Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Herní enginy pro simulaci a analýzu komplexních systémů

Diplomová práce

Autor: Michal Horák
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mls Karel, Ing. Ph.D.

Hradec Králové 08/2023

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 10.8.2023 Michal Horák

Poděkování:

Děkuji vedoucímu diplomové práce doktoru Karlu Mlsovi za metodické vedení práce.

Anotace

Herní enginy pro simulaci a analýzu komplexních systémů

Cílem této diplomové práce je zjistit, zda je možné využívat herní enginy pro simulace komplexních systémů, kde se využívá systém agentů. Motivací pro tento experiment je ten, že univerzitou využívaný program NetLogo se ukázal být v určitých směrech těžko nastavitelný a neumožňuje rozšířit program jakýmkoliv způsobem.

Tato diplomová práce je rozdělena na dvě části, kdy v první části se probírá teoretická část, kdy v jednotlivých kapitolách budou popsány komplexní systémy, systémy agentů, nevýhody a výhody prostředí NetLogo a stručný přehled herních enginů.

V druhé části bude vysvětlen diplomový projekt, který byl vytvořen v herním enginu Unity a budou vysvětleny prvky projektu a zásadní problémy, které musely být řešeny, včetně návrhů na další rozšíření.

V závěru bude řečeno, zda je možné využít herní enginy pro simulování komplexních systémů.

Zdrojové kódy a spustitelná aplikace k diplomovému projektu jsou v repositáři GIT na adrese: https://github.com/voklik/DiplomovaPrace

Annotation

Title: Game engines for simulation and analysis of complex systems

The aim of this diploma thesis is to find out whether it is possible to use game engines for simulations of complex systems where an agent system is used. The reason for the creation of this experiment is that the NetLogo program used by the university turned out to be difficult to adjust in certain directions and does not allow the program to be extended in any way.

This diploma thesis is divided into two parts, where the theoretical part is discussed in the first part, where complex systems, agent systems, disadvantages and advantages of the NetLogo environment and a brief overview of game engines will be described in individual chapters.

The second part will explain the thesis project that was created in the Unity game engine and explain the elements of the project and the main problems that had to be solved, possible further extensions.

In the conclusion, it will be said whether it is possible to use game engines for simulating complex systems.

The source codes and executable application for the diploma project are in the GIT repository at:

https://github.com/voklik/DiplomovaPrace

Obsah

1 Úvod ... 1

2 Cíl práce ... 2

3 Teoretická část ... 3

3.1 Komplexní systémy a agentové systémy ... 3

3.2 Příklady využití simulačních prostředí ... 7

3.3 Hledání cest ve světě ... 9

3.3.1 Hledání cest ve 2D světech ... 9

3.3.2 Hledání cest ve 3D světech ... 13

3.3.3 2,5D světy ... 13

3.4 Stromy chování ... 14

3.4.1 Behavior Tree ... 16

3.4.2 Selector ... 16

3.4.3 Sequence ... 16

3.4.4 Inverter ... 16

3.4.5 Node ... 17

3.4.6 Variace stromu chování ... 17

3.5 Prostředí NetLogo ... 18

3.6 Herní enginy ... 21

3.6.1 Engine Unreal ... 22

3.6.2 Engine Unity ... 24

3.6.3 Engine CryEngine ... 26

3.6.4 Engine GameMaker: Studio ... 27

3.6.5 Engine Construct ... 29

4 Praktická část – projekt ... 30

4.1 Představení projektu a nastavení simulace ... 30

4.2 Strom chování ... 34
4.3 Základní komponenty editoru Unity ... 38
4.3.1 Transform ... 38
4.3.2 Camera ... 39
4.3.3 RigidBody ... 40
4.3.4 Collider ... 41
4.3.5 Material ... 42
4.3.6 Light ... 43
4.3.7 Canvas ... 44
4.3.8 Canvas Scaler ... 45
4.3.9 Rect Transform ... 45
4.3.10 Graphic Raycaster ... 46
4.3.11 Raw Image ... 47
4.3.12 Event System ... 47
4.3.13 Standalone Input Module ... 48
4.3.14 Scroll Rect ... 48
4.3.15 Button ... 49
4.3.16 Text ... 50
4.4 Systém navigace a systém agentů ... 51
4.4.1 NavMesh ... 51
4.4.2 NavMesh agent ... 51
4.4.3 Off-Mesh Link ... 53
4.4.4 NavMesh Obstacle ... 54
4.5 Řešené problémy ... 55
4.6 Rozdíl projektu oproti simulaci Predátor – kořist v systému NetLogo ... 56
4.7 Možná rozšíření projektu ... 57

5 Závěry a doporučení 59

6 Seznam použité literatury 62

Seznam obrázků

Obrázek [1]: 2D svět, který není příkladem klasické hledání cesty ... 9
Obrázek [2]: 2D svět, který může být příkladem hledání cesty ... 10
Obrázek [3]: 2,5D svět, který může být příkladem hledání cesty ... 10
Obrázek [4]: Technika záplavy ... 11
Obrázek [5]: Příklad konečného automatu ... 15
Obrázek [6]: Příklad hierarchického konečného automatu ... 15
Obrázek [7]: Příklad stromu chování pro zloděje ... 17
Obrázek [8]: Příklad simulace v aplikaci NetLogo(2D) ... 19
Obrázek [9]: Příklad simulace v aplikace Netlogo3D ... 20
Obrázek [10]: Příklad Unreal engine – blueprint visual script ... 23
Obrázek [11]: Příklad Unity uživatelského rozhraní ... 26
Obrázek [12]: Diagram představuje historii vývoje CryEnginu ... 27
Obrázek [13]: GameMaker:Studio editor ... 28
Obrázek [14]: GameMaker:Studio ... 29
Obrázek [15]: Construct 3 ... 30
Obrázek [16]: Rozhodovací strom ... 34
Obrázek [17]: Komponenta Transform ... 39
Obrázek [18]: Komponenta Camera ... 40
Obrázek [19]: Komponenta RigidBody ... 41
Obrázek [20]: Komponenta Mesh Collider ... 42
Obrázek [21]: Komponenta Material ... 43
Obrázek [22]: Komponenta Light ... 44
Obrázek [23]: Komponenta Canvas ... 45
Obrázek [24]: Komponenta Canvas Scaler ... 45
Obrázek [25]: Komponenta RectTransform ... 46
Obrázek [26]: Komponenta Graphic Raycaster ... 46
Obrázek [27]: Komponenta Raw Image ... 47
Obrázek [28]: Komponenta Event System ... 48
Obrázek [29]: Komponenta Standalone Input Module ... 48
Obrázek [30]: Komponenta Scroll Rect ... 49

Obrázek [31]: Komponenta Button 50
Obrázek [32]: Komponenta Text... 50
Obrázek [33]: Komponenta Nav Mesh Agent.. 52
Obrázek [34]: Komponenta Agent Type 53
Obrázek [35]: Komponenta Off Mesh Link 54
Obrázek [36]: Komponenta Nav Mesh Obstacle.. 55
Obrázek [37]: Model Vlk – Ovce v aplikaci NetLogo 56
Obrázek [38]: Ukázka diplomového projektu 57

1 Úvod

Komplexní systémy se využívají k simulování scénářů v kontrolovaném prostředí. Existuje několik způsobů, jak lze simulaci zpracovat od pouhé textové podoby, kdy se zadají vstupní údaje a následně simulátor vrátí výsledek simulace, až po vizuální simulátory, které mohou uživateli v průběhu simulace zobrazit průběžný stav a upozornit, že v daném okamžiku se provádí něco, co by mohlo uživatele zajímat.

V této době existuje nespočet různých simulačních modelů z oblastí sociálních, armádních, vědeckých či lékařských témat, které jsou vytvořeny na specializovaných simulačních prostředích, jenž mohou být programátorsky omezeny, protože nepodporují většinu programových struktur a metod.

Cílem této diplomové práce je zjistit, zda je možné využít herní enginy pro vizuální simulace komplexních systémů. Herní enginy nejsou programátorsky omezeny a lze v některých přepsat i části jádra enginu.

2 Cíl práce

Cílem této diplomové práce je zjistit, zda je možné využívat herní enginy pro simulace komplexních systémů, kde se využívá systém agentů.
Výzkumné otázky jsou tyto:

- Je možné využít herní engine pro komplexní systémy?
- V případě, že by bylo možné využít herní engin, tak co by byla nutná základna, aby se toto řešení mohlo veřejně využívat?

3 Teoretická část

V této kapitole bude vysvětlena teorie ke komplexním a agentovým systémům, k simulačním prostředím, jak lze hledat cesty ve světech, jak lze u agentů vytvořit systém chování a jaké hlavní herní enginy existují. Více lze se dozvědět ve zdroji [4],[8].

3.1 Komplexní systémy a agentové systémy

Agentové systémy jsou označovány počítačové programy, které slouží k simulování prostředí, ve kterém se vyskytují autonomní agenti, kteří zpracovávají svůj kód a mohou nějakým způsobem i kooperovat s jinými agenty v prostředí [20]. V případě, že agenti spolupracují, tak roste počet možností, co dokáží zvládnout, oproti situaci, kdy agenti nekooperují. Agentové systémy se vyznačují i tím, že dokáží dosáhnout cíle, který nejen pro jednotlivé autonomní agenty je nedosažitelný, nebo těžko dosažitelný, ale že i v tomto směru mohou překonat i monolitní systémy. Inteligence agentových systémů může obsahovat metody, funkce, procedury, algoritmické vyhledávání anebo i možnost strojového učení, kdy se agent dokáže poučit z vlastních zkušeností, nebo učení z dodaných dat agentovi.

Agenti mohou být rozděleni do několika typů, podle komplexnosti:

- Pasivní agenti, kteří nemají žádný svůj cíl a mohou představovat překážku v systému, nebo představovat prostředek, který bude využit jinými agenty.
- Aktivní agenti, kteří mají jednoduché cíle. Příkladem těchto agentů je vlk a ovce z modelu Kořist-Predátor.
- Komplexní agenti, kteří mají už složité cíle a mohou obsahovat stromy chování anebo i strojové učení, kdy budou upravovat své chování na základě toho, co už sami zažili.

Agenti mohou být rozděleni i do kategorii, podle spolupráce [16]:

- Spolupracující agenti, kteří spolupracují s jinými agenty, aby dosáhli společně stanoveného cíle. Příkladem spolupracujících agentů může být simulace fotbalového zápasu, kdy agenti musí v týmu spolupracovat, jinak žádný jedinec nemá možnost, že by vyhrál.

- Soupeřící agenti, kteří pracují samostatně a v případě některých simulačních modelů mohou i škodit jiným agentům. Příkladem agenta může být simulace dopravy na silnici, kdy agenti představují dopravní prostředek, který se vždy snaží dostat do vedení, aby se dostal do svého cíle co nejdříve.
- Kombinace kooperujícího a soupeřícího agenta, kdy agent se rozhoduje dle situace, zda bude kooperovat, anebo pracovat sám a soupeřit. Příkladem tohoto chování by byl model Vězňovo dilema, kdy jsou dva agenti Vězni, kteří mohou mlčet, nebo zradit druhého vězně, aby dostali menší trest. Pokud zradí jenom jeden agent, tak tento agent dostane menší trest a druhý agent dostane větší trest. Pokud oba agenti zradí, tak oba dostanou střední trest. Pokud oba agenti mlčí, tak oba dostanou minimální trest.

Agenty ještě můžeme rozlišit, jak směřují ke splnění svého cíle:

- Agenti orientující se jenom na cíl. Je nutné, aby bylo možné znát předchozí, současný stav a informace o cíli. Agenti volí akce na základě situace, která je závislá na prostředí.
- Agenti orientující se na užitkovost jsou varianta, která vychází z Agenta orientující se jenom na cíl. Jedinou změnou je to, že tyto agenti musí při své činnosti brát ohled i na efektivitu. V každém kroku modelu agent musí poskytovat informaci o efektivnosti a aktuální stav práce. Toto hlášení efektivnosti lze využít jako zpětnou vazbu, která pomůže zvýšit výkon anebo kontrolovat platnosti a účinnosti systému.
- Agenti, kteří se učí. Nejinteligentnější typ agentů, kteří se učí z uživatelského vstupu, z předchozí historie pokusu, z vlivu prostředí a svého okolí. Tyto agenti začínají jako konvenční agenti, kteří znají jenom základní informace. Po každé činnosti, ať už dopadla činnost úspěchem, či neúspěchem, se agent poučí přes učící komponentu. Další komponentou je komponenta kritiky, která upřesňuje, jestli se agent z posledního učení zhoršil, či zlepšil. Další nová komponenta je komponenta výkonnosti, která je zodpovědná za měření výkonu systému a má zajistit plynulé fungování systému. Poslední komponenta je generátor problémů, která má zajistit to, že agent bude získávat nové zkušenosti a bude se zlepšovat se zvyšující se zátěží.

Prostředí, kde se agenti vyskytují, můžeme rozdělit na několik kategorii [17]:

- Virtuální
- Diskrétní
- Kontinuální

Dále lze rozdělit i prostředí na několik kategorií v závislosti na přístupu k informacím o prostředí, pokud je možné získat informace:

- Úplně pozorovatelné prostředí, kdy je možné přistoupit ke všem informacím o stavu agenta. Příkladem je hra Šachy, kdy hrací deska je úplně pozorovatelné prostředí a tahy soupeře jsou úplně pozorovatelné.
- Částečně pozorovatelné prostředí, kdy je přístup jen k omezené množině informací o stavu agenta. Příkladem je řízení dopravního prostředku, kdy řidič má omezeny pozorovací úhly, protože vždy bude něco tvořit překážku v pohledu. Příkladem překážek může být porost, jiné dopravní prostředky, budovy, a i samotné vozidlo řidiče, kdy není zaručena jistota, že řidič něco nepřehlédl přes rám vozidla.
- Deterministické prostředí, kdy ke každé akci je možné přesně definovat, jaká reakce bude vyvolána.
- Stochastické prostředí, kdy každá možná akce agenta je specifikována pravděpodobností, že bude vykonána a pravděpodobnost, že bude akce provedena není rovna 100 %, kdy můžeme definovat, která akce bude následovat, pokud existuje více jak jedna možnost pro právě provedenou akci.
- Dynamické prostředí, které se samo mění následkem akcí agentů a může se měnit i počet agentů v průběhu simulace. Příkladem dynamického prostředí může být horská dráha, přestože určitá dráha je trvale stejná, tak okolí této dráhy se může měnit.
- Statické prostředí, které zůstává vždy stejné a nemění se vlivem akcí agentů.
- Diskrétní prostředí, kdy počet možných akcí v prostředí je konečný. Příkladem může být hra Šachy, kde, přestože počet tahů může být odlišný hra od hry, tak celkový počet tahů je konečný.

- Kontinuální prostředí, kdy neexistuje konečný počet akcí agentů. Příkladem mohou být samořídící auta, která fungují v kontinuálním prostředí a jejich akce řízení, parkování atd. nelze specifikovat konečným číslem.
- Epizodické prostředí, kdy akce agenta v jedné periodě neovlivní další periodu. Příkladem může být detekce defektních prvků z transportního pásu. Robot, který bere defektní prvky z pásu, v jedné periodě vezme defektní prvky, které byly v určité číselné posloupnosti, ale tento fakt už není platný v další periodě, protože číselná posloupnost se může lišit.
- Sekvenční prostředí, kdy akce agenta v jedné periodě ovlivní další periodu, kdy může ovlivnit agenta samotného, jiné agenty anebo samotné prostředí. Příkladem může být hra Dáma, kdy pohyb v aktuálním tahu ovlivní všechny budoucí tahy.
- Dimenzionální prostředí, kdy agenti ve svém chování a plánování berou v potaz i samotné prostředí, ve kterém se vyskytují.

V praktických příkladech užití výše zmíněných prostředí je možné, že samotný systém bude kombinovat několik kategorii.

Pro agenty je charakteristické:

- Agenti pracují úplně, nebo částečně autonomně.
- Žádný agent nemá úplný přehled o celém prostředí, ve kterém se vyskytuje, nebo samotný systém je příliš komplexní, aby agent byl schopen zpracovat informace z úplného přehledu o celém prostředí, pokud by tyto informace vlastnil.
- Decentralizace, kdy žádný agent není vytvořen tak, aby vykonával funkci kontrolera, který by ovládal všechny ostatní agenty anebo i prostředí. Pokud by se něco takového stalo, tak efektivnost takového agentového systému je snížena na úroveň efektivnosti monolitického systému.

3.2 Příklady využití simulačních prostředí

V dnešní době se využívají simulační prostředí k řešení problémů pro většinu teoretických oblastí. Více se lze dozvědět ve zdroji [12].

- Sociální systémy

Modely zabývající se faktory, které ovlivňují úmrtnost či porodnost obyvatel, vliv poměru etnik v prostředí na kriminalitu a cenu nemovitostí či modely vlivu určitých prvků na migraci obyvatelstva do měst z vesnic.

- Dopravní systémy

Modely, které simulují dopravu a hlavním faktorem bude změna křižovatek, značek a silničních pravidel.

- Simulace přírody

Modely, které simulují chování zvířat anebo rozšiřování rostlin. Tyto modely simulují, jak moc ovlivní faunu a flóru, pokud člověk zasáhne do přírody. I ten nejmenší zásah může mít dalekosáhlé následky. Příkladem může být situace, kdy je přepraven nový druh do prostředí, a to může mít negativní vliv na své nové okolí. Příkladem tohoto zásahu je přepravení zvířat do Austrálie, kde je takový čin zakázán, pokud není přeprava povolena.

- Šíření a vývoj pandemií [10],[22]

Existuje spousta modelů, které se zabývají tématem šíření nemocí. Vlivem „SARS-CoV-2“ (dále jako „Covid“) pandemie v roce 2019 vzniklo ještě více takovýchto modelů, přestože nelze označit tyto modely za přesné, protože o viru Covid se stále neví všechny informace, protože samotný virus mutuje, což nelze dlouhodobě modelovat.

- Vojenské simulace

Modely, které slouží pro simulace válečných scénářů, při kterém se mění počty jednotek, vybavení a taktika při postupu v simulaci, a možné simulace činnosti teroristických skupin.

Přestože se tady jedná o simulace agentových prostředí, tak by se nemělo zapomenout ani na simulace agentových prostředí s uživatelským vlivem. Příkladem je Bohemia Interactive Simulations (BISim), což je vojenský simulátor od české společnosti Bohemia Interactive, který je nabízen státním vojenským organizacím. Prostředí slouží pro trénink osob, aby byly připraveny na většinu možných situací, ve kterých se mohou vyskytnout. Tento simulátor je nabízen v několika variacích a jsou dodávány i simulační repliky prostředí vozidel a kokpitů, aby osoba byla připravena na řízení, bez přímého ohrožení. Je možné využívat i brýle virtuální reality pro lepší autentický zážitek. Toto prostředí umožňuje provádět simulace, kdy jedna osoba rozdává rozkazy počítačovým agentům (přátelským i nepřátelským) a zbývající osoby ovládají jenom svou postavu. Tento simulátor je rozdělen na dvě části, kdy by neměly tyto části být spuštěny na jednom zařízení. První částí je serverová aplikace, která by měla fungovat na výkonných zařízení, protože samotný simulátor má značné požadavky. Druhá část je uživatelská část, která může být spuštěna i na průměrném zařízení, ale pokud uživatel plánuje využít brýle pro virtuální realitu, tak se požadavky zvyšují.

- Simulace přesunů obyvatelstva.
- Simulace vlivu okolních podmínek na stavební a technické řešení.
- Simulace multi-agentových systému v automatizaci [9].

3.3 Hledání cest ve světě

Pro práci s agenty ve světě je vždy třeba hledat cesty, kterými by se agenti dostali ke svým cílům. Světy můžeme rozdělit na 2D a 3D světy, kdy obě varianty mají několik způsobů, jak se dá hledat cesta ve světě. Více ve zdrojích [2-3] a [18].

3.3.1 Hledání cest ve 2D světech

Hledání cest ve 2D světech [24] je založeno na tom, kdy svět je tvořen čtvercovou sítí, kdy se uživatel dívá z pozice na nebi směrem k zemi. Tento popis 2D světa je z toho důvodu, že se vyskytuje spousta videoher, kde uživatel nahlíží ze své pozice z boku světa a kamera se přesouvá doleva, doprava, dolů, nahoru. Na obr. 1 – 3 jsou příklady, kde lze vidět tyto dvě variace.

Obrázek [1]: 2D svět, který není příkladem klasické hledání cesty
zdroj:https://unity.com/how-to/beginner/unity-good-2d-development

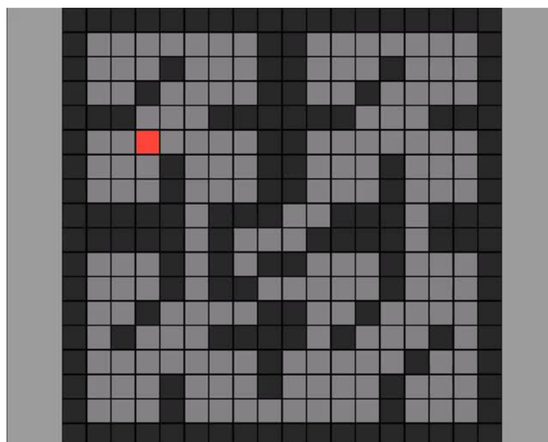
Obrázek [2]: 2D svět, který může být příkladem hledání cesty, zdroj:https://heroes.thelazy.net/index.php/Combat

Obrázek [3]: 2,5D svět, který může být příkladem hledání cesty, zdroj:https://store.steampowered.com/app/1536610/OpenTTD/

Technika Záplavy

Technika [19], která má stanovený počátek, odkud by „Záplava“ začala. Principem záplavy je postupná záplava ze startovního bodu, který má hodnotu 0. Buňky mřížky se označují číslem, které představuje minimální počet kroků, než se z počátku dostane agent k této buňce. Případně tuto techniku lze využít na ocenění cesty k cíli, kdy místo počtu kroků z počátku do vybraného políčka bude hodnota představovat náročnost

cesty, takže agent by si mohl v některých případech vybrat daleko delší cestu, ale s minimální náročností.



Obrázek [4]: Technika záplavy, zdroj:https://gfycat.com/lightaromaticcaecilian

Tuto techniku lze rozdělit na 2 možnosti. V prvním případě se záplava může rozšiřovat pouze 4 směry (doleva, doprava, nahoru, dolů) a ve druhém případě se přidávají i diagonální směry (nahoru doleva, nahoru doprava, dolů doleva, dolů doprava). Na obrázku 4 ve druhém případě může nastat situace, že pokud z červeného políčka, který představuje počátek, by se vydalo nahoru, tak se může dostat i nahoru doleva, což by se nemohlo stát u čtyř-směrného pohybu, protože by tam byla překážka a muselo by se jít delší cestou ke stejnému políčku.

Dijkstrův algoritmus

Algoritmus [13] pro hledání nejkratší cesty v grafu s ohodnocenými vrcholy. Jeden z nejvíce využívaných algoritmů. Princip tkví v tom, že z počátečního vrcholu se navštíví všechny okolní vrcholy a u každého uzlu se zapíše „cena cesty“, což se určí z ceny hrany mezi vrcholy. Do kolekce se zapisují postupně vrcholy, které se navštívily. Jakmile se ohodnotí všechny vrcholy, které sousedí s počátkem, tak se přejde k prvnímu uzlu v kolekci a opět se ohodnocují sousední vrcholy, kdy cena je rovna součtu ceny hrany a hodnoty vrcholu, ze kterého se provádí ohodnocení. Pokud se narazí na vrchol, který má už nastavenou hodnotu, tak se kontroluje, zda nová hodnota není nižší, než aktuální hodnota a pokud ano, tak se hodnota aktualizuje. Jakmile všechny vrcholy jsou z kolekce odstraněny, protože se provedla aktualizace cen sousedních vrcholů, tak už zbývá najít nejkratší cestu z cílového vrcholu do počátečního vrcholu. Toto nalezení cesty

se provádí tak, že se z cílového vrcholu vydává k vrcholům, které mají nejnižší hodnotu, dokud se nedostane do počátečního vrcholu.

Nejlepší první hledání

Metoda [13], která může být rychlejší než ostatní způsoby, ale podmínkou je, aby vrcholy grafu měly geografickou pozici X a Y, kdy je následně možné se vydat směrem k cíli právě přes souřadnicový systém. Příkladem je to, že Počáteční vrchol má pozici vlevo dole [0,0] a cílem je vrchol vpravo nahoře. Pomocí tohoto algoritmu by se vydalo přímou cestou, kdy by se každým krokem prohledal uzel [X+1, Y+1], takže v 10. kroku se nalezne cíl a začne se skládat zpáteční cesta. U tohoto algoritmu není zaručeno, že nalezená cesta bude nejkratší, ale celkový čas, který je stráven hledáním cesty může být značně kratší, než je u jiných algoritmů, protože není nutné projít všechny vrcholy grafu. Příkladem, kdy celkový čas strávený hledáním jakékoliv cesty k cíli, je ten, kdy v předchozím příkladu grafu s počátkem [0,0] a cílem [10, 10] by byly překážky na přímé cestě.

3.3.2 Hledání cest ve 3D světech

Metoda A*

Variace [14] Dijkstrova algoritmu [13], kdy každý vrchol má prioritu, kterou je zpracován. Tato priorita je určena touto funkcí:

f (n) = g (n) + h (n) f(n) = g(n)+h(n) f(n)=g(n)+h(n)

- f(n) představuje komplexní prioritu vrcholu. Vždy by se měl vybírat vrchol, který má nejvyšší komplexní prioritu, protože představuje nejnižší aktuální cenu za přechod k dalšímu vrcholu.
- g(n) celková cena za dostání se k danému vrcholu z počátečního bodu
- h(n) je odhadovaná cena vrcholu z cílového vrcholu. Tato funkce je právě heuristická část tohoto algoritmu.

Metoda D*

Algoritmus [13], který je podoben algoritmu A*, ale jediný rozdíl je v tom, že se nehledá cesta z počátečního vrcholu, ale vyhledává se cesta z cílového vrcholu, což ušetří čas strávený zpětným dohledáním cesty, který se vyskytuje u předchozích algoritmů.

3.3.3 2,5D světy

Tyto světy jsou částečnou kombinací 2D a 3D světů, kdy nelze říci, kterou techniku hledání cesty lze využít, protože záleží na tom, jak samotný svět je tvořen.

Příkladem 2,5D světa je videohra DOOM, kdy svět je tvořen ve 2D, ale využívá techniky, aby to vypadalo, že vše je ve 3D.

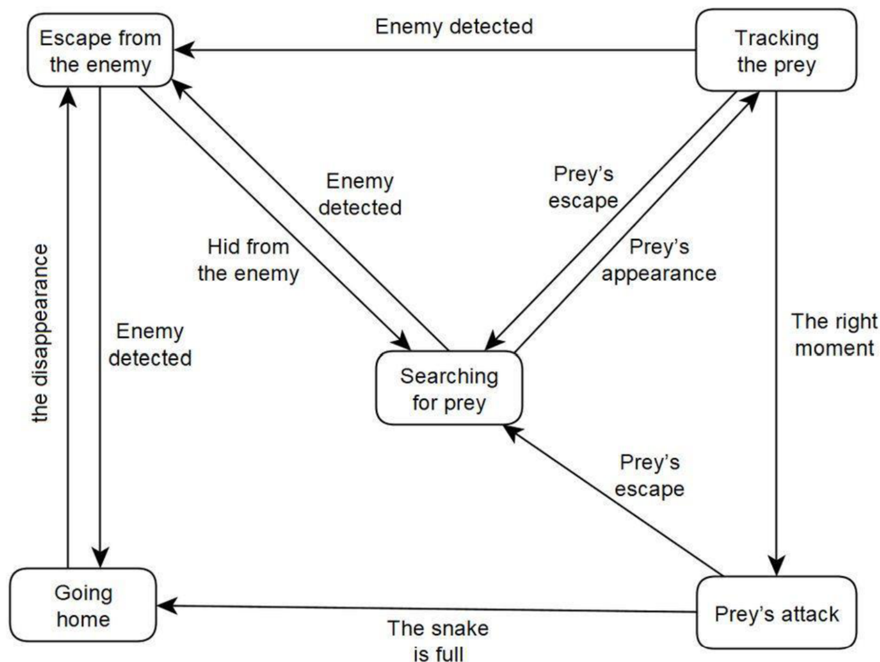
3.4 Stromy chování

Aktivity spojené s umělou inteligencí (AI) jsou označovány jako chování pro charaktery neovládané hráčem „Non-Player Character“ (NPC). Navržení logiky pro různé charakterové entity v počítačových hrách je důležité téma, které definuje, jakým způsobem reaguje NPC na nastalé situace a jakým způsobem se chová v čase bez událostí, přičemž vždy záleží na podmínkách [1].

Existuje několik způsobů, jak definovat chování NPC: místo následujících odrážek udělat nadpisy, odrážka s velkým množstvím textu nevypadá dobře

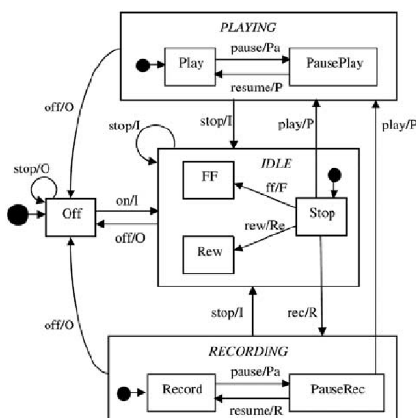
- Systém pravidel („Rule-Base system“) je nejjednodušší způsob, jak omezeně kontrolovat chování NPC, kdy chování je funkcí založenou na podmínkách v okamžiku rozhodování. Pravidla jsou posuzována proti skupině znalostí, zda se podmínky rozhodnutí X splnily. Příkladem je hra Pac-Man, kdy se NPC duch rozhoduje tak, že postupuje kupředu, dokud nenarazí na překážku a následně se náhodně otočí do strany a zkusí postupovat tímto směrem. Tento systém umožňuje snadno pochopit a implementovat pravidla, ale je téměř nemožné vytvořit komplexní model chování.

- Konečný automat („Finite-state Machines“) je způsob, jak definovat stav NPC v určitém čase a umožňuje tvořit komplexní chování. Základem tohoto způsobu je definování stavu NPC v aktuálním čase, a to je dosaženo tím, že tyto „stavy“ tvoří diagram, kdy je definováno, z jakého stavu se dostane kam. NPC může být pouze v jednom stavu. Každá hrana diagramu představuje podmínku, která spustí přechod ze stavu X do stavu Y, pokud je splněna podmínka. Každý stav v diagramu musí znát do jakých stavů může vstoupit za podmínky Z.



Obrázek [5]: Příklad konečného automatu, zdroj: https://mind-simulation.com/en/blog/tech/using-finite-state-machines-to-model-behavior.html

Škálovatelnost této metody je hlavní nevýhoda, protože v současné době u her i u simulací je potřeba značné množství stavů. Tato nevýhoda se může částečně odstranit metodou Hierarchického konečného automatu. Tato metoda funguje stejným způsobem jako Konečný automat a jediná změna je v tom, že se stavy zapouzdřují do skupin, které mají stejnou oblast logiky.



Obrázek [6]: Příklad hierarchického konečného automatu, zdroj:https://www.researchgate.net/figure/Hierarchical-FSM-for-a-tape-recorder_fig2_220459077

3.4.1 Behavior Tree

Stromy chování („Behavior Tree“) je nejčastější a nejvíce používaný způsob, jak vytvořit snadno škálovatelné a hierarchické komplexní chování. NPC prochází strom chování od hlavního uzlu, který je nejvýše a postupně pokračuje směrem dolů. Oproti způsobu Konečného automatu není potřeba, aby uzel ve stromě znal podmínky přechodu do dalšího uzlu, a proto je tato metoda zvaná jako „bez stavu“. Strom chování využívá několik typů uzlů a výsledkem zpracování každého uzlu je jeden ze stavů Splněno, Probíhá, nebo Neúspěch, podle toho, jak se zpracovala logika uzlu. Škálovatelnost stromu vězí v tom, že se uzlům typu Selektor a Sekvence přidělují uzly potomci a pouze se změní list potomků. Nejlepším způsobem, jak pracovat se stromy chování, je využívat některý z vizuálních editorů, protože u složitých stromů může být zmatečné programovat list potomků manuálně. Příkladem obyčejného stromu chování je obrázek [7].

3.4.2 Selector

Úkolem selectoru je to, že prochází své uzly potomky. V případě, že potomek vrátí stav Splněno, nebo Probíhá, tak selektor ukončuje svůj průběh a vrací stav výš k rodiči. V případě, že dostal Neúspěch, tak se přesune k dalšímu svému potomkovi a pokud dostane Neúspěch i od posledního potomka, tak vrací Neúspěch o úroveň výš k rodiči.

3.4.3 Sequence

Úkolem Sequence je procházet všechny své potomky uzly. V případě, že dostane Neúspěch, nebo Probíhá, tak vrací stav o úroveň výš k rodiči. Pokud všichni potomci uzly vrátí Úspěch, tak samotná sekvence vrátí Úspěch o úroveň výš k rodiči.

3.4.4 Inverter

Inverter má jediný cíl a to obrátit hodnotu, kterou dostane od svého jediného potomka. Z Úspěchu vytvoří Neúspěch a naopak. Neměl by měnit výsledek stavu Probíhá. Příkladem využití obraceče by byl selektor, který má 2 potomky – 2 sekvence. První sekvence by začínala uzlem „V dosahu“ a druhá sekvence by začínala uzlem Obraceč, který by měl potomka „V dosahu“. V případě druhé sekvence by proběhl pohyb

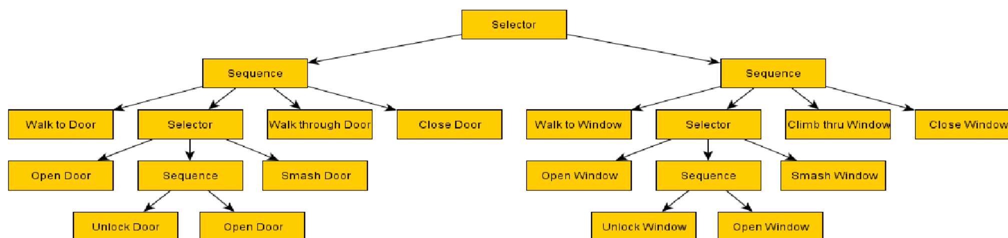
k cíli a v případě první sekvence by se provedla určitá činnost u objektu, ke kterému se směřovalo.

3.4.5 Node

Node nemá definován cíl a je jenom na programátorovi, jak definuje vnitřní logiku, podle které bude vždy vracet Splněno, Probíhá, nebo Neúspěch.

3.4.6 Variace stromu chování

Výše popsané typy uzlů ve stromě mohou být pozměněny a vytvořeny nové uzly, pokud si to projekt žádá, ale všechny návrhy stromů budou obsahovat vše zmíněné. Existuje několik typů modifikací stromu chování. Prvním typem je Strom chování, který umožňuje rozhodování, kdy se sleduje více cílů. Druhým typem je Strom chování s využitím AI, které hodnotí svoji strukturu a snaží se optimalizovat svou práci. Třetím typem je Strom chování, kdy je přidán prvek času a ke každému uzlu může být nastaveno zpožděný oproti předcházejícímu uzlu. Čtvrtým typem je Strom chování, kdy se rozhoduje s určitou pravděpodobností, že se vybere určitý uzel potomek. Posledním typem je Strom chování, kdy v této modifikaci by měl umět ovládat více větví potomků v jednom okamžiku, ale tato modifikace je nebezpečná v tom, že může způsobit to, že bude následována větev, která by nevedla k optimálnímu splnění cíle.



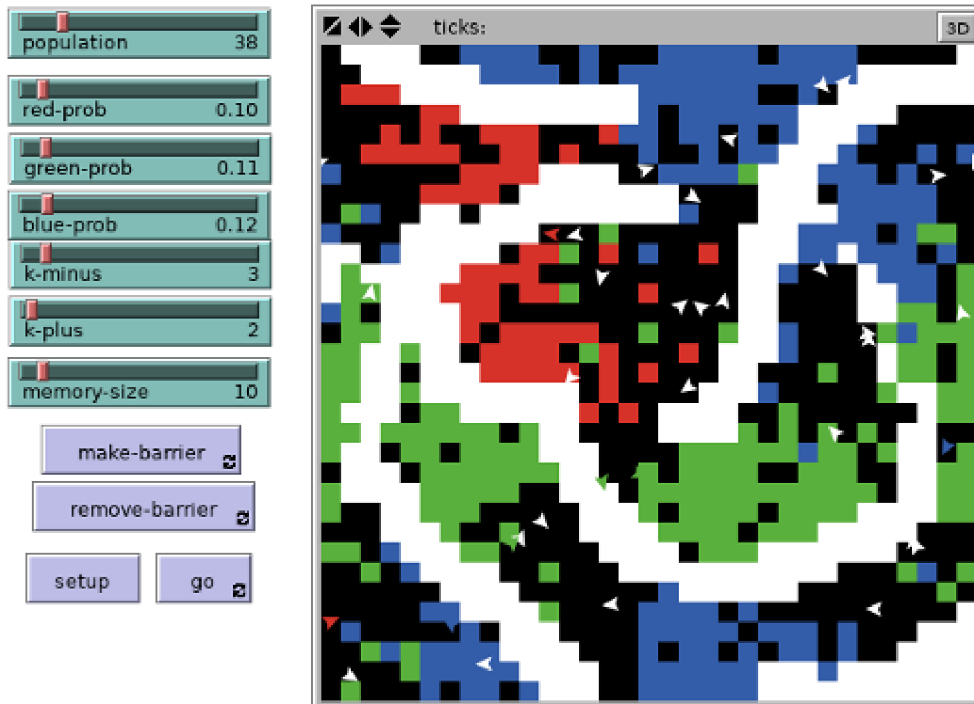
Obrázek [7]: Příklad stromu chování pro zloděje, jak se bude rozhodovat při vniknutí do budovy zdroj: https://outforafight.wordpress.com/2014/07/15/behaviour-behavior-trees-for-ai-dudes-part-1/

3.5 Prostředí NetLogo

Netlogo je multi-agentový programovací jazyk a modelovací prostředí, které slouží pro simulaci přírodních a sociálních fenoménů. Toto prostředí je často využíváno pro modelování složitých systémů, které se vyvíjí v průběhu času. Systém dává možnost vydávat pokyny stovkám až několika tisícům nezávislých agentů, kteří pracují současně a každý zpracovává svůj program nezávisle na ostatních. Cílem je možnost prozkoumat souvislosti mezi chováním na mikroúrovni jednotlivce a vzorce chování na makroúrovni, kdy se agenti ovlivňují svou interakcí. Netlogo umožňuje uživateli vytvářet simulace a zkoumat možnosti, jak se změní chování agentů, pokud se nastaví určitý prvek jinak oproti výchozímu nastavení. Prostředí je dostatečně jednoduché, aby umožnilo studentům i výzkumníkům snadno vytvářet simulační modely, přestože nemusí umět dostatečně programovat. Více lze nalézt ve zdroji [23].

Uživatelské rozhraní lze rozdělit na tři části.

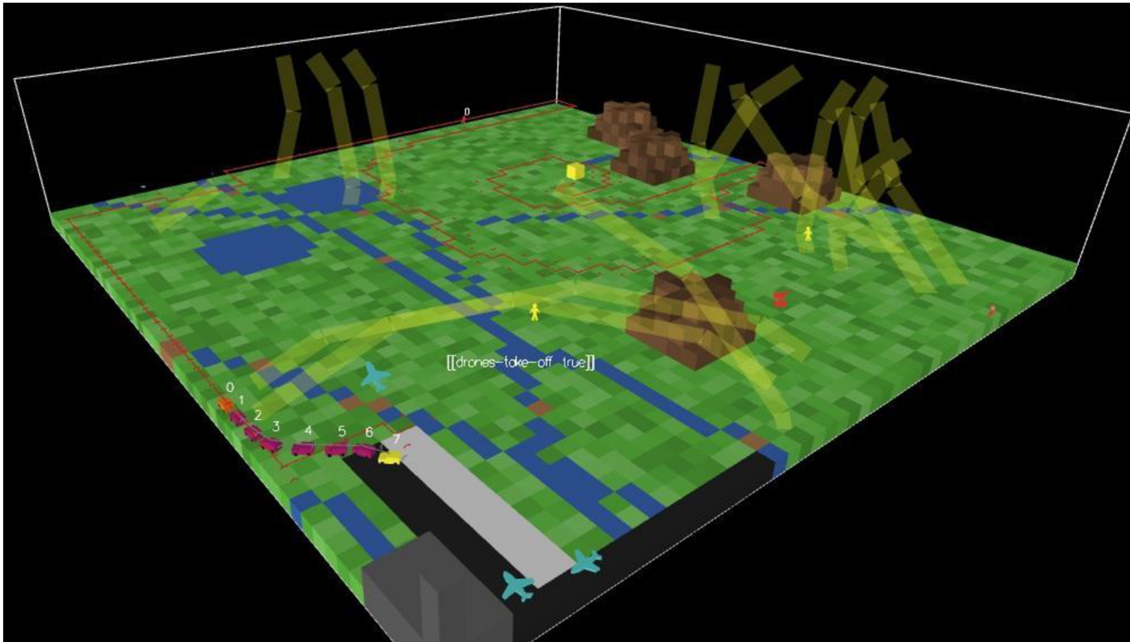
- První část je samotná simulace, které je znázorněno 2D mřížkou, kdy jeden „čtvereček“ se nazývá „Patch“, a nezávislými agenty, kteří se nazývají „Turtle“, neboli želva, protože ikona želvy je jedna z hlavních ikon pro agenty. Každý patch má velikost, pozici, barvu a nemůže se pohybovat. Agenti mají vlastnosti ikonu, barvu, natočení a pozici. Patch je také typem určitého agenta. Zbylé 2 typy agentů jsou „Linky“ a „Observer“ („Pozorovatel“) a nejsou viditelné. Link je typ agenta, který spojuje 2 agenty Turtle a v okamžiku, kdy zanikne jeden z agentů, tak zaniká i tento agent Link. Agent Pozorovatel není přímo pasivní, jak by napovídal název, ale dává příkazy ostatním agentům.
- Druhou částí jsou prvky, které nastavují simulaci a nemají vlastní kód oproti ostatním částem. Do této části patří posuvníky, zaškrtávací políčka a další prvky, které uživatel upravuje. Podle modelu simulace je možno měnit posuvníky i během simulace.
- Poslední částí jsou tlačítka, která mají svůj vlastní kód, který se vykoná při stisknutí tlačítka.



Obrázek [8]: Příklad simulace v aplikaci NetLogo(2D), zdroj: http://web.cse.ohio-state.edu/~stiff.4/cse3521/netlogo.html

Samotná simulace probíhá v „tazích“, kdy každý agent vykoná svůj kód, který má deklarovaný v metodě „TO GO“. Tato metoda se volá pouze jednou pro každého agenta za jeden tik. Nový tik se nezahájí, dokud se nedokončí předchozí tik všech agentů. Výhodou tohoto prostředí je i to, že umožňuje vygenerovat tabulkový soubor, který obsahuje proměnné agentů, které se zvolily pro sledování.

V rámci programu Netlogo existuje samostatná aplikace Netlogo 3D, která umožňuje simulovat 3D prostředí. Tato aplikace má stejný základ jako Netlogo (2D), ale neexistuje sjednocená verze, která by umožňovala pracovat s 2D a 3D modely z jedné aplikace. Modul může být 2D, nebo 3D, ale nelze mít model, který by byl spustitelný v aplikaci Netlogo 2D a současně i na Netlogo 3D.



Obrázek [9]: Příklad simulace v aplikace Netlogo3D, zdroj:https://github.com/lrq3000/netlogo-mas-drones

Přestože je prostředí jednoduché na programování, tak vzniká problém, kdy není možné vytvořit složité struktury kódu. Příkladem je to, že neexistuje dědičnost mezi agenty, systém pohybu agentů je příliš jednoduchý a programátor si musí vytvořit vlastní plánování. Tyto zmíněné nedostatky se týkají Netloga (2D).

Z pohledu historie je NetLogo (rok 1999) další generací multi-agentových prostředí, kdy předcházel systém StarLogo. Samotné prostředí je psáno v programovacím jazyce Java, proto může být spuštěno na většině platforem. Využívání tohoto prostředí není zpoplatněno. Při nainstalování se současně s prostředím instaluje i základní balíček simulačních modelů a dokumentace.

3.6 *Herní enginy*

Pro herní enginy [1]neexistuje přesná definice, přestože to lze jistě označit za software, které slouží pro tvoření videoher. Toto označení ovšem nedefinuje, co by takový herní engine měl umět. Hlavním cílem herního enginu je znovuvyužití částí kódů a herních assetů v jiných hrách. Asset představuje soubor a může obsahovat 3D modely, obrázky, zvukové stopy anebo jiné typy, které herní engine podporuje.

Za první skutečný herní engine by mohl být považován engine Unreal od společnosti Epic, když v roce 1998 byl zahájen prodej hry Unreal. To je první známý okamžik, kdy byl rozdělen projekt na samostatný engine a samotnou hru. Dříve platilo to, že byl smíšen kód enginu a samotné hry.

Herní engine by měl splňovat tato kritéria:

- Umět renderovat 2D nebo 3D grafiku v renderovacím enginu.
- Umět kontrolovat Vstupní události (Například událost pro klávesnice, myši, dotyková zařízení a další hardware prvky).
- Umět řídit herní smyčku, která by měla být zpracována každý snímek (Frame). Tato herní smyčka je většinou označována jako „Update“ metoda.
- Fyzikální engin, který bude zjišťovat kolize a umožňovat reagovat určitým způsobem na tyto kolize.
- Ovládání zvuků.
- Grafická scéna. Prvek, který umožňuje nastavení grafických prvků na obrazovce a nastavení vztahů mezi těmito prvky.
- Ovládání animací pro 2D obrázky anebo pro 3D modely.
- Správa paměti.
- Správa vláken, která umožňuje, aby hra vytvořená v herním enginu mohla být nastavena takovým způsobem, že bude pracovat s více vlákny.

[1] V dalších podkapitolách budou popsány hlavní enginy, které slouží pro vývoj her. Existuje spousta enginů, ale ty ostatní budou pouze vyjmenované, a i u hlavních enginů budou popsány pouze základní informace, protože už samotný jeden engine by představoval jednu několikaset stránkovou knihu. Pro více informací se doporučuje na stránkách enginu přečíst dokumentaci. Nejvíce bude popsán engine Unity, protože bude využíván pro diplomový projekt, který bude doprovázet tuto diplomovou práci.

Vše výše zmíněné by měly být povinné prvky, které by měl herní engine obsahovat, ale existují prvky, které mohou rozšiřovat funkčnost enginu. Mezi tyto prvky patří:

- Možnost vytvoření scriptů.
- Možnost integrace umělé inteligence.
- Přidání síťové vrstvy.
- Integrace streamovacích služeb.
- Možnost uživatelských modifikací.
- Možnost přidání jazykových mutací.
- Možnost enginu exportovat aplikaci na jiné platformy.

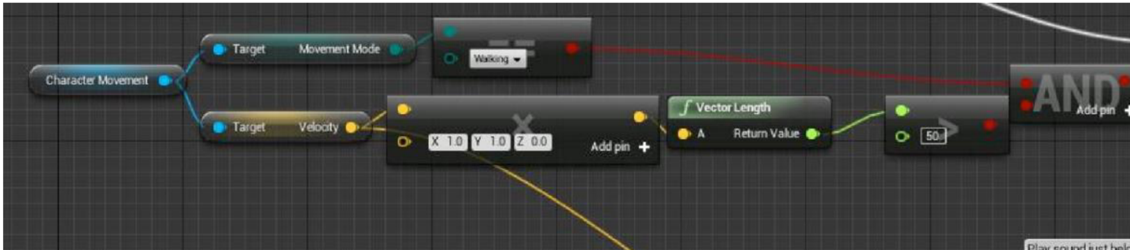
3.6.1 Engine Unreal

Herní engine, který využívá jazyk C++.

Jak bylo zmíněno výše, tak Engine Unreal se považuje za první herní engine, který byl využit u hry Unreal z roku 1998 a byl to první herní engine, který byl oddělen od kódů samotné hry.

Engine Unreal pochází od společnosti Epic a v okamžiku psaní této diplomové práce zastává většinu herního průmyslu. Dřív se za licenci platila značná částka, ale v důsledku konkurence enginu Unity společnost Epic přešla na podobný typ licence, kdy základní verze je zadarmo pro nekomerční účely. Tento krok nejen zvýšil počet vývojářů ale i fanouškovských studentů tohoto enginu. Původní účel Unreal enginu byl pro vytváření 3D her, kde se střílí z pohledu první osoby, ale postupem let se engine rozrostl a je schopen zvládnout vývoj jakékoliv hry a to včetně 2D prostředí. Stejně jako Unity engine má Unreal engine velkou komunitu a existuje i komunitní trh, kde se mohou nabízet assety.

Jak je zmíněno výše, tak se programuje v jazyce C++ a hlavní síla tohoto enginu je v takzvaných „Blueprint Visual Scripting“, neboli vytváření vizuálních viditelných přednastaveních.



Obrázek [10]: Příklad Unreal engine – blueprint visual script, zdroj: Game engines a survey [25]

Tento Blueprint Visual Scripting (BVS) systém umožňuje, aby se rychle testovaly změny v aplikaci a bylo je možné vidět za běhu aplikace v Unreal enginu (Tímto je myšleno to, že hra samotná není vyexportovaná do samostatné spouštěcí aplikace), jak se u jednotlivých objektů rozhoduje a v jakém stavu je právě aktuální objekt. Tento systém je výborným řešením pro stromy chování, protože lze vidět, jak objekt se rozhoduje a není nutno vytvářen strom chování v kódu, jako v případě Unity.

Kromě BVS systému je možné tradičně programovat třídy i ve Visual Studiu pro C++. Další výhodou Unreal enginu je to, že má otevřený kód („open source“), což umožňuje vývojářům i samotné komunitě provádět změny v enginu, když by změny byly pro vývoj potřeba. Těmito změnami se myslí jen lokální změny u vývojářů aplikace/hry.

V době vytváření této diplomové práce byl proveden průzkum fór pro vývojáře, kdy se řešila otázka, který herní engine je nejlepší pro určité vývojáře a výsledkem v případě Unreal enginu bylo to, že Unreal je komplexní herní engine a pro grafiky poskytuje nástroje, se kterými může snadno pracovat s programovou částí vývoje pomocí BVS. Pro programátory „ne-grafiky“ bude seznámení s tímto enginem hodně náročné.

3.6.2 Engine Unity

Herní engine, který využívá programovací jazyk C#.

Herní engine Unity byl poprvé představen na Světové Konferenci Vývojářů (v originále „Worldwide Developer Conference“) v roce 2005, kdy tuto akci organizovala společnost Apple, a od svého představení získal jedno z předních míst v herním průmyslu. Hlavní výhody tohoto enginu je možnost vytvořit spustitelný program hry/aplikace na široké spektrum zařízení. V okamžiku psaní této diplomové práce Unity podporuje více jak 19 různých platforem, včetně počítačů, herních konzolí, mobilních zařízení a zařízení virtuální reality.

- Mobilní zařízení: IOS, TVOS, Android, Android TV, BlackBerry, Tizen
- Počítače: Univerzálně všechny Windows systémy, zařízení s operačním systémem Mac a Linux.
- Internet: Aplikace pro WebGL
- Herní konzole[2]: Sony Playstation (PSP, PS3, PS4, PS5), Microsoft Xbox (Xbox 360, Xbox One, XboxSeries X/S) Nintendo (3DS, Switch), cloudové hry pro službu Google Stadia.
- Virtuální/Rozšířená realita: Oculus, Playstation VR, Google ARCore, Apple Arkit, Windows Mixed Reality, HoloLens, Magic Leap, Steam VR, Google CardBoard, GearVR, DayDream.

Právě tato široká podpora zařízení a jednoduchost tvoření v Enginu Unity získala velkou tvůrčí základnu od roku 2005. Základní Unity licence je pro „indie“ vývojáře, kteří tvoří nízkorozpočtové hry, a společnosti zadarmo do doby než roční zisk z prodeje produktu, který byl vytvořen v Unity, nepřesáhne 100 000$. Existuje i placená PRO verze, která obsahuje mnoho pokročilých vlastnosti. Příkladem je nízkoúrovňový přístup ke grafickým procedurám a GPU profiler, kdy tyto prvky jsou důležité pro optimalizaci využívání zdrojů zařízení.

[2] V případě herních konzolí je potřeba mít koupenou licenci od společnosti, která vlastní herní konzoli. Tato licence se váže k finálnímu produktu.

Unity obsahuje vizuální editor a integrované vývojářské prostředí, které umožňuje rychlé testování nové verze aplikace. Pracuje se s jednoduchými scénami, které mohou být spuštěny a testovány, aniž by hra musela být dokončena a může být spuštěna, aniž by byla vytvořena jediná část kódu. Unity je dostatečně modulární a flexibilní pro komplexní mechaniky, které jsou vybudovány ze spousty malých částí.

Základem všeho ve scéně je „Herní objekt", neboli „GameObject". Scéna je kontejner pro tyto herní objekty. Každý herní objekt má vždy nějaké komponenty, kdy povinně je 3D pozice ve scéně neboli Transform. Přestože by hra byla 2D, tak se vždy pracuje s 3D Transform. Existuje velké množství komponent pro herní objekty, ať už pro práci s grafikou či pro práci s fyzickým rendererem. Hlavní částí pro herní objekty jsou „skripty", což jsou C# třídy, které lze rozdělit na „akční třídy" a „klasické třídy".

Akční třídy by v Unity představovaly ty třídy, které by dědily Unity třídu MonoBehaviour, která poskytuje spoustu vlastnostií ale hlavní jsou metody Start a Update. Tyto dvě metody zajišťují smyčku v Unity, kdy metoda Start se zavolá při inicializaci třídy, ale oproti klasické třídě to není konstruktor. Pokud je potřeba předat nějaké proměnné při konstrukci, tak nejlepší způsob je vytvořit speciální metodu, která tyto proměnné nastavuje. Metoda Update je speciální metoda, která se volá pravidelně v Unity. Počet volání metody Update je závislé na omezení Frame Per Second neboli počet snímku za vteřinu a spolu s tím i související výpočetní složitost a časová náročnost metody Update.

Unity poskytuje možnost vytvářet před-vytvořené herní objekty a poskytuje spoustu nástrojů pro práci s animacemi, modely, texturami, audio nástroje, a i různé optimalizační nástroje.

Další výhodou enginu je i funkční obchod „Unity Marketplace", který umožňuje prodávat před-vytvořené herní objekty, či celé před-vytvořené scény za reálnou měnu, nebo se nabízí zadarmo jako učící materiály.

Obrázek [11]: Příklad Unity uživatelského rozhraní, zdroj: Game engines a survey [25]

V době vytváření této diplomové práce byl proveden průzkum fór pro vývojáře, kdy se řešila otázka, který herní engine je nejlepší pro určité vývojáře a výsledkem v případě Unity engine bylo to, že Unity je komplexní herní engine a pro programátory poskytuje nástroje, se kterými může snadno pracovat s grafickou částí vývoje a s animacemi.

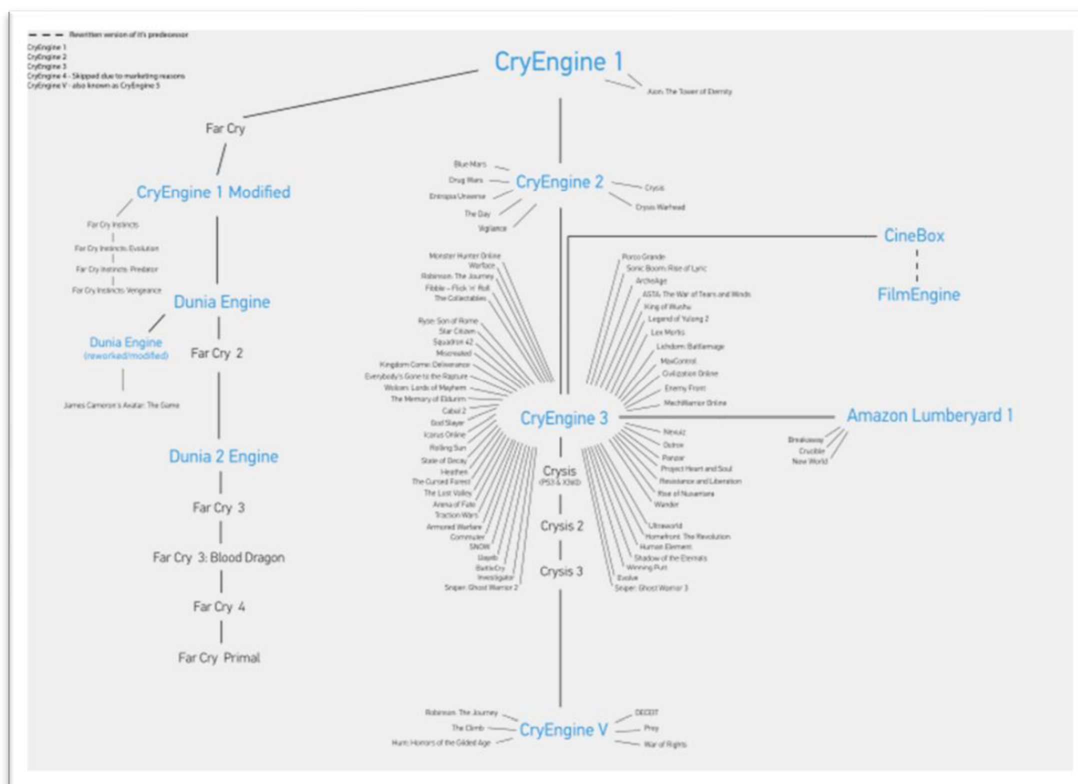
3.6.3 Engine CryEngine

Herní engine, který využívá jazyk C++.

Engine CryEngine byl vytvořen společností Crytek pro hru Far Cry (2004) a všechny její díly, ale hlavní hrou, která proslavila tento engine, byla série Crysis (první hra 2007) a proslavila tento engine, jako engine, který zvyšuje pokrok v technologických požadavcích.

Původně, než byl engine využit pro svou první hru Far Cry 1 od společnosti Ubisoft, bylo plánované využít engine pouze jako technologické demo pro společnost Nvidia, které by představovalo technologické možnosti jejich grafických karet. Společnost Ubisoft využila původní CryEngine pro několik her Far Cry (1) a následně odkoupila od společnosti Crytek licenci pro jejich engine, a tak vznikla samostatná větev CryEnginu zvaná „Dunia“. Tato licence nebyla časově omezena a je stále používána.

S původním CryEnginem současná Dunia Engine obsahuje podle odhadů informovaných zdrojů už pouze 1-5 % kódů. Dunia Engine je stále používána a v roce 2021 vydala společnost Ubisoft Far Cry 6.



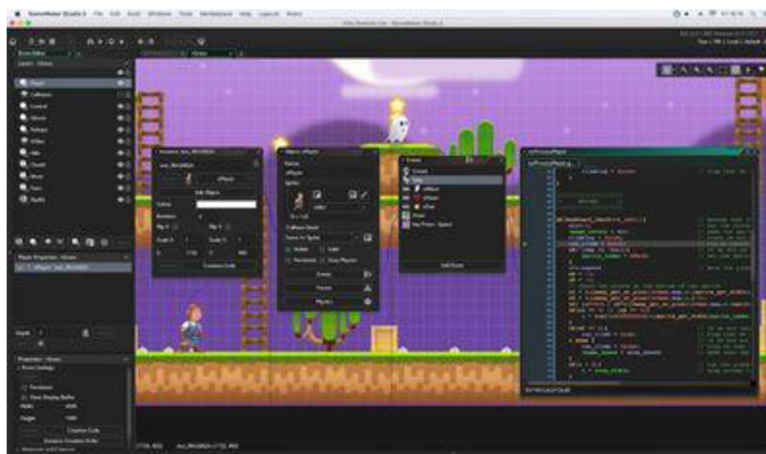
Obrázek [12]: Diagram představuje historii vývoje CryEnginu, zdroj: https://en.wikipedia.org/wiki/CryEngine

V době vytváření této diplomové práce byl proveden průzkum fór pro vývojáře, kdy se řešila otázka, který herní engine je nejlepší pro určité vývojáře a výsledkem v případě CryEngine bylo to, že CryEngine dokáže vytvořit velké věci, ale vyžaduje oproti enginům Unity a Unreal velké množství času na seznámení s enginem a cena licence je vyšší. Pro studium enginu je licence zdarma, a přestože není otevřený kód, tak se na GitHubu nalézá celý zdrojový kód enginu.

3.6.4 Engine GameMaker: Studio

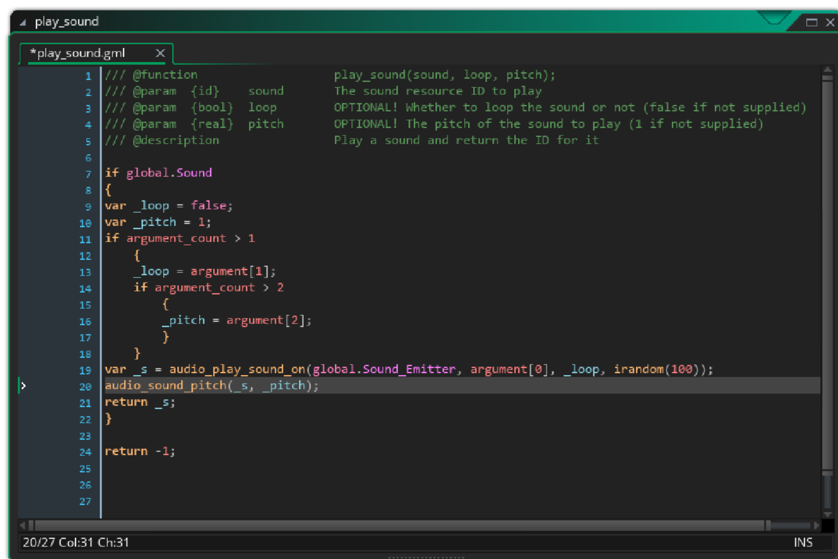
GameMaker: Studio (GMS) je komerční systém pro tvorbu her a engine, které podporuje mnoho platforem. Oproti výše zmíněným herním enginům je GMS vytvořen

pro uživatele ne-programátory a umožňuje „drag and drop“ editor neboli editor, kde si uživatel vybere komponentu z nabídky a přetáhne ji do editoru. Tato možnost umožňuje vytváření her na abstraktu logiky, která je jinak nutná. Tento postup se nazývá „Visual programming language“ (VPL), neboli visuální programovací jazyk a byl vytvořen na základě MIT „Scratch computer language learning environment“, což je programovací prostředí, které bylo vytvořeno pro děti ve věku 8 až 16 let, aby se naučily logiku, která je skrytá za programováním. Je možno vytvářet 2D i 3D hry.



Obrázek [13]: GameMaker:Studio editor, zdroj: https://www.macrumors.com/2017/09/02/gamemaker-studio-2-debuts-on-macos/

Kromě VPL obsahuje GameMaker: Studio svůj vlastní skriptovací jazyk Game Maker Language, který umožňuje práci s pokročilou funkcionalitou. Tento jazyk se označuje jako jazyk, který je podoben JavaScriptu a jazykům z C rodiny.



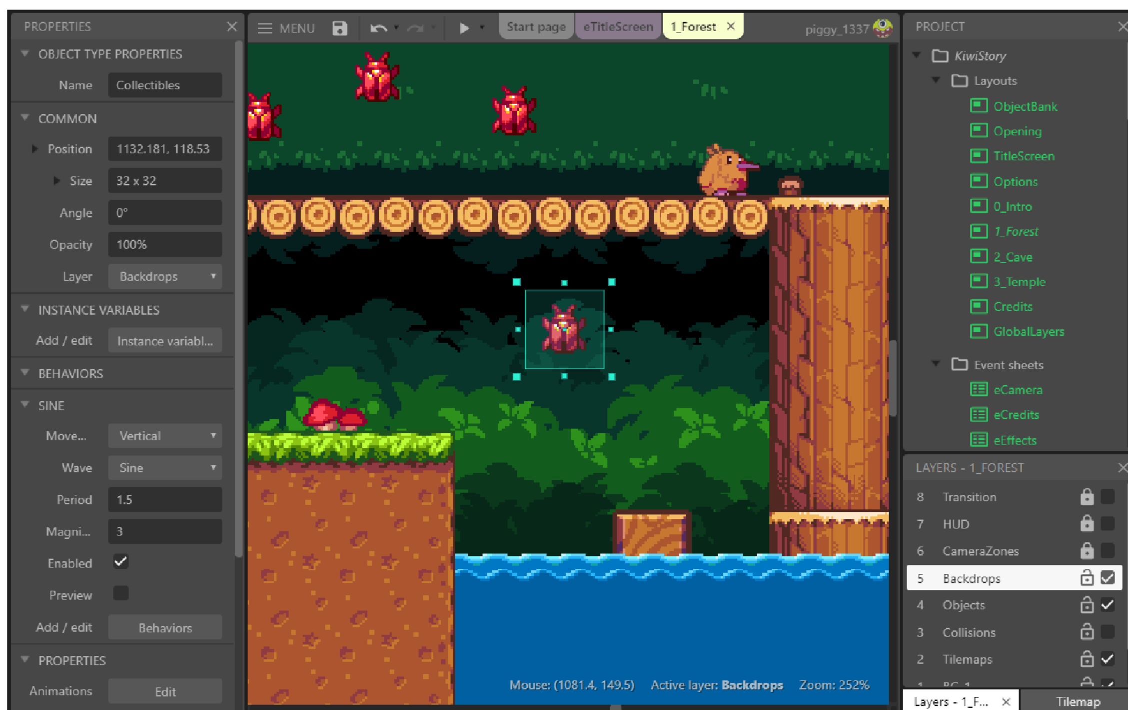
Obrázek [14]: GameMaker:Studio, příklad GamerMakerLanguage, zdroj: https://en.wikipedia.org/wiki/CryEngine

Engine GameMaker je zmíněn pouze jako učební nástroj a pro skutečný vývoj her a simulací není využíván, přestože existuje určité množství her, které se vytváří pro komerční účely na tomto enginu.

3.6.5 Engine Construct

Herní engine, který využívá jazyk C++ a Javascript.

Construct 1 a 2 byly podobné enginu jako GameMaker: Studio, ale umožňovaly vytvářet pouze 2D hry. Tento nedostatek byl odstraněn v Construct 3, který umožňuje 2D i 3D hry. Základem je opět „Drag and drop“ vizuální editor, ve kterém se tvoří většina logiky, která reaguje na události a na chování. Rozšířením jazyka je Jacascript a přestože podporuje většinu desktopových a mobilních platforem, tak Construct míří primárně na HTML5 s Javascriptem.



Obrázek [15]: Construct 3, zdroj: https://editor.construct.net/

Engine Construct 3 je zmíněn pouze jako učební nástroj a pro skutečný vývoj her asimulací. Construct 3 není využíván pro komerční účely. Vývoj enginu Construct 2 byl zastaven 1.7.2020. a vývoj je nyní zaměřen na Construct 3.

4 Praktická část – projekt

V této kapitole bude popsán praktický projekt a komponenty, které byly využity v herním enginu Unity, a srovnání projektu k simulaci v Netlogu Predátor – kořist.

Zdroje [5-7][15],[21] byly využity při vytváření praktického projektu. První 3 zdroje instruují, jak lze naprogramovat určitý systém v Unity a poslední zdroj je samotná dokumentace herního enginu Unity.

4.1 Představení projektu a nastavení simulace

Praktickým projektem této práce je simulátor sociální interakce mezi vlky, ovcemi a travou. Celý svět a agenti budou tvořeni dynamicky. Uživatel před zahájením simulace si může nastavit určité vlastnosti zvířat a nastavení generování světa. Vlastnosti vlků a ovcí se nastavují odděleně.

V projektu není simulováno to, že zvíře uteče před útočníkem, protože oproti aktuálnímu stavu projektu by se musela řešit velikost mapy a zda zvíře je schopno si uvědomit, že je jeho kořist už v nedohlednu, což by se muselo řešit přes systém „Pohledu“, kdy by každé zvíře mělo kužel pohledy muselo by se zjišťovat, zda vidí, a taky přes systém Čichu, kdy by se musel současně simulovat i směr větru, což výpočetně už přesahuje pouhou simulaci, která má za úkol zjistit, zda herní engine je možné využít pro simulace.

Vlastnosti zvířat:

- Počáteční populace. V případě hodnoty 0 se zvíře daného druhu negeneruje a v případě 1 neexistuje možnost reprodukce.
- Možnost reprodukce.
- Typ potravy (Masožravci, Býložravci a Všežravci).
- Věk dospívání, kdy zvíře přestane být mládětem a odtrhne se od následování své rodiny.
- Maximální zdraví. Při dosažení hodnoty 0 zvíře umírá.
- Regenerace zdraví za čas. Množství zdraví, které se obnoví zvířeti za vteřinu, pokud netrpí hladem a žízní.
- Věk umírání, kdy zvířeti začne pravidelně ubývat zdraví.
- Maximální energie, kdy při dosažení maximální hodnoty může zvíře zplodit dalšího jedince s jedincem svého druhu opačného pohlaví.
- Regenerace energie za čas, která ovlivní množství obnovené energie za vteřinu.
- Hodnota potravy, která představuje hodnotu doplnění potravy pro zvíře, které napadne toto zvíře, protože má hlad.
- Čas mezi porody, který představuje čas, kdy samice nemůže být oplodněna po porodu.
- Čas do porodu, který představuje čas, kdy samice bude březí. Po vypršení času nastane porod nových jedinců.
- Hlad za čas, který představuje, jak rychle bude zvířeti růst hlad. Vyšší hodnota znamená, že zvíře bude častěji řešit tuto potřebu.

- Žízeň za čas, který představuje, jak rychle bude zvířeti růst žízeň. Vyšší hodnota znamená, že zvíře bude častěji řešit tuto potřebu.
- Spánek za čas, který představuje, jak rychle bude zvířeti růst potřeba spánku. Vyšší hodnota znamená, že zvíře bude častěji řešit tuto potřebu.
- Maximum Hladu, kdy vyšší hodnota představuje to, že zvíře nebude řešit tuto potřebu tak často.
- Maximum Žízně, kdy vyšší hodnota představuje to, že zvíře nebude řešit tuto potřebu tak často.
- Maximum Spánku, kdy vyšší hodnota představuje to, že zvíře nebude řešit tuto potřebu tak často.
- Síla útoku, kdy vyšší hodnota znamená, že zvíře zraní jiné zvíře za více životů. Vyšší hodnota by měla být nastavena u masožravců a všežravců.

Vlastnosti Rostlin[3]:

- Počáteční populace. V případě hodnoty 0 se rostlina daného druhu negeneruje a v případě 1 neexistuje možnost reprodukce.
- Možnost reprodukce.
- Věk dospívání, kdy po jeho dosažení rostlina začne možnost šířit okolo sebe semínka a tímto způsobem se rozmnožovat.
- Věk umírání, kdy rostlině začne pravidelně ubývat zdraví.
- Maximální energie, kdy při dosažení maximální hodnoty může rostlina šířit semínka ve svém okolí
- Regenerace energie za čas, které ovlivní množství obnovené energie za vteřinu.
- Hodnota potravy, která představuje hodnotu doplnění potravy pro zvíře, které napadne tuto rostlinu, protože má hlad.

Čas mezi rozmnožováním, které představuje čas, po kterém rostlina rozšíří počet svého druhu ve svém okolí.

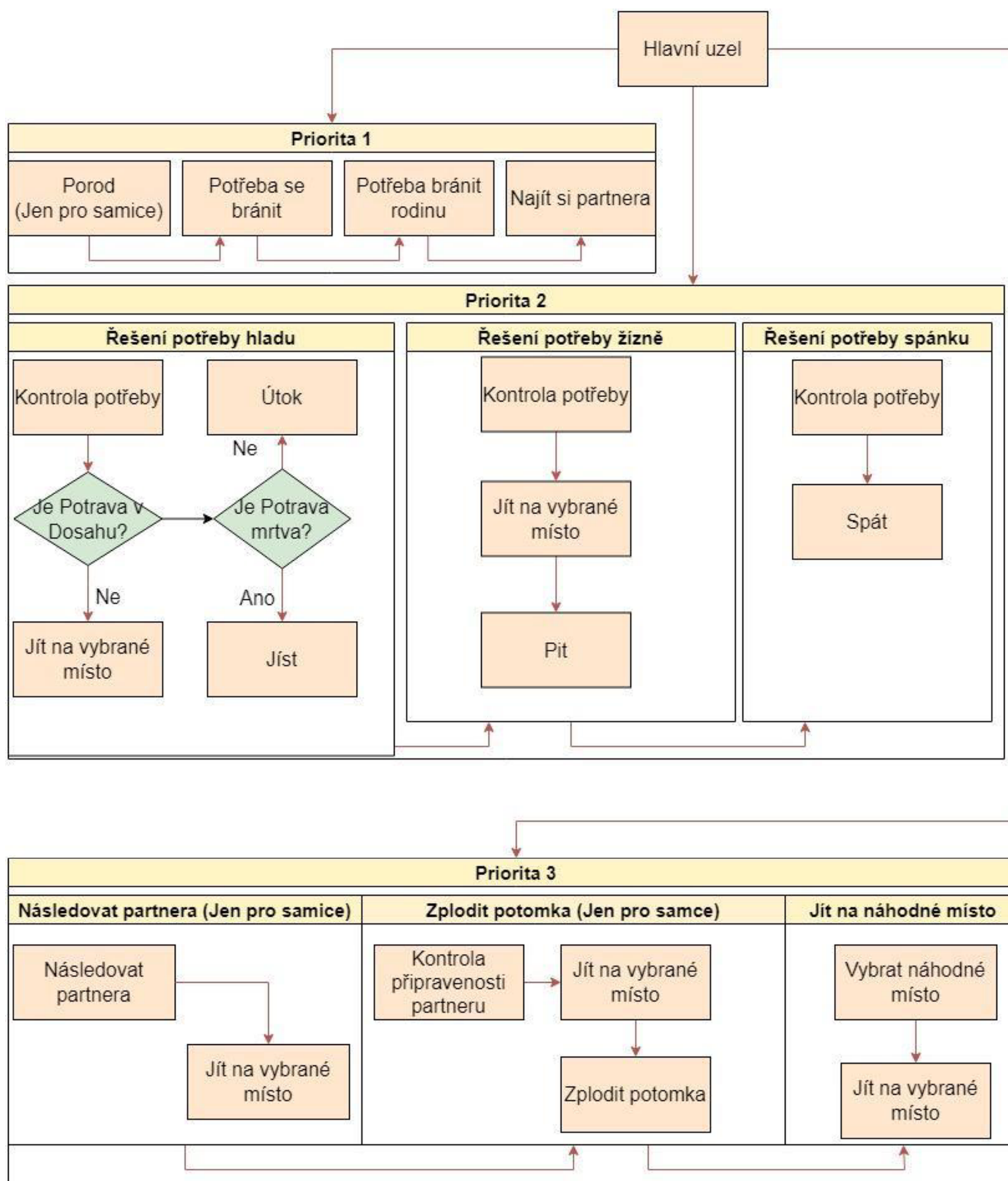
[3] U Rostlin existuje vlastnost „Čas mezi porody“, protože Rostliny a Zvířata mají abstraktního předka Entitu, která obsahuje sdílené vlastnosti. Kvůli názvu proměnné by bylo nesprávné tvořit duplicitní části kódu v Rostlinách a Zvířatech.

Vlastnosti generování světa:

- Násobič výšky, který nastavuje, jak výškově rozdílná bude mapa. Počáteční hodnota (a současně minimální možná hodnota) je 1.6, protože při nižší hodnotě by nebyla dostupná voda na vygenerované mapě.
- Počet bloků do šířky. Počet bloků po ose X ve světě. Určuje šířku světa.
- Počet bloků do hloubky. Počet bloků po ose Z. určuje hloubku světa.

4.2 Strom chování

V této kapitole jsou popsány stromy chování v podobě, ve které byly implementovány v praktickém projektu. Vysvětlení základních pojmů je v teoretické části této práce.



Obrázek [16]: Rozhodovací strom, zdroj: vlastní tvorba v https://app.diagrams.net/

Jak bylo zmíněno v kapitole 4.1, tak v projektu jsou Zvířata a Rostliny.

Rostliny nemají stromy chování a jejich kód je zpracováván v metodě Update, kde je vše definováno a jediné změny mohou nastat, když uživatel změní nastavení simulace. U zvířat je stejný systém vytváření stromu chování jako na obrázku 16, ale dochází ke dvěma změnám podle pohlaví zvířete. Existuje hlavní uzel, který se zpracovává v Update metodě. V tomto hlavním uzlu jsou vytvořeny tři skupiny, kde existují kolekce uzlů pro určité činnosti. Strom chování v aplikaci funguje tak, že se prochází skupina X, kde se zpracuje skupina uzlů A, a pokud skupina uzlů A vrátí status Úspěch, nebo Probíhá, tak se procházení skupiny A ukončí a opustí se i hlavní uzel se stavem, který obdržel ze skupiny A, toto platí pouze pro první nalezený úspěch, či probíhající stav, který se dostane na úroveň hlavního uzlu. Pokud se ze skupiny A dostane neúspěch na úroveň hlavního uzlu, tak se pokračuje k následující skupině, kde se opakuje stejný postup. Nemůže nastat situace, že by se prošly všechny skupiny a na úroveň hlavního uzlu se vrátil status neúspěch, protože ve skupině 3 je vytvořena kolekce „Jít na náhodné místo“, které se provede vždy, když zvíře nemá, co by provádělo.

První skupina obsahuje tyto kolekce uzlů:

- Porod, který je jenom pro samice. V tomto případě vznikne v okolí samice 1 až N zvířat, kdy je v kódu zajištěno, že se potomci všech zvířat rodí rovnoměrně zastoupeni 1:1 pohlaví, kdy existuje v projektu místo, kde se zapisuje, které pohlaví měl poslední potomek, který se narodil.
- Potřeba se bránit, kdy se zajišťuje to, že zvíře se brání, pokud je napadeno jiným zvířetem.
- Potřeba bránit rodinu, kdy se zjišťuje, zda partner anebo potomci jsou v ohrožení a probíhá/bude probíhat útok na ně. Potomky jsou v tomto případě myšleni přímí potomci. Není simulováno to, že by Jedinec považoval vnuka za součást rodiny, kterou by měl bránit. Okamžikem, kdy jedinec dospívá, tak se osamostatňuje a není nadále pod ochranou rodičů.

- Potřeba si najít Partnera, kdy základní podmínkou je to, že jedinec, aby si mohl najít partnera, musí být dospělý a oba jedinci nesmí mít partnera. Pro jednoduchost simulace není vytvářeno dvoření mezi jedinci a pouze si zaregistrují partnera do svých záznamů, jakmile v dosahu je jedinec, který je svobodný, tak proběhne nahlášení danému objevenému jedinci, že jsou pár.

Druhá skupina obsahuje tyto kolekce uzlů:

- Řešení potřeby hladu [4]je tvořeno z několika částí, kdy se na začátku zjišťuje, zda jedinec má hlad, což se vždy zjišťuje, zda stav hladu je >= 70%. Pokud takto není, tak se tato kolekce ukončí a pokračuje se k další potřebě. Pokud má jedinec hlad, tak začne zjišťovat, zda ve svém okolí není něco, co by mohl sníst. Pokud nic v okolí není, tak se tato činnost ukončuje a pokračuje se u další kolekce. Pokud existuje zdroj potravy, tak se jedinec vydá za tímto zdrojem potravy. Pokud je zdroj potravy živé zvíře, tak je lovené zvíře upozorněno a vzniká u tohoto zvířete potřeba se bránit a u jeho rodičů/partnera potřeba bránit rodinu. V okamžiku, kdy se jedinec přiblíží ke zdroji potravy, tak se řeší, zda je zdroj potravy živé zvíře, mrtvé zvíře anebo rostlina. V případě živého zvířete začíná souboj mezi těmito jedinci. V případě mrtvého zvířete anebo rostliny se jedinec začne krmit a činnost se ukončuje, jakmile hlad se sníží pod 10 %.
- Řešení potřeby žízně je tvořeno z několika částí, kdy se na začátku zjišťuje, zda jedinec má žízeň, což se vždy zjišťuje, zda stav žízně je >= 70%. Pokud takto není, tak se tato kolekce ukončí a pokračuje se u další kolekce. Pokud má jedinec žízeň, tak začne zjišťovat, zda ve svém okolí není voda. Pokud v okolí voda není, tak se tato činnost ukončuje a pokračuje se u další kolekce. Pokud existuje zdroj vody, tak se jedinec vydá za tímto zdrojem. Jakmile jedinec je u vody, tak se zahajuje proces pití, kdy se bude postupně snižovat stav žízně a jakmile klesne pod 10 %, tak se činnost ukončuje.
- Řešení potřeby spánku je tvořeno tak, že se zjistí, zda potřeba spánku je >= 70. Pokud takto není, tak se tato kolekce a skupina ukončí a pokračuje se u další

[4] V případě, že jedinec má hlad a nemá ve svém okolí žádný zdroj potravy, tak se jedinec nevydává hledat přímo jídlo, protože to bude náhodně objeveno přes „Jít na náhodné místo“ ve skupině třetiřadých priorit.

skupiny. Pokud má jedinec potřebu spát, tak se zastaví na místě a nebude se pohybovat, dokud neklesne potřeba pod 10 %, kdy se probudí.

Třetí skupina obsahuje tyto kolekce uzlů:

- Následování partnera (jen pro samice), kdy samice, pokud má partnera, se bude držet v dosahu partnera a bude ho následovat, pokud se od něj vzdálila.
- Zplození potomka (jen pro samce), kdy samec, pokud má partnerku a partnerka je připravena mít další potomky, tak oplodní partnerku.
- Jít na náhodné místo je kolekce, kdy jedinec vypotřeboval všechny možné činnosti a žádná neskončila ve stavu Dokončeno, nebo Probíhá. Jedinec si vybere ze svého seznamu bloků terénu, které aktuálně má v dosahu a následně se vydá na tuto pozici.

4.3 Základní komponenty editoru Unity

Při představení systému Unity bylo zmíněno, že systém pracuje tak, že každý objekt ve scéně editoru se nazývá GameObject neboli herní objekt a k těmto objektům se přidávají komponenty, které jsou univerzální a nejsou jakkoliv omezovány na počet stejných komponent ve scéně. Existuje omezení určitých komponent - u jednoho herního objektu může být přiřazena každá určitá komponenta pouze jednou. Každá komponenta má svůj vlastní kód, který se vykonává ve smyčce. Kód nelze nijak upravovat a ani není možné ho prohlížet. Jediný kód, který vývojář může měnit na základní úrovni systému Unity je přes skripty, což jsou třídy v programovacím jazyce C#. Tyto skripty se editují v externím programu, který se spustí při otevření skriptu. V závislosti na přiřazených komponentách u herního objektu je možnost přidávat metody, které jsou vypsány v dokumentaci Unity, které využívají komponenty a vrací informace o nějaké události z komponenty. V případě přidání metody, která využívá komponentu, do kódu skriptu, který je přiřazen k hernímu objektu, který nemá přiřazenou potřebnou komponentu, tak se kód této komponenty nebude vykonávat. Každá proměnná skriptu, která má otevřenost Public tak se může nastavit přímo v editoru Unity. Z tohoto důvodu nelze přímo zjistit defaultní hodnoty herních objektů ve scéně a předpřipravených herních objektů, které jsou uloženy jako šablony herních objektů, pokud jsou hodnoty veřejných vlastností nastaveny před editor. Skripty mohou tradičně využívat C# třídy, které existují v rámci projektu a nejsou přiřazeny.

Všechny komponenty, které budou v podkapitolách zmíněny, byly využity v praktickém projektu. Každá komponenta bude popsána obecně, protože vždy záleží na tom, jak komponenty spolupracují jako celek a nelze v tomto případě specifikovat využití jednotlivých komponent.

Více o komponentách lze zjisti na stránkách dokumentace o Unity [15].

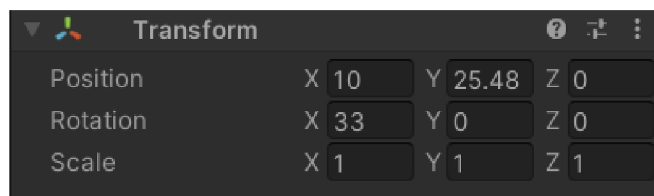
4.3.1 Transform

Každý herní objekt má tuto komponentu, která představuje umístění ve scéně, jaké stupně rotace má objekt a jak je objekt roztažen na osách šířka, výška a hloubka. Pokud

je herní objekt v hierarchii objektů ve scéně zařazen pod jiný herní objekt, tak tato komponenta představuje lokální pozici ve scéně vůči rodičovskému objektu.

Příklad: Pozice [0,0,0] by představovala u objektu bez rodiče to, že bude umístěn na pozici [0,0,0]. V případě pozice [1,2,3] u objektu s rodičem je pozice objektu [X+1, Y+2, Z+3], kdy X, Y, Z se získává od pozice rodiče. Toto se aplikuje napříč stromem hierarchie, kdy není limit na úroveň zanoření herního objektu pod jiným objektem.

Unity nabízí možnost v kódu skriptu pohybovat herním objektem kupředu, zpět, otočení, dle rotace objektu, a nastavení pozice, ať už lokální, nebo globální pozici lze nastavovat, protože objekt bez rodiče je zařazen pod scénou a lokální pozice je stejná jako globální pozice



Obrázek [17]: Komponenta Transform, zdroj: vlastní výstřižek z editoru Unity

4.3.2 Camera

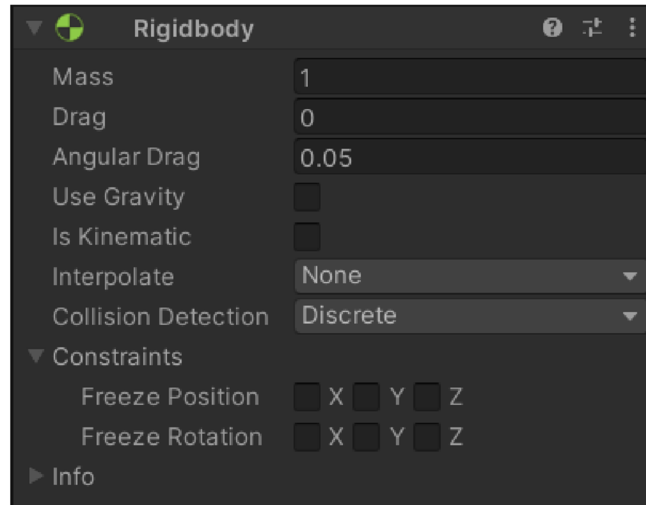
Herní komponenta, která musí být ve scéně minimálně 1x, protože jinak obrazovka „Hra“, kde je vidět v reálném čase renderování obsahu světa, neukazuje nic. U kamery lze nastavit, jak se bude renderovat obraz, který kamera vidí. Lze nastavit, co je možné vidět, jaké bude pozadí kamery, kde nebude nic vidět. Jestli se jedná o 2D/3D obraz, jak velké je zorné pole, jak se bude ořezávat obraz v závislosti na vzdálenosti herního objektu a spoustu dalších vlastností. Počet kamer není omezen ve scéně a je možné přepínat mezi nimi. Je možné současně renderovat více kamer, kdy se toto může využít jako monitory na kamery anebo zpětná zrcátka u auta.

Obrázek [18]: Komponenta Camera, zdroj: vlastní výstřižek z editoru Unity

4.3.3 RigidBody

Komponenta, která slouží pro reprezentaci těla, ať už 2D anebo 3D. Pomocí této komponenty se nastavuje vliv gravitace na herní objekt. Gravitace může být i záporná, kdy herní objekt bude letět vzhůru k nebi. Je možné nastavovat, jak bude objekt těžký a zda se jedná o kinematický objekt, kdy fyzikální část enginu nepracuje s tímto objektem a jediná možnost, jak se objekt bude pohybovat, je přes skript, který bude měnit pozici z Transform komponenty. Nastavuje se možnost detekce kolize, kdy je možné rozlišit, zda se jedná o diskrétní, nebo pokračující kolizi. Rozdíl je ten, že diskrétní kolize je ovlivněna pouze dalšími herními objekty, které mají tuto komponentu, a pokračující kolize je ovlivněna herními objekty bez této komponenty, ale obsahují komponentu Collider. Komponenta umožňuje zamknutí pozice a rotace herního objektu.

V aplikaci je využita tato komponenta tak, aby herní objekty se držely u terénu a spolupracovala s Collidery



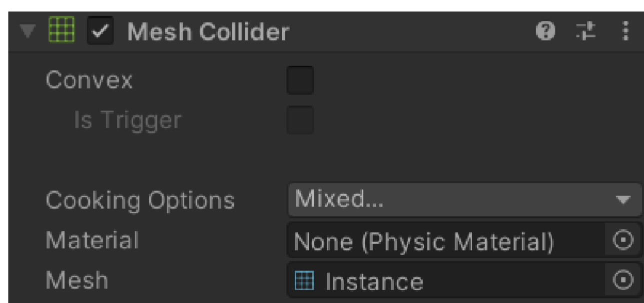
Obrázek [19]: Komponenta RigidBody, zdroj: vlastní výstřižek z editoru Unity

4.3.4 Collider

Komponenta, která slouží pro detekci kolize mezi herními objekty, které mají také Collider. Existuje několik druhů Colliderů, kdy je funkčnost stejná, ale je jiná podoba Collideru. Box Collider má podobu kostky, Sphere Collider má podobu koule, Capsule Collider má podobu oválu a Mesh Collider nemá přímo definovaný tvar. Mesh je objekt, který je tvořen vrcholy a celkově tvoří síť polygonů. Mesh Collider je možné speciálně nastavit, zda je konvexní, kdy může kolidovat s ostatními Mesh collidery. Každý Collider může být nastaven jako Trigger neboli spouštěč, kdy fyzikální engine bude ignorovat tento herní objekt při svých výpočtech, což způsobuje to, že se sníží výpočetní náročnost, protože ke kolizi může dojít jenom u jiného herního objektu s Colliderem, kdy vstoupí do takto nastaveného Collideru. Tato komponenta umožňuje u skriptu využívat metody Vstup/Stání/Odchod v oblasti Collideru na kolizi/spouštěči. Je možné přiřadit fyzikální materiál ke Collideru, pokud je potřeba definovat tření dvou objektů anebo odrazivost herního objektu od jiného objektu.V projektu je tato komponenta využívána u zvířat a rostlin, kdy pod těmito herními objekty jsou další 2 objekty, TargetArea a BodyArea. TargetArea obsahuje Collider, který je nastaven jako spouštěč a má větší dosah. Pokud vstoupí do této oblasti někdo, tak si to herní objekt, přímo zvíře anebo rostlina, zapíše do svého seznamu současně viděných herních objektů. BodyArea obsahuje Collider, který je nastaven na kolize a má menší dosah. Tento Collider umožňuje to, že pokud se dva

herní objekty k sobě přiblíží, tak se zabrání tomu, aby se modely těchto dvou herních objektů narušily nebo prolnuly

Důvod takového využití Colliderů u podřazených herních objektů je ten, že nelze v kódu skriptu přesně říci, ze kterého Collideru je volána metoda Vstup/Stání/Odchod, pokud herní objekt má více komponent typu Collider. Toto rozlišení na dva Collidery je nutné, protože bylo potřeba mít dvě oblasti, u kterých je potřeba detekovat kolize. Detekce objektů, které jsou v dosahu vnímání zvířete anebo rostliny, a detekce objektů, které by narušily prostor těla zvířete anebo rostliny.

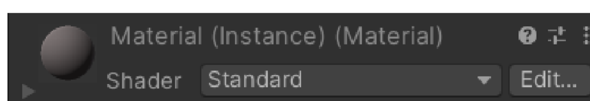


Obrázek [20]: Komponenta Mesh Collider, zdroj: vlastní výstřižek z editoru Unity

4.3.5 *Material*

Grafický materiál, který nemá souvislost s fyzikálním materiálem, který je zmíněn u Colliderů. Tento materiál určuje barvu herního objektu anebo umožňuje nastavení textury herního objektu a případně kombinace, kdy se nastavuje barevná paleta textury. Tato komponenta umožňuje nastavit celkové grafické chování herního objektu, který má model. Hlavní možností je, jak bude model pracovat se světlem a přes který grafický Shader se bude objekt renderovat a jaké obrázky se budou používat pro mapy normály, výšky, odrazivosti a jaký je koeficient „kovovosti“ textury a další.

V aplikaci je tato komponenta využita u terénu, kdy v kódu je terénu přiřazena barva v závislosti na výšce z globální pozice herního objektu.



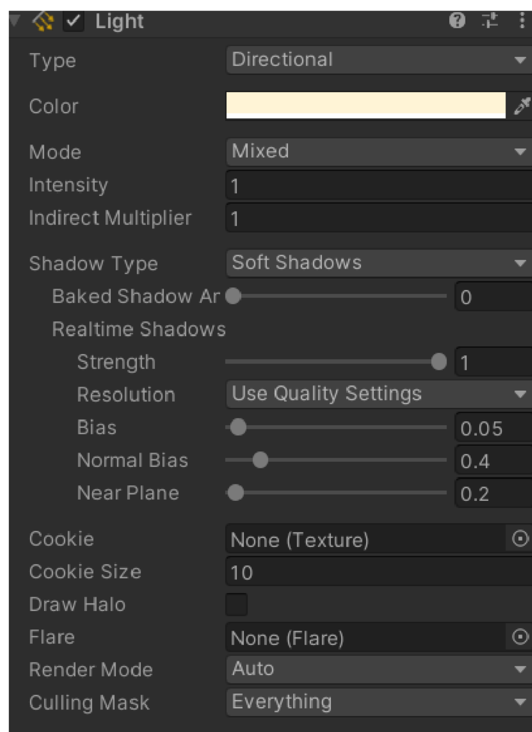
Obrázek [21]: Komponenta Material, zdroj: vlastní výstřižek z editoru Unity

4.3.6 Light

Komponenta, která poskytuje světlo do prostředí kolem sebe. Tato komponenta je vždy jednoho typu, kdy může být **směrové světlo**, které svítí jedním směrovým vektorem a světlo přechází přes celou scénu ve všech výškových úrovních, **bodové světlo**, kdy ozařuje prostor kolem sebe do podoby koule, **výsečové světlo**, kdy svítí ve výseči nastaveným směrem, a **oblastní světlo**, kdy je vyznačena oblast, ve které světlo svítí. Komponentě je možné nastavit barvu a intenzitu světla, Je možné nastavit i vlastnosti stínu, který je tvořen tímto světlem, a může se nastavit, zda světlo tvoří stín, nebo pokud tvoří stíny, tak jestli budou mít stíny ostré hrany, nebo jestli budou stíny mít zjemnělé hrany, a další vlastnosti pro stín. Dodatečné vlastnosti od Cookie jsou prvky, které ovlivňují renderování světla, kdy se určuje maska textury pro stíny a možnost vyřazení objektů pomocí masky, které se nastavuje Herním objektům.

Při práci se světlem je nutné být obezřetný, protože to může vést k vysokým výpočetním nárokům aplikace. V aplikaci je tato komponenta využita pouze jako Slunce, které svítí celou dobu simulace.

V aplikaci je tato komponenta použita pro bodové světlo, které představuje slunce.

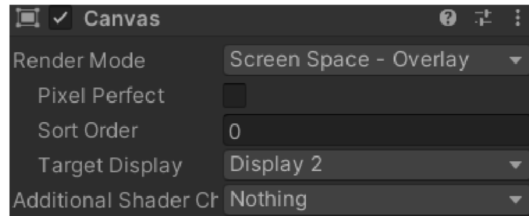


Obrázek [22]: Komponenta Light, zdroj: vlastní výstřižek z editoru Unity

4.3.7 Canvas

Komponenta, která musí být přiřazena u herního objektu, který je v hierarchii herních objektů nejvýše oproti všem herním objektům, které obsahují komponenty, které patří do kategorie Uživatelské rozhraní. Pokud není herní objekt s komponentou Canvas nejvýše v hierarchii, tak se stane to, že herní objekt, který má komponentu Uživatelské rozhraní a je v hierarchii výše, tak se nebudou vykreslovat komponenty Uživatelského rozhraní.

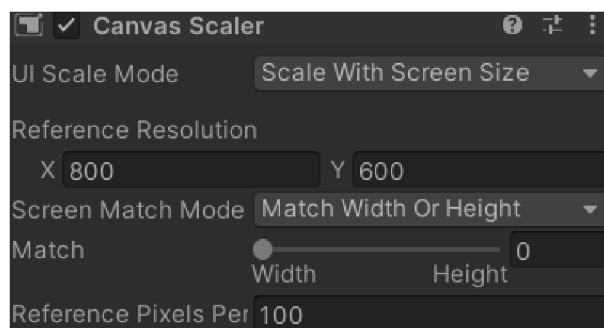
V aplikaci je tato komponenta využita pro každou oblast, ve které se nachází text a nejedná se o tlačítko. Příkladem je obrazovka, kde se nastavují parametry před spuštěním simulace, protože na této obrazovce v Canvasu probíhá celé nastavování.



Obrázek [23]: Komponenta Canvas, zdroj: vlastní výstřižek z editoru Unity

4.3.8 Canvas Scaler

Komponenta, která slouží pro přizpůsobení všech komponent Uživatelského rozhraní, které se nalézají v hierarchii objektů pod objektem, který obsahuje tuto komponentu. Toto existuje z toho důvodu, že existuje nespočet velikosti monitorů a na příliš velkém monitoru bude uživatelské rozhraní příliš malé a na malém monitoru se pravděpodobně nevejde celé uživatelské rozhraní. Je možné nastavit, zda Canvas zůstává na fixní velikosti na počtu pixelů, zda se uživatelské rozhraní mění s velikostí monitoru, nebo zda uživatelské rozhraní zůstává stejné velikosti, bez ohledu na velikost monitoru. Dále se nastavuje velikost rozlišení, zda rozhraní má být zúženo jedním směrem a pokud textury sprity mají možnost Pixel per Unit, tak kolik je nastaveno na současné rozlišení.

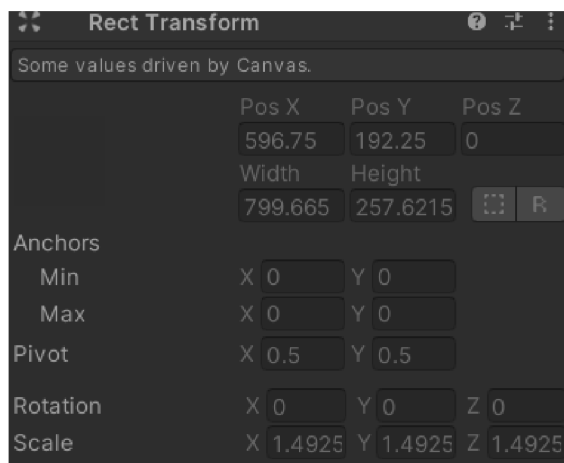


Obrázek [24]: Komponenta Canvas Scaler, zdroj: vlastní výstřižek z editoru Unity

4.3.9 Rect Transform

Komponenta, která patří do kategorie uživatelského rozhraní, kdy tato komponenta musí být přiřazena hernímu objektu, které je zařazeno v hierarchii herních objektů pod objekt, který obsahuje komponentu Canvas, protože jinak nebude vykreslena. Tuto komponentu lze využít jako pozadí na uživatelském rozhraní, kdy se nastaví pozice,

stupeň rotace a měřítko, které určuje, o kolik je plocha větší oproti základní velikosti. Anchors neboli kotva určuje kotevní body v levém dolním rohu a horním pravém rohu plochy. Pivot určuje středový bod plochy, kolem které se bude provádět rotace.

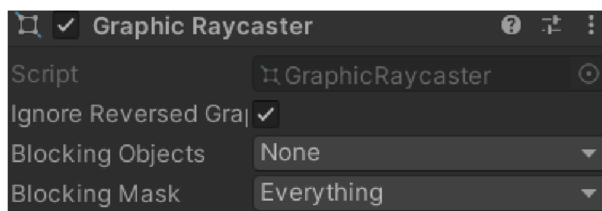


Obrázek [25]: Komponenta RectTransform, zdroj: vlastní výstřižek z editoru Unity

4.3.10 Graphic Raycaster

Komponenta, která zajišťuje vrhání paprsků proti prvkům uživatelského rozhraní, kdy je zajištěno, že nebude zasahovat herní objekty ve scéně. Je možné nastavit, které typy herních objektů budou blokovat vržený paprsek a které objekty s nastavenou maskou budou blokovat vržený paprsek. Je možnost, zda má se při výpočtech i zjišťovat odvrácená strana vrženého paprsku.

V aplikaci je tato komponenta využita k tomu, že se od pozice kurzoru myši vrhá paprsek do scény a pokud zasáhne zvíře, tak se vypíšou jeho informace v informačním poli.

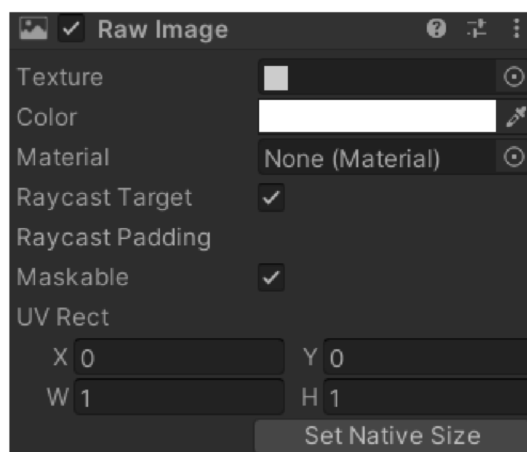


Obrázek [26]: Komponenta Graphic Raycaster, zdroj: vlastní výstřižek z editoru Unity

4.3.11 Raw Image

Komponenta, která patří do kategorie uživatelského rozhraní, kdy tato komponenta musí být přiřazena hernímu objektu, které je zařazeno v hierarchii herních objektů pod objekt, který obsahuje komponentu Canvas, protože jinak nebude vykreslena. Tato komponenta slouží pro vykreslení obrázku na základě uživatelského rozlišení. Je možné nastavit Texturu, což je samotný obrázek a zda se mění barevné schéma obrázku. Dále je možné nastavit materiál, který dodá obrázku určité vlastnosti daného materiálu, zda se obrázek může stát cílem vrženého paprsku a zda se má obrázek oříznout směrem do šířky a do výšky.

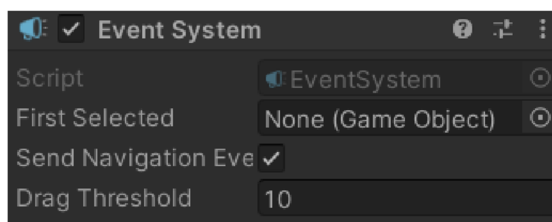
Příkladem je obrazovka, kde se nastavují parametry před spuštěním simulace, protože na této obrazovce v jsou právě Raw Image, kdy při umístění myši na těchto obrázcích zobrazí dodatečné informace.



Obrázek [27]: Komponenta Raw Image, zdroj: vlastní výstřižek z editoru Unity

4.3.12 Event System

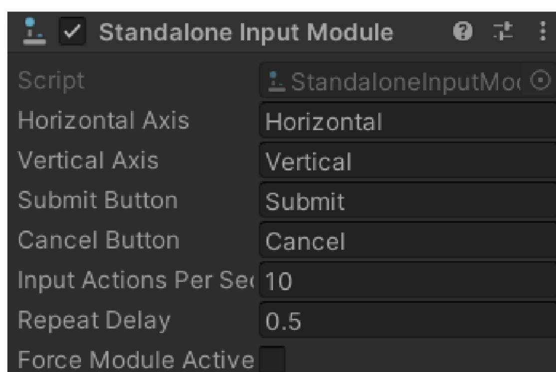
Komponenta, která zajišťuje hlášení události v aplikaci. Tato komponenta vrací herní objekt, který byl první vybraný v každé smyčce Update. Je možné nastavit, zda má Event system dovolit navigační události, které jsou spojené s myší a tlačítky (Move, Submit a Cancel) a jaká je oblast pro tažení myši v pixelech. Tato komponenta musí v aplikaci existovat, jakmile je potřeba práce s myší nebo klávesnicí a dalšími zařízeními.



Obrázek [28]: Komponenta Event System, zdroj: vlastní výstřižek z editoru Unity

4.3.13 Standalone Input Module

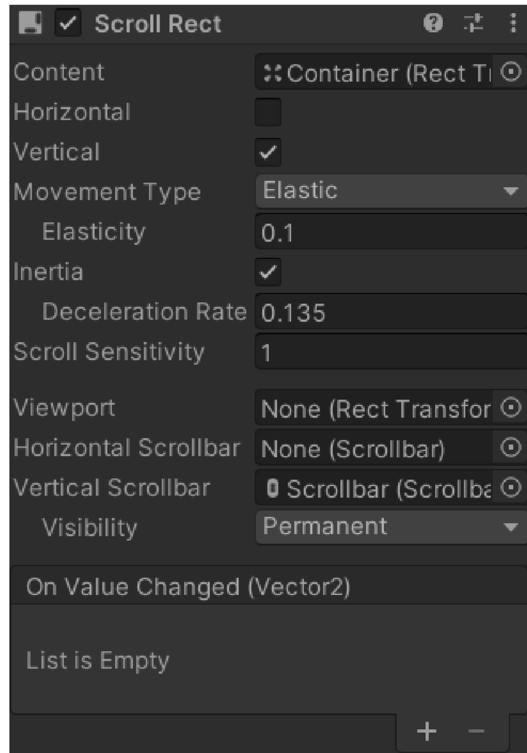
Modul, který slouží pro zjišťování události z myši či z kontroleru. Je možné nastavit osy pohybu, tlačítka Potvrzení, Ukončení, kolik akcí za vteřinu je možné a minimální zpoždění mezi akcemi a zda je tato komponenta nuceně zapnuta.



Obrázek [29]: Komponenta Standalone Input Module, zdroj: vlastní výstřižek z editoru Unity

4.3.14 Scroll Rect

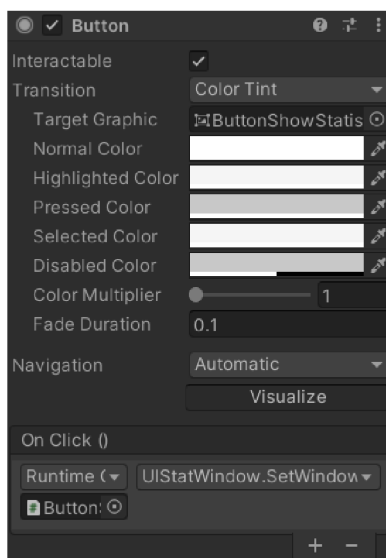
Komponenta, která slouží pro rolování plochy komponenty Rect Transform. Této komponentě se nastaví komponenta Rect Transform od objektu, který představuje plochu na uživatelském rozhraní, kde bude plocha, a následně lze nastavit, zda se dá rolovat plocha horizontálně či vertikálně, vlastnosti, jakým způsobem se roluje plocha. Dále se nastavuje Viewport, což je další Rect Transform plocha, která je v hierarchii výše nad obsahem rolování. Pokud je potřeba, aby bylo vidět, jaká část plochy je právě ukázána, tak je možné přiřadit Rolovací lištu. Tato komponenta nabízí událost Na změně hodnoty proveď, kdy se vrací nová pozice rolovací plochy.



Obrázek [30]: Komponenta Scroll Rect, zdroj: vlastní výstřižek z editoru Unity

4.3.15 Button

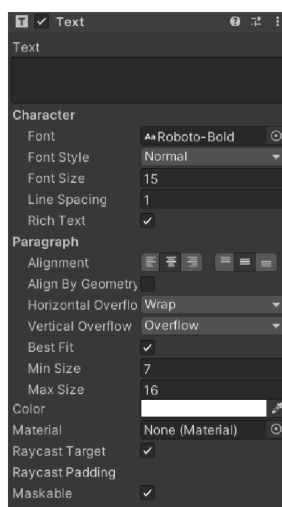
Komponenta, která představuje tlačítko v Uživatelském rozhraní, kdy lze nastavit barvy pro určité případy, ale u této komponenty je nejdůležitější nastavení skriptu a nastavení metody ve skriptu, která se spustí při kliknutí na tuto komponentu. Není omezen počet skriptů a volaných metod při kliknutí. Skripty, které jsou určené, tak by měly být tvořeny tak, aby je bylo možné znovu využít při podobné situaci na jiném tlačítku. Toto přiřazování skriptu by mělo být provedeno manuálně v editoru, protože v systému Unity nelze jednoduše přiřadit metodu k tlačítku, aby ji zpracovalo při kliknutí.



Obrázek [31]: Komponenta Button, zdroj: vlastní výstřižek z editoru Unity

4.3.16 Text

Komponenta, která představuje textové pole na uživatelském rozhraní. Je možné nastavit všechny tradiční prvky z textových editorů, Font písma, velikost, styl, vzdálenost mezi řádky. Nově je možnost nastavení, zda se jedná o Rich text, který se pokusí automaticky zmenšovat/zvětšovat velikost písma, podle velikosti herního objektu, ke kterému komponenta patří. Je možné nastavit, jak by se měl text zarovnávat, jak se pracuje s přetečením textu. Nová věc oproti textovým editorům je zarovnání textu podle geometrie, což způsobí to, že se text bude zarovnávat ke geometrické křivce.



Obrázek [32]: Komponenta Text, zdroj: vlastní výstřižek z editoru Unity

4.4 Systém navigace a systém agentů

Navigace v prostoru pro tento projekt je řešena přes Unity NavMesh system [15], který zařizuje správu pohybu agentů. Existují i další řešení, ale tanebyla implementována.

4.4.1 NavMesh

Datová struktura, která popisuje průchozí prostředí ve světě. NavMesh umožňuje NavMesh agentům najít cestu z bodu A do bodu B, pokud oba body existují v NavMesh struktuře a existuje spojení mezi nimi. Tato struktura se tvoří tak, že se například na kostku přidá komponenta NavMesh a na jedné straně kostky (v editoru Unity vždy ta horní stěna) si vytvoří NavMesh nad touto stěnou. NavMesh je vždy vytvořen o několik jednotek vzdálenosti výš, než je samotná stěna a při zapnutí aplikace v debugovacím módu v editoru, či v exportované aplikaci není tento NavMesh viditelný.

4.4.2 NavMesh agent

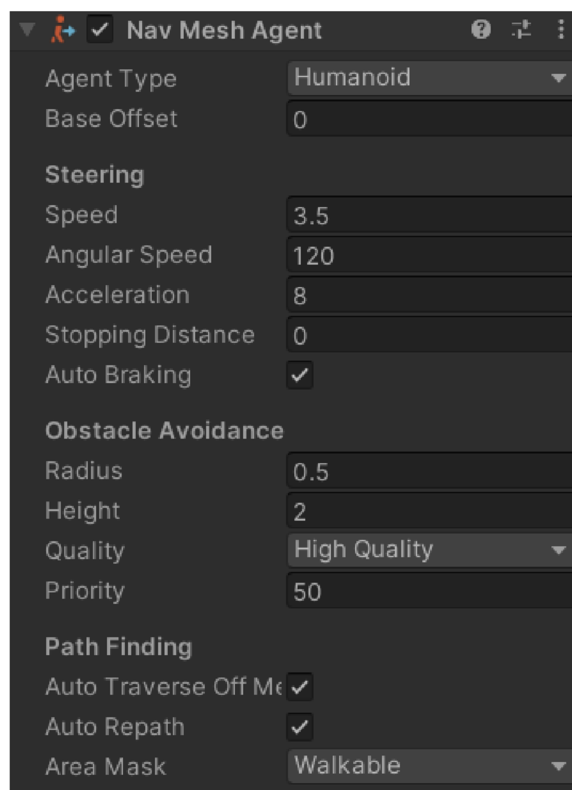
Komponenta, která se přidá k hernímu objektu, který by měl znázorňovat agenta, který by se měl pohybovat po nějaké ploše, která má NavMesh. Samotná komponenta má několik vlastností, které se nastavují.

Agent type určuje velikost agenta a tvoří skupiny agentů dle velikosti. Typ má definovaný, jak vysoké kroky může dělat, aby se dostal na platformu, která není na stejné výškové úrovni, jako platforma, na které agent aktuálně stojí. Jak vysoký je agent, což může být využito tak, že agent může/nemůže najít cestu přes NavMesh, pokud cesta vede přes tunel a myš může projít skrz tunel, zatímco kočka by nemohla najít cestu skrz tunel a musela by hledat jinou cestu. Rádius znázorňuje, jak moc široký je agent. Příkladem je to, že široký agent by nemohl projít mezi dvěma překážkami. Všichni agenti při hledání cesty jsou znázorněni válcem. Poslední vlastnost typu agentů je sklon prostředí, kdy se můžou pohybovat po stoupajícím prostředí. Příkladem je to, že by agent neměl být schopen vylézt po útesu, který se blíží k 90 stupňům.

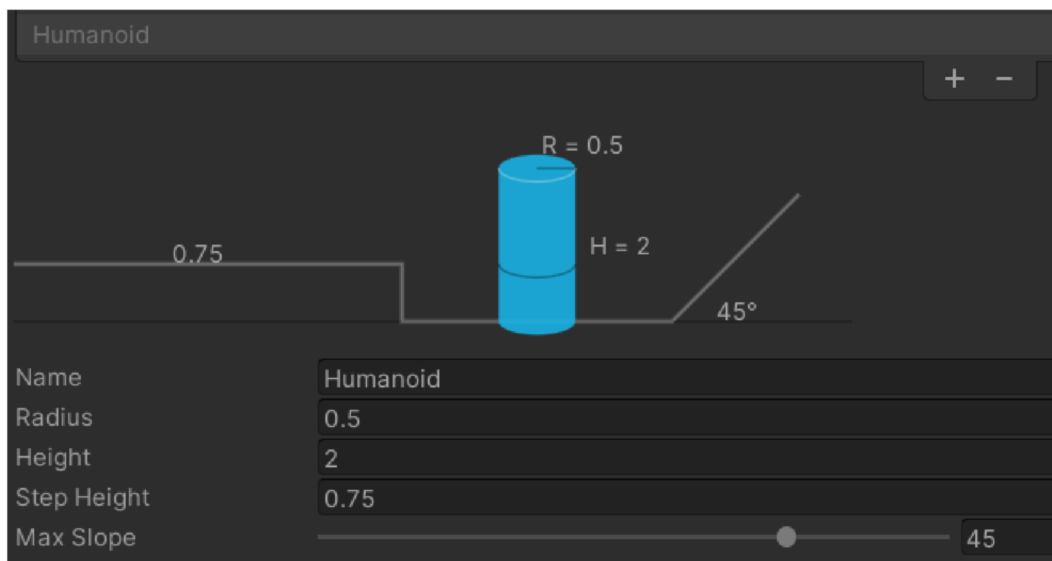
- Base Offset, který určuje, jak vysoko je válec, který znázorňuje agenta při hledání cesty, nad herním objektem
- Steering neboli řízení, které určuje rychlost pohybu po NavMesh. Umožňuje nastavit rychlost pohybu kupředu, rychlost otáčení, jak rychle

agent získává rychlost ze stavu stání, jak daleko před nastaveným cílem má zastavovat a zda má sám snižovat rychlost a zastavit, než by mělo dojít ke kolizi s jiným herním objektem.

- Obstacle Avoidance neboli vyhýbání se překážkám. Nastavuje se rádius při vyhýbání, výška, kdy by se měl vyhýbat překážka. Příkladem by bylo vyhýbání se krápníkům ze stropu v jeskyni. Dále je možné nastavit kvalitu vyhýbání a prioritu, kdy oba parametry určují výpočetní náročnost při hledání cesty.
- Path Finding neboli hledání cesty. Je možné nastavit, zda agent může vyhledávat cestu přes Off Mesh Links na dalších NavMesh strukturách, zda Agent si může sám naplánovat novou cestu, pokud aktuální cesta přestane být validní a jaké Area Mask neboli Masky prostředí jsou průchozí pro daného agenta. Příkladem



Obrázek [33]: Komponenta Nav Mesh Agent, zdroj: vlastní výstřižek z editoru Unity



Obrázek [34]: Komponenta Agent Type, zdroj: vlastní výstřižek z editoru Unity

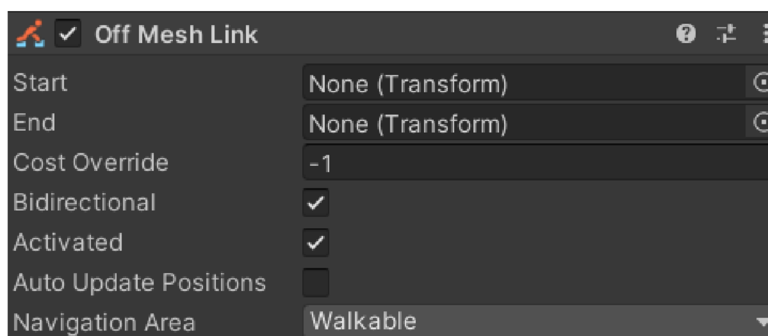
4.4.3 Off-Mesh Link

Komponenta, která umožňuje vytvořit spojení mezi dvěma navigačními meshy, které nejsou přímo spojené. Pohyb agenta, který využije tuto komponentu k pohybu lze připodobnit k chůzi po laně, protože mezi Počátečním a Konečným bodem se pohybuje po přímé lince. Během cesty po tomto spojení neplatí na herní objekt fyzikální pravidla, které by jinak platily.

Tato komponenta má několik vlastností:

- Start neboli 3D pozice, odkud je možné použít tuto cestu.
- End neboli 3D pozice, kam cesta vede.
- Cost Override neboli náročnost cesty pro agenta. Výchozí hodnota -1 znamená, že cena je určena jenom délkou cesty.
- Bidirectional, což umožňuje používat cestu obousměrně.
- Activated, zda je cesta aktivní.
- Auto update Position, kdy hodnota Pravda, znamená, že se Off Mesh Link automaticky připojí k navigačnímu Meshi, pokud se pohne cílový bod (End) linku. Pokud hodnota je Nepravda, tak se počáteční bod (Start) nebude přesouvat, pokud by se stalo to, že by se cílový bod (End) přesunul.

- Navigation Area, což definuje vrstvu navigačního Meshe, kterou mohou využít navigační agenti, kteří mají nastaveno, že mohou procházet právě nastavenou Navigation Area, kterou má Off Mesh Link.



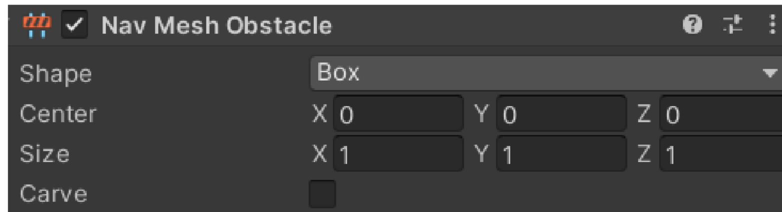
Obrázek [35]: Komponenta Off Mesh Link, zdroj: vlastní výstřižek z editoru Unity

4.4.4 NavMesh Obstacle

Komponenta, která se přiřazuje k herním objektům, které představují překážku pro navigační agenty a znemožní jim plánovat cestu přes pole působnosti této komponenty.

Tato komponenta má několik vlastností:

- Shape, kdy lze určit, zda pole působnosti je tvořeno kvádrem, nebo se jedná o kapsli.
- Center, kdy se nastavuje počátek překážky.
- Size, který určuje, jak velké pole působnosti má překážka. V případě kvádru se nastavuje velikost pro osy X, Y, Z a v případě kapsle se nastavuje výška a poloměr.
- Carve, které tvoří díru v navigačním meshi, pokud je hodnota Pravda. Pokud je hodnota Nepravda, tak navigační je mapa pod celým objektem, ale tvoří překážku, která neumožňuje cestu. V případě hodnoty Pravda, tak navigační mapa nevytvoří pod objektem spojovací body pro navigaci a v poli působnosti překážky je díra. Rozdíl je v tom, že pokud by překážka byla odstraněna v případě hodnoty Nepravda, tak navigační agenti by mohli najít cestu, která dříve nebyla možná, ale v případě hodnoty Pravda agenti nebudou moci nalézt cestu.



Obrázek [36]: Komponenta Nav Mesh Obstacle, zdroj: vlastní výstřižek z editoru Unity

4.5 Řešené problémy

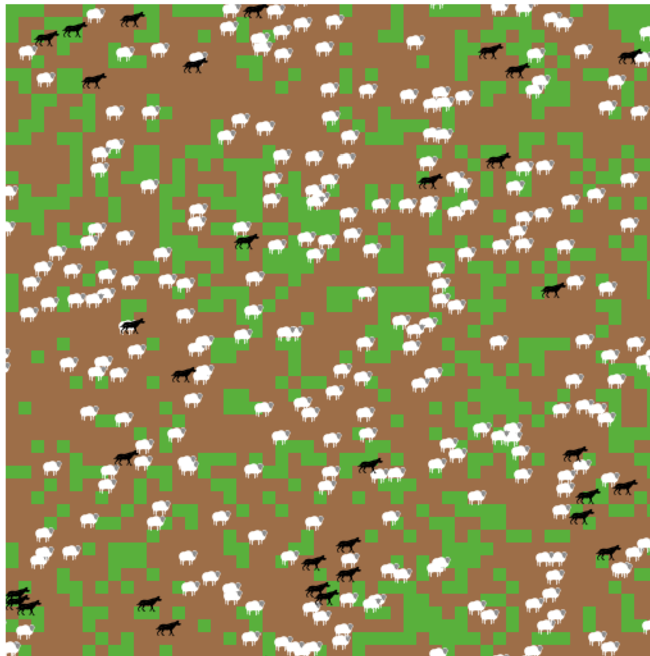
V průběhu vytváření praktické části se objevilo několik problémů.

Hlavním problémem herního enginu Unity je referencovatelnost. Jak bylo zmíněno v kapitole 4.4 o herních objektech, tak pokud má herní objekt přiřazen skript (třídu programování), která má veřejné proměnné, tak je možno v editoru Unity přiřadit proměnou přímo ve scéně pro tento herní objekt, což je využíváno často. Problém vzniká, jakmile se v herní třídě provede úprava názvu proměnné, kdy se následně zruší refence u všech herních objektů, které mají tento skript. Tomuto problému se v tomto enginu nelze vyhnout a několikrát vzniknul problém, kdy se přejmenovala proměnná kvůli nesrozumitelnému názvu a následně se musela opravit reference v editoru. Tomu to se dá vyhnout v případě dynamického generovaní objektů, ale vždy existují některé herní objekty, které jsou pevně vytvořeny v editoru a jsou zodpovědné za generování dalších herních objektů a u těchto herních objektů může vzniknout tento problém.

Vyskytl se problém s ukončením pohybu agenta (zvířete), pokud došel na svou pozici a následně měl zůstat na místě a nic nedělat, protože komponenta Nav Mesh Agenta, která zajišťuje pohyb po navigačním Meshi, způsobovala to, že se celý herní objekt neustále otáčel náhodným směrem. Tento problém byl vyřešen tak, že se přepíná Area Mask, která dovoluje agentovi procházet po Meshi, který patří do povolené Area Mask, na oblast, která neexistuje, a to zamezuje pohybu a otáčení modelu. Původní Area Mask se nastavuje jenom na dobu, než agent splní svůj původní cíl, nebo nedostane nový cíl. Tento způsob byl inspirován zdrojem [11].

4.6 Rozdíl projektu oproti simulaci Predátor – kořist v systému NetLogo

Přestože je téma stejné, tak se projekt liší celkově v komplexností a novými prvky oproti simulaci v systému NetLogo (dále jen pouze jako „v NetLogu“).

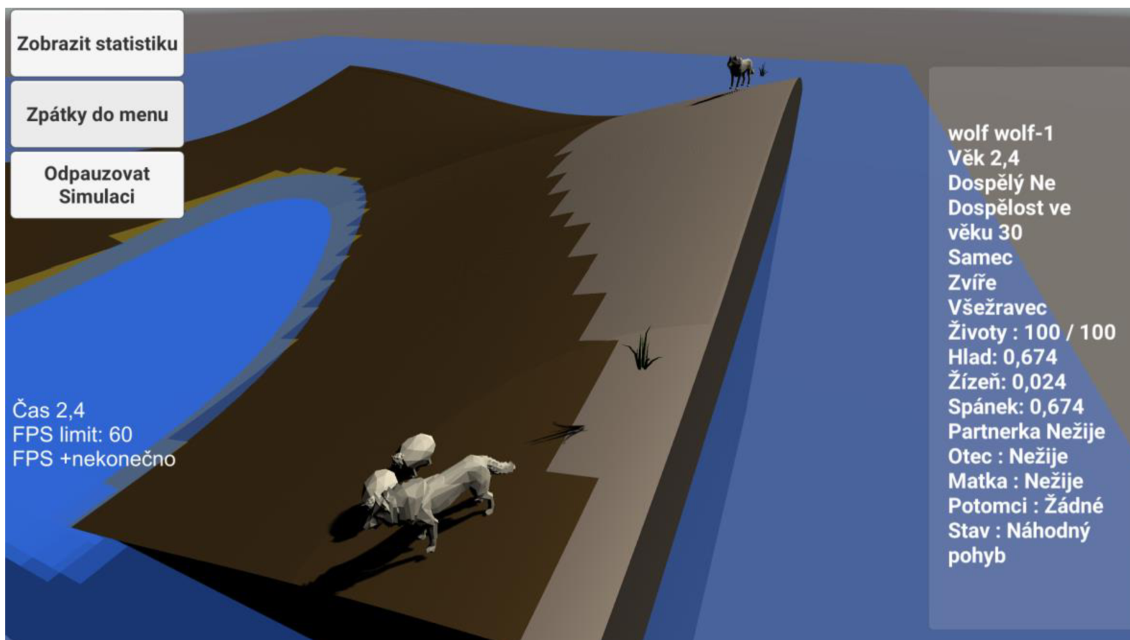


Obrázek [37]: Model Vlk – Ovce v aplikaci NetLogo, zdroj: [*https://ccl.northwestern.edu/netlogo/models/WolfSheepPredation*](https://ccl.northwestern.edu/netlogo/models/WolfSheepPredation)

Rozdílné prvky:

- Systém pohybu
  - V NetLogu je pohyb zcela náhodný.
  - V projektu je pohyb vždy cílený, pokud se nejedná o náhodný pohyb, protože zvíře nemá žádné potřeby, které by plnilo.
- Svět
  - V NetLogu je svět 2D a jedná se pouze o čtvercovou síť X řádků a Y sloupců, kdy každý čtverec je buď hnědý jako hlína, nebo je zelený jako tráva. Pokud ovce je na zeleném čtverci, tak přemění čtverec na hnědý.
  - V projektu je 3D svět, který je dynamicky generovaný dle zadaných parametrů (velikost světa do výšky, šířky a hloubky). Podle výšky terénu proti maximální výšce světa je na polygonu terénu nastaveno, zda se jedná o vodu, hlínu či kámen a dostává svou texturu. Pokud se jedná o vodu, tak jsou vytvořeny herní objekty „Voda“, které zvíře rozpoznává a může se u objektu napít.

- Rozmnožování
    - V Netlogu je v každém kroku přesně definovaná pravděpodobnost, že se zvíře „rozdělí“ na 2 zvířata
    - V projektu je rozpoznáváno, zda zvíře je samec/samice a zda je zvíře v partnerském vztahu s jiným zvířetem opačného pohlaví. Samice mají přesně definovaný čas, jak dlouho trvá březost a jak dlouho po porodu nemohou znova být březí.



Obrázek [38]: Ukázka diplomového projektu, zdroj: vlastní diplomový projekt

Nové prvky v projektu:

- Systém rozhodování
- Vytváření rodin
    - Samec a samice vytváří pár a plodí potomky. Rodina se drží pospolu, kdy se následuje otec rodiny.
- Potřeby hladu, žízně a spánku.

4.7 Možná rozšíření projektu

Tento praktický projekt byl vytvořen na ověření teorie, zda by se dal herní engine využít pro vytváření simulačních modelů. Projekt představuje pokus o vytvoření jednoho takové modelu, ale existuje spousta možností, jak tento model/projekt rozšířit. Pokud by

se měl herní engine začít využívat pro simulační modely v analytickém a vědeckém směru, tak bude potřeba vytvořit základ pro toto využívání. V herním enginu Unity se označují takovéto základy jako šablony a lze je stáhnout, po případné koupi, pokud nejsou zadarmo zveřejněny, a využít je pro své projekty. Tyto šablony mají nejblíže k tradičnímu označení „knihovna" anebo „framework" u programování.

Něco ve stylu následujícího odstavce by mělo být na začátku praktické části – to je to, co chcete zkoumat. Co/jaké prvky je potřeba pro tvorbu simulace – a pak třeba jaké komponenty v unity jednotlivým prvkům odpovídají.

Nutný základ pro tyto šablony by měl obsahovat generování světa, generování herních objektů s a bez agentů, spravovat systém navigačního meshe, umožňovat vizuálně vytvářet a přeorganizovávat stromy chování u jednotlivých druhů agentů a dovolovat i vytvářet nové akce chování.

Jelikož v Unity je možné stažené šablony upravovat, tak není omezen potenciál, protože uživatel by si mohl naprogramovat sám dodatečné funkce, nebo pozměnit fungování samotné šablony. Příkladem takovéto změny může být vypnutí generování světa a nechat generovat agenty na předem vytvořené prostředí v editoru. Dalším příkladem může být přidání nové vlastnosti pro agenta, kdy se následně mohou přepsat akce chování, aby nová vlastnost dosáhla své zamýšlené funkčnosti.

Bez takového základu by vytváření simulačních modelů byl problém pro uživatele, kteří nemají programátorské znalosti na vyšší úrovni, protože návrh dynamicky vytvořeného světa, systém agentů a samotný systém pohybu je značně složitý, přestože se mohou využít speciální komponenty enginu Unity, které byly v případě praktického projektu využity. Pokud by nebyly využity komponenty agentů a navigačního meshe, tak by uživatel samotný musel vytvořit systém hledání cesty, což sice existuje několik algoritmů pro to, ale pro osobu, která má minimální zkušenosti s programováním, tak se jedná o složitý úkol a nelze říci, který algoritmus by byl nejlepší pro specificky generované prostředí.

5 Závěry a doporučení

Cílem této práce bylo zodpovědět otázku, zda lze využít herní enginy jako simulační prostředí pro testování hypotéz při určitých podmínkách.

V teoretické části práce byly představeny hlavní enginy, které by mohly být použity, a hlavní témata, která je nutné řešit, přestože je použit herní engine pro simulace, který uživateli usnadňuje tvorbu simulačního prostředí. Tím, že je využit herní engine pro vytvoření simulačního prostředí, tak uživatel nemusí tvořit složité systémy, které by zajišťovaly renderování grafického prostředí, fyzikální engine, který by zajišťoval kolize, a další důležité části, které jsou potřebné.

Co uživateli zbývá rozhodnout na začátku tvorby simulačního prostředí je to, zda bude prostředí statické, či dynamické. V případě statického prostředí je to ulehčení práce pro uživatele, protože nemusí tvořit složitý systém tvorby terénu a také má naprostou kontrolu nad prostředím, kdy může kdykoliv ovlivnit jenom jeden herní objekt ve scéně editoru, aniž by ovlivnit jakýkoliv další herní objekty stejného typu. Uživatel dále musí vytvořit systém pohybu, kdy se musí objevit cesta ke zvolenému cíli z bodu A do bodu B, a systém chování, který zařizuje to, že agent se rozhoduje, co právě bude dělat a umožňuje tvořit stromy chování, které jsou základem komplexního chování. Tento systém chování by měl být tvořen ze základních akcí, které by nemělo být možné více rozdělit, a měl by umožňovat jistou úroveň modifikovatelnosti, kdy by měl být strom chování být snadno rozšířen o další akce, či celé větve chování.

Výhodou herního enginu oproti prostředí NetLogo, které bylo zmíněno v teoretické části, je ten, že herní agenti vykonávají svůj kód nezávisle na ostatních, takže všichni agenti pracují svým tempem a současně všichni jsou současně aktivní. Toto je označeno to, že Agenti v NetLogu pracují po kolech, kdy kolo se ukončí, jakmile všichni agenti provedou kód ve smyčce, takže se může stát, že se celá simulace zastaví, protože jeden agent nedokázal dokončit svou smyčku.

V praktické části byl popsán praktický projekt, který byl vytvořen v herním enginu Unity a jednalo se o simulátor, kdy se dynamicky generovalo prostředí, ve kterém se následně vygenerovaly entity Ovce, Vlk a Rostlina v počtu dle nastavení před generováním. Agenti se snaží přežít a jejich chování je ovlivněno nastavením vlastností před spuštěním simulace. Jelikož praktický projekt slouží pouze pro ověření výše zmíněné hypotézy, tak nebyly vytvořeny animace pro agenty a jsou to jenom 3D modely, které byly „Zakoupeny" za 0€ z komunitního tržiště Unity. Pro pohyb byla použita komponenta NavMesh neboli navigační mesh, který slouží pro nalezení cesty k cíli a zařizuje následný pohyb agentů, pokud je správně vytvořen terén, který vyhovuje podmínkám této komponenty.

Úplně na závěr zbývá odpovědět na otázku, zda lze herní engine využít pro simulace komplexních systémů.

Odpověď zní, že ano.

Herní engine usnadňuje vytváření simulačních prostředí, protože spoustu systémů už obsahuje. Pro akademické účely by bylo potřeba vytvořit komunitní balíček tříd a předpřipravených herních objektů, které by zajistily větší uživatelskou přívětivost, pokud uživatel neovládá programování na vyšší úrovni. V závislosti na velikosti simulace, kdy záleží na počtu agentů a na časové náročnosti kódu agentů by byla potřeba verze takového komunitního balíčku, kdy by byly grafické modely co nejvíce omezeny v počtu polygonů a samotné části kódu by byly omezeny, aby se zvýšil výpočetní potenciál simulace.

Na otázku, jaký by byl minimální základ pro to, aby se dal využít herní engine pro veřejné účely už bylo zodpovězeno v předchozí kapitole „Možné rozšíření projektu".

Základ by musel obsahovat:

- Systém navigace, který by podporoval 2D i 3D světy a obsahoval už vytvořené třídy, které by byly schopny dostatečně rychle vypočítávat cestu.

- Strukturu pro vytváření uzlů chování a správu stromu chování, kdy by bylo výhodou, kdyby bylo možné pracovat s vizuálním stromem místo kódovým provedením a upravovat strom pouhým přetahováním uzlů z nabídky na místo ve stromě chování.
- Základní systém agentů, který by obsahoval strukturu událostí, kdy agent by měl už vytvořený kolizní systém a metody s nimi spojené, které by bylo možné v potomcích přepsat. Dále další metody, které by sloužily pro komunikaci mezi agenty a kolekce, kam by se ukládaly informace o prostředí a ostatních agentech.
- Možnost si nechat vygenerovat svět dynamicky podle zadaných parametrů.
- Systém záznamů, který by bylo možné exportovat a uživatel mohl zapisovat.

Všechny body výše by musely splňovat to, že by řešení bylo dostatečně modulární a umožňovalo uživateli přepsat všechny metody v potomkovi. Dále by bylo vhodné vytvořit několik odlišných projektů, které by představily možnosti základu. Vše by muselo být vytvořeno tak, aby byly výpočetní časy minimální.

6 Seznam použité literatury

[1] Behavior trees for computer games - researchgate. (n.d.). https://www.researchgate.net/publication/312869797_Behavior_Trees_for_Computer_Games

[2] Abd Algfoor, Z., Sunar, M. S., & Kolivand, H. (2015). A comprehensive study on pathfinding techniques for Robotics and video games. International Journal of Computer Games Technology, 2015, 1–11. https://doi.org/10.1155/2015/736138

[3] Becker-Asano, C., Ruzzoli, F., Hölscher, C., & Nebel, B. (2014). A multi-agent system based on Unity 4 for virtual perception and wayfinding. Transportation Research Procedia, 2, 452–455. https://doi.org/10.1016/j.trpro.2014.09.059

[4] Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences, 99(suppl_3), 7280–7287. https://doi.org/10.1073/pnas.082080899

[5] Brackeys. (2017, April 16). How to make RTS Camera Movement in Unity. YouTube. https://www.youtube.com/watch?v=cfjLQrMGEb4&ab_channel=Brackeys

[6] DapperDino. (n.d.). Tooltip-ui-tutorial/assets at master · Dapperdino/Tooltip-ui-tutorial. GitHub. https://github.com/DapperDino/Tooltip-UI-Tutorial/tree/master/Assets

[7] GameDevChef. (n.d.). GameDevChef/Behaviourtrees: Tutorial project for YouTube. GitHub. https://github.com/GameDevChef/BehaviourTrees

[8] Gembarski, P. C. (2020). Agent collaboration in a multi-agent-system for analysis and optimization of Mechanical Engineering Parts. Procedia Computer Science, 176, 592–601. https://doi.org/10.1016/j.procs.2020.08.061

[9] González-Briones, A., Mezquita, Y., Castellanos-Garzón, J. A., Prieto, J., & Corchado, J. M. (2019). Intelligent multi-agent system for water reduction in automotive irrigation processes. Procedia Computer Science, 151, 971–976. https://doi.org/10.1016/j.procs.2019.04.136

[10] Maglio, P. P., & Mabry, P. L. (2011). Agent-based models and systems science approaches to public health. American Journal of Preventive Medicine, 40(3), 392–394. https://doi.org/10.1016/j.amepre.2010.11.010

[11] Navmesh how to check if full path available? C#. Navmesh How to check if full path available? C# - Unity Answers. (n.d.). https://answers.unity.com/questions/1197626/navmesh-how-to-check-if-full-path-available-c.html

[12] Niazi, M., & Hussain, A. (2011). Agent-based computing from multi-agent systems to agent-based models: A visual survey. Scientometrics, 89(2), 479–499. https://doi.org/10.1007/s11192-011-0468-9

[13] Pathfinding. Hledání cesty. (n.d.). https://wikigcs.cyou/wiki/Pathfinding#Hierarchical_path_finding

[14] Programmer think. A * algorithm for path planning. (n.d.). https://programmer.ink/think/a-algorithm-for-path-planning.html

[15] Technologies, U. (n.d.). Unity user manual 2021.3 (LTS). Unity. https://docs.unity3d.com/Manual/index.html
[16] Types of environments in AI. GeeksforGeeks. (2022, August 24). https://www.geeksforgeeks.org/types-of-environments-in-ai/
[17] Understanding agent environment in AI. KDnuggets. (n.d.). https://www.kdnuggets.com/2022/05/understanding-agent-environment-ai.html
[18] Wikimedia Foundation. (2022, July 1). Pathfinding. Wikipedia. https://en.wikipedia.org/wiki/Pathfinding#Dijkstra's_algorithm
[19] Wikimedia Foundation. (2022, July 28). Flood fill. Wikipedia. https://en.wikipedia.org/wiki/Flood_fill
[20] Wikimedia Foundation. (2022, October 7). Multi-agent system. Wikipedia. https://en.wikipedia.org/wiki/Multi-agent_system
[21] YouTube. (2020, June 9). Ai in Unity Turorial. behavior trees. YouTube. https://www.youtube.com/watch?v=F-3nxJ2ANXg&ab_channel=GameDevChef
[22] Izewski, Nate, "A NetLogo COVID-19 Virus Simulation Model for Determining Better Strategies at Handling a Virus Outbreak" (2022). Graduate Academic Symposium. 97. https://scholar.valpo.edu/gas/97
[23] NetLogo: A simple environment for modeling complexity - researchgate. (n.d.). https://www.researchgate.net/publication/230818221_NetLogo_A_simple_environment_for_modeling_complexity
[24] Zarembo, I., & Kodors, S. (2015). Pathfinding algorithm efficiency analysis in 2D GRID. Environment. Technology. Resources. Proceedings of the International Scientific and Practical Conference, 2, 46. https://doi.org/10.17770/etr2013vol2.868
[25] A. Andrade, Game engines: a survey Virtual Campus Lda. Av. Fernão de Magalhães, 716, 1 PT 4350-151, Porto, Portugal – http://virtual-campus.eu

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Michal Horák**
Studium:	I2000041
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Herní enginy pro simulaci a analýzu komplexních systémů**
Název diplomové práce AJ:	Game engines for simulation and analysis of complex systems

Cíl, metody, literatura, předpoklady:

Cíl: Navrhnout a ověřit využití herních enginů pro simulační účely komplexních systémů.

Osnova:

1) Teoretické základy komplexních systémů

2) Popis vybraných herních enginů

3) Praktický projekt v enginu Unity

4) Zhodnocení výsledků

ALEXANDRU, Catalin (ed.). *Modeling and simulation in engineering*. BoD–Books on Demand, 2012.

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Karel Mls, Ph.D.
Oponent:	Ing. Milan Kořínek

Datum zadání závěrečné práce: 9.9.2021