VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

OVĚŘOVÁNÍ SPRÁVNOSTI HERNÍCH PROSTŘEDÍ POMOCÍ DAVU AUTONOMNÍCH AGENTŮ

GAME ENVIRONMENT CHECKING BY A SWARM OF AUTONOMOUS AGENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TINA HEINDLOVÁ**
AUTHOR

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

BRNO 2018

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Studentka: **Heindlová Tina**
Program: Informační technologie
Název: **Ověřování správnosti herních prostředí pomocí davu autonomních agentů**
Game Environment Checking by a Swarm of Autonomous Agents
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s prostředími pro vývoj moderních počítačových her typu Unreal Engine nebo Enfusion
2. Navrhněte algoritmy pro řízení autonomní entity, která má projít prostředím a ověřit, zdali je vytvořeno v souladu s návrhem a odhalit případné nesprávně utvořená místa.
3. Implementujte agentní systém řízený výše navrženými algoritmy a na dodaném prostředí ověřte přínosnost navrženého systému.
4. Experimentujte s parametry systému (umístění agenta, granularita prostředí) a s jeho rozhodovacími strategiemi. Porovnejte agentovy schopnosti nacházení chyb v prostředí pro různá nastavení.
5. Diskutujte dosažené výsledky, uveďte úspěšnost nalezených nežádoucích míst v prostředí na základě předchozích experimentů a navrhněte další možnosti zlepšení do budoucna.

Literatura:

- Wooldridge, M.: An Introduction to MultiAgent Systems, 2nd Edition, Willey, 2009
- Dudek, G., Jenkin, M., Milios, E., Wilkes, D.:Robotic exploration as graph construction, IEEE Transactions on Robotics and Automation, Volume: 7, Issue: 6 , 1991

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato práce se věnuje práci s enginem Enfusion a jak s ním začít pracovat. Popisuje, jak v něm vytvořit minihru pro seznámení s enginem. Dále je zde navržen a popsán algoritmus zpětného navrácení, který byl použit pro detekci nekonzistentních stavů v herním světem. Práce popisuje tvorbu takového světa a testování na něm. Práce tento algoritmus testuje a diskutuje dosažené výsledky. V závěru jsou navrženy další možné metody řešení.

Abstract

This thesis studies engine Enfusion and how to start with it. It describes the creation of a minigame to introduce the engine. Further, a backtracking algorithm for the detection of inconsistent states in the game worlds is introduced. Thesis describes how to create a simple world and its testing. The algorithm is tested and the testing results are discussed. In the end, it describes more methods of solutions.

Klíčová slova

zpětné navrácení, engine, detekce, nekonzistentní stavy, minihra

Keywords

backtracking, engine, detection, inconsistent states, minigame

Citace

HEINDLOVÁ, Tina. *Ověřování správnosti herních prostředí pomocí davu autonomních agentů*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František Zbořil, Ph.D.

Ověřování správnosti herních prostředí pomocí davu autonomních agentů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Doc. Ing. F. Zbořila Ph.D. Další informace mi poskytli pánové P. Benýšek a P. Šafář z firmy Bohemia Interactive. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Tina Heindlová
30. července 2019

Poděkování

Tímto bych ráda poděkovala Doc. Ing. F. Zbořilovi Ph.D. a pánům P. Benýškovi a P. Šafářovi za jejich čas a rady při řešení práce.

Obsah

Kapitola 1

Úvod

Hry byly odjakživa součástí našeho života. Ať už mluvíme o sportovních hrách jako jsou fotbal, hokej či tenis, nebo o stolních hrách jako je třeba Člověče, nezlob se, člověk tráví velkou část svého volného času hraním her. Nebylo divu, že po vzniku prvních počítačů se hry přesunuly i tam. Počet počítačových her, které od té doby vzniklo, je obrovské množství a v dnešní době je jen málokdo, kdo žádnou z nich nehrál či o nějaké aspoň neslyšel. Počítačové hry se tak staly rychle naší součástí a nároky na hru se stávají větší a větší, neboť si hráči při hraní přejí zažít nezapomenutelný zážitek. Proto je velmi důležité, aby byly hry co možná nejlépe otestovány. Tato práce se věnuje testování herního prostředí, které vzniklo v herním enginu Enfusion.

Tento engine i herní prostředí vyvinula firma Bohemia Interactive, a proto je druhá kapitola věnována popisu této firmy a hrám, které vytvořila. Mezi nejznámější z nich jsou série ArmA a DayZ. V této kapitole bylo popsáno, o čem dané hry jsou, kdy vznikly a na jakém světě se odehrávají. Dále je zde vysvětleno, co je to engine a jaké volně dostupné enginy existují.

Třetí kapitola se věnuje stavovému prostoru a metodám procházení takového prostoru. Je zde popsáno, co to takový stavový prostor je, jakým způsobem se dělí a bylo zde uvedeno několik příkladů, jak fungují a jaká mají hodnotící kritéria. Dále je zde vysvětleno, co to je agent a agentní systém a jaký mají vztah k dané práci.

Čtvrtá kapitola se věnuje samotnému enginu Enfusion. Co to je, jakým programovacím jazykem se v něm píše, jak a pomocí čeho se tvoří herní světy v takovém enginu. Co to je překážka a jaké nekonzistentní stavy se ve světě vyskytují.

Engine Enfusion je momentálně stále ještě ve vývoji a neexistují veřejné návody k práci s ním. Jediný oficiální, ještě nezveřejněný, návod je tutoriál „steal the car“. V páté kapitole je popsáno, o čem má tutoriál být, jaké prvky se v něm používají a jak se vytvoří. Tato kapitola seznámí čtenáře s prací v tomto enginu a naučí ho správně vytvořit misi.

Šestá kapitola se věnuje návrhu testování herního prostředí pomocí enginu Enfusion. Popisuje metodu řešení hledání nekonzistentních stavů, co bylo použito k testování, použitý algoritmus a jakým způsobem se detekují některé nekonzistentní stavy.

Sedmá kapitola se věnuje popisu implementace metody popsané v kapitole 6. Jaké funkce byly použity a jaké problémy se musely řešit.

Osmá kapitola se zabývá testováním této metody a jaké byly výsledky při testování programu.

Závěr shrnuje výsledky práce, zabývá se jejími úspěchy a neúspěchy a jaké jsou možné úpravy do budoucna.

Kapitola 2

Bohemia Interactive

Bohemia Interactive [3] je česká vývojářská společnost zabývající se vývojem herního softwaru a výzkumem v oblasti 3D grafiky, umělou inteligencí a fyzikální simulací v reálném čase. Firmu založili bratři Španělové v roce 1999. První hra vydaná touhle společností se nazývá Operace Flashpoint a byla vydaná roku 2001. Od té doby vydali mnoho dalších her jako je například série Arma, Take on Mars či DayZ.

Kromě her tato firma vyvíjí i herní enginy, ve kterých tyto hry vznikly. Příkladem takového herního enginu je Enfusion, který bude detailněji popsán v kapitole 4.

2.1 Hry

DayZ [5] je počítačová hra od firmy Bohemia Interactive. Je to online hra s otevřeným světem. Hráč se ocitne ve fiktivní zemi Chernarus, jejíž rozloha čítá 230 km^2. Zemi zasáhl neznámý virus, který většinu obyvatel proměnil v běsnící infikované. Cílem hráče je přežít, jak nejdéle to bude možné. Po celou hru hráč získává různé věci (bonusy), které může využít k lepší obraně a vylepšit tak podmínky k přežití. Pokud hráčova postava ve hře zemře, přijde o všechny věci, které ve hře během života získal, a začíná hru nanovo. Na serveru je dohromady šedesát hráčů, kteří se snaží přežít v tomto nehostinném světě, kde na ně číhají strasti nejen infikovaných lidí, ale i samotných hráčů, kteří mohou útočit i na sebe navzájem.

ArmA je další počítačovou hrou od firmy Bohemia Interactive. Hra byla vydána v roce 2006 a od té doby vzniklo již mnoho dodatků (tzv. datadisky [4]). Od té doby již vyšly další dvě pokračování. Obsahuje jak mód pro samostatně hrajícího hráče (tzv. singleplayer), tak pro hru ve skupině (tzv. multiplayer). Odehrává na smyšleném ostrově Sahrani v Atlantském oceánu. Ten je rozdělen na dvě části: sever a jih. Na každé straně vládne jiný režim. Na jihu vládne prozápadní demokratická monarchie a na severu komunistická Demokratická republika Sahrani (zkráceně DRS). Hráč se vžije do role řadového vojáka US ARMY, která provádí v jižní části výcvik tamních ozbrojených složek. Právě v okamžiku, kdy celý americký kontingent opouští po dlouhé době klidu ostrov, zaútočí armáda DRS na hraniční město Corazol a jižní armáda je spolu s dosud nestaženými americkými vojáky nucena se bránit. [1]

Obrázek 2.1: DayZ [5]

2.2 Engine

Engine [6] je vývojové prostředí pro tvorbu her zahrnující mnoho funkcionalit, které urychlují vývoj her. Obsahuje většinou vykreslovací engine pro vykreslování 2D nebo 3D scény, fyzikální engine pro simulaci fyzikálních zákonů jako je detekce kolizí, gravitace, vzájemné působení těles, pak zvukový a skriptovací engine, engine pro tvorbu animací, umělé inteligence atd.

Důvodem vzniku takového enginu bylo zlevnit a urychlit vývoj her. Představuje totiž znovupoužitelnou část hry. Mezi nejznámější volně dostupné herní enginy patří Unreal [11], Unity [10] a GameMaker [7]. Kromě těchto veřejně dostupných enginů existují i takové, které si firmy vytváří samy, a které pak používají při tvorbě her. Příkladem je třeba Enfusion od firmy Bohemia Interactive. Tyto enginy jsou privátní nebo proprietární a vytvářet hry v nich mohou jen pracovníci firmy či vývojáři, kteří s danou firmou spolupracují nebo mají zakoupenou licenci.

Kapitola 3

Metody prohledávání stavového prostoru

Stavový prostor je definován jako SP = (S, O), kde:

S je množina stavů úlohy S = s_i a i = 1, 2, ...

O je množina operátorů O = o_j a j = 1, 2, ...

Metody prohledávání takového prostoru se dělí na: slepé, informované a metody lokálního prohledávání.

„Slepé metody jsou metody, které nevyužívají žádné informace, které by mohly usnadnit řešení úlohy. Jejich použití je proto oprávněné pouze v případech, kdy o řešených úlohách žádné informace skutečně nemáme. Informované metody jsou metody, které naopak využívají nějaké informace o řešené úloze – tyto informace pak usnadňují, resp. umožňují řešení této úlohy. Metody lokálního prohledávání jsou metody, které místo systematického prohledávání stavového prostoru prohledávají pouze okolí aktuálního stavu. Jsou vhodné především pro řešení optimalizačních problémů.“ [9]

Metody se hodnotí podle kritérií na úplné, optimální a podle časové a prostorové složitosti. Existuje-li řešení úlohy a metoda jej najde, je pak úplná. Pokud navíc najde nejlepší řešení ze všech možných, je tato metoda optimální.

Tato kapitola se věnuje prohledáváním stavového prostoru, jaké existují metody, které z nich jsou úplné, optimální a jakou mají složitost. Dále je zde vysvětleno, co je to agent a agentní systém.

3.1 Prohledávání do šířky

Prohledávání do šířky (Breadth First Search) je jednou ze slepých metod. Metoda pracuje s frontou OPEN, která obsahuje všechny uzly určené k expanzi. Do fronty se umístí počáteční uzel. Pokud je fronta prázdná, je prohledávání označeno jako neúspěšné (úloha nemá řešení). Jinak se vybere první uzel fronty a otestuje, zda není cílový. Pokud ano, je prohledávání ukončeno, označeno jako úspěšné a vrátí se cesta od kořenového uzlu k cílovému. Pokud ne, uzel se expanduje a všechny bezprostřední následníci jsou umístěny do fronty OPEN. Následně se vybírá další uzel k testování, zda není hledaný a případně se bude též expandovat. Metoda pokračuje dokud nebylo nalezeno první řešení nebo dokud se nevyprázdní fronta. [9]

Metoda je úplná a optimální.

Označí-li se symbolem b maximální počet bezprostředních následníků všech expandovaných uzlů (tzv. faktor větvení) a symbolem d hloubka stromu, ve které se nachází řešení, pak:

prostorová i časová složitost je: $O_{BFS}(b^{d+1})$

3.2 Prohledávání do hloubky

Prohledávání do hloubky(Depth First Search) je další ze slepých metod. Metoda pracuje se zásobníkem OPEN, který obsahuje všechny uzly určené k expanzi. Do zásobníku se umístí počáteční uzel. Pokud je zásobník prázdný, je prohledávání označeno jako neúspěšné (úloha nemá řešení). Jinak se vybere první uzel z vrcholu zásobníku a otestuje, zda není cílový. Pokud ano, je prohledávání ukončeno, označeno jako úspěšné a vrátí se cesta od kořenového uzlu k cílovému. Pokud ne, uzel se expanduje a všechny bezprostřední následníci jsou umístěny do zásobníku OPEN. Následně se vybírá další uzel k testování, zda není hledaný a případně se bude též expandovat. Metoda pokračuje dokud nebylo nalezeno první řešení nebo dokud se nevyprázdní zásobník. [9]

Metoda není úplná a tím pádem není ani optimální. Metodu lze modifikovat, aby byla úplná. Modifikace spočívá v eliminaci stejných stavů a předků v OPEN. Tato metoda je pak úplná, ale nikoli optimální.

Časová i prostorová složitost nemodifikované metody je: $O_{DFS}(\infty)$

Časová složitost modifikované metody je: $O_{DFSmodif}(m)$

Prostorová složitost modifikované metody je: $O_{DFSmodif}(b^m)$

3.3 Backtracking

Backtracking je velmi podobný metodě prohledávání do hloubky neboli DFS. Metoda pracuje se zásobníkem OPEN, do kterého se umístí počáteční uzel. Pokud je zásobník prázdný, je prohledávání označeno jako neúspěšné (úloha nemá řešení). Jinak jde-li na uzel z vršku zásobníku aplikovat první/další operátor, tak se tento operátor aplikuje. V opačném případě se odstraní testovaný uzel z vrcholu zásobníku a opět se testuje, zda není prázdný a případně se aplikují další operátory. Je-li vygenerovaný uzel uzlem cílovým, je prohledávání ukončeno, označeno jako úspěšné a vrátí se cesta od kořenového uzlu po cílový. V opačném případě se nový uzel uloží na vrchol zásobníku a postup se opakuje. Backtracking namísto DFS generuje jenom jednoho následníka a při návratu postupně další namísto expandování všech. Při modifikaci se nový uzel neukládá, pokud se již v zásobníku nachází. [9]

Stejně jako DFS není metoda úplná ani optimální, ale při modifikaci se stává úplnou.

Časová i prostorová složitost nemodifikované metody je: $O_{BT}(\infty)$

Časová složitost modifikované metody je: $O_{BTmodif}(m)$

Prostorová složitost modifikované metody je: $O_{BTmodif}(b^m)$

3.4 Agent a agentní systémy

Agent je aktivní prvek systému vytvořený člověkem k nějakému předem zamýšlenému účelu. Agent je autonomní, reaktivní, proaktivní a má sociální schopnosti. Autonomnost znamená, že jedná v prostředí bez přímého vlivu okolí a má plnou kontrolu nad svým jednáním. Tato realizace bývá řízena programem. Reaktivita znamená, že je schopen adekvátně a pohotově reagovat na změny prostředí. Proaktivní agent je schopen se ujímat iniciativy a sám ovliv-

ňovat prostředí za účelem dosažení svých cílů. Sociální schopnosti říkají, že dokáže jednat ve skupině agentů, spolupracovat a řešit konflikty.

Agentní systém je agent a prostředí. Agent vnímá prostředí svými senzory a na základě podnětů, resp. současného a všech minulých podnětů se rozhoduje o svém dalším jednání v prostředí. [13]

Prostředí lze klasifikovat podle toho, zda je:

- spojité / diskrétní – stavy prostředí tvoří diskrétní / spojitou množinu, číslicové počítače jsou diskrétní systém – umělí agenti vnímají prostředí jako diskrétní
- přístupné / nepřístupné – agent svými senzory vnímá celé / část prostředí, ve kterém se nachází
- statické / dynamické – ve statickém prostředí, na rozdíl od dynamického, se jeho stav nemění, pokud agent nejedná
- deterministické / nedeterministické – v deterministickém prostředí je jeho následující stav dán aktuálním stavem, nebo případnou akcí prováděnou agentem
- epizodní / neepizodní – epizody rozdělují běh systému na sobě nezávislé části
- strategické (MAS) – více agentů jedná v prostředí současně

V této práci to bude postava procházející prostředí za účelem nalezení nekonzistencí s návrhem. Realizace agenta bude podrobněji popsána v podkapitole 6.1. I když je možné použít agentů více, v daném problému byl z hardwarových důvodů použit jenom jeden.

Kapitola 4

Enfusion

Jak již bylo zmíněno v kapitole 2, Enfusion je engine vyvíjený a využívaný firmou Bohemia Interactive a slouží k tvorbě 3D her. Engine umožňuje vytvářet svět, animaci, hudbu, skriptování atd.

Tato kapitola se věnuje práci s enginem. Nejdříve je probrán jazyk, ve kterém se Enfusion skripty vytvářejí. V práci je pak zmíněno použití objektů, jak vypadá a jak se tvoří svět v enginu, řešení ohraničení světa (neboli konec mapy), rozpohybování objektu a jaké nekonzistentní stavy se v prostředí mohou nacházet.

4.1 Jazyk

Skripty se píší v jazyce velmi podobném jazykům C#, C++ a C. Je to objektově orientovaný jazyk. Třída je generická šablona pro množinu objektů stejného typu. Definuje jejich chování a atributy. Tento jazyk je silně typovaný. To znamená, že při překladu se provádí silná typová kontrola. Překladač kontroluje správné použití datových typů.

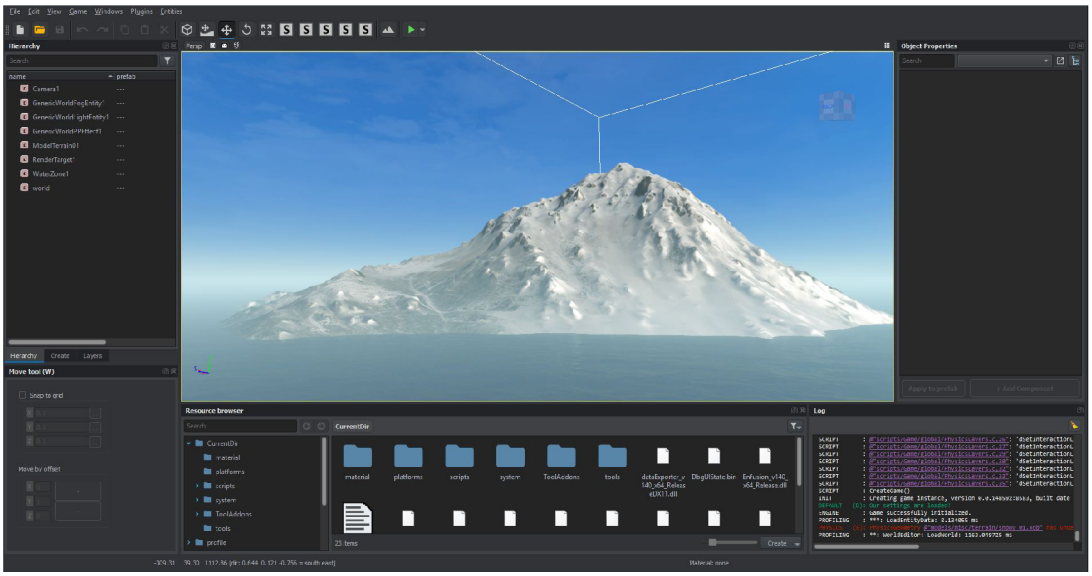
Veškeré metody a proměnné jsou implicitně veřejně viditelné (angl. *public*) a tím pádem jsou viditelné všemi ostatními třídami. Je-li potřeba tuto vlastnost změnit, použije se klíčové slovo *private* či *protected*. *Private* zakáže používat metody a proměnné třídám, které je nevlastní. *Protected* povolí použití u tříd jim vlastní i jim odvozeným.

Metody jsou virtuální a jeho potomek je může přepsat příkazem *override*. Třída může dědit jenom po jednom rodiči klíčovým slovem *extends*. Objekty je nutné vytvořit a zničit pomocí příkazů *new* a *delete*. Veškeré proměnné jsou při vytvoření inicializovány na výchozí hodnotu. Výchozí hodnoty závisí na datových typech. U číselných datových typů je výchozí hodnota 0 či 0.0. U řetězců je to "". U boolovských hodnot je to hodnota false. Vektor je uspořádaná trojice čísel. Neinicializován obsahuje hodnoty „0 0 0“.

Tento engine momentálně ještě neobsahuje mnoho funkcionalit, které se pravděpodobně budou přidávat v budoucnu. Například ačkoli lze vytvořit dvourozměrné pole, není možné do něj bez chybové hlášky zapisovat (ale existuje zde možnost vytvořit pole vektorů, který byl popsán výše).

4.2 Objekt

Svět se vytváří pomocí 3D objektů ve formátu *autodesk fbx*[2], který je běžně podporovaný mnoha programy pro tvorbu 3D objektů. S objekty lze hýbat, otáčet, měnit velikost či jim přidat pomocí kódu účel, jenž budou vykonávat.



Obrázek 4.1: Enfusion - World Editor

Objekt jako takový nemá žádný tvar a nezabírá žádný prostor. Aby byl objekt viditelný pro hráče hry a aby měl nějaký tvar, je nutné mu ho nastavit. Po přidání objektu do světa ve *World Editoru*, se v pravé části obrazovky objeví *Object Properties*, kde je možné přidat objektu funkce, nastavit jeho vlastnosti a přidat další komponenty. Přidáním komponenty *MeshObject* vznikne okénko, kde je možné vybrat tvar objektu uložených ve formátu *autodesk fbx*. Tímhle vznikne viditelný objekt světa.

Takový objekt, ale ještě nezabírá žádnou plochu a neřeší kolize s jinými objekty. Aby objekt zabíral prostor, je nutné mu přidat komponentu *RigidBody*, kde po nastavení *ModelGeometry* vznikne propojení s *MeshObjectem*.

Takhle propojený objekt může být statický či dynamický. Je-li objekt nastaven jako statický, neovlivňuje ho gravitace. To znamená, že zůstane stát na místě, kam ho programátor postaví. Dynamický objekt je ovlivňován gravitací. Ta udává, jakou silou bude objekt přitahován k objektu pod ním, pokud zde nějaký je. Jinak bude propadat do bezedné propasti. V enginu Enfusion je nastavena na hodnotu 9,81, což je hodnota tíhového zrychlení v České republice.

Pohybovat s objektem lze dvěma způsoby. Buď pomocí změny souřadnic, nebo mu lze nastavit sílu, která začne na objekt působit a tím pádem ho rozpohybuje.

První možnost je velmi jednoduchá na pochopení i programování. Každý objekt má souřadnice *x y z*, které udávají, kde se ve světě nachází. Tyto souřadnice lze nastavit ve *World Editoru* (a to rovnou dvěma způsoby) či ve skriptovacím editoru pomocí funkce *setOrigin()*. Tato funkce patří třídě entita (kterým objekt je) a přijímá vektor jako parametr. Neinicializován obsahuje hodnoty „0 0 0“. Při nastavování souřadnic objektu znamenají tato čísla „x y z“ souřadnici.

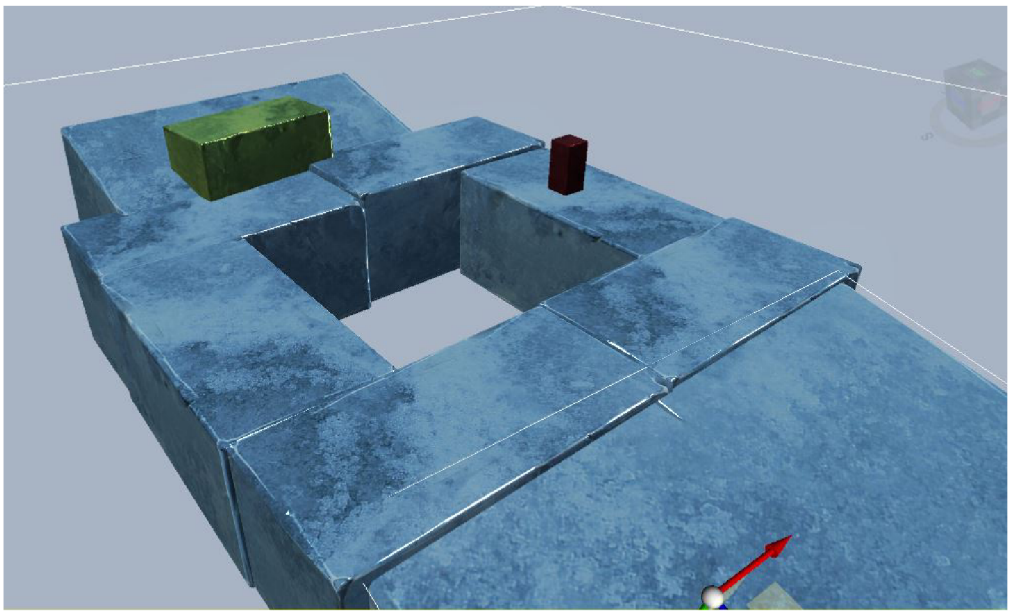
Druhým způsobem pohybu je nastavit hodnotu vektoru *velocity*, tím začne působit na objekt síla, která dá objekt do pohybu. Tuto metodu není možné použít ve *World Editoru*, ale jen ve skriptovacím editoru. Hodnota se nastavuje pomocí funkce *setVelocity()*, která přijímá jako parametr objekt a vektor. Funkce není součástí třídy entita, ale je veřejně viditelná všemi ostatními třídami, proto je nutné specifikovat pro jaký objekt se funkce bude používat. Vektor udává jakou silou se bude objekt pohybovat a jakým směrem. Například

při nastavení první hodnoty na 1, začne na objekt působit síla a objekt se pohne v x-ové souřadnici o velikost 1. Je-li potřeba s objektem pohnout daleko, musí se buď tato síla nastavit na vysokou hodnotu, nebo volat funkci opakovaně, neboť na objekt působí třecí síla, která ho časem zastaví.

4.3 Svět

Svět se skládá z velkého množství objektů, které jsou různě poskládány. Nejdůležitější objektem je země, která se bude testovat, a po níž budou hráči chodit. Země může být složená z jedné či více libovolných statických objektů. Dále se k nim přidají další objekty (statické či dynamické), které vytvoří svět s danou atmosférou. Například přidáním desítek stromů vznikne les. Stromem se samozřejmě nedá procházet, takže je nutné ho při procházení obejít.

Světy od firmy Bohemia Interactive mívají rozlohu několik stovek kilometrů čtverečních, ale bohužel z důvodu slabého hardwarového výkonu, nebylo možné testování provést na daném světě či jej jen otevřít a prozkoumat. Z tohohle důvodu byl vytvořen menší a jednodušší svět, na kterém proběhlo testování. Tento svět má rozlohu několik metrů čtverečních a je složen z několika jednoduchých kvádrových objektů, které na sebe navazují. Uprostřed byla vytvořena díra pro testování správné detekce děr. Kromě díry se zde nachází překážka (též byl použit objekt kvádru), s níž se postava musí při chůzi nějak vypořádat. Například obejít, přelézt atd. Bohužel nelze 100% ověřit, zda je nově vytvořený menší svět ve všem ekvivalentní se světem, který má být testován, neboť nemohl být prozkoumán. Toto může vést k malým nesrovnalostem, které se možná budou měnit při závěrečném testování.



Obrázek 4.2: Nově vytvořený testovací svět

Celá mapa je ohraničena ze čtyř stran neviditelnými bariérami. Tyto bariéry brání herní postavě dostat se za ni a označují konec mapy. Jsou to neviditelné, ale hmotné objekty, ohraničující mapu. Z důvodu nemožnosti prozkoumání bariér, se tato možnost při vytvoření menšího světa nevyužívá. Tento svět končí ze všech čtyř stran propastmi, které by měl

program detekovat jako díry. Tímhle způsobem vzniklo mnoho dalších míst k detekci. Konec mapy lze vidět na obrázku 4.2.

4.4 Překážky

Herní svět obsahuje mnoho objektů, které ho tvoří, jak již bylo zmíněno v podkapitole 4.3. Jsou to stromy, kameny, budovy atd. Překážka je objekt zabírající prostor, jímž nelze projít. Pro procházení mapy je to komplikace, neboť se musí nějakým způsobem vyřešit, aby se daná překážka obešla a postava mohla bez problému projít celý herní svět.

Jakmile nějaký hýbající se objekt narazí na překážku, vytvoří se zpětná síla, která tento objekt zastaví či zpomalí. Velikost síly závisí na rychlosti pohybujícího se objektu, neboli síle nárazu objektu do překážky.

Této vlastnosti testovací program využívá, aby zjistil, že se před postavou nachází překážka a není možné dále pokračovat v chůzi daným směrem.

4.5 Nekonzistentní stavy

Jednou z nejdůležitějších nekonzistencí návrhu je chybějící pevná část země. Toto místo není možné rozeznat pouhým okem. Hráči i vývojáři se jeví jako normální textura země, ale obsahuje v sobě nepevné části, tudíž při vstupu herní postavy na tuto část textury začne postava padat do bezedné propasti, ze které není možné se dostat. Místo bylo nazváno jako díra. Pohled postavy z díry lze vidět na obrázku 4.3 ze hry Kingdom Come Deliverence [8].

Obrázek 4.3: Pohled postavy z díry ze hry Kingdom Come Deliverence [12]

Dalším nekonzistentním stavem je, když se postava dostane mezi dva objekty, které jsou umístěny moc blízko sebe. Postava začne vyvozovat neočekávané chování a stává se neovladatelnou. Může se například točit dokolečka, odletět na druhou stranu mapy či se seknout mezi těmihle objekty. Tyto stavy bývají velmi těžko objevitelné, protože se do nich lze dostat jen při správném natočení a rychlosti postavy. Z tohohle důvodu nebylo možné tyto

stavy simulovat a tím pádem nebudou testovacím programem testovány. Tato funkcionalita se plánuje přidat v budoucnu.

Kapitola 5

Tutoriál „Steal the car“

Protože je Enfusion stále ještě ve vývoji, neexistuje v současné době velké množství návodů k práci s tímto vyvíjeným enginem. Jediné, co zatím existuje, je oficiální (doposud nezveřejněný) výukový úkol, jehož cílem je vytvořit minihru. Tento úkol umožní uživateli lépe se seznámit s prací s enginem.

Jak již název napovídá, jedná se o misi, kde hráčská postava má za úkol ukrást auto. Tato mise se skládá z nalezení auta, následného zjištění, že nemá klíče, nalezení oněch klíčů a doprava auta z bodu A do bodu B. Minihra má ovšem několik podmínek, které se musí dodržet. Hra nesmí hráči umožnit vstoupit do auta bez klíčů, musí mu oznámit, že nemá klíče, ukázat, kde se klíče zhruba nacházejí, a kam má s autem dojet.

Uživatel má k dispozici veškeré prostředky pro tvorbu mise (triggery, postavu, ...), bez nichž by misi nebyl schopen vytvořit (pokud by si je nevytvořil sám, ale to by musel mít dřívější znalosti s prostředím).

5.1 Prvky

Nejdůležitějším prvkem mise je postava. Postava je tvořena fyzickou postavou a dvěma kamerami. Na obrázku 5.1 je znázorněno postavení obou kamer vůči dané postavě. Mezi jednotlivými kamerami lze při hraní libovolně přepínat a vybírat si tak úhel pohledu.

Postavu tvoří mnoho skriptů pro pohyb a vývoj takové herní postavy trvá i několik let. Jako většina herních postav v dnešní době se pohybuje pomocí kláves WASD a pomocí klávesy E provádí veškeré interakce (například zvedá klíče, vstoupí do auta atd). Drží-li se s klávesou pohybu i klávesa *shift*, bude postava místo chůze běhat. Po stisknutí klávesy *ctrl* se postava přikrčí a při dalším stisknutí klávesy *ctrl* si postava lehne. V obou případech lze s postavou pohybovat pomocí kláves pohybu. Ležící postava se zvedne do dřepu pomocí klávesy *mezerník* a krčící se postava si stoupne též stisknutím klávesy *mezerník*. Tím pádem postava může chodit, běhat, krčit se a plazit se.

Dalším prvkem je auto, které má hráč odcizit. Nastoupí-li postava do auta, změní se aktivní prvek na auto (hráč bude ovládat auto), přičemž hráčská postava musí zmizet. Auto se ovládá stejně jako postava.

Mezi důležité prvky patří i:

- EventMissionStart a EventMissionEnd - udávají, že daná mise začala či skončila.
- Trigger - podmínka spouštějící událost. Např. hráč dojel s autem na konkrétní místo a mise je splněna.

Obrázek 5.1: Postava

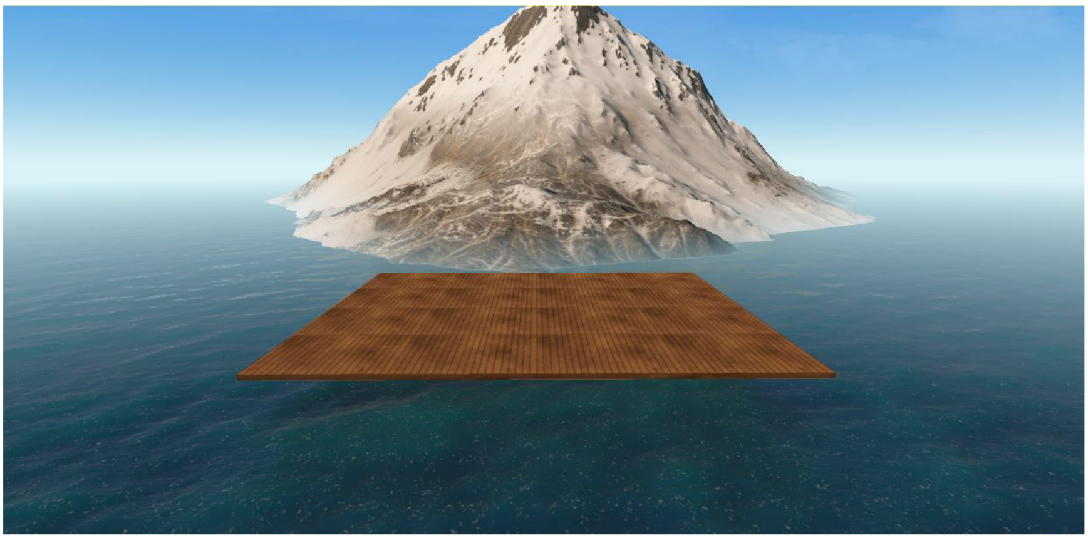
- EventPlayerSelect - propojí daný prvek s hráčskou postavou. Toto se například používá s prvkem EventMissionStart, který vytvoří misi pro konkrétní postavu.
- EventTimer - vykoná prvek po uběhnutí časového intervalu, který se nastaví. Například mise začne po uběhnutí 2 sekund, nikoliv bezprostředně po spuštění minihry.
- EventObjectCreate - vytvoří podúkol. Například najdi klíče.
- EventWayPoint - ukáže hráči, kde se něco nachází. Ku příkladu ukáže místo, kde se nacházejí klíče.
- EventHint - vypíše nápovědu. Na příklad: „Abys ukradl auto, potřebuješ nejdřív klíče.“
- InteractionPoint - slouží k interakci s nějakým objektem. Například zvednutí klíčů.
- ObjectChangeState - oznámí, že je podúkol splněn/nezdařil se, apod.

5.2 Tvorba minihry

V podkapitole 5.1 byly představeny nejdůležitější prvky pro tvorbu minihry. V této části bude vysvětleno, jak všechny prvky využít pro vytvoření minihry.

Minihra se bude tvořit ve světě *Default.ent*. Svět se nachází v levé liště nazvané *Resource browser* ve složce *ReforgerTemplate* a v ní ve složce *worlds*. Ta obsahuje jednoduchou placatou plochu, již lze vidět na obrázku 5.2. Tato plocha bude stačit pro vytvoření minihry. Svět je připraven pro osazení plochy potřebnými prvky ke splnění zadaného úkolu „Steal the car“.

Nejdříve je nutné přidat herní postavu. Pro její přidání je nutné v dolní liště rozkliknout *ReforgerTemplates -> templates -> character* a *Character.et* přetáhnout do herního světa. Tím se vytvoří postava *Character1*. Přidanou postavu lze vidět kromě v herním světe i v levé liště na stránce *Hierarchy*. Pro vyzkoušení ovladatelnosti postavy lze minihru spustit. Ta se spustí zelenou šipkou v horní liště. Postava i ovládání bylo popsáno v podkapitole 5.1.

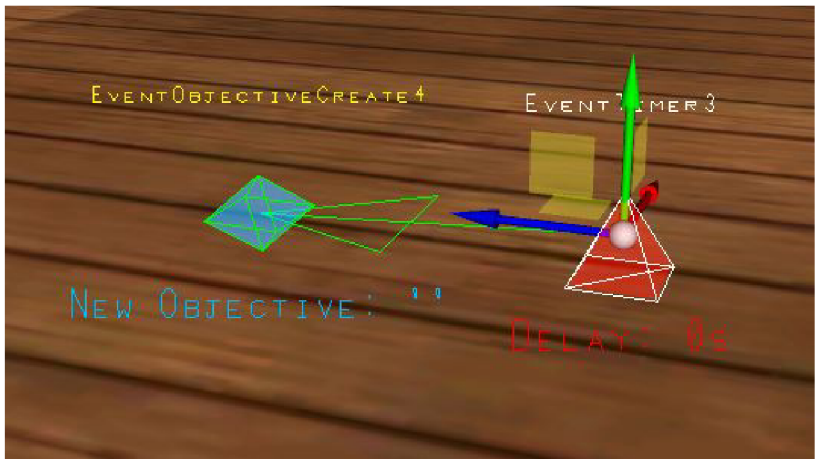


Obrázek 5.2: Svět pro tvorbu minihry

Stejným způsobem se přidá auto. To se nachází též v dolní liště ve složkách *Reforger-Templates -> templates -> vehicles -> car*. Tam se nachází *CarOffroad_02.et*, který se stejným způsobem jako postava *Character.et* přesune do herního světa.

Zbylé prvky, které byly zmíněny v podkapitole 5.1 se nacházejí v levé liště na stránce *Create* ve složce *_C_Placeable*. Prvky se též přidávají do herního světa přetáhnutím.

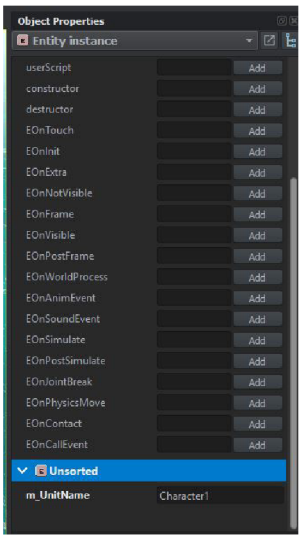
Aby minihra správně fungovala, je důležité tyto prvky správně propojit. Propojení prvků se provádí tak, že se nejdříve kliknutím myši označí cílový prvek (kam má směřovat šipka), poté se klikne na zdrojový prvek současně se zmáčknutou klávesou *Ctrl* a nakonec se stiskne klávesa L. Tím se vytvoří spojení mezi dvěma prvky. Správnost lze ověřit šipkou, která by měla směřovat ze zdrojového do cílového prvku. Šipku lze vidět na obrázku 5.3.



Obrázek 5.3: Propojení objektu EventTimer a EventObjectiveCreate

Aby mise mohla začít, je nutné přidat prvky *EventMissionStart* a *EventPlayerSelect*. Tyto dva prvky je nezbytné mezi sebou propojit. Následně je nutné navázat prvek *Event-PlayerSelect* s postavou *Character1*, aby bylo jasné, na kterou postavu se mise váže. Při kliknutí na prvek *EventPlayerSelect* se v pravé liště nazvané *Object Properties* na konci

nachází v okně *Unsorted* políčko *m_UnitName*. *Object Properties* lze vidět na obrázku 5.4. Zde se napíše jméno postavy (zde *Character1*). Tím se vytvoří spojení mezi postavou *Character1* a *EventPlayerSelect*. Propojení lze ověřit pomocí šipky, která se vytvoří mezi těmihle dvěma prvky. Tímhle způsobem se vytvořila mise pro postavu *Character1*, která se zapne po spuštění minihry.



Obrázek 5.4: Object Properties

Aby mise nezačala ihned po spuštění minihry, je vhodné přidat prvek *EventTimer*, který po nastavení času zpozdí začátek mise o nastavený čas. Zpoždění se nastaví v liště *Object Properties* v okně *Unsorted* v políčku *m_timeDelay*. *EventTimer* je nutné propojit s prvkem *EventPlayerSelect* pomocí šipky.

Cíl mise oznamuje prvek *EventObjectiveCreate*, který je potřeba přidat, aby hráč věděl, jaký je cíl mise. Ten se propojí s prvkem *m_timeDelay* a v pravé liště se nastaví název a cíl mise v okně *Unsorted*. Název mise se během hry nezobrazuje. Slouží jen pro rozlišení více misí a nastavuje se v políčku *m_ObjectiveName*. Cíl mise se nastavuje v *m_ObjectiveTitle* a zobrazuje hráči, jaký je cíl mise. V této minihře je zde například *Steal the car*.

Prvkem *EventWayPointSet* se ukáže text na mapě během puštěné minihry. Po propojení s *EventObjectiveCreate* ukáže na mapě libovolný text, který byl napsán v políčku *m_WaypointText*. V této minihře je to například označení umístění auta, které lze vidět na obrázku 5.5. Mimo jiné lze i změnit barvu textu či přidat obrázek pomocí políček *m_WaypointImage* a *m_Color*.

Cílem mise je ukrást auto. Nejdříve ho hráč musí najít, pak následuje zjištění, že pro splnění mise je potřeba nejprve najít klíče. K tomu slouží *Trigger*, což je oblast, do které když hráč vejde, se něco stane. Po propojení s prvkem *EventHint* se při hraní na obrazovce objeví text, který oznámí hráči, že v autě nejsou klíče a že je nejdříve potřeba je najít. *EventHint* obsahuje políčko *m_HintText* pro vypsání textu na obrazovku při hraní a políčko *m_ShowTime*, které udává, jak dlouho se bude text na obrazovce zobrazovat.

Dále vytvoří podúkol *Find keys*, aby hráč věděl, co musí udělat pro splnění mise *Steal the car*. To se vytvoří pomocí prvku *EventObjectiveCreate*. Ten se propojí s *EventHint*. *EventObjectiveCreate* obsahuje políčka *m_ObjectiveName* a *m_ObjectiveTitle*. Do *m_ObjectiveName* se napíše označení podúkolu (například *Mission 1a*). Toto označení slouží pro prvek *Event-*

Obrázek 5.5: Ukázka prvku EventWayPointSet ve hře

ObjectiveChangeState, aby mohl změnit konkrétní stav podúkolu (jako je třeba splněno, zrušeno atd). Do *m_ObjectiveTitle* se napíše daný podúkol (třeba najdi klíče).

EventObjectiveCreate se propojí s *EventWayPointSet* oznamující hráči při hraní, kde se klíče od auta nacházejí. Ten se propojí s *InteractionPoint*, aby bylo možné klíče z daného místa vzít. Ten obsahuje políčka: *m_IconPath*, *m_IconText*, *m_MaxDistance*, *m_MaxAngle*, *m_Enabled*, *m_OnlyFromCurrentUnit* a *m_Once*. Do políček *m_IconPath* a *m_IconText* se nastavuje ikona a text, který se při možné interakci bude zobrazovat. V minihře stačí, aby se zobrazoval text: „Vezmi klíče (E)“. Do políček *m_MaxDistance* a *m_MaxAngle* se vepíše maximální vzdálenost a úhel interakce. Toto lze nechat, jak bylo původně nastaveno (2 a 30). Políčka *m_OnlyFromCurrentUnit* a *m_Once* je potřeba zaškrtnout, aby s klíči mohl hráč interagovat a aby je nemohl vzít víckrát než jednou.

InteractionPoint se propojí s *EventObjectiveChange*. Ten, jak již bylo zmíněno, obsahuje políčko *m_ObjectiveName*, do kterého se napíše název, který byl napsán v *EventObjectiveCreate*. Dále obsahuje políčko *m_NewState*, ve kterém je možné vybrat stav podúkolu. Stavy, které je možné vybrat, jsou: *Completed*, *Ongoing*, *Failed* a *Cancelled*. Na daném místě je potřeba oznámit hráči, že klíče našel a vzal si je, takže se nastaví stav *Completed*, což mu oznámí, že podúkol splnil.

EventObjectiveChange se propojí s *EventWayPointSet*. Ten oznamuje, že je možné nastoupit do auta. Pak se přidá další *InteractionPoint*, který se dá k autu a umožní pak hráči nastoupit do auta při interakci s ním. Je potřeba nastavit stejné parametry jako u *InteractionPoint* u klíčů.

InteractionPoint se propojí s *EventPlayerSelect*. Ten se naváže na auto *CarOffroad_1* pomocí pole *m_UnitName*.

Aby se při nastoupení do auta postava *Character1* nezobrazovala, je nutné ji pomocí prvku *EventBase* schovat. Ten se propojí s prvkem *EventPlayerSelect*, který zvolí auto jako herní prvek a hráč začne ovládat auto a ne postavu. Jako všechny ostatní prvky i *EventBase* obsahuje kód, který je možný mu přidat. Doteď to nebylo nutné použít, ale je-li potřeba schovat postavu *Character1*, tak je vhodné tuto vlastnost využít. V pravé liště je možné najít mnoho funkcí, které se mohou přepsat. V této minihře se přepíše funkce *EOnCallEvent*. Při stisknutí tlačítka *EDIT* se objeví skriptovací editor, do kterého je možné kód psát a vytvoří funkci se všemi parametry. Do funkce se vepíše

g_GameBase.ShowUnitByName(„Character1“, false). Tím bylo docíleno toho, že postava *Character1* při nastoupení do auta zmizí.

Mise *Steal the Car* tím byla splněna a je vhodné hráči oznámit pomocí *EventObjectiveChangeState*, že se mu povedlo auto odcizit a pomocí *EventObjectiveCreate* mu říct, že je potřeba ho dopravit na požadované místo. Toto místo je vidět na obrazovce pomocí *EventWayPointSet*. Například je to „Doprav auto sem!“.

Oblast, do které má hráč dopravit auto, je *Trigger* propojený s autem *CarOffroad_1*, který zaznamená přijetí auta do této oblasti, a s dalším *EventObjectiveChangeState*, který změní stav úkolu „Doprav auto sem!“ na splněno.

Je dobré přidat prvky *EventWayPointClear*, *EventHint*, který oznámí, že je mise splněna, a *EventTimer*, aby se text stihl zobrazit na obrazovce a nezmizet po ukončení mise. Nakonec se přidá *EventMissionEnd*, který ukonči misi.

Tímhle způsobem se vytvoří mise „Steal the Car“. Misi je samozřejmě možné doplnit o mnoho dalších prvků, ale pro seznámení s enginem je to dostačující. Kompletní misi lze nalézt v souboru mujStealtheCar.ent.

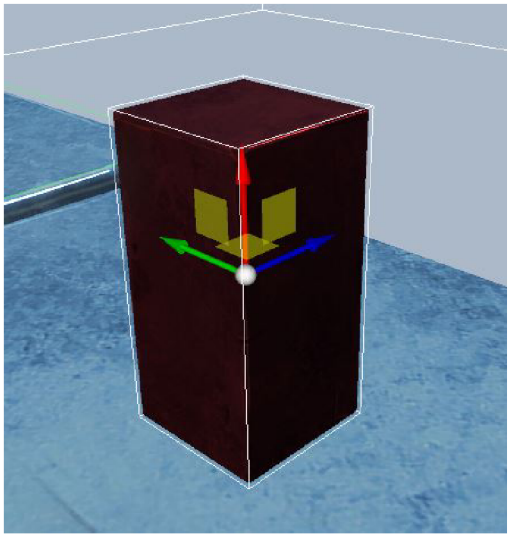
Kapitola 6

Návrh testování

Tato kapitola se věnuje návrhu řešení. Je zde popsáno, jaké prvky jsou použity, jak je nastavit, aby fungovaly správně a jakým způsobem se detekují nekonzistentní stavy.

6.1 Obálka

Cílem testování je odhalit místa, kam by se postava mohla dostat a kde by mohla propadnout, i když by to nemělo být možné. Postava jako taková obsahuje mnoho funkcí a akcí, které smí vykonávat. Pro testování (v engine) je potřeba jen několik z nich, proto se nemusí testovat pomocí (herní) postavy, nýbrž si stačí vytvořit (libovolný) objekt, jenž bude substituovat postavu pouze s těmi vlastnosti, které testovací program využívá. Tento objekt se nazývá obálka. Je tedy autonomním agentem. Obsahuje všechny funkce, které jsou pro testování objektu nezbytné a vnější vlivy mají na obálku stejný efekt jako na postavu (na příklad gravitace působí na obálku stejnou silou jako na postavu atd).



Obrázek 6.1: Obálka

Obálka se vytvoří tak, že se ve *World Editoru* do světa přetáhne prvek, na nějž se váže kód, který bude obálka vykonávat. Následně se přidají komponenty popsané v podkapitole 4.2. V této práci byl jako *MeshObject* nastaven kvádr, který splňuje vlastnosti postavy a zabírá přibližně stejnou plochu jako stojící postava. Bude-li se chtít testovat postava,

která se plazí či se hýbe v podřepu, lze *MeshObject* nastavit na jiný objekt a přiblížit vlastnosti obálky danému jevu.

Protože je svět tvořen z podobných kvádrů jako je tvořena obálka, byla obálka obarvena pomocí změny materiálu, aby nesplývala se zemí.

Obálka se musí postavit správným směrem k zemi, jinak by se mohlo stát, že testování neproběhne správně. Toho lze docílit buď otočením objektu, nebo zadáním hodnot v *Transformation*, který se nachází v pravé části *World Editoru* u *MeshObject*. *Transformation* obsahuje 3 políčka, do nichž je možné napsat číselné hodnoty udávající otočení objektu. Jsou to hodnoty *angleX, angleY* a *angleZ*. Aby obálka byla správně postavena je potřeba nastavit hodnoty *angles*. Proměnná *angleX* se nastaví na hodnotu 0, proměnná *angleY* na hodnotu 180 a proměnná *angleZ* na hodnotu -90. Tímto způsobem se obálka natočí správným směrem k zemi a okolí.

S objekty se hýbe pomocí síly, a protože lze na kvádr působit ze 6 různých směrů, musí se nastavit v *CenterOfMass* správně hodnoty. Tyto hodnoty udávají, jak budou působit síly na obálku. Špatné nastavení těchto hodnot může znamenat nekorektní a neočekávané chování obálky při pohybu. Například se může rolovat po ploše. Tento efekt není pro pohyb postavy ideální, protože se postava při obyčejném pohybu nekutálí. Je tedy potřeba zařídit, aby se obálka hladce hýbala po povrchu země. *CenterOfMass* obsahuje 3 různé hodnoty „X Y Z“. Protože je obálka situována na výšku, je nutné nastavit hodnotu X na hodnotu -1 a ostatní hodnoty na hodnotu 0. Při testování plazící se postavy, budou tyto hodnoty nastaveny na jinou hodnotu, aby se simulovalo plazení správně.

Obálka představující plazící osobu by bylo situována na šířku, tím pádem by se hodnoty musely nastavit jinak. Pokud se vezme stejný *MeshObject* jako při stojící postavě, tak při nastavení hodnot *CenterOfMass* na stejné hodnoty jako při chodící postavě by se obálka při prvním pohybu zvedla na výšku. Ta by pak prošla mapu stejným způsobem jako chodící postava a testování by nebylo provedeno správně. V tomto případě se hodnota X nastaví na 1 a zbylé hodnoty na 0. Bez tohohle nastavení se obálka nebude pohybovat jako plazící se postava.

6.2 Průchod a detekce děr

Cílem testování je odhalit nekonzistentní části mapy. Nejdůležitější z nich jsou tzv. díry. Co to je díra, bylo popsáno v podkapitole 4.3. Obálka se hýbe po mapě, hledá díry i případné překážky a zaznamenává je do pole. Výsledkem testování je pak pole oznamující, zda se na mapě nacházejí nějaké díry a překážky se souřadnicemi, kde se nacházejí. K testování byla použita metoda *backtracking*. Ten byl popsán v podkapitole 3.3.

Program prohledá prostor o velikosti $X \cdot Y$, kde X udává šířku prohledávaného prostoru a Y jeho délku. Z důvodů velkého množství možných míst, které lze prohledat, byly v testovacím programu nastaveny hodnoty X = 20 a Y = 10. Obálka projde prostor o velikosti $20 \cdot 10$ podle zadaných pravidel. Toto nastavení může zapříčinit, že se neotestuje celá mapa, ale jen její část. Aby se otestovala celá mapa je nezbytné nastavené hodnoty změnit (nebo při výkonným počítači přidat do světa obálek více a rozložit je, aby si nepřekáželi).

Algoritmus se vykonává následovně:

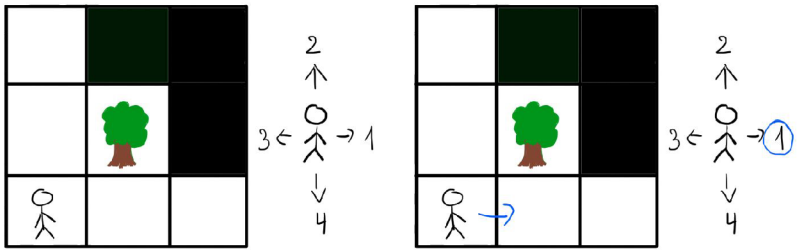
```
Vytvoř a inicializuj matice move[0..x; 0..y] na nulu;
vytvoř inicializuj pole vektorů positions[0..x; 0..y] na nulu;
vytvoř a inicializuj pole history[0...x*y];
nastav i = 0, j = 0, h = 0;
```

```
opakuj:
    když h < 0: skonči testování;
    otestuj výskyt díry;
        je tam:
            do matice move[i; j] označ díru a všechny
               okolní body jako neprůchozí;
            jdi zpět;
    otestuj výskyt překážky:
        je tam:
            do matice move[i; j] označ překážku;
            jdi zpět;
    když move[i; j] = 0:
        nastav move[i; j] = 1;
        když i != x:
            ulož pozici obálky do vektoru position[i; j];
            do pole history[h] = k;
            zvyš h o 1;
            jdi doprava;
            zvyš hodnotu i o 1;
    když move[i; j] = 1:
        nastav move[i; j] = 2;
        když j != y:
            ulož pozici obálky do vektoru position[i; j];
            do pole history[h] = k;
            zvyš h o 1;
            jdi nahoru;
            zvyš hodnotu j o 1;
    když move[i; j] = 2:
        nastav move[i; j] = 3;
        když i != 0:
            ulož pozici obálky do vektoru position[i; j];
            do pole history[h] = k;
            zvyš h o 1;
            jdi doleva;
            sniž hodnotu i o 1;
    když move[i; j] = 3:
        nastav move[i; j] = 4;
        když j != 0:
            ulož pozici obálky do vektoru position[i; j];
            do pole history[h] = k;
            zvyš h o 1;
            jdi dolů;
            sniž hodnotu j o 1;
        jinak:
            jdi zpět;
    jinak
     jdi zpět;
```

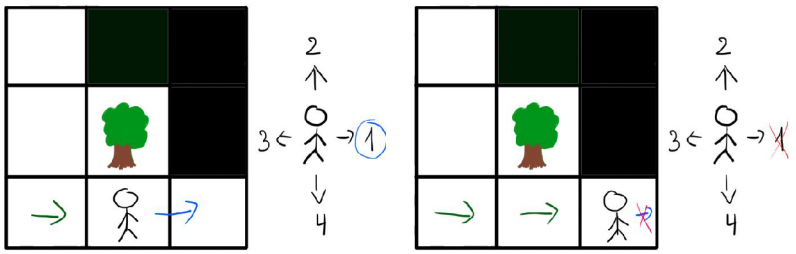
Vynoření (neboli jdi zpět) funguje následovně:

```
když h <= 0:
    nastav h na -1;
jinak:
    opakuj:
        sniž h o 1;
        do k nastav history[h];
        když move[k] obsahuje jiné hodnoty než pohybové:
            skonči;
        přesuň obálku na pozici positions[k];
```

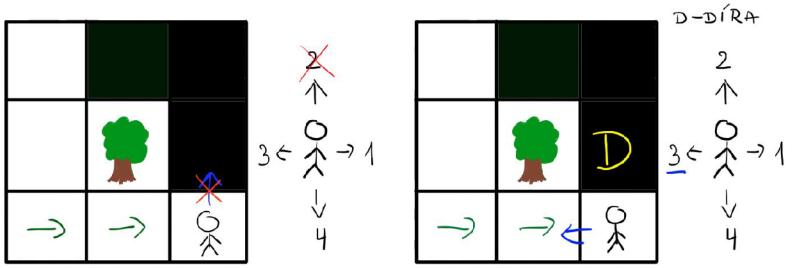
Fungování algoritmu je znázorněno za obrázcích 6.2 až 6.4, které zobrazují prvních pět kroků. V pravá části jde vidět jak obálka vybírá aplikovaný směr.



Obrázek 6.2: Ukázka pohybu obálky po mapě 3x3 s detekcí překážky a děr a první krok.



Obrázek 6.3: Druhý a třetí krok.



Obrázek 6.4: čtvrtý a pátý krok.

V podkapitole 4.3 bylo vysvětleno, že díra je místo bez pevné části, které když obálka navštíví, tak začne padat. Při zkoumání chování obálky na takovém místě, bylo zjištěno, že při padání obálka snižuje hodnotu proměnné *velocity[1]*. To znamená, že když se obálka pohybuje směrem dolů, je proměnná *velocity[1]* záporná, a když se pohybuje směrem nahoru, je kladná. To bylo provedeno tak, že se do světa bez země přidala obálka. Ta vypisovala hodnoty vektoru *velocity*. Při spuštění programu, začala obálka padat a vypisovat hodnoty. Bylo zjištěno, že čím obálka byla níž, tím menší byla hodnota *velocity[1]*. Při hodnotě kolem -30 obálka zmizela, přestala vypisovat hodnoty a nebylo ji již možné na mapě vidět.

Jednou z možností řešení detekce díry je pohnout s obálkou zpátky na místo, kde stála předtím než vydala daným směrem. Pokud by byla obálka schopná dostat se na danou pozici, bylo by místo bez díry. Jenže pokud by se zde vyskytoval schod, nebylo by možné, aby se obálka vrátila zpátky. Spadla by na schod a při pokusu vrátit se, by na něj narazila. Toto by šlo otestovat, pomocí zjištění překážky při návratu. Ačkoli se metoda zdá funkční a programovatelná, nebyla při detekci použita.

Další možností detekce je pomocí třídy *contact*. Tuto třídu lze využít jenom ve funkci *EOnContact* a tato funkce se volá jen tehdy, dojde-li ke kolizí obálky s jiným objektem. Více informací o této funkci se nachází v kapitole 7. Při padání do díry, ale k žádným kolizím nedochází, a proto není možné tuto metodu použít. K detekci díry by bylo potřeba použít hodnotu proměnné *velocity[1]*.

Algoritmus detekce děr by vypadal asi takhle:

```
Zjisti hodnotu velocity;
když velocity[1] < -25 pak:
     nastav move[k] = -2;
     přesuň obálku na předchozí pozici;
```

Nejlepší vyzkoušená možnost detekce bylo pomocí pomocného objektu, který se vložil pod úroveň země. Při pádu by obálka jednou skončila na tomto objektu a dotyk obálky s pomocným objektem lze snadno detekovat. Je-li výška obálky podobná jako výška pomocného objektu, znamená to, že obálka propadla do díry. Toto místo je následně označeno jako díra.

Algoritmus detekce děr pomocí pomocného objektu:

```
Když Y souřadnice < Y souřadnice pomocného objektu + 1:
     označ díru do pole move[i; j];
     vyznač okolní body jako neprůchozí;
     jdi zpět;
```

Test na překážky probíhá pomocí třídy *contact*. Ten v sobě obsahuje impulse při kontaktu obálky s jiným objektem.

Test probíhá následovně:

```
Když je impulse dost veliký:
     označ překážku do pole move[i; j];
     jdi zpět;
```

Tímhle algoritmem projde obálka zadaný prostor a otestuje zadané nepřesnosti. Výsledkem průchodu je matice *move* obsahující hodnoty značící díry, překážky a bezproblémové části.

Kapitola 7

Implementace

Tato kapitola se věnuje implementaci pohybu obálky po okolí, detekci děr a překážek v okolí.

7.1 Popis detekce

Obálka je instance entity *GenericEntity*, která již má definované některé funkce. Tyto funkce se spustí při nastavení správných příznaků masky událostí a při splnění daných podmínek. Například funkce *EOnContact()* se vykoná pouze tehdy, dotkne-li se obálka nějakého jiného objektu. Je-li smazán příznak či nebyl nastaven, funkce se nevykoná, i když se podmínka pro spuštění splnila (obálka se dotýká či dotkla jiného objektu). Ačkoli funkce existují, nemají uvnitř žádný kód a ten je nutné přidat pomocí přepsání funkcí klíčovým slovem *override*. V řešení bylo využito funkcí *EOnInit(IEntity owner)* a *EOnContact(IEntity owner, IEntity other, Contact contact)*.

Kromě již existujících funkcí, byly vytvořeny pomocné funkce. Nejdůležitější jsou pro pohyb obálky do stran a pro otestování překážky v cestě. Funkce pro pohyb jsou: *JdiDoprava()*, *JdiNahoru()*, *JdiDoleva()* a *JdiDolu()*. Všechny tyto pohybové funkce mají jako parametr entitu *owner*. Funkce hýbou s entitou *owner* pomocí funkce *SetVelocity()*. Ta obsahuje parametry entitu a vektor. Rychlost a směr se nastaví pomocí vektoru *rychlost*. Při pohybu doprava se nastaví hodnota vektoru $rychlost[0] = 5$. Při pohybu nahoru se nastaví hodnota vektoru $rychlost[2] = 5$. Při pohybu doleva se nastaví hodnota vektoru $rychlost[0] = -5$. Při pohybu dolů se nastaví hodnota vektoru $rychlost[2] = -5$.

Testování překážky probíhá pomocí funkce *TestPrekazky*. Funkce má jako parametr třídu *contact*. Třída v sobě obsahuje doplňující informace o kolizi objektů. Obsahuje v sobě funkci *GetNormalImpulse()*, která vrací vektor impulsu kolizi objektů. Je-li hodnota dostačující, místo je označené jako neprůchozí a informace se uloží do pole *move* (pole bude popsáno níže). Čím vyšší hodnota impulsu, tím větší je náraz obálky do překážky. Bylo vypozorováno, že při hodnotě proměnné $velocity = +5/-5$ bude hodnota součtu absolutních hodnot proměnných *GetNormalImpulse()*[0] a *GetNormalImpulse()*[2] při nárazu větší než 2. Při pohybu po rovné ploše bez překážky je to malinko menší než 2. Tím pádem je-li $abs(GetNormalImpulse()[0]) + abs(GetNormalImpulse()[2]) > 2$, nachází se zde překážka. Ta se zaznamená do pole a místo je označeno jako neprůchozí.

Na začátku se vytvoří pole *move*, který označuje prostor, který je nutný projít. Ve verzi Enfusion, na kterém probíhal vývoj této práce, nebylo plně implementovaná podpora dvourozměrného pole, proto bylo implementováno jako jednorozměrné. Toto pole simuluje 2D prostor. Dále se vytvoří proměnné *posX* a *posY*. Tyto dvě proměnné značí šířku a délku

pole. V programu byly tyto hodnoty nastaveny následovně: $posX = 20$ a $posY = 10$. Velikost pole *move* byla nastavena na hodnotu 200 (neboli $X \cdot Y$). Kdyby šlo implementovat jako matice, tak by se vytvářela následovně: *move*[20][10]. Tyto hodnoty lze v budoucím testování libovolně změnit. Tím bylo vytvořeno jednorozměrné pole představující matici o délce *posX* a šířce *posY*. Hodnoty pole se inicializují na hodnoty 0. Tuto činnost dělá engine *Enfusion* automaticky, proto není nutné to dělat. Pro práci s polem je použita proměnná k, která může nabývat hodnot 0...199. Hodnota k se mění v závislosti na tom, kterým směrem se obálka pohybuje či kde se momentálně nachází.

Pro uložení aktuální pozice v prostředí slouží pole vektorů *positions*. Pole má též velikost $X \cdot Y$ a při přesunu obálky na další pozici se do *positions*[k] uloží pozice obálky v prostředí. Při návratu se obálka přesune na danou pozici podle tohohle pole.

Metoda využívá *backtracking*, proto je potřeba zaznamenávat veškeré tahy, které byly dosud provedeny a na jakém místě, aby bylo možné se na tato místa vrátit a vynořit se. K tomu bylo vytvořeno pole *history* o velikosti $4 \cdot X \cdot Y$. $X \cdot Y$ udává velikost pole a čtyřka značí počet možných pohybů na jednom místě. Pro indexaci slouží proměnná h, která se zvyšuje při libovolném nezpětném pohybu. Při vynořování se proměnná snižuje. Proměnná h značí počet pohybů, které je nutno vykonat při cestě z aktuální pozice po počáteční. Pole *history*[h] obsahuje aktuální pozici obálky (v poli *move*). Díky tomu (a díky poli *positions*) je možné obálku přesunout na předešlé pozice.

První volanou funkcí je konstruktor objektu. Ten vytvoří obálku, která byla pojmenována *Chuze*. Dále se zde nastavují příznaky pro provádění funkcí *EOnInit()* a *EOnContact()*. Tyto příznaky nastavuje funkce *SetEventMask()*.

Funkce *EOnInit()* se volá hned po konstruktoru. Tato funkce se volá jednou a to na začátku. Slouží k inicializaci a nastavení proměnných. Parametrem této funkce je *IEntity owner*. Je to entita samotné obálky. V této funkce se nastavuje pozice obálky do vektoru *positions*[0] pomocí funkce *GetOrigin()*. Tuto funkci obsahuje entita obálky a vrací vektor obsahující její souřadnice.

Funkce *EOnContact()* se volá vždy, dojde-li ke kolizi objektů. Parametry této funkce jsou *IEntity owner, IEntity other, Contact contact*. *Owner* značí entitu obálky, *other* je entita, které se obálka dotýká, a *contact* je třída udávající doplňující informace o kolizi objektů. Například materiál entit, impuls při kolizi, pozici kolize atd. Tato funkce vykonává veškeré testování, pohyby atd. Nejdříve se otestuje, zda se na místě nenachází díra. Následně se testuje výskyt překážky. Oba principy testováni byly popsány v podkapitole 6.2. Aby se při prvním pohybu po zpětném přesunu obálky nezačala chovat nepředvídatelně, je potřeba nastavit úhly obálky na počáteční hodnoty, zastavit ji nastavením hodnot *velocity* na 0 a zamezit její rotaci. Úhly se nastavují pomocí funkce *SetAngles()*, která bere jako parametr vektor obsahující úhly natočení a je součástí třídy *owner* (neboli je součástí obálky). Rotace se zamezuje funkcí *SetYawPitchRoll*, jejíž parametrem je vektor rotace a je též součástí třídy *owner*. Než se obálka posune, volá se dotyková funkce víckrát. Aby nedocházelo k špatné detekci překážky, ukládá se ID překážky a pozice obálky a při dalším dotyku se zjišťuje, zda jsou nové hodnoty s těmito hodnotami shodné. Tím se zajistí, že obálka nedetekuje víc překážek než se na místě nachází.

Následně se testuje hodnota v poli *move* a podle ní se rozhodne, co má obálka vykonat. Když se proměnná $move[k] = 0$, nastaví se hodnota $move[k]$ na hodnotu 1. Když se proměnná $move[k] = 1$, nastaví se hodnota $move[k]$ na hodnotu 2, atd. Tím se označí, že na dané pozici byl vykonán daný pohyb a při návratu se už nebude vykonávat. V programu bylo nastaveno, že prvním pohybem je pohyb doprava. Dalším je nahoru, pak doleva a jako poslední se vykoná pohyb dolů. Pohyb doprava je uložen v proměnné $move[k]$ jako hodnota

0, pohyb nahoru jako hodnota 1, pohyb doleva jako hodnota 2 a pohyb dolů jako hodnota 3. Nachází-li se zde hodnota -1, značí to překážku, hodnota -2 značí díru a hodnota 4 značí, že všechny pohyby na místě už byly aplikovány. Při těchto třech hodnotách se obálka vynoří a přesune zpátky na předešlou pozici, kde ještě může aplikovat nějaký pohyb.

Po prozkoumání celé mapy je na konci vypsáno pole *move* (jako dvourozměrné), které obsahuje výskyty děr, překážek a bezproblémových částí.

Kapitola 8

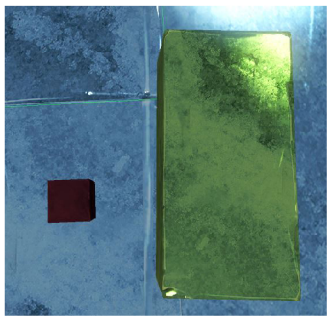
Testování

Tato kapitola se věnuje testování programu na testovacím světě. Ukazuje, jak správně dokáže program detekovat překážky a díry. Jaké výsledky vrací program při testování a jaké odchylky jsou od reálné mapy.

8.1 Detekce překážky

Obálka byla vložena do testovacího světa, který byl popsán v podkapitole 4.3 a podle popisu v podkapitole 6.1 správně nastavena. Dále byl umístěn pomocný objekt, který obálka detekuje, když spadne. Barva obálky byla nastavena na tmavě červenou, aby se odlišovala od modrého kvádru představující zemi. Překážky mají barvu zelenou a pomocný spodní objekt je zobrazen tmavě zeleně, aby se dalo správně odlišit objekty od sebe.

Nejdříve bylo testováno, zda obálka dokáže detekovat překážku. Ta byla postavena kousek od obálky a jejím cílem je detekovat celou hranu překážky. Pozici obálky a překážky a výsledek testování lze vidět na obrázku 8.1.



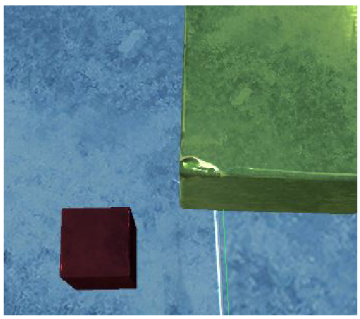
```
{84,4,4,4,4,4,4,4,4,-1,4,4,4,4,4,4,4,4,-1,0}
{4,4,4,4,4,4,4,4,4,-1,4,4,4,4,4,4,4,-1,4,-1}
{4,4,4,4,4,4,4,4,4,-1,4,4,4,4,4,4,-1,4,4,-1}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,-1,4,-1,4,4,4,-1}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,-1,4,4,4,4,-1,0}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,-1,4,-1,4,-1,0}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,-1,0}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,-1,4,-1,-1}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,-1,4,4,-1,-1,-1,4,4}
{-1,4,4,-1,-1,4,4,4,-1,4,4,4,4,4,4,4,4,4,-1,-1}
```

Obrázek 8.1: Pozice obálky a překážky s výsledkem testování

Dále bylo otestováno nalezení rohu překážky. To lze vidět na obrázku 8.2.

Levý obrázek ukazuje, který prostor v herním světě obálka otestovala. Při testování začíná na levém dolním okraji a testuje okolí směrem doprava a nahoru. Na pravém obrázku lze pak vidět výsledek testování. Kvůli špatnému přístupu k poli byl při výpisu prohozen směr nahoru a dolů, tím pádem se zdá, že se obálka hýbala doprava a dolů, nikoli nahoru. Na tento omyl je potřeba myslet při správné interpretaci výsledků.

Při prozkoumání výsledků zapsané v poli, si lze všimnout velkého množství hodnot -1. Tyto hodnoty oznamují, že se zde nachází překážka. U obrázku 8.1 je velké množství



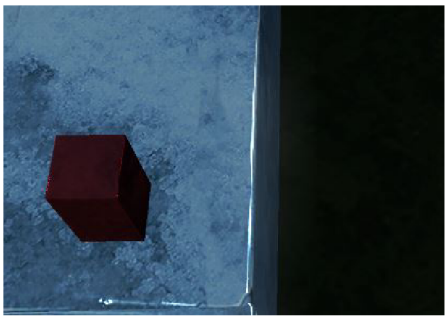
```
{84,4,4,4,4,4,4,4,4,4,4,-1,-1,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,-1,-1,4,4,4,4,4,-1}
{4,4,4,4,-1,4,4,4,4,-1,4,4,4,4,4,4,4,4,4,-1}
{4,4,4,4,4,-1,4,4,4,-1,4,4,-1,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,-1,4,-1,0,-1,4,4,4,-1,4,4,4,4,-1}
{4,4,4,4,4,4,4,4,-1,4,-1,-1,-1,-1,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,-1,4,-1,4,4,4,4,4,-1,4,4,4}
{4,4,4,4,4,4,4,4,-1,4,-1,-1,4,-1,4,-1,4,-1,4,4,-1,4,-1}
{4,4,4,4,4,4,4,4,-1,-1,0,-1,-1,4,4,-1,4,4,4,-1,4}
{4,4,4,4,4,4,4,4,4,4,-1,-1,-1,-1,-1,4,4,4,-1,4,4,4}
```

Obrázek 8.2: Nová pozice obálky a překážky s výsledkem testování

na pravé části a na obrázku 8.2 je to roh, kdy je mnoho hodnot -1 soustředěno kolem určité oblasti. V obou případech si lze všimnout malého zkosení, to je způsobeno neočekávaným chováním fyzikálních jevů v enginu.

8.2 Detekce díry

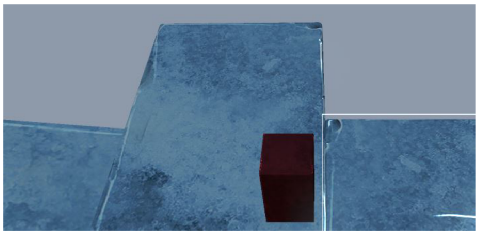
Jako další jev, který je nutný zachytit, je díra. Obálka byla postaveno blízko díry a bylo testováno, zda je schopná ji detekovat. Na obrázku 8.3 lze vidět její umístění a výsledek testování.



```
{84,4,4,4,4,4,4,4,4,4,4,4,4,-2,4,4,4,0,0,0}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,-2,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,-2,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,-2,4,4,4,4,4,-2,4,4,4,0}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,0}
```

Obrázek 8.3: Umístění obálky a díry k detekci + výsledek testování

Jako další bylo otestováno, zda je obálka schopná detekovat roh díry. Na obrázku 8.4 lze vidět testovaný svět a pozici obálky a výsledek testování.



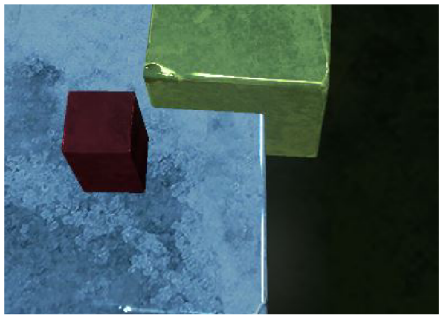
```
{84,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,-2}
{4,-2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,-2,4,4,4}
{4,4,4,-2,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,0}
```

Obrázek 8.4: Nové umístění obálky a díry k detekci + výsledek testování

Hodnoty -2 oznamují výskyt díry. Na obou obrázcích lze vidět přibližné umístění děr v herním světě i ve výsledném poli. Při bližším zkoumání lze vidět, že obálka byla schopna nalézt díry a detekovat v přibližném místě jako se nacházejí v herním světě.

8.3 Detekce díry a překážky

Jako poslední test bylo otestováno nalezení díry i překážky. Umístění veškerých prvků k detekci a obálku lze vidět na obrázku 8.5.



```
{78,4,4,4,4,4,4,4,4,4,4,4,-2,4,4,4,0,0,0}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,-2,4}
{4,4,4,4,4,-1,4,4,-1,4,4,4,4,4,-2,-1,4,4,4,4}
{4,4,4,4,-1,4,4,-1,0,-1,4,4,4,4,4,4,4,4,4,4}
{4,4,4,4,4,4,-1,0,0,0,-1,4,4,4,4,4,4,4,4,4}
{4,4,4,4,-1,-1,0,0,0,0,0,0,4,4,4,4,4,4,4,0,0}
{4,4,4,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

Obrázek 8.5: Umístění prvků k detekci a obálky + výsledek testování

Jak již bylo zjištěno v předchozím testování, výsledky detekce nejsou 100% přesné, ale lze odhadnout, zda se v herním světě nachází překážka, díra a kde se přibližně nacházejí. Při zkoumání výsledků poslední detekce lze zjistit, že se kolem obálky po pravé straně nachází díra a nahoru od ní překážka.

8.4 Shrnutí testování

Během testování bylo zjištěno, že hodnoty ve výsledném poli nejsou 100% přesné. Zobrazuje to, zda se na mapě nacházejí nějaké překážky a díry a jejích přibližné pozice. Nezobrazuje to ale přesnou lokaci ani velikosti daných jevů. Špatná detekce může být způsobena malým testovacím prostorem či neočekávaným chováním fyzikálních jevů v enginu. Není zcela možné simulovat pohyb, detekci a prostor pomocí dvourozměrného pole, neboť je zde mnoho rušících prvků, které na obálku působí a pohyb a náraz pak není ve všech směrem pořád stejný. Tyto nesrovnalosti lze vidět na všech obrázcích z testování. Zobrazuje to shluk minusových hodnot, ale nezobrazí přesnou velikost a tvar. Testovací program spíše oznámí, že v tomto okolí je nějaká nesrovnalost, než přesné umístění a tvar překážky či díry.

Kapitola 9

Závěr

Cílem práce bylo seznámit se s vývojem her a herních prostředí. Toto seznámení probíhalo v enginu Enfusion od firmy Bohemia Interactive a to výukovou formou, kdy bylo cílem vytvořit misi „Steal the car“. Mise se tvořila ve World Editoru za pomoci již definovaných prvků. Tyto prvky bylo potřeba správně propojit a některým z nich přidat vykonávající se kód. Na konci vznikla spustitelná minihra, kde hráč musí odcizit auto. Toto byl první krok pro seznámení se s enginem. Dále bylo potřeba prozkoumat prvky enginu a jazyk, ve kterém se následně implementovala další část práce.

Po seznámení se s enginem a World Editorem bylo třeba vytvořit herní svět, kde by probíhalo testování navrženého algoritmu pro hledání nekonzistentních stavů. Tento svět byl tvořen z kvádrů, které se na mapě různě rozmístili a znamenali buď zemí či překážku. Díra byla simulována jako strana kvádru představující zemi.

Pro otestování herních prostředí bylo potřeba navrhnout metodu řešení průchodu herního prostředí a detekce nekonzistentních stavů, aby zvýšil hráčům herní požitek z hraní. K tomu byl využit agent, jehož úkolem je detekovat nekonzistentní stavy. Protože se jedná jen o jednoho agenta (a ne víc) byla použita metoda tzv. backtracking. Agent na základě této metody prošel prostor pomocí veškerých možných pohybů a otestoval výskyt překážek a děr. Výstupem programu je pole označující pozice nalezených překážek a děr.

Z testování vyplynulo, že lze tuto metodu použít, ale její výsledky nejsou úplně přesné. To může být zapříčiněno neočekávaným chováním fyzikálních jevů a enginu Enfusion. Mnoho metod a jevů má funkcionalitu skrytou a nelze úplně odhadnout chování objektů ve světě (například náraz do objektů může zapříčinit rotaci objektu atd.). Kvůli tomu je výstup programu na některých pozicích nepřesný a může se stát, že označí místo jako díru či překážku, i když zde žádná není. Těchto pozic ale nebývá mnoho a proto je nebudeme označovat za reálné překážky a díry, i když jsou v poli označeny. Z tohohle důvodu nelze metodu použít na vyhledávání pozic nekonzistentních stavů, ale na oblasti, kde se dané stavy nacházejí.

Do budoucna se plánují určité úpravy, aby detekce fungovala přesněji. Plánuje se přidat detekce uváznutí objektu mezi dvěma jinými. Tento stav se ale nejdřív musí důkladně prozkoumat, a protože je nebylo momentálně možno nijak simulovat, takže nebylo možné určit, jak se obálka v dané situaci zachová. Plánuje se tento stav nějak vytvořit a prostudovat.

Další možnou (a pravděpodobně) plánovanou úpravou je detekovat místa pomocí více než jednoho agenta. Tím se zařídí projití celého prostoru (a ne jen části, jak je tomu v současné verzi algoritmu). Tito agenti by si rozdělili plochu a každý z nich by prošel svou část. Následně by vypsali pozice nekonzistentních míst.

Tyto testy (uskutečněné i plánované) sníží počet nekonzistentních stavů v počítačových hrách a zamezí možnostem snížit herní prožitek z hraní.

Literatura

[1] *ArmA*. [Online; navštíveno 26.07.2019].
URL https://cs.wikipedia.org/wiki/ArmA

[2] *Autodesk fbx*. [Online; navštíveno 30.07.2019].
URL https://en.wikipedia.org/wiki/FBX

[3] *Bohemia Interactive*. [Online; navštíveno 30.07.2019].
URL https://www.bohemia.net/

[4] *Datadisk*. [Online; navštíveno 30.07.2019].
URL https://cs.wikipedia.org/wiki/Roz%C5%A1%C3%AD%C5%99en%C3%AD_hry

[5] *DayZ*. [Online; navštíveno 26.07.2019].
URL https://www.bohemia.net/games/dayz

[6] *Engine*. [Online; navštíveno 30.07.2019].
URL https://cs.wikipedia.org/wiki/Hern%C3%AD_engine

[7] *GamerMaker*. [Online; navštíveno 30.07.2019].
URL https://www.yoyogames.com/gamemaker

[8] *Kingdome Come Deliverence*. [Online; navštíveno 30.07.2019].
URL https://www.kingdomcomerpg.com/cs

[9] *Metody prohledávaní stavového prostoru*. FIT VUT v Brně, [Online; navštíveno 26.07.2019].
URL https://www.fit.vutbr.cz/study/courses/IZU/private/1819izu_2.pdf

[10] *Unity*. [Online; navštíveno 30.07.2019].
URL https://unity.com/

[11] *Unreal*. [Online; navštíveno 30.07.2019].
URL https://www.unrealengine.com/en-US/

[12] Sulková, K.: *Pohled postavy z díry ze hry Kingdom Come Deliverence*. [Online; navštíveno 26.07.2019].
URL https://1url.cz/RMIVG

[13] Zbořil, F.: *Úvod do agentních a multiagentních systémů*. FIT VUT v Brně, [Online; navštíveno 26.07.2019].
URL http://www.fit.vutbr.cz/~zborilf/study/AGS/AGS01_Uvod.pdf