kľúčov a vyberienajkratšiucestu. Operácia nájdenia optimálnej cesty je volaná pre nájdenie cesty z koreňovej miestnosti k prvému špeciálnemu kľúču, od špeciálneho kľúča k nasledovnému špeciálnemu kľúču a od posledného špeciálneho kľúča k cieľu úrovne.

Ná nájdenie najkratšej cesty sa používaalgoritmus `A*` [7], ktorý vráti pole krokov, kde každý krok je jedna miestnosť, cez ktorú je potrebné prejsť. Cez tieto miestnosti prechádza hypotetický hráč, reprezentovaný triedou `LevelSolverState`, ktorá sleduje inventár kľúčov. V každom kroku sa kontroluje či je nutné prejsť cez zamknuté dvere, a ak áno, či má hypotetický hráč dosť kľúčov na otvorenie dverí. Proces nájdenia optimálnej cesty je ilustrovaný v diagrame 16.

4.3.7 Načítavanie herných prostriedkov

Hra definuje podporný objekt `ResourceProvider`, ktorý načíta grafickú reprezentáciu herných objektov a písma pre zobrazovanie textu v hre. Tieto dáta sú načítané prostredníctvom funkcií obsiahnutých v PyGame. Všetky textúry sú uložené v jednom súbore vo forme atlasu. Referenciu na tento objekt môžu herné objekty získať prostredníctvom `DependencyInjection`.



Obr. 16: Proces nájdenia optimálnej cesty

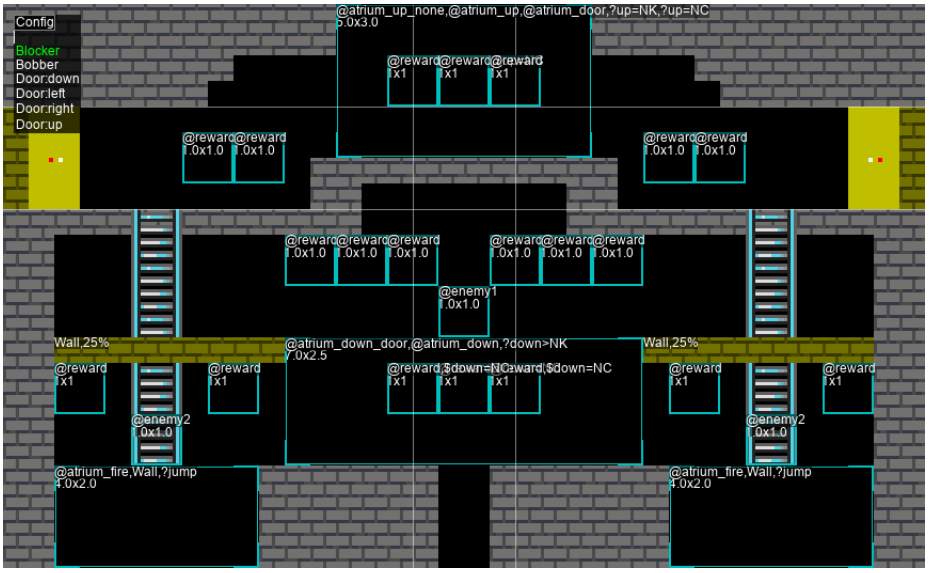
Obr. 17: Grafické prostriedky uložené v atlase

Všetky možné súčasti miestnosti sú triedou `RoomPrefabRegistry` načítané z `json` súborov. Každá miestnosť obsahuje zoznam herných objektov, ktoré sa majú vytvoriť pri generovaní. V súbore sú tieto herné objekty reprezentované svojím menom. Aby bolo možné z ich mena vytvoriť inštanciu objektu, sú všetky objekty, ktoré je dovolené používateľom umiestniť, uložené v triede `ActorRegistry`.

4.3.8 Editor súčastí miestnosti

Na tvorbu jednotlivých súčastí miestnosti je implementovaná aplikácia editora, ktorá prostredníctvom grafického rozhrania umožňuje definovať herné objekty, súčasti a taktiež vytvoriť a konfigurovať zástupné objekty.

Editor sa spustí prostredníctvom konzolového rozhrania, kde sa ako argument k príkazu špecifikuje `json` súbor, v ktorom je uložená súčasť, ktorú sa používateľ rozhodol editovať (ak špecifikovaný súbor ešte neexistuje, je vytvorený). Editor zdieľa rovnakú architektúru ako samotná hra s tým rozdielom, že namiesto objektu hráča je vytvorený objekt triedy `LevelEditor`. Tento objekt prevádzkuje samotné editovanie ako aj ukladanie zmien do súboru.



Obr. 18: Aplikácia editora

Používateľ môže vytvoriť nové objekty ich vyhľadaním prostredníctvom mena a kliknutím pravého tlačidla myši na miesto, kde má byť objekt vložený. Je možné kliknutím ľavého tlačidla vybrať objekt a ťahaním myšou ho presunúť na novú pozíciu. Taktiež je možné zmeniť veľkosť zvoleného objektu pomocou ťahania zvýraznenej oblasti na stranách vybraného objektu. Stlačením klávesu `DELETE`, je vybraný objekt vymazaný, stlačením kombinácie `CTRL+D` je zvolený objekt duplikovaný.

Po vybratí zástupného objektu je možné jeho konfiguráciu upraviť písaním textu. Konfigurácia je tvorená jednoduchým zásobníkovým programovacím jazykom, prostredníctvom ktorého je možné nastaviť, ktorým herným objektom alebo súčasťou bude zástupný objekt nahradený. Príkazy sú oddelené čiarkou. Neplatná konfigurácia je indikovaná zmenou farby textu na červenú. Konfigurácia obsahuje nasledovné príkazy:

- % + percento ⇒ je vybratá hodnota zo zásobníka. Podľa náhodnej šance danej percentom, je buď hodnota vložená naspäť alebo nahradená prázdnou hodnotou
- % + percento + ? ⇒ sú vybraté dve hodnoty zo zásobníka. Podľa náhodnej šance danej percentom, je buď vložená naspäť prvá hodnota alebo druhá
- $ + podmienka ⇒ je vybratá hodnota zo zásobníka. Ak je daná podmienka platná, je hodnota vložená naspäť inak je nahradená prázdnou hodnotou
- ? + podmienka ⇒ sú vybraté dve hodnoty zo zásobníka. Ak je daná podmienka platná, je prvá hodnota vložená naspäť, inak je vložená druhá hodnota
- @ + názov skupiny súčasti ⇒ vloží skupinu súčasti na zásobník
- názov herného objektu ⇒ vloží herný objekt na zásobník

Stlačením tlačidla "Config" sa otvorí menu, prostredníctvom ktorého je možné nastaviť, ktoré požiadavky súčasť spĺňa, pre výber ako koreňovú miestnosť, alebo je možné vybrať parametre generovania, prípadne pridať súčasť do skupín pre výber do zástupných objektov.

Stlačením klávesy `ENTER` je možné vyskúšať editovanú súčasť. Za funkcionalitu je zodpovedná trieda `TestPlayController`, ktorá vytvorí dočasnú kópiu editovanej súčasti a na pozícií myši je vytvorený objekt hráča. Stlačením tlačidla "Config" sa otvorí menu, prostredníctvom ktorého je možné aktivovať generovanie miestnosti s aktuálne editovanou súčasťou ako koreňovou súčasťou a nastaviť parametre, podľa ktorých generovanie prebieha. Stlačením klávesu `TAB` je testovací svet znovu vygenerovaný alebo opätovným stlačením klávesu `ENTER` je možné vrátiť sa naspäť do editora.

Všetky zmeny sú automaticky ukladané. Editor implementuje aj históriu zmien, stlačením kombinácie `CTRL+Z` je možné vrátiť späť poslednú akciu alebo `CTRL+SHIFT+Z` je možné vrátenú akciu znovu aplikovať.

4.4 Overenie riešenia

Projekt spĺňa požiadavky pre interakciu hráča s herným prostredím a pre generovanie úrovne a miestností podľa stanovených parametrov, čo bolo overené testovaním hry manuálne.

Názov	Popis testu	Výsledok testu
Editor miestností	Používateľ použije editor na vytvorenie miestnosti. V rámci tohto procesu otestuje: vytváranie, presúvanie, zmenu veľkosti a mazanie objektov, vrátenie zmien a automatické ukladanie zmien.	Miestnosť bola úspešne vytvorená a otestované funkcie editora fungujú podľa špecifikácií.
Generovanie obsahu miestnosti	Používateľ použije editor na vytvorenie koreňovej súčasti miestnosti a viacerých súčastí miestnosti. Prostredníctvom funkcie editora na vyskúšanie miestnosti otestuje spojenie súčasti na vytvorenie miestnosti.	Program úspešne použije súčasti miestnosti na vytvorenie miestnosti.

Generátor úrovne	Používateľ manuálne špecifikuje parametre pre generátor a vygeneruje úroveň. Miestnosti úrovne budú overené prostredníctvom testovacích nástrojov.	Bola vygenerovaná úroveň podľa špecifikovaných parametrov.
Základné herné objekty	Prostredníctvom ovládania hernej postavy otestuje: zber drahokamov, kontakt s nepriateľmi, použitie rebríkov, zber farebných kľúčov a odomknutie dveri.	Testované herné objekty fungujú podľa špecifikácií.
Pokrok cez úroveň	Prostredníctvom testovacích nástrojov používateľ pozbiera všetky špeciálne kľúče, odomkne cieľ úrovne a prostredníctvom neho ukončí hru.	Testované herné objekty fungujú podľa špecifikácií.
Celkový herný test	Prostredníctvom ovládania hernej postavy používateľ pozbiera všetky špeciálne kľúče, odomkne cieľ úrovne a prostredníctvom neho ukončí hru.	Používateľ úspešne prešiel cez úroveň a ukončil hru.
Optimalizácia obtiažnosti	Používateľ špecifikuje parametre úrovne a vloží ich do optimalizatora obtiažnosti. Následne porovná parametre najlepšej úrovne z pôvodnými parametrami.	Výsledné parametre sa zvyšovaním počtu generácií optimalizátora približujú k špecifikovaným parametrom v primeranej miere.

5 Zhodnotenie

Cieľom práce bolo vytvorenie generátora, ktorý umožňujepoužívateľovi jednoducho vytvoriť herné prostredie pre trénovanie umelej inteligencie. Generátor tvorí neustále nové úrovne nastavením iba malého počtu parametrov. Súčasťou práce je aj replikácia mechaník hry Montezuma's Revenge, čo umožňuje trénovanie agenta umelej inteligencie bez potreby emulácie starších zariadení.

Výsledkom práce je generátor prostredia a herná aplikácia, ktorá umožní hranie hry človekom alebo agentom UI.

Aplikácia je implementovaná v jazyku Python [12]. Jej súčasťou je aj implementácia prostredia pre knižnicu Gymnasium [5]. Grafický výstup je generovaný cez knižnicu Pygame [11]a grafická reprezentácia objektov používa obrázky z balíčka DawnLike [4]distribuovaného pod licenciou CC-BY 4.0.

Generátor prostredia, ktorý dokáže vytvárať neustále nové úrovne, môže pomôcť k robustnejšiemu tréningu agentov umelej inteligencie pri spätnoväzbovom učení. Herná aplikácia umožňuje tréning vykonávať bez potreby použitia emulátora, čo pomôže spustiť viacero inštancií trénovania paralelne. Prostredie bolo úspešne integrované s výskumným frameworkom školiteľa a zbehli pilotné testy agentov na tomto prostredí.

Literatúra

[1] BELLEMARE, M. G.; NADDAF, Y.; VENESS, J. et al.: The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR*, ročník abs/1207.4708, 2012, 1207.4708. Dostupné z: http://arxiv.org/abs/1207.4708

[2] COBBE, K.; HESSE, C.; HILTON, J. et al.: Leveraging Procedural Generation to Benchmark Reinforcement Learning. *CoRR*, ročník abs/1912.01588, 2019, 1912.01588. Dostupné z: http://arxiv.org/abs/1912.01588

[3] DALTON, S.; FROSIO, I. GARLAND, M.: Accelerating Reinforcement Learning through GPU Atari Emulation. 2020, 1907.08467. Dostupné z: https://arxiv.org/abs/1907.08467

[4] DragonDePlatino DawnBringer: DawnLike - 16x16 Universal Rogue-like tileset v1.81 | OpenGameArt.org. Január 2014. Dostupné z: https://opengameart.org/content/dawnlike-16x16-universal-rogue-like-tileset-v181

[5] Farama: Basic Usage - Gymnasium Documentation. December 2024. Dostupné z: https://gymnasium.farama.org/introduction/basic_usage/

[6] GABROVŠEK, P.: Analysis of maze generating algorithms. *IPSI Transactions on Internet Research*, ročník 15, č. 1, 2019: s. 23–30. Dostupné z: http://www.ipsitransactions.org/journals/papers/tir/2019jan/p5.pdf

[7] HART, P. E.; NILSSON, N. J. RAPHAEL, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, ročník 4, č. 2, 1968: s. 100–107, doi:10.1109/TSSC.1968.300136. Dostupné z: https://ieeexplore.ieee.org/document/4082128

[8] JUSTESEN, N.; TORRADO, R. R.; BONTRAGER, P. et al.: Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation. 2018, 1806.10729. Dostupné z: https://arxiv.org/abs/1806.10729

[9] KAISER, L.; BABAEIZADEH, M.; MILOS, P. et al.: Model-Based Reinforcement Learning for Atari. 2024, 1903.00374. Dostupné z: https://arxiv.org/abs/1903.00374

[10] MNIH, V.: Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. Dostupné z: https://people.engr.tamu.edu/guni/csce642/files/dqn.pdf

[11] Pygame contributors: Pygame Front Page — pygame v2.6.0 documentation. August 2024. Dostupné z: https://www.pygame.org/docs/

[12] Python Software Foundation: The Python Tutorial — Python 3.13.1 documentation. December 2024. Dostupné z: https://docs.python.org/3/tutorial/index.html

[13] SUTTON, R. S.; BARTO, A. G. et al.: *Reinforcement learning: An introduction*, ročník 1. MIT press Cambridge, 1998.

[14] TOGELIUS, J. LUCAS, S.: Evolving robust and specialized car racing skills. Január 2006, s. 1187 – 1194, doi:10.1109/CEC.2006.1688444.

[15] YU, X. GEN, M.: Introduction to evolutionary algorithms. Springer Science & Business Media, 2010, doi:10.1007/978-1-84996-129-5. Dostupné z: https://link.springer.com/book/10.1007/978-1-84996-129-5

Príloha A: Obsah digitálnej časti práce

Digitálna časť práce obsahuje zdrojový kód vo forme Python projektu v pôvodnej forme a taktiež v inštalovateľnom balíčku typu wheel, pre inštaláciu projektu ako knižnica.

Obsiahnuté sú tu tiež všetky zdroje programu, t.j. grafické prvky, súčasti miestností a písma. Tieto súbory sa nachádzajú v priečinku `src/pg_gen/assets`. Všetky použité zdroje sú pod licenciou, ktorá umožňuje ich ďalšiu distribúciu. Kópia alebo odkaz na relevantnú licenciu je umiestnený popri súbore, na ktorý sa vzťahuje.

- Zdrojový kód sa nachádza v priečinku: `PlatformGameGenerator`
- Inštalovateľný balíček je súbor: `pg_gen-1.0.0-py3-none-any.whl`

Zdrojový kód obsahuje súbor `pyproject.toml` vyjadrujúci knižnice a verziu Pythonu, potrebnú pre jeho spustenie. Na manažment knižníc bol použitý program `rye` verzie `0.44.0`, ktorý je možné stiahnuť cez internet z adresy `https://rye.astral.sh/`. Po inštalácií danej aplikácie je možné nainštalovať potrebné knižnice príkazom `rye sync`. Tento príkaz vytvorí Python virtuálne prostredie, ktoré je možné aktivovať príkazom vhodným pre Váš operačný systém.

Projekt definuje príkazy, ktoré služia na spustenie komponentov práce bez potreby písania ďalšieho kódu. Tieto príkazy je možné spustiť po aktivácií Python prostredia.

- `start` ⇒ vygeneruje úroveň a sputí interaktívnu hru
- `start-editor <súbor>` ⇒ spustí editor a otvorí súčasť miestnosti definovanú v špecifikovanom súbore

Príloha B: Návod na použitie programu

Inštalácia programu ako knižnica

Aby bol program použiteľný ako knižnica, je potrebné ho nainštalovať z wheel súboru. Tento súbor je súčasťou C3 alebo je k dispozícií na internetovej adrese `https://bt7s7k7-2.github.io/PlatformGameGenerator/pg_gen-1.0.0-py3-none-any.whl`. Tento súbor, kedže je štandardným formátom pre Python knižnice, je možné nainštalovať prostredníctvom Vami zvoleného inštalačného programu. Nižšie sú uvedené príklady inštalácie prostredníctvom niektorých programov.

```
pip install 'pg_gen-1.0.0-py3-none-any.whl'
rye add pg_gen --path 'pg_gen-1.0.0-py3-none-any.whl'
```

Použitie programu priamo

Program potrebuje pre svoju činnosť zdroje. Tieto sú manažované globálnymi objektami, ktoré treba inicializovať.

```
import pygame
from pg_gen.generation.RoomPrefabRegistry import RoomPrefabRegistry
from pg_gen.level_editor.ActorRegistry import ActorRegistry

pygame.init()
ActorRegistry.load_actors()
RoomPrefabRegistry.load()
```

Ďalším krokom je zvoliť si obtiažnosť, ktorú požadujete.

```
from pg_gen.generation.RoomParameter import UNUSED_PARAMETER
from pg_gen.difficulty.DifficultyReport import DifficultyReport

target_difficulty = DifficultyReport()
target_difficulty.set_all_parameters(UNUSED_PARAMETER)
target_difficulty.set_parameter(RoomParameter.REWARD, 500)
target_difficulty.set_parameter(RoomParameter.JUMP, 10)
target_difficulty.set_parameter(RoomParameter.ENEMY, 100)
target_difficulty.set_parameter(RoomParameter.SPRAWL, 50)
```

Následne je potrebné vytvoriť optimalizátor a vložiť zvolené parametre. Optimalizátor ďalej príjima argumenty `random`, kde je potrebné vložiť náhodný generátor - tu je možné nastaviť konzistentný seed. Optimalizátor má možnosť nastaviť populáciu jedincov genetického algoritmu cez parameter `max_population` a počet iterácií genetického algoritmu cez parameter `max_generations`.

```
from random import Random
from pg_gen.game_core.Universe import Universe
from pg_gen.difficulty.DifficultyOptimizer import DifficultyOptimizer

universe = Universe()
optimizer = DifficultyOptimizer(universe, target_difficulty=target_difficulty, random=Random(108561))
```

Nie je potrebné špecifikovať všetky parametre. Nepoužité parametre ostanú ako `UNUSED_PARAMETER`. Tieto parametre budú mať po optimizácií náhodnú hodnotu, preto je vhodné nastaviť ich nastaviť na konštatnú hodnotu. Takto je možné nastaviť všetky parametre, ktoré sa menia pri optimalzácií.

```
from pg_gen.generation.RoomParameter import RoomParameter

optimizer.get_parameter(RoomParameter.REWARD).override_value(0.5)
```

Po zvolení parametrov a obtiažnosti je možné spustiť optimalizátor.

```
optimizer.initialize_population()
optimizer.optimize()
```

Po optimizácií je možné vybrať najvhodnejšieho kandidáta a použiť jeho úroveň. Následne je potrebné aktivovať koreňovú miestnosť a vložiť do nej objekt hráča.

```
from pg_gen.generation.RoomController import RoomController
from pg_gen.actors.Player import Player
from pg_gen.support.Point import Point
from pg_gen.support.constants import ROOM_HEIGHT, ROOM_WIDTH

best_candidate = optimizer.get_best_candidate()
map = best_candidate.get_map()
universe.map = map

room_controller = RoomController.initialize_and_activate(universe, map.get_room(Point.ZERO), None)
room_controller.world.add_actor(Player(position=Point(ROOM_WIDTH / 2, ROOM_HEIGHT / 2)))
```

Pre použitie hry interaktívne, stači spustiť `InteractiveGameLoop`.

```
from pg_gen.game_core.InteractiveGameLoop import InteractiveGameLoop

game_loop = InteractiveGameLoop(universe)
game_loop.run()
```

Pre použitie s modelom, je potrebné vytvoriť vlastný pygame-ový `Surface` a vložiť ho do `GameLoop`. Následne je možné vkladať vstup cez objekt `InputState` a manuálne simulovať update. Keďže mimo interaktívneho prístupu nie je herná slučka naviazaná na reálny čas, je potrebné stanoviť čas, ktorý prejde medzi každý update krokom. Napríklad: ak chceme mať 20 krokov cez jednu simulovanú sekundu, tak čas medzi krokmi je $1/20$ sekúnd.

```
from pg_gen.game_core.GameLoop import GameLoop
from pg_gen.support.constants import CAMERA_SCALE
from pg_gen.game_core.InputState import InputState

surface = pygame.display.set_mode((CAMERA_SCALE * ROOM_WIDTH, CAMERA_SCALE * ROOM_HEIGHT))
self.game_loop = GameLoop(surface, self.universe)

input_state = self.universe.di.inject(InputState)
input_state.clear()

input_state.left = True
input_state.jump = True
```

```
# [...]

game_loop.update_and_render(1 / fps)
```

Použitie Gymnasium prostredia

Gymnasium prostredie rieši spustenie hernej slučky automaticky. Stačí inicializovať globálne objekty a vložiť úroveň. Prostredie je automaticky registrované, stačí importovať package `pg_gen_gym`.

```
import pg_gen_gym
```

Následne je možné vytvoriť prostredie.

```
env = gymnasium.make("gymnasium_int/PgEnv", render_mode="rgb_array", level=level)
observation, info = env.reset()
```

Prostredie podporuje dva render módy:

- `human` ⇒ simulácia prebieha v reálnom čase a je otvorené okno, kde je možné vidieť výstup z hry
- `rgb_array` ⇒ simulácia prebieha najrýchlešie ako je možné, výstupom modelu je array obsahujúci grafický výstup hry

Pri použití `rgb_array`, kedže sa neotovrí okno, je potrebné manuálne inicializovať grafický systém pygame-u. Najjednoduchší spôsob, ako to dosiahnuť, je vytvoriť jednopixelové skryté okno, ktoré nebude na nič využité.

```
if env.render_mode == "rgb_array":
    pygame.display.set_mode((1, 1), flags=pygame.HIDDEN)
```

Ako parameter `level` je možné dodať:

- string ⇒ ako explicitný názov miestnosti, ktorá bude vygenerovaná; toto je možné použiť pre testovanie konkrétnej mechaniky
- callback ⇒ funckia dostane ako argument referenciu na `Universe` a musí vrátiť `Map` objekt; vo funkcií spustite generáciu ako v predošlej sekcii

Toto je jednoduchý loop, pre využitie prostredia:

```
episode_over = False
while not episode_over:
    action = env.action_space.sample()
    observation, reward, terminated, truncated, info = env.step(action)

    episode_over = terminated or truncated
    print(action, observation, info, reward)

env.close()
```

Príloha C: Harmonogram práce

Zimný semester

Cieľom práce v zimnom semestri bolo vytvoriť základ generovania úrovní. Preto bolo potrebné vytvorenie komponentu Generátor úrovne, na jeho podporu komponent Editor súčastí a na jeho vyskúšanie komponent Hra.

Týždeň	**Plán aktivít**
1.	Plánovanie hernej architektúry
2.	Práca na hernom komponente do stavu možnosti prechodu cez miestnosti hernou postavou
3.	Práca na generovaní úrovne do stavu generovania rozloženia miestností
4.	Práca na editore do stavu možnosti tvorby koreňových súčastí
5.	Implementácia herných mechaník
6.	Implementácia herných mechaník
7.	Implementácia herných mechaník
8.	Implementácia herných mechaník
9.	Práca na generovaní miestností a kombinovanie súčastí
10.	Práca na generovaní miestností a kombinovanie súčastí
11.	Práca na generovaní miestností a kombinovanie súčastí
12.	Práca na generovaní miestností a kombinovanie súčastí
13.	Práca na odovzdaní BP1

V rámci práce na implementácií herných mechaník a generovaní miestností, je aj práca na funkcionalite editora. Nové funkcie boli pridané práve vtedy, keď boli potrebné pre testovanie súčasti, na ktorej sa práve pracovalo. Práca na odovzdaní BP1 pokračovala aj po skončení výučby počas prázdnin.

Letný semester

Cieľom letného semestra je práca na optimalizátore obtiažnosti. Súčasťou je aj implementácia podporných systémov optimalizátora, t.j. meranie obtiažnosti miestností a nájdenie optimálnej cesty cez úroveň. Tiež je dôležitou súčasťou tiež práca na záverečnej správe projektu.

Týždeň	Plán aktivít
1.	Plánovanie architektúry optimalizatora
2.	Plánovanie architektúry optimalizatora
3.	Implementácia podporných systémov optimalizátora
4.	Implementácia podporných systémov optimalizátora
5.	Implementácia podporných systémov optimalizátora
6.	Implementácia podporných systémov optimalizátora
7.	Práca na optimalizátore obtiažnosti
8.	Práca na optimalizátore obtiažnosti
9.	Práca na optimalizátore obtiažnosti
10.	Práca na optimalizátore obtiažnosti, Práca na záverečnej správe projektu
11.	Implementácia Gymnasium prostredia, Práca na záverečnej správe projektu
12.	Práca na záverečnej správe projektu

Kontrola a overovanie funkčnosti projektu prebiehala zároveň s vývojom funckionality. Finálne overenie funkcionality prebehlo popri písaní záverečnej správy projektu.