UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Umělá inteligence v PC hrách a její modelování

Jan Januš

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan Januš**
Osobní číslo:	**I09137**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Umělá inteligence v PC hrách a její modelování**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Autor představí základní problematiku umělé inteligence a její propojení do počítačových her. V praktické části představí vybrané editory pro umělou inteligenci a na konkrétní mapě představí metody učení a využívání umělé inteligence v řízení a chování postav.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

***FUNGE, John David. Artificial intelligence for computer games: an introduction. Wellesley, Mass.: Peters, c2004, x, 146 p. ISBN 15-688-1208-6.**
***BUCKLAND, Mat. Programming game AI by example. 1. vyd. Texas: Wordware Publishing, 2005, 495 s. ISBN 15-562-2078-2**

Vedoucí bakalářské práce: **Ing. Soňa Neradová**
Katedra softwarových technologií

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 5. 5. 2013 Jan Januš

Poděkování

Na tomto místě bych velice rád poděkoval své vedoucí práce paní Ing. Soně Neradové za její ochotu a přínosné rady, díky kterým se podařilo bakalářskou práci realizovat. Dále bych chtěl poděkovat celé své rodině za podporu v průběhu studia.

Anotace

Práce představuje základní principy použití umělé inteligence (UI) při vytváření PC her. Teoretická část je rozdělena na tři části: Vysvětlení základních pojmů z této oblasti, dělení UI z hlediska herních žánrů a nakonec na popis mechanismů UI v konkrétní hře. V praktické části je představen editor na tvorbu UI, ve kterém je vymodelována mapa s určitou UI postav, které se na ní pohybují.

Klíčová slova

UI, umělá inteligence, PC hra, modelování, A* algoritmus

Title

Modeling of Artificial Intelligence in PC Games

Annotation

This thesis introduces the basic principles of using artificial intelligence (AI) in creating PC games. The theoretical part is divided into three parts: An explanation of basic concepts, dividing AI in terms of game genres and finally a description of AI mechanisms in a specific game. In the practical part an AI editor is introduced. There is a modelled map, with a set of AI characters, created in this editor. These characters move on the map.

Keywords

AI, artificial intelligence, PC game, modeling, A* algorithm

Obsah

Seznam zkratek

UI	Umělá inteligence
AI	Artificial Intelligence
FPS	First Person Shooter, Frames Per Second
TPS	Third Person Shooter
3D	3 Dimensional
A*	A Star
TBA*	Time Bounded A Star
PR LRTA*	Path Refinement Learning Real-Time A Star
D LRTA*	Dynamic Learning Real-Time A Star
kNN LRTA*	k Nearest Neighbors Learning Real-Time A Star
F.E.A.R.	First Encounter Assault and Recon
FSM	Finite-State Machine
NPC	Non-Player Character
STRIPS	Stanford Research Institute Problem Solver
NOLF	No One Lives Forever
GOAP	Goal-Oriented Action Planning
MMOGs	Massively Multi-Player Online Games
RTS	Real-Time Strategy
RPG	Role-Playing Game
PC	Personal Computer

Seznam obrázků

Úvod

Umělá inteligence v počítačových hrách (dále jen UI v PC hrách) je důležitým aspektem kvalitních her. Výrazně ovlivňuje případný úspěch, či neúspěch titulů na trhu. Správně vytvořená UI v PC hrách by měla být schopna poskytnout hráčům adekvátní výzvu při prozkoumávání virtuálního světa.

Hlavním cílem této práce je přiblížení principů UI, která se používá v moderních a úspěšných PC hrách. Jaké jsou metody její tvorby a jaká úskalí musí vývojáři při její tvorbě zdolat. Důraz je kladen na praktické vyzkoušení a otestování těchto principů v daném editoru a snaha o to, vymodelovat UI co nejvíce podobnou skutečnosti.

Práce je rozdělena na teoretickou a praktickou část. V teoretické části je věnována pozornost představení základních principů tvorby UI v PC hrách, kde je také mimo jiné popsáno použití oboru Teorie grafů v oblasti pohybu herní UI. Dále je v této části představeno použití systémů konečných stavů, Fuzzy logiky a neuronových sítí v rozsahu chování umělé inteligence v PC hrách. V další kapitole je rozdělení UI dle herních žánrů. Poslední kapitola teoretické části se věnuje popisu mechanismů UI v konkrétní PC hře. V praktické části je představen editor na modelování map společně s UI.

Dané téma bylo zvoleno pro nadšení z umělé inteligence jako takové. UI je obecně chápána jako soubor systémů, které při řešení daného úkolu užívají podobné postupy, jaké by při řešení stejných činnostech používal člověk. Tudíž se snaží přiblížit realitě. Herní UI je odvětví umělé inteligence, které si klade za cíl právě co nejvíce přiblížit počítačové hry, skrze UI použitých systémů, realitě.

1 Principy herní UI

„Je to jeden z nejdůležitějších stavebních prvků každé dobré hry. Na první pohled je ovšem téměř neviditelný a bývá tedy často opomíjen. Čím je navíc složitější, tím hrozí autorům větší nebezpečí, že se obrátí proti nim.“ (Vávra, 2011)

UI jakožto umělá inteligence obecně, je schopnost počítače jednat jako člověk. To znamená, aby se počítač choval co nejpřirozeněji a aby navozoval dojem lidské bytosti (TechTerms, 2010). Tento princip umělé inteligence se tedy neliší, pokud se jedná o herní UI, nebo UI používanou v robotice, či jinde. UI si vždycky klade za cíl co nejvíce se přiblížit lidskému chování, a nahradit tak člověka v požadovaných úkonech ať je to v jakémkoliv oboru.

1.1 Rozdíl mezi herní a lidskou inteligencí

Důležité je si hned vyjasnit, že herní UI se od té lidské velice liší. Aby se dala nazývat skutečnou inteligencí, musela by být schopna se učit z různých situací, zkrátka přemýšlet o tom co dělá jako člověk. To je nejdůležitější věc, která UI ještě chybí. Dalo by se říci, že herní UI nikdy nebude chytřejší než člověk, který ji vytvořil. Ovšem chovat se a jednat přirozeně, tak jak by to udělal člověk, je hlavním cílem vývojářů kteří danou UI vytvářejí. Snaží se tedy co nejvíce nabýt dojmu, že spoluhráč či nepřítel je živá bytost, která se umí inteligentně pohybovat po určitém prostoru, schovávat se za překážky, spolupracovat se svými spolubojovníky, zintenzivnit hráčův zážitek a navodit tak tu správnou atmosféru dané hry. Špatná UI, která by neposkytnula hráči určitou výzvu, by se neblaze odrazila ve finální prodejnosti určité hry, tudíž tento aspekt vývojáři nemohou opomíjet.

(Vávra, 2011)

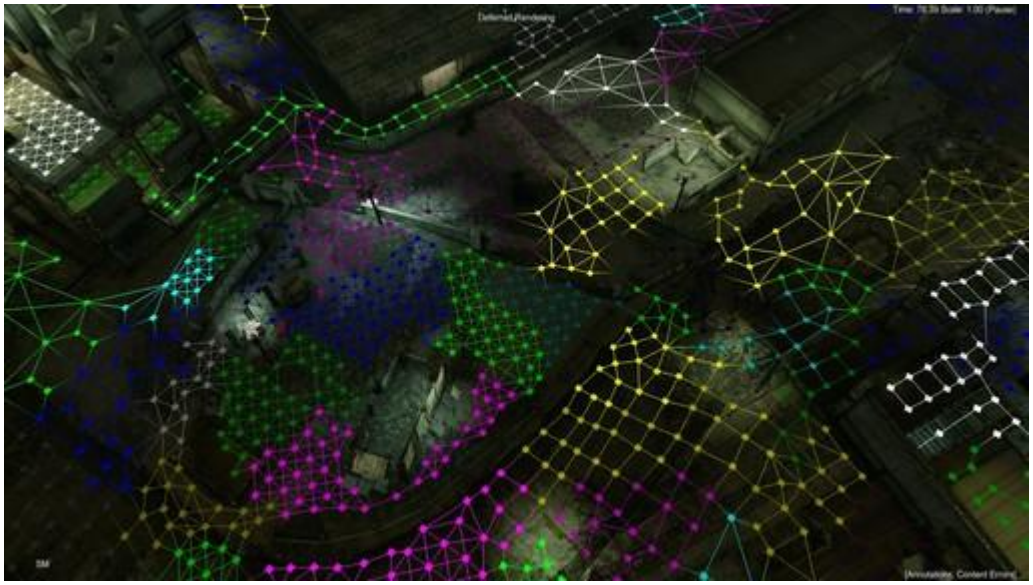
1.2 Pohyb v prostoru

Od každé kvalitní UI lze očekávat, že se bude schopna inteligentně pohybovat v prostoru. K navigaci kam má jít, se využívá metoda zvaná Pathfinding, což je metoda hledání nejvhodnější cesty k cíli. Pathfinding využívá pro nalezení správné cesty heuristické procházení grafu, přičemž tento graf sestává z tzv. Waypointů nebo Navigačních sítí, kterými je konkrétní plocha, kde se může UI postava pohybovat, poseta.

(Hruška, 2008)

1.2.1 Waypointy

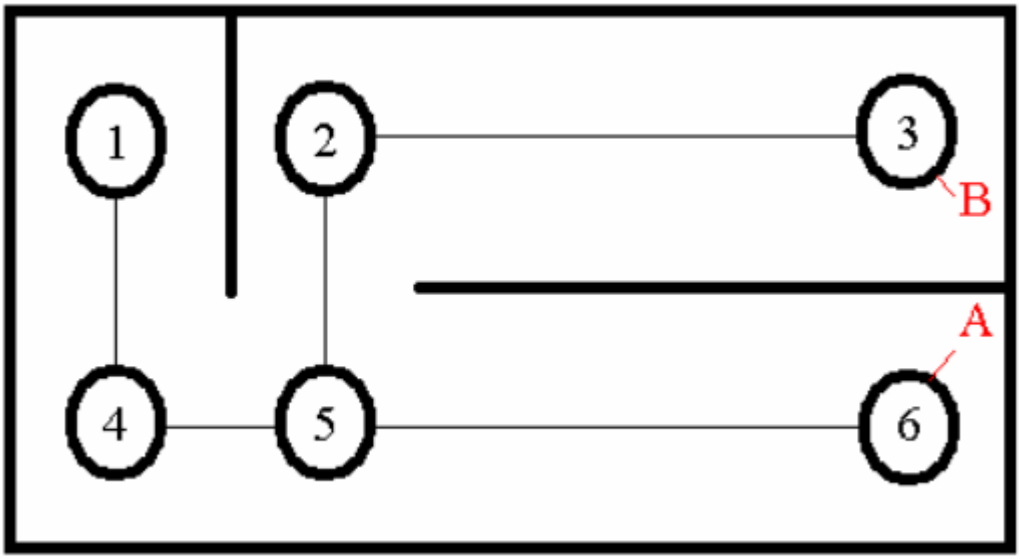
Jsou to body na mapě, nejčastěji propojené čarami a tvořící tak celou síť. Waypointy mohou být tvořeny ručně přímo vývojáři daného produktu, ovšem dnes již existují technologie na automatické počítání těchto pomyslných cest. Počítačem řízená postava se tak pohybuje po těchto cestách, dokud se nedostane ke kýženému cíli (Vávra, 2011).



Obrázek 1 – Síť Waypointů ve hře Killzone 2

Zdroj: (Champandard, 2011)

UI postava, která se má dostat z bodu A do bodu B, jak je znázorněno na obrázku (Obrázek 2). Postupuje po Waypointech, přičemž na každém z nich se rozhoduje, na jaký bod se přesunout dále, pomocí algoritmů Pathfindingu (Graham, a další, 2003).

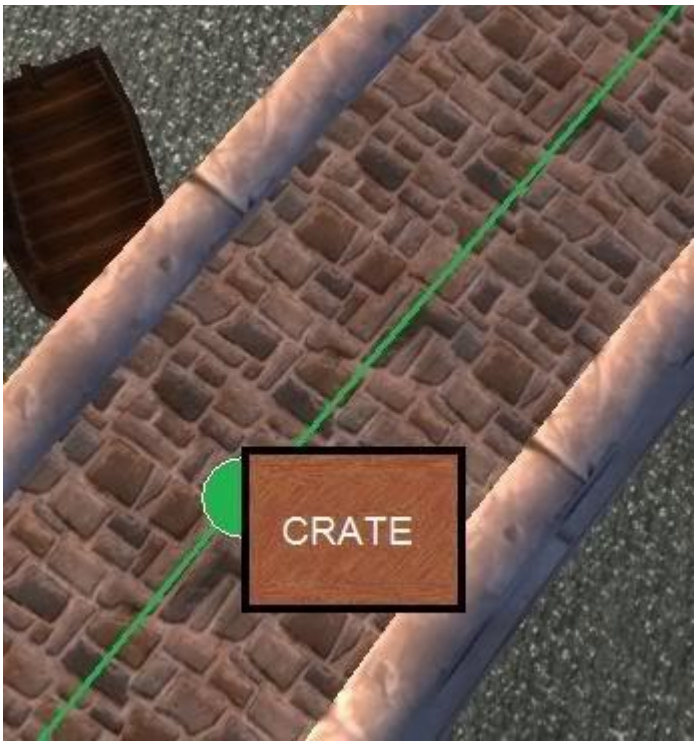


Obrázek 2 – Ukázka Waypointů

Zdroj: (Graham, a další, 2003)

Waypointy nacházejí své užití především na statických menších mapách, kde se nevyskytuje mnoho překážek nebo při užití skriptů. V dynamických mapách kde je hojnost výskytu překážek velká, a je zde také velká míra interakce s předměty, není jejich použití vhodné (Graham, a další, 2003). Tyto mapy představují riziko zastavení UI postavy o předmět, který je v jeho cestě a původně tam být neměl. Lze si představit situaci, kdy hráč

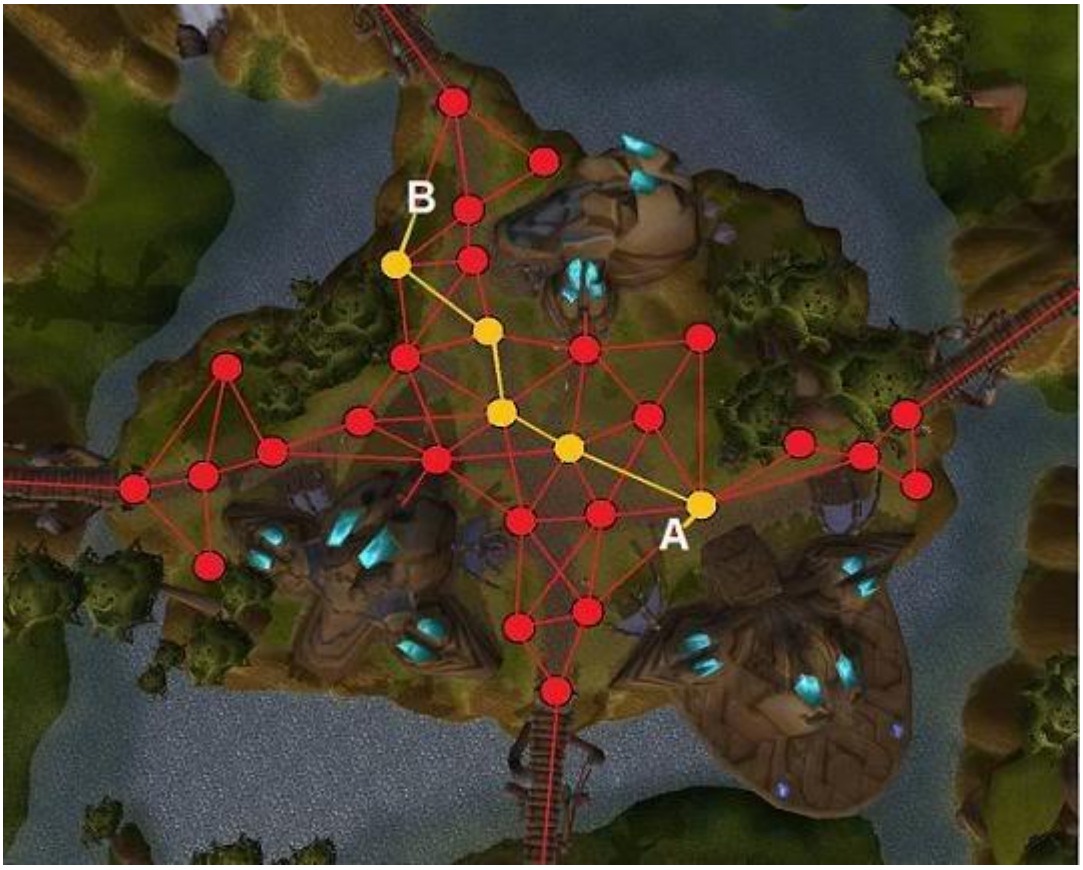
v interaktivní mapě s propracovanou fyzikou přesune těžkou bednu do cesty UI postavě, která ho má následovat (Obrázek 3). V nejhorším případě, se UI postava o bednu zastaví a přestane hráče následovat. V případě použití fyziky by tlačila bednu před sebou, protože by nemohla nalézt jinou vhodnou cestu k obejití zmíněné překážky. Toho lze docílit přidáním více Waypointů, za účelem nalezení vhodnější cesty okolo překážky. Ovšem vývojáři tyto situace nikdy nedokážou předvídat a z toho důvodu je použití Waypointů na těchto mapách zcela nevhodné.



Obrázek 3 – Překážka na waypointové cestě

Zdroj: (PaulT, 2008)

Další nevýhodou Waypointů je občasný trhaný pohyb UI postav. Na mapě pokryté Waypointy se snaží postava dostat z místa A do místa B. K dispozici má celou síť navigačních bodů. Ovšem tyto body nemusí být vždy umístěny v přímce tak, aby postava nemusela za svoji trasu k cíli změnit ani jednou směr, jak je vidět na obrázku (Obrázek 4). Ve výsledku to vypadá, jakoby se postava na každém bodě pootočila a nešla rovnou za cílem, ale někam jinam, i když cíle nakonec dosáhne (PaulT, 2008).

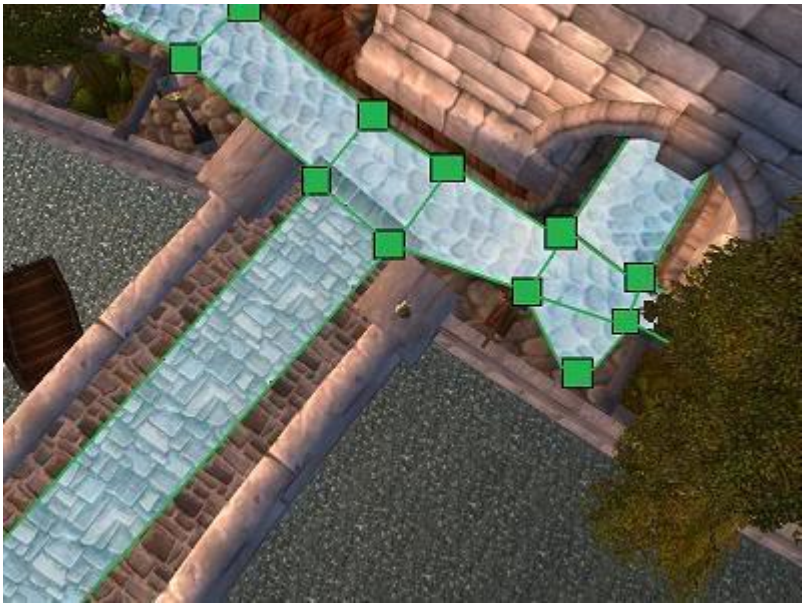


Obrázek 4 – Navigace z místa A do B za pomocí Waypointů

Zdroj: (PaulT, 2008)

1.2.2 Navigační síťě

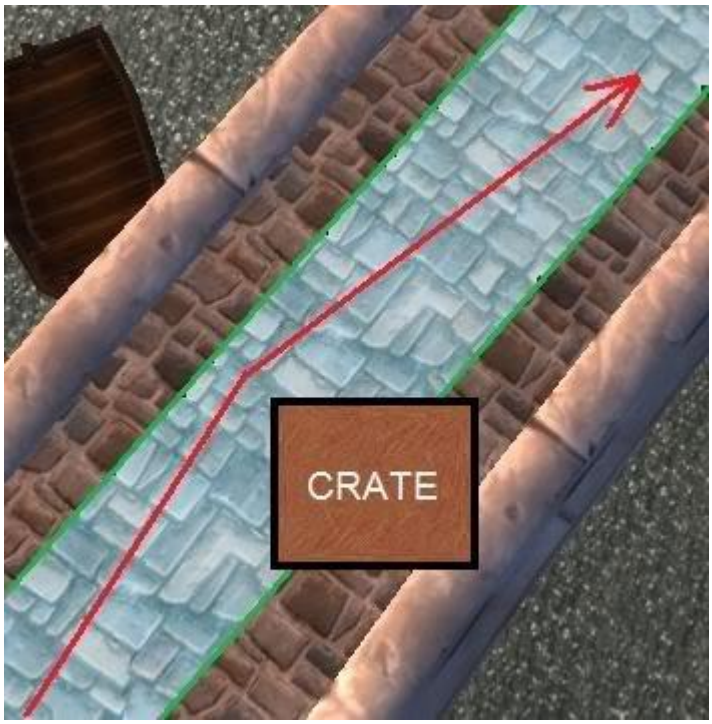
Navigační sítě jsou skupiny konvexních polygonů, které pokrývají povrch mapy nebo různých 3D objektů a určují tak, kde se mohou UI postavy pohybovat. Polygony se počítají automaticky, v závislosti na oblasti působnosti, a každý z nich reprezentuje bod pro algoritmy Pathfindingu (Graham, a další, 2003). Ukázka takové sítě je znázorněna na obrázku (Obrázek 5).



Obrázek 5 – Ukázka navigační sítě

Zdroj: (PaulT, 2008)

Odpadají zde nedokonalosti Waypointů jako trhaný pohyb, či neschopnost obejít překážku. Při užití navigační sítě UI postava zkrátka překážku obejde, protože má i v okolí překážky určený povrch, kde se může pohybovat.



Obrázek 6 – Překážka v navigační síti

Zdroj: (PaulT, 2008)

Jak již bylo zmíněno, trhaný pohyb se nevyskytuje, jelikož lze dva body na navigační síti spojit přímkou. UI postava potom dosáhne cíle přímo, bez nutnosti zbytečné změny směru. Cesta nemá charakter křivky, jak tomu bylo u waypointové logiky.

Obrázek 7 – Navigace z místa A do B za pomocí navigační sítě

Zdroj: (PaulT, 2008)

Ve výsledku se navigační síť doplní o speciální body jak je vidět na obrázku (Obrázek 8). Ty rozmístí vývojáři a v podstatě se navigační síť stará o Pathfinding, zatímco body určí UI činnost, kterou má vykonat (PaulT, 2008). Například bude muset držet hlídku mezi dvěma body, bude se muset přikrčit za překážkou ke znesnadnění své vlastní likvidace hráčem, či provádět mnoho dalších činností.

Obrázek 8 – Navigační síť doplněná o speciální body

Zdroj: (PaulT, 2008)

1.2.3 Algoritmy Pathfindingu

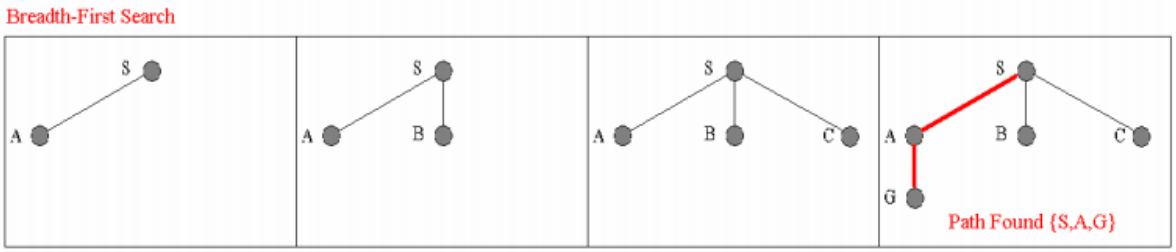
Algoritmy hledání vhodné cesty, tedy algoritmy, které využívá Pathfinding mohou být rozděleny na dvě hlavní kategorie, přičemž obě jsou založeny na procházení grafu (Graham, a další, 2003). Graf v logice herní UI představuje již zmíněné Waypointy, či navigační sítě. UI, která se chce dostat z jednoho bodu do druhého, zjišťuje nejvhodnější cestu k cíli právě těmito algoritmy.

Neřízené algoritmy:

Tento postup lze přirovnat k myši, běhající v bludišti a snažící se najít cestu ven. Myš se nezaobírá plánováním další cesty, nýbrž zkouší všechny možné, i když většinou slepé uličky k nalezení cíle. Pro potřeby herní UI, kde je velký důraz kladen na chování co nejvíce se přibližující skutečnosti, je tento způsob nevhodný. Nicméně může být použit v kombinaci s řízeným algoritmem. Neřízený algoritmus rychle pohybuje UI, zatímco řízený na pozadí hledá vhodnější cestu kam se ubírat dále (Graham, a další, 2003).

Nejvýznamnější algoritmy z kategorie neřízených algoritmů vyhledávání jsou *Breadth-first* a *Depth-first* (Graham, a další, 2003).

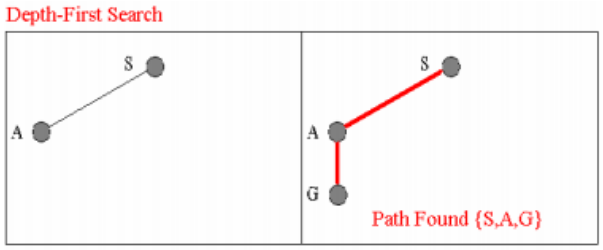
Breadth-first algoritmus prohledává postupně všechny body, přímo napojené na aktuální bod. V případě nenalezení cíle se algoritmus přesune na tyto, v předešlém kroku prohledávané body, a opakuje proces pro každý z nich, dokud nenalezne cílový bod (Graham, a další, 2003). Prohledané body se umístí do seznamu CLOSED, zatímco body určené v daném kroku k prohledání jsou v seznamu OPEN (Hruška, 2008). Tímto se zamezí znovu procházení bodů jednou již navštívených. Tento algoritmus prohledává graf do šířky.



Obrázek 9 – Breadth-first algoritmus

Zdroj: (Graham, a další, 2003)

Depth-first algoritmus naproti tomu prohledává graf do hloubky. Nejdříve tedy prohledá potomka aktuálního bodu, ovšem neprohledá všechny potomky, ale drží se linie onoho jediného a prohledává opakovaně jeho potomky do hloubky grafu. Až v případě nalezení slepé uličky, se vrací na začátek a proces opakuje pro dalšího potomka úplně prvního bodu. Tento algoritmus má dobrou šanci najít vhodnou cestu již po prohledání malého prostoru z celkového grafu, tudíž je využívanější než algoritmus *Breadth-first* (Graham, a další, 2003).



Obrázek 10 – Depth-first algoritmus

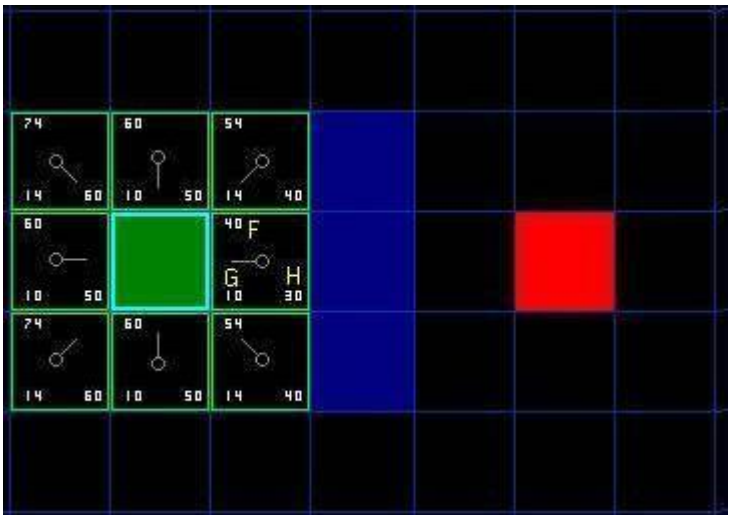
Zdroj: (Graham, a další, 2003)

Řízené algoritmy:

Všechny řízené algoritmy Pathfindingu mají jednu věc společnou. Neprocházejí slepě celý graf, ale rozhodují se, na který bod se vydat, v závislosti na ohodnocení daných cest. Za pomoci tohoto ohodnocení dokáže algoritmus najít nejoptimálnější cestu k cíli (Graham, a další, 2003).

Nejvyužívanější řízený algoritmus pro potřeby Pathfindingu v PC hrách se nazývá *A* (A star)*.

*A** algoritmus nejprve umístí počáteční bod hledání do seznamu OPEN. Dále umístí do seznamu OPEN také body, které jsou jeho přímými potomky (kromě bodů, na které se nedá vstoupit, voda, překážky atd.) a uloží si počáteční bod hledání jako jejich rodiče. Nyní počáteční bod přesune ze seznamu OPEN do CLOSED, protože již není třeba na něj vcházet znovu (Lester, 2005). Tento bod je světle modře ohraničený na obrázku (Obrázek 11).



Obrázek 11 – A* algoritmus výchozí stav

Zdroj: (Lester, 2005)

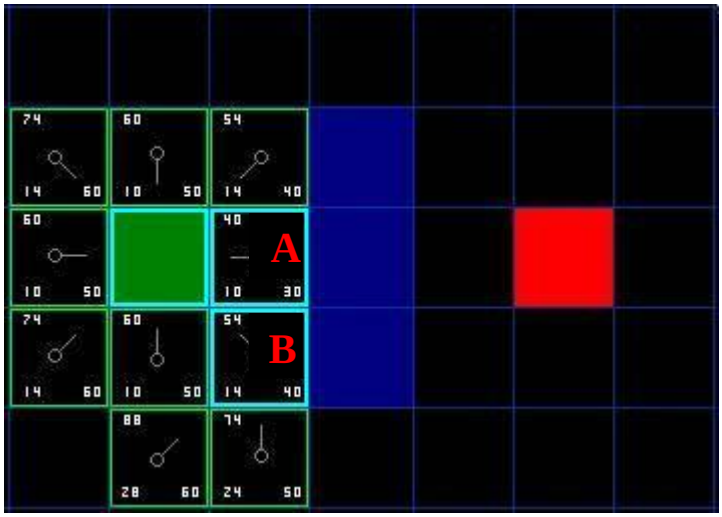
V této chvíli se algoritmus nachází ve výchozím stavu pro další rozhodování, na jaký bod z těch, které má v seznamu OPEN vstoupit. Rozhodne se na základě rovnice:

$$F(n) = G(n) + H(n)$$

kde $G(n)$ – je hodnota, vyjadřující délku mezi počátečním bodem a potencionálním dalším bodem v cestě, tj. konkrétním bodem označeným písmenem n. $H(n)$ – je odhadovaná vzdálenost mezi konkrétním bodem n a cílovým bodem. Tato vzdálenost je brána vzdušnou čarou, nejsou tak brány v potaz překážky. Této hodnotě se také říká heuristika, protože se jedná o odhad.

(Lester, 2005)

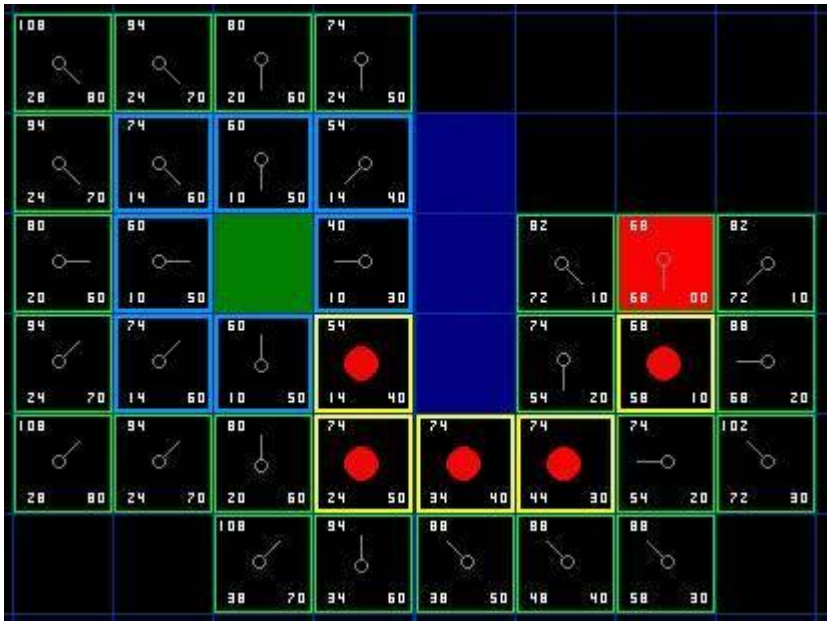
Bod, který má hodnotu F nejmenší z bodů v seznamu OPEN, je vybrán jako další bod v cestě a následně vložen do seznamu CLOSED. Dále se zkontrolují všechny jeho přilehlé body kromě těch, na které se nedá vstoupit nebo těch, které jsou umístěny v seznamu CLOSED. Pokud nově kontrolované body nejsou v seznamu OPEN, vloží se do něho a jako rodič se jim nastaví aktuální prvek s onou nejmenší hodnotou F. V případě, že nějaký přilehlý bod je v OPEN seznamu obsažen, je nutné pro tento bod zkontrolovat, zda je jeho hodnota G menší, než G hodnota při použití aktuálního bodu. Tento problém je znázorněn na obrázku (Obrázek 12). Bod A je aktuální a bod B je ze seznamu OPEN kontrolován. Jak lze vidět, pokud by byla cesta vedena skrze bod A, součet G hodnot při dosažení bodu B by byl 20 (pohyb horizontálně vpravo a vertikálně dolů tzn. 10 + 10 = 20), nicméně v případě vedení cesty rovnou na bod B, by byla hodnota G pouze 14 a tudíž menší. Pokud tedy bude hodnota G kontrolovaného bodu menší, tento bod se uloží jako rodič, přepočítají se hodnoty F a G a umístí se do seznamu CLOSED (Lester, 2005).



Obrázek 12 – A* algoritmus průběh

Zdroj: (Lester, 2005)

V této chvíli se proces opakuje, dokud se neumístí do seznamu CLOSED cílový prvek. Poté se postupuje zpět po rodičích a tímto postupem se vyznačí vhodná cesta (Lester, 2005).



Obrázek 13 – A* algoritmus výsledek

Zdroj: (Lester, 2005)

Algoritmus *A** však není jediný, který se dnes používá, pokud se jedná o herní Pathfinding. Existuje celá řada dalších algoritmů, nicméně se povětšinou jedná o modifikace již zmíněného *A**. Není ovšem na škodu o nich vědět. Jsou to algoritmy jako například *TBA**, *PR LRTA**, *D LRTA** a *kNN LRTA** (Gonzáles-Calero, a další, 2011).

1.3 Chování UI postav

1.3.1 FSM

FSM je systém konečného počtu stavů. Dá se přirovnat k určitému objektu z reálného světa. Například turnikety na letištích, či v metru jsou jednoduché FSM. Mají dva stavy a to zamčeno a odemčeno. Lidé mohou pomocí akce vhození mince přepnout turniket z jednoho stavu do druhého.

Tyto systémy se používají v herní UI, k řízení chování postav. Jednotlivé stavy FSM představují chování dané NPC postavy, které se může měnit v závislosti na určité situaci (Hruška, 2008).

Například UI postava, na kterou je spuštěna palba hráče a jejíž životy klesnou pod určitou mez, přepne své chování (stav FSM) na ukrytí se před palbou. Rozhodne se tedy přepnout stav chování na základě klesajících životů.

Jak již bylo zmíněno výše, FSM tedy řídí chování NPC postav, zatímco algoritmus A* se stará o jejich Pathfinding (Orkin, 2006).

1.3.2 Fuzzy logika

Jedná se o logiku, která je nadstavbou klasické logiky výrokové, ve které může být výrok pravda nebo nepravda (True nebo False). Ve Fuzzy logice existují určité hodnoty mezi těmito dvěma výroky (Hruška, 2008).

V herní UI se Fuzzy logika využívá, společně s FSM. FSM reprezentuje zkrátka logiku rozhodování se podle dvou výroků a to ANO a NE. Fuzzy logika tak dává UI postavě více možností v rámci jejího chování.

NPC postavy tak mohou pomocí Fuzzy logiky určovat, do jaké míry chtějí vykonávat konkrétní akce. Rozhodují se na základě Fuzzy hodnoty. Čím je tato hodnota vyšší, tím je i vyšší touha po splnění určité akce a naopak. Tato hodnota může mít i vliv na přepnutí stavů FSM (Hruška, 2008).

Jako příklad lze uvést UI postavu, která střílí po hráči a docházejí jí tak náboje. Čím méně jich má, tím je touha po získání dalších větší (Hruška, 2008).

1.3.3 Neuronové sítě

Neuronová síť v umělé inteligenci představuje výpočetní model, který si bere za vzor chování určitých biologických struktur, jako je například lidské myšlení. Neuronové sítě jsou složeny z neuronů, které si vzájemně mezi sebou předávají signály (Hruška, 2008).

V herní UI se neuronové sítě používají k učení NPC postav různým situacím, které se ukládají do určitého souboru. Jako příklad lze uvést UI postavu, která se přesouvá na určité místo. Postava přejde velkou překážku, za kterou je schován hráč, bez povšimnutí a ten nic netušící UI postavu napadne zezadu. Tato situace, kdy se hráč schovává za překážkou, je

zapsána do souboru. Při opětovném sehrání této situace, již UI postava ví, že se za překážku má podívat (Hruška, 2008).

1.4 Zjištění nepřítele

„Kvalita UI se většinou v plné kráse projeví teprve v boji. Stěžejní roli v tom, aby vypadal boj, co nejpřirozeněji, však hrají „smysly" umělé inteligence, tedy zrak a sluch." (*Vávra, 2011*)

1.4.1 Zrak

Jak docílit toho, aby UI postava hráče viděla a reagovala na něho určitou akcí a naopak neviděla, pokud doopravdy vidět není? Tento problém je řešený způsobem neviditelné čáry, která je vysílána z hlavy UI postavy mnohokrát za vteřinu ve směru jejího pohledu. Potom se už jen stačí jednoduše rozhodnout. Pokud čára narazí na hráče, UI postava ho zaznamená, pokud narazí na překážku, tak nikoliv. V případě, že je hráči vidět pouze končetina a zbytek těla jeho virtuálního já je schovaný za překážkou, by tedy UI postava nezareagovala a to je z hlediska přiblížení se co nejvíce reálnému chování špatně. Proto se nejdříve vykreslí čára např. k trupu hráče, pokud je za překážkou, vykresluje se ke končetinám. Celý výpočet čar je velmi náročný na výkon počítače, jelikož jich narůstá s přibývajícími postavami na mapě.

(Vávra, 2011)

Problém může nastat ve hrách, kde jsou důležitou součástí hratelnosti stíny a celkové nasvícení. V takových hrách by UI postava neměla zaznamenat hráče, který je nejen za překážkou, ale také ve stínu. Tento případ se vyřeší podmínkou, pokud je na hráčovu postavu vrženo množství světla nad nějakou určitou mez, je zaznamenán. V opačném případě nikoliv.

(Vávra, 2011)

„Největší masakr je, když hráč sice stojí ve stínu, ale za ním se nachází světlo, takže ve skutečnosti je zcela jasně viditelná jeho silueta. S touhle situací si dodnes většina her moc poradit neumí." (Vávra, 2011)

1.4.2 Sluch

Další věc, která je důležitá pro správné, reálné chování UI je její sluch. Jinými slovy zvuky, které UI postavy přinutí zaznamenat a reagovat na hráče. Pokud se UI nachází na stejné mapě s hráčem bez značných překážek, stačí pouze vyřešit vzdálenost, na kterou je daný zvuk slyšet. Obtížnější ovšem je, pokud se mapa, po které se hráč s UI postavami nachází, skládá z různých místností a velkých překážek, které by mohly jakkoliv utlumit zvuk vydaný hráčem. V tomto případě se musí brát v potaz reálné šíření zvuku a jeho výpočet implementovat dané UI. Nicméně tento proces by byl natolik složitý a jeho efektivita v podobě postřehnutí této realističnosti hráčem tak malá, že se vývojáři spokojí s jednoduchou detekcí zvuku. V praxi to vypadá tak, že hráč vystřelí, UI detekuje zvuk,

ovšem po hráči půjdou jen ty postavy, které mu jsou na blízku, tudíž např. z jedné místnosti.

(Vávra, 2011)

1.5 Skripty

Skripty, což je v podstatě přesně dané chování UI, jsou dnes hojně využívané. Vývojáři mnohokrát musí UI přímo sdělit, co má v daném okamžiku dělat (Vávra, 2011).

Situace, kdy má např. postava řízená UI hráče někam dovést, či s ním promluvit v daném určitém okamžiku je do detailu naskriptovaná. Vývojáři naprogramovali přesné chování oné postavy, tudíž má přesně vytyčenou Waypointovou cestu kudy dojít k cíli. Pokud se hráč někde opomene, UI postava na něj zavolá. Všechno to připomíná předem připravené divadlo.

(Vávra, 2011)

Ve velkém množství dnešních FPS jsou skripty hojně využívány. Hráč se pohybuje po mapě bez velké volnosti a vše okolo něho je pevně dané. V praxi to vypadá tak, že hráč dojde na mapě k určitému bodu, či zmáčkne nějaké tlačítko a spustí se skript. Tento skript sestává z různých akcí. V již zmíněných FPS se např. zjeví nepřátelské UI postavy a začnou se schovávat a střílet po hráči. Jejich zjevování případně ustane, pokud hráč provede určitou činnost, jako umístí nálož atp. Následně mu je umožněna cesta dál, kde na něho čeká další skript. UI se v těchto případech omezuje pouze na vyčkávání, až hráč konkrétní skript spustí.

(Vávra, 2011)

UI se pohybuje pouze po daných trasách, tudíž odpadá práce s Waypointovou sítí a omezí se jen na pouhé schování za překážku a střílení po hráči. Na první pohled to však vypadá, že se UI chová inteligentně, nicméně pokud se jí podaří hráče zastřelit, je hráč povětšinou nucen projít tento skript znovu a UI postavy se chovají naprosto stejně jako při předešlém pokusu. Navíc hráč nemůže udělat nic, čím by skript překazil, kromě splnění předem daného úkolu např. již zmíněného umístění nálože.

I přes všechny nedostatky mají skripty své opodstatnění. NPC postavy FPS žánru zkrátka nežijí tak dlouho, aby si hráč všiml nějakého markantního rozdílu mezi živou inteligencí a UI, a o to přeci jde. Z důvodu krátké životnosti těchto postav by bylo i neefektivní, kdyby pro jejich pohyb vývojáři tvořili celou navigační nebo Waypointovou síť.

Příklad FPS, kde jsou skripty nadmíru využívány je celá série Call of Duty.

2 Typy UI z hlediska herních žánrů

2.1 Akční hry

Akční hry jsou klasické hry, kde je zjednodušeně hlavním úkolem střílet, sekat, mlátit do NPC protivníků (pokud se nejedná o hraní přes internet proti lidským hráčům) s určitou UI. Pro zajímavost, tento typ her patří k nejvíce výdělečným, co se finanční stránky týče. Je zkrátka pro svou přímočarost velice oblíben, avšak aspekt špatné UI v takové hře, ji dokáže výrazně znehodnotit a naopak (Millington Ian, 2009).

Mohou být rozděleny z hlediska pohledu, kde je umístěna kamera hráče. Buď hráč vnímá virtuální dění kolem sebe přímo z očí postavy, za kterou hraje tj. pohled z první osoby (FPS) nebo jsou pomyslné oči hráče zasazeny do určité vzdálenosti za postavou tj. pohled z třetí osoby (TPS).

Nejvýznamnější požadavky na UI v akčních hrách jsou:

- **Pohyb** – počítačem řízené postavy by se měly umět pohybovat po určité mapě, s tímto souvisí Pathfinding
- **Útok (střelba)** – schopnost rychlého a přesného míření, v rámci obtížnosti jakou si hráč zvolí
- **Umění rozhodovat se** – tímto je myšleno např. nabíjení zbraně při docházení munice, zkrátka jednoduché úkony spojené s konkrétní postavou
- **Vnímavost** – vnímání odlišnosti mezi nepřítelem a spoluhráčem, na koho zaútočit a na koho ne
- **Pathfinding** – schopnost plánování vhodné cesty na grafu, ať už waypointovém, či na navigační síti
- **Taktické myšlení** – používá se např. k nalezení krytu před střelbou, či vpádu nepříteli do zad

(Millington Ian, 2009)

Nejvýznamnějším zástupcem tohoto žánru je například série Doom, která odstartovala vlnu FPS her. Ze současných zástupců lze jmenovat FPS F.E.A.R., kterému je věnována celá kapitola (UI ve hře F.E.A.R.), a novější série Crysis. Z TPS žánru je nutné zmínit sérii Assassin's Creed a Tomb Raider.

Do této skupiny také patří Adventury a hry typu MMOGs, což jsou hry hrající se online s lidskými spoluhráči. Tyto hry ovšem nepotřebují propracovanou UI a spokojí se většinou s jejími základy (Millington Ian, 2009). MMOGs většinou potřebují UI pouze na to, aby kontrolovala různé NPC postavy a ty se uměly pohybovat a zaútočit na hráče. O žádném složitějším rozhodování zde nemůže být řeč.

Podobné klasickým FPS hrám, jsou také hry RPG. V nich hráč kontroluje své virtuální já také z pohledu první nebo třetí osoby podobně jako je tomu u FPS. Rozdíl tkví ve snaze se více přiblížit realitě v RPG hrách. Hráče se tyto hry snaží více vtáhnout do děje tím, že jak postupuje rozlehlým světem, dostává zkušenosti a může tak vylepšovat nějaké své dovednosti. Tyto hry se pyšní povětšinou velkou volností pohybu, nicméně nenabízejí tolik akce. Je to jakýsi simulátor života ve velmi nadnesené formě.

2.2 Závodní hry

Typ závodních her, může být jakýmsi simulátorem řízení vozidla v reálném světě nebo se naopak nemusí vůbec přibližovat realitě. Nicméně z hlediska UI je důraz kladen hlavně na pohyb. Jako v každé hře, hráč vyžaduje určitou výzvu. Hra by zřejmě nebavila, pokud by vozidla ovládány UI postrádaly jakousi dravost a nechaly pokaždé hráče dojet na první pozici bez sebemenší námahy.

Pohyb po cestě, či silnici je zajištěn pomocí waypointových křivek, po kterých se UI vozidla pohybují. Těmto křivkám jsou přiřazena speciální data, podle kterých vozidla přizpůsobují rychlost v zatáčkách, či podle situace. Nutností je také implementovat kolizní systém, díky němuž se vozidla snaží vyhýbat. Dále je implementováno jednoduché chování při vystrčení vozidla mimo danou křivku pohybu. V takovém případě by se mělo být schopno vrátit na vozovku a pokračovat v jízdě (Millington Ian, 2009).

Snad nejznámější série závodních her je Need for Speed. Tato série začala již před dlouhou dobou, nicméně nové díly jsou vydávány dodnes. Need for Speed ovšem nepatří mezi úplně reálné simulátory a jízdní model je přikloněn spíše k hratelnosti než k realitě. Mezi velmi povedené a zároveň realistické závodní hry patří Richard Burns Rally nebo série her Toca Race Driver.

2.3 Real-Time strategie

RTS jsou hry, kde je hráč zasazen do jakési role vůdce národu, jednotky, rasy, zkrátka čehokoliv. Podstatou ovšem je, že je kamera hráče zasazená tak, aby viděl mapu z výšky. Je tomu tak, protože v tomto typu her hráč musí ovládat skupiny NPC postav, stavět svá města, či základny a starat se o jejich chod. Celý tento systém by nešel provést jinak, než když bude hráči poskytnut co možná největší přehled o dění na mapě.

Nejvýznamnější požadavky na UI v RTS hrách jsou:

- **Pathfinding** – schopnost nalézt vhodnou cestu na mapě, pokud hráč pošle jednotku na určité místo, Pathfinding ručí za to, že na místo dorazí, a že se nezastaví někde po cestě
- **Pohyb skupin** – možnost uskupovat vybrané skupiny do formací a schopnost se v těchto formacích pohybovat

- **Taktická a strategická UI** – RTS jsou o strategii, UI v těchto hrách musí mít taktické a strategické myšlení, aby byla hráči pravou výzvou
- **Umění rozhodovat se** – každý druh jednotky (lučištník, rytíř) má svůj vlastní systém rozhodování

(Millington Ian, 2009)

Mezi velice významné RTS série patří bezesporu Warcraft a Starcraft od studia Blizzard, ovšem předkem všech dnešních RTS her je stařičká Dune 2.

2.4 Sportovní hry

Sportovní hry jsou založeny hlavně na týmové spolupráci. Z toho také vychází UI. Hráčovi spoluhráči mu musejí být nápomocní a musejí dodržovat správné rozestavění. Samozřejmě zde musí být zakomponována věc náhody, aby se občas stalo, že se vyskytne mezera v obraně a tak dále. To vše závisí na obtížnosti, kterou hráč zvolí, či na kvalitách týmu, za který hraje. Většinou se jedná o hry jako hokej, fotbal a basketbal. Ústřední roli v těchto hrách hraje míč nebo puk. UI musí být schopna částečně předpovídat, kam se puk odrazí, zkrátka počítat s fyzikou onoho hracího předmětu (Millington Ian, 2009).

Sportovní hry jsou série jako například NHL, FIFA a Pro Evolution Soccer.

2.5 Tahové strategie

Žánr tahových strategií se velice podobá klasickým RTS. Rozdíl tkví především v tahovém systému. Opět je kamera umístěna vysoko, aby hráč mohl mít přehled o dění na mapě. Tahové strategie, jak již název napovídá, se hraje na tahy. Hráč má určitý počet tahů, kterými může spravovat svou základnu, pohybovat s jednotkami a tak dále. Po ukončení jeho tahu dojde na tah jeho nepřítele, tedy UI (pokud ovšem nehraje proti lidským protivníkům). Celé to připomíná šachy z reálného světa.

Co se týče UI, je velice podobná RTS. Využívá se Pathfinding, pohyb skupin, taktické a strategické myšlení a umění rozhodovat se (Millington Ian, 2009).

Mezi tahové strategie patří bezesporu série Heros of Might and Magic. Nicméně do tohoto žánru lze zařadit i hry typu Worms.

3 UI ve hře F.E.A.R.

FPS F.E.A.R. je i v dnešní době, což je 8 let od jejího vydání, špičkou ve svém žánru a zapsala se tak do myslí hráčů po celém světě.

Studio Monolith Productions, které je tvůrcem her No One Lives Forever, Aliens vs Predator 2, Condemned: Criminal Origins nebo stařičké FPS Blood (1997) a mnoha dalších, vytvořilo roku 2005 směsici hororu a FPS akce, která nesla právě již zmiňovaný název F.E.A.R.

Tato hra představila naprosto revoluční systém UI, díky němuž se povedlo vytvořit doposud nevídané reálné chování postav. F.E.A.R., z hlediska UI, zkrátka jen těžko hledá konkurenta i přesto, že je relativně dlouho na světě.

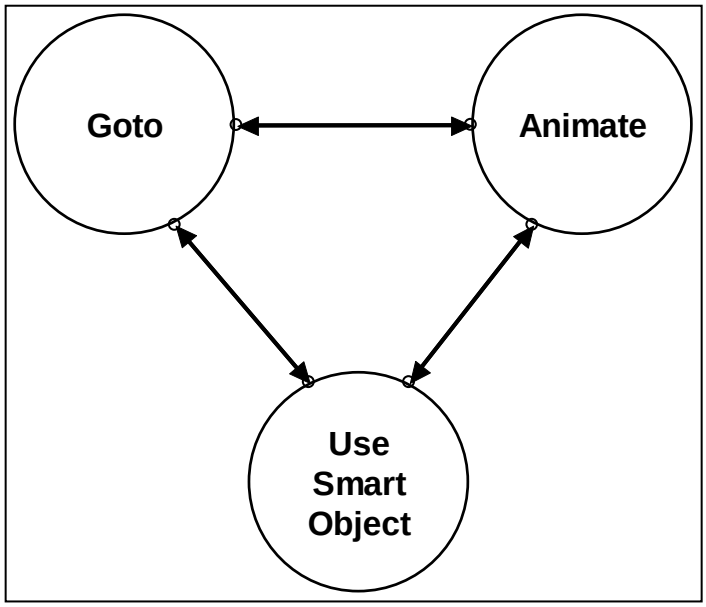
Obrázek 14 – F.E.A.R.

Zdroj: (Orkin, 2006)

UI v dnešních hrách je tvořena za pomocí algoritmu A* a určitého FSM. Ve hře F.E.A.R. je použit FSM pro NPC postavy jen se třemi stavy pro řízení jejich chování, zatímco A* algoritmus plánuje jejich pohyb i posloupnost jejich akcí. Vývojáři zavedli také jakýsi plánovací systém, který nesmírně zlepšuje chování jednotlivých postav a tím ho přibližuje blíže k realitě (Orkin, 2006).

3.1 Tři stavy

Jak je vidět na obrázku (Obrázek 15), vývojáři z Monolith ve svém produktu uplatnili třístavový FSM. Jedná se o stavy *Goto, Animate* a *Use Smart Object*. *Use Smart Object* je v podstatě nadstavbou stavu *Animate,* protože v tomto stavu se skrývá využívání různých objektů virtuálního světa, což je jen a pouze přehrávaní animací ze stavu *Animate*. Stav *Goto* lze chápat také jako blízký stavu *Animate*, jelikož postava chtějíc se někam dostat, nedělá nic jiného, než že animuje svůj pohyb na určité místo. Je důležité si uvědomit, že jakákoliv UI v herním světě v podstatě, při každé své činnosti využívá animace. Při střelbě se opakuje animace střelby, při skrývání se opakuje animace skrčení a vykukování za překážkou (Orkin, 2006).



Obrázek 15 – Tři stavy FSM ve F.E.A.R.

Zdroj: (Orkin, 2006)

Mezi jednotlivými stavy FSM se zvolna přepíná. Problém ovšem nastává při položení otázky kdy přesně přepnout z jednoho stavu na druhý a jaké parametry jim nastavit. Tato logika může být přímo uvnitř každého FSM, což by znamenalo, že každý charakter mající FSM by měl svou logiku rozhodování. Ve F.E.A.R. se vývojáři rozhodli, aby tuto logiku měl na starost speciální plánovací systém namísto jednotlivých FSM (Orkin, 2006).

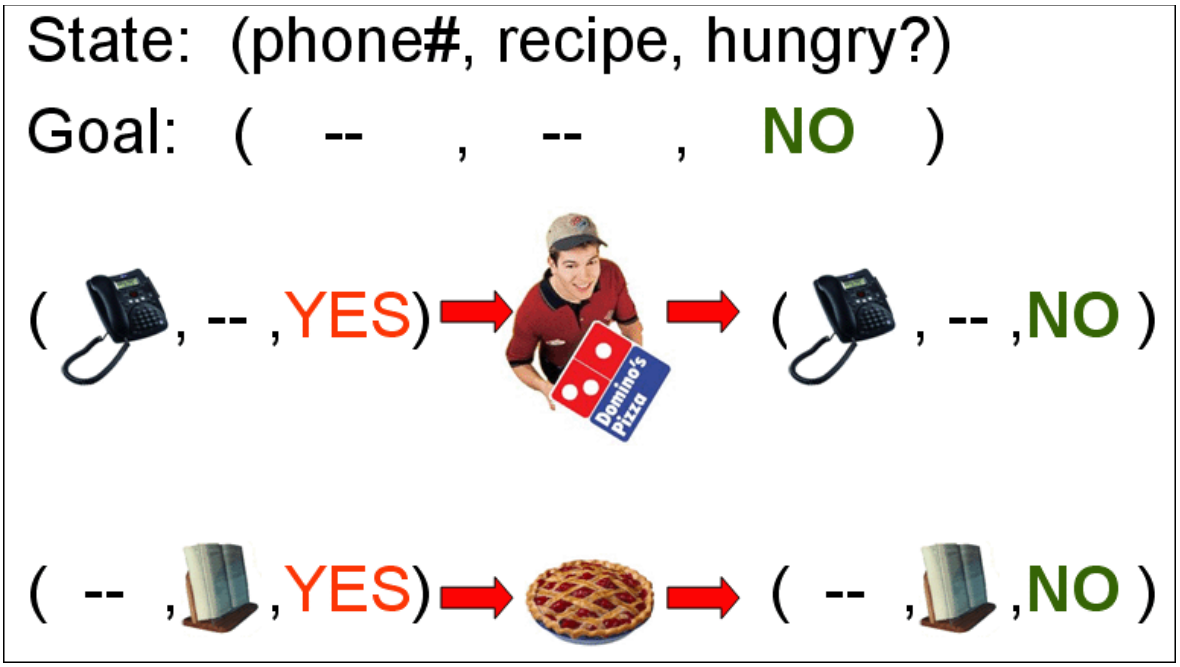
3.2 Rozdíl mezi FSM a plánováním

FSM přímo říká UI, co přesně má udělat a jak se chovat v daných situacích. Je to procedurální způsob. Na rozdíl od toho, plánovací systém pouze UI upozorní, jaké jsou její cíle a dostupné akce a nechá UI samotnou rozhodnout, jakými akcemi a v jakém pořadí daného cíle dosáhne. Plánování lze chápat jako deklarativní způsob. Plánování je tedy proces hledání sekvence akcí, k dosažení nějakého cíle. Plánovací systém ve F.E.A.R. je založen na systému STRIPS (Orkin, 2006).

3.3 Systém STRIPS

Tento systém vznikl v roce 1970 na Univerzitě ve Stanfordu. STRIPS se skládá z cílů a akcí, kde cíle popisují nějaký vytoužený stav světa, kterého se dá dosáhnout. K akcím přísluší různé předpoklady a efekty, které mají určitý význam na svět. Akce může mít ovšem efekt na stav světa pouze tehdy, pokud jsou splněny veškeré její předpoklady (Orkin, 2006).

Tento systém lze snáze pochopit na příkladu, kde se vyskytuje hladový člověk, jehož jméno není podstatné. Člověk si může objednat pizzu, ale pouze za předpokladu, že má číslo na pizzerii. Pizza není jediná možnost jak ukojit hlad. Může si ještě například upéci koláč, ale jen pokud má recept. Člověk má tedy k dispozici dvě akce, jak se dostat k cíli což je jinými slovy stav světa, kde nemá hlad. Pokud se nachází ve stavu světa, kdy má recept, upeče koláč. Pokud má číslo do pizzerie, nechá si přivést pizzu. Pokud má obě dvě věci, jsou dvě možné cesty k dosažení cíle a rozhoduje se na základě ohodnocení akcí (viz. kapitola Ohodnocení akcí), nicméně pokud nemá ani jednu, cíle nelze dosáhnout (Orkin, 2006).



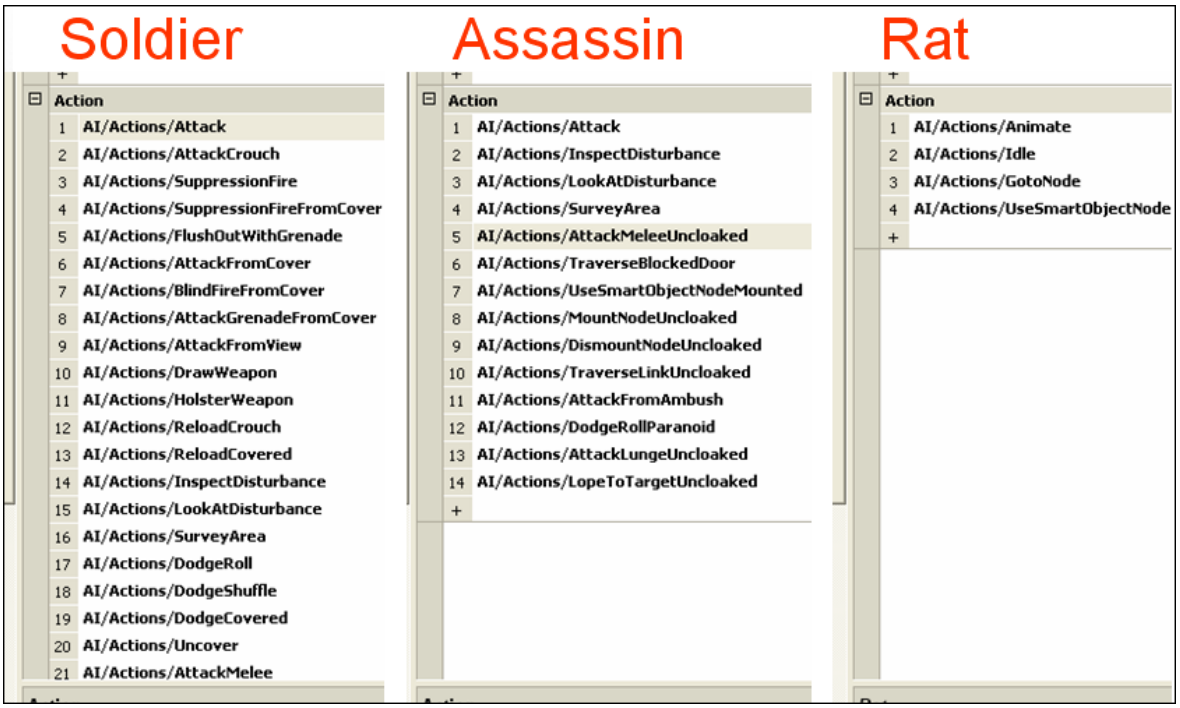
Obrázek 16 – STRIPS příklad

Zdroj: (Orkin, 2006)

Samozřejmě tento příklad je co nejvíce zjednodušen za účelem základního pochopení problematiky. Zmíněné akce mohou mít ještě nejrůznější předpoklady a těmito předpoklady být zřetězeny do dalších akcí (Orkin, 2006). Například pokud má člověk číslo do pizzerie, musí mít ještě na pizzu peníze. Jestliže nemá, musí podniknout další akci a to cestu do banky. Cestu do banky může učinit buď chůzí, nebo jízdou autem. K jízdě v autě musí mít benzín. Pokud nemá, musí jet natankovat a tak dále.

3.4 Plánování ve F.E.A.R.

Na obrázku (Obrázek 16) jsou vidět dvě akce k uspokojení hladového člověka. Co když ale bude jíst jen zeleninu a ne pizzu a sladkosti? V takovém případě se vytvoří akce speciálně pro tohoto člověka vegetariána. To znamená, že dvě akce uspokojí hladového člověka a jedna akce uspokojí člověka vegetariána. Takovýto systém aplikovali vývojáři ve F.E.A.R. Každá postava má jinou skupinu akcí, díky kterým může dosáhnout daných cílů. Pro objasnění, obyčejný voják má cíle hlídkovat ve skladišti, a pokud spatří nepřítele, zlikvidovat ho. Voják se při plnění cílů bude chovat jinak, než například hbitý vrah, který bude hlídkovat po skladišti způsobem určeným danými akcemi přímo pro tento typ postavy (Orkin, 2006). Na obrázku (Obrázek 17) lze vidět skupiny akcí pro tři typy postav, vojáka, vraha a krysu. Každý z nich bude dosahovat cílů pomocí těchto akcí.



Obrázek 17 – Skupiny akcí podle typů postav

Zdroj: (Orkin, 2006)

3.5 Výhody systému plánování

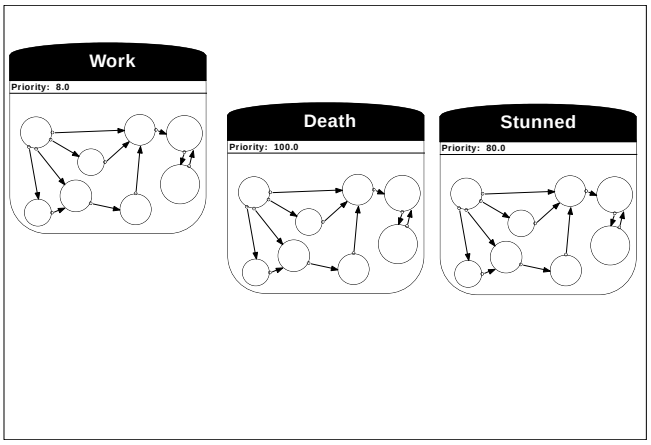
Plánování chování UI má tři základní výhody. Jsou to *oddělené akce od cílů*, *vrstvení jednoduchých chování* k vytvoření jednoho komplexnějšího chování a nakonec *obohacení UI o schopnost dynamicky řešit problémy* (Orkin, 2006).

3.5.1 Oddělení akcí od cílů

V minulých produktech společnosti Monolith Productions jako je NOLF2 měl každý cíl, za kterým UI šla svůj vnořený FSM. Neexistoval způsob jak oddělit cíl od plánu, ke splnění tohoto cíle. Pokud vývojáři chtěli přidat chování k nějakému typu postavy, museli vytvořit novou větev přímo u příslušného FSM. Postupem času se tyto FSM staly nesmírně

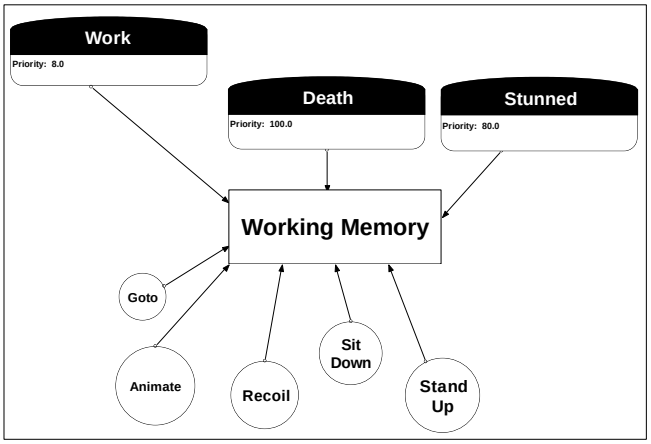
obsáhlými a každá úprava v nich by mohla mít neblahé následky na celý produkt. S využitím systému plánování lze každé UI postavě dát svůj vlastní seznam akcí. Navíc tento oddělený systém umožňuje i jednoduše a s minimální časovou náročností vytvářet UI pro nové postavy, kombinováním akcí a cílů postav, již vytvořených (Orkin, 2006).

V předešlých titulech jako je NOLF2 mezi sebou cíle nesdílely žádné informace, což mělo za následek občas neblahé zkraty v UI. Každý cíl se musel nejdříve ukončit se všemi akcemi, které k němu náležely, a poté se teprve mohl začít řešit další cíl. Ve F.E.A.R. za pomoci oddělení akcí od cílů mezi sebou jednotlivé cíle sdílejí informace skrze pracovní paměť a tudíž na sebe mohou navazovat přirozeně (Orkin, 2006).



Obrázek 18 – Cíle a akce V NOLF2

Zdroj: (Orkin, 2006)



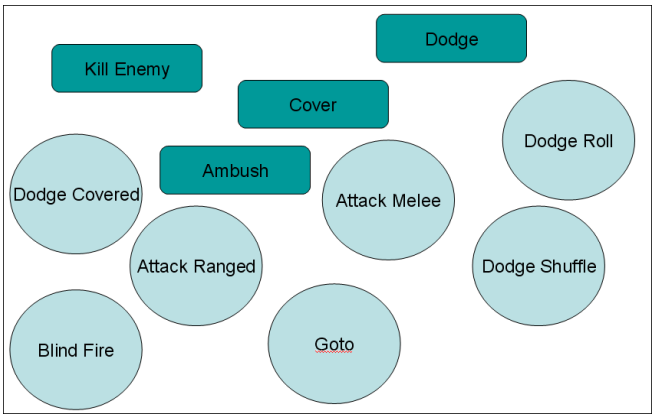
Obrázek 19 – Oddělené cíle a akce ve F.E.A.R.

Zdroj: (Orkin, 2006)

3.5.2 Vrstvení chování

Jak je vidět na obrázku (Obrázek 20), voják, jakožto UI postava, může mít cíl *KillEnemy*, kterého docílí akcí *AttackRanged*. To je první, základní vrstva chování. Druhá může být, aby voják uskočil, pokud je na něho zamířeno kurzorem. Lze tedy přidat cíl *Dodge*,

kterého lze dosáhnout akcemi *DodgeRoll* a *DodgeShuffle*. Další vrstva může být například, aby voják zaútočil pažbou své zbraně v případě, že se hráč přiblíží. V tomto kroku už existuje cíl *KillEnemy*, tudíž stačí přidat pouze akci *AttackMelee* (Orkin, 2006). Takto lze přidávat nové akce a cíle do konečné podoby realistického chování postav.



Obrázek 20 – F.E.A.R. vrstvení chování

Zdroj: (Orkin, 2006)

Jádrem této výhody je, že vývojáři mohou nadefinovat různé akce a cíle bez nutnosti jakýchkoliv vazeb mezi nimi. UI si sama tyto vazby určí v závislosti na jednotlivých cílech, předpokladech a efektech akcí (Orkin, 2006).

3.5.3 Dynamické řešení problémů

Dynamické řešení problémů znamená, že UI je schopna na základě svých předešlých neúspěchů v dostání cíle, přehodnotit své chování a pokusit se dosáhnout cíle za pomoci jiných akcí a činností. Kupříkladu hráč svým virtuálním tělem blokuje dveře, které se UI postava snaží otevřít a uskutečnit tak cíl *KillEnemy*, o kterém ví, že je za nimi. Pokud selže v otevírání, pokusí se dveře vykopnout, pokud i to selže, pokusí se proskočit blízkým oknem a dostat se k hráči tímto způsobem (Orkin, 2006).

3.6 Odlišnosti v plánovacím systému F.E.A.R. a v systému STRIPS

Vývojáři Monolith Productions se rozhodli, že pojmenují jejich plánovací systém GOAP. Jak již bylo zmíněno, tento systém je založen na plánovacím systému STRIPS, nicméně GOAP musel být pozměněn, aby se dal aplikovat na dnešní hry (Orkin, 2006). Tato změna přinesla různé odlišnosti od STRIPS systému.

3.6.1 Ohodnocení akcí

V předchozím příkladě se vyskytoval člověk, který měl za cíl uspokojit svůj hlad. Na výběr měl dvě akce, zavolat do pizzerie pokud má číslo, nebo upéct koláč pokud má recept. Pokud má recept i číslo rozhoduje se na základě ohodnocení těchto akcí. Každé akci je přiděleno ohodnocení v podobě čísla. Objednání pizzy má například 1.0, zatímco pečení koláče 2.0. V této fázi nejsou všechny cíle, akce a jejich předpoklady nic jiného než ohodnocený graf s body, na který se dá aplikovat algoritmus *A**. *A** algoritmus tedy

vyhledá posloupnost nejníže ohodnocených akcí k uspokojení nějakého cíle. Jak je vidět, *A** algoritmus se ve F.E.A.R. nepoužívá jen na Pathfinding po navigačních sítích, ale také pro plánování posloupnosti akcí, kde body grafu jsou stavy světa, tedy cíle, a jednotlivé hrany k nim vedoucí jsou akce k docílení těchto stavů (Orkin, 2006).

A*	Navigation	Planning
Nodes:	NavMesh Polys	World States
Edges:	NavMesh Poly Edges	Actions

Obrázek 21 – Porovnání A* algoritmu pro navigaci a plánování

Zdroj: (Orkin, 2006)

3.6.2 Odstranění vkládacích a odebíracích seznamů

STRIPS systém obsahuje pro efekty akcí vkládací a odebírací seznamy. V GOAP systému se vývojáři rozhodli tyto seznamy odstranit a vložit předpoklady a efekty akcí do pevně daného pole. Například akce *AttackRanged* má předpoklad, že zbraň, se kterou se tato akce provede, musí být nabita. A akce *Reload*, což je akce nabití zbraně, má za výsledný efekt nabití zbraně (Orkin, 2006).

3.6.3 Procedurální předpoklady a efekty

Ve F.E.A.R. je akce napsaná jako třída v jazyce C++. Existuje funkce *CheckProceduralPreconditions()*, která vrací True, pokud se například UI postava potřebuje přemístit z krytu, na který je vedena ustavičná palba do většího bezpečí. Pokud by vrátila tato funkce False, postava by musela zůstat v krytu, jelikož by pro přemístění nebyly splněny potřebné předpoklady. Pro efekty je to obdobné. Zdrojový kód třídy *Action:*

```
class Action
{
      // Symbolic preconditions and effects,
      // represented as arrays of variables.
      WORLD_STATE m_Preconditions;
WORLD_STATE m_Effects;

      // Procedural preconditions and effects.
      bool CheckProceduralPreconditions();
      void ActivateAction();
};
```

(Orkin, 2006)

3.7 Chování jednotek

Vývojáři aplikovali ve F.E.A.R. speciální systém pro koordinaci chování UI postav v jednotce. Jednotkou je myšleno větší uskupení UI postav, které spolu vzájemně spolupracují a komunikují mezi sebou. Chování jednotek se dá rozdělit do dvou kategorií. Je to *jednoduché* a *složité chování*. *Jednoduché chování* se stará o to, aby se postavy v jedné jednotce následovaly, navzájem kryly palbou a pohybovaly se na určitá vytyčená místa. *Složité chování* zahrnuje obcházení nepřítele, koordinované útoky, přivolání posil a stahování se z nebezpečných situací (Orkin, 2006).

3.7.1 Jednoduché chování jednotek

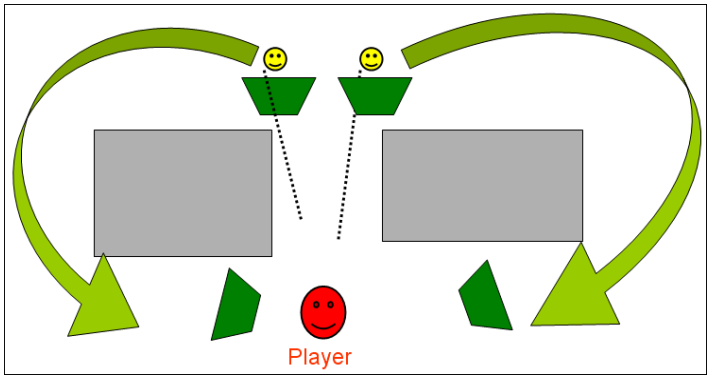
Jednoduché chování jednotek vychází ze čtyř základních chování. *Get-to-Cover* chování přinutí všechny členy jednotky, kteří nejsou v řádném krytu, aby ho vyhledali a schovali se za něho. Chování *Advance-Cover* je modifikací předešlého. V podstatě opět donutí členy jednotky vyhledat vhodné krytí, nicméně u toho budou při tomto typu postupovat vpřed k nepříteli. *Orderly-Advance* je postup jednotky na určité místo, přičemž jsou všichni členové seřazeni za sebou kryjící jeden druhého. A poslední chování je *Search*, kde je jednotka rozdělena na dvojice. Každý člen ve dvojici potom kryje druhého a postupně tak prohledávají okolí (Orkin, 2006).

UI se tak dynamicky snaží plnit tato chování v závislosti na cílech a rozkazech. Systém chování jednotky vždy vyhledá vhodnou UI postavu. Té zadá rozkaz k vykonání nějaké činnosti, ovšem postava se může sama rozhodnout, zda je lepší splnit rozkaz, nebo se věnovat svému cílu. V případě, kdy se dvě UI postavy kryjí před hráčovou palbou, se aktivuje systém chování a zadá jedné postavě, aby opětovala palbu, zatímco druhá postava se bude snažit nalézt lepší kryt. Pokud se stane, že hráč naruší průběh chování UI, například hozením granátu za novou překážku, systém chování selže, protože UI nemůže nový kryt využít a tím se naruší dané chování jednotky (Orkin, 2006). UI poté čeká na další rozkaz nebo se rozhodne pro uspokojení svého cílu, což může být útěk. Celý tento systém se velmi přibližuje realitě.

3.7.2 Složité chování jednotek

Pravdou je, že vývojáři neimplementovali žádné složité chování jednotek. Toto chování vyvstává z kombinace věcí již hotových, konkrétně z jednoduchého chování jednotek a schopností rozhodování se jednotlivých UI postav. Často lze nabít dojmu, že se jedná o složitější chování, než doopravdy je (Orkin, 2006).

V předchozím příkladě byly dvě UI postavy kryjící se za překážkou. Systém chování rozkáže jedné postavě najít lepší úkryt. Pokud tato postava narazí na překážky typu menší zeď, či okno při hledání lepšího krytu, přeskočí je a může se tak dostat do boku nepřítele, což může vypadat jako obíhání hráče. Nejedná se ovšem o nic jiného než vedlejší efekt, prosté snahy nalézt úkryt (Orkin, 2006). Nicméně ve F.E.A.R. to opravdu funguje a lze nabít dojmu, že proti hráči stojí lidští protivníci.



Obrázek 22 – Složité chování jednotky vyplynulé ze situace

Zdroj: (Orkin, 2006)

K umocnění zážitku byla zavedena komunikace mezi jednotlivými členy jednotky. Pokud UI postava zůstane z jednotky sama, zavolá o posily, což hráč slyší a nabude dojmu, že za okamžik proti němu bude stát více protivníků. Celá komunikace ještě více vtahuje hráče do akce.

Vývojáři tedy zavedli v této hře zcela nový systém plánování akcí, a chování jednotek i postav samotných. Jak již bylo řečeno, UI ve F.E.A.R., je jedna z nejpovedenějších v novodobé historii FPS žánru.

4 Crysis Sandbox editor 2

Crysis Sandbox editor 2 je prostředek pro práci s herním enginem CryEngine 2.

Herní engine obecně je v podstatě pohon hry. Určuje, jaká je UI postav a jak vypadá grafika ve hře, za pomoci nejrůznějších technik. Všechny tyto techniky vznikají za jediným účelem a to je nabití dojmu co nejreálnějšího prostředí. Svět, ve kterém se hráč pohybuje, by měl navozovat pocit, že je opravdu v přírodě, kde se sluneční paprsky odrážejí od hladin řek, či v opravdovém městě. Ovšem existují výjimky, kdy se autoři ve svých herních enginech nesnaží o realističnost, nicméně ve velké míře toto pravidlo platí.

Naprogramovat herní engine je velice složité. Z tohoto důvodu firmy, které tak učiní, prodávají své enginy, aby poháněly jiné hry od jiných studií, než ke kterým byl původně určen. Příkladem může být například herní engine Unreal od studia Epic Games, který má již mnoho verzí a pohání nesčetně PC her.

Jedním z herních enginů je CryEngine 2 od německého studia Crytek. Tento engine pohání FPS hru Crysis z roku 2007 a pro práci s ním, pro vytváření map a modelování UI, byl vytvořen právě Crysis Sandbox editor 2. V podstatě celá hra Crysis byla v tomto editoru vytvořena. Editor umožňuje otevřít projekty již hotových map (úrovní) přímo od vývojářů a nahlédnout tak jak byly designovány nebo vytvořit své vlastní úrovně. Editor lze získat po zakoupení originální kopie hry Crysis.

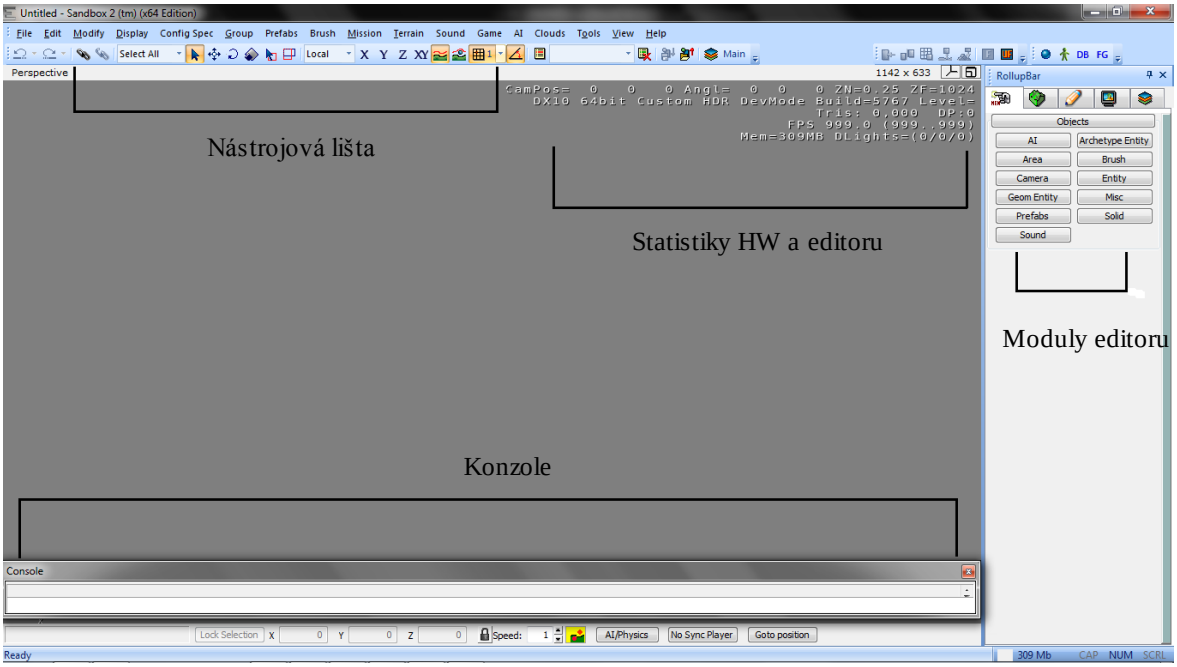
Crysis byla kritiky velice chválená nejen díky revoluční grafice, ale také kvůli povedené UI.

4.1 Základní práce s editorem

Jak již bylo zmíněno výše, Crysis Sandbox editor 2 umožňuje vytvářet své vlastní mapy, které jsou po vytvoření a uložení projektu k nalezení ve složce s hrou *Crysis/Game/Levels*. Tyto projekty reprezentuje daná složka s jeho názvem, ve které je po počátečním uložení soubor **.cry*. Po vytvoření terénu mapy a případném nanesení textur se ve složce projektu objeví soubory **.pak* a **.bak*.

4.1.1 Představení uživatelského rozhraní

Na obrázku (Obrázek 23) je znázorněno výchozí okno editoru. V horní části se vyskytuje nástrojová lišta s různými tlačítky. Tato tlačítka slouží především pro práci s vkládanými objekty jako např. jejich přemisťování. Nad lištou se nacházejí nejrůznější záložky. Z nich je nejznámější záložka *File*, která slouží, jak je známo z jiných editorů, či programů, k uložení projektu a práci s ním. Dále se vyskytují záložky na přizpůsobení editoru dle potřeb uživatele, na změnu grafických detailů pro pomalejší nebo naopak rychlejší PC, na vytvoření jednotlivých misí na konkrétní mapě, vytvoření terénu, UI a tak podobně.



Obrázek 23 – Výchozí okno editoru

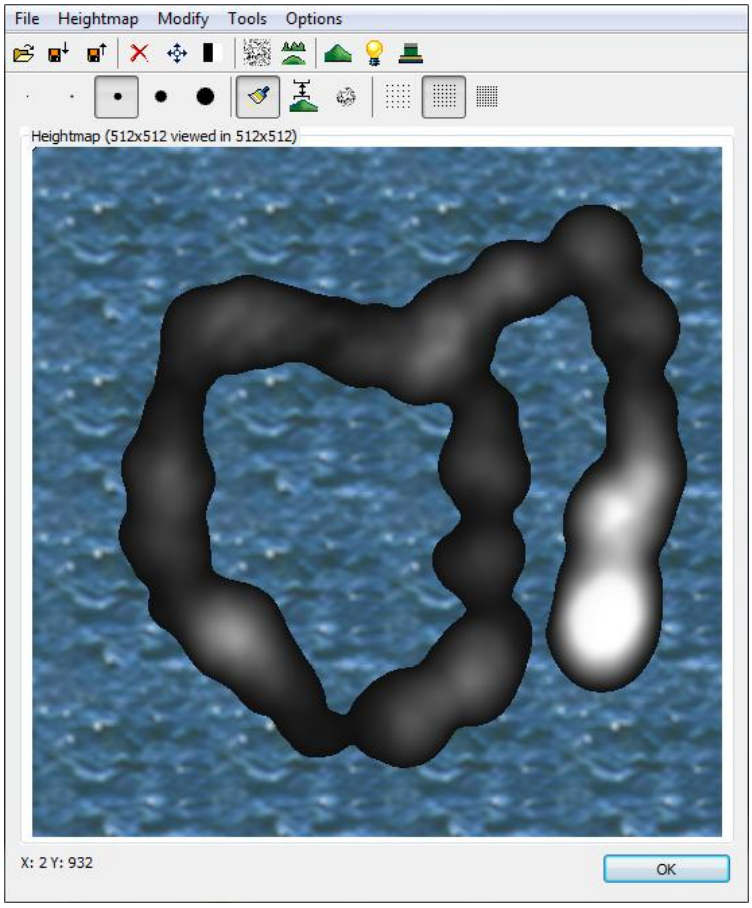
Zdroj: Autor

V dolní části se nachází konzole pro psaní užitečných příkazů. V pravém horním rohu zobrazovacího okna jsou různé statistiky a informace o mapě, či použitém hardwaru. Konečně v pravé části editoru se nacházejí jednotlivé moduly pro objekty, terén, zobrazení, či přehled vrstev.

4.1.2 Vytvoření a úprava terénu

V Sandbox editoru 2 se začíná tvořit terén z vody. Nejdříve se vytvoří nová mapa, které se zadá v příslušné tabulce rozměr a název. Poté se tlačítkem *Generate surface texture* vytvoří zmíněná voda. Všechny tyto možnosti jsou dostupné ze záložky *File*. Je možné si povšimnout části vodní plochy, která je ohraničená šedými hranami. Tato plocha je plocha, kde se bude vytvářet terén.

Nyní je možné v záložce *Terrain* terén různě upravovat. K hrubé úpravě povrchu mapy slouží možnost *Edit terrain*, kde se zobrazí půdorys vytvořené mapy, zatím tedy jen vodní plochy. Pomocí nástroje (štětce) a klikání na vodní plochu lze základní tvar terénu libovolně upravovat. Je zde také k nalezení možnost pro automatické vygenerování terénu.



Obrázek 24 – Vytvoření terénu na mapě

Zdroj: Autor

V pravé části editoru se nachází modul pro terén, v němž je záložka *Modify*, kde se vyskytují funkce pro jemnější a lokální úpravy povrchu mapy. Tyto funkce se nazývají *Flatten*, *Smooth* a *Rise/Lower*.

Funkce *Flatten* vyrovnává terén do dané výšky. Velikost vyrovnávané plochy je dána velikostí štětce, přičemž existuje vnější kruh a vnitřní kruh, ve kterém je působení funkce silnější. Dále je možno měnit parametr *Hardness*, který určuje sílu efektu funkce na daný povrch. Poslední parametr je *Height*, tedy výška vyrovnání. Pokud je terén v místě aktivace této funkce vyšší než zadaná výška *Height*, funkce *Flatten* tuto část sníží a naopak.

Smooth je funkcí, která vyhlazuje ostré hrany nadělané buď automatickým generováním terénu, nebo zkrátka jeho nešikovnou úpravou. Jako parametry jsou u této funkce uvedeny *Hardness* a velikost štětce.

Poslední funkcí je *Rise/Lower*, která, jak již název napovídá, zvyšuje nebo snižuje povrch určený velikostí štětce podle zadané výšky. Tato funkce je vhodná k modelování různých skalisek či propastí.

4.2 Vytvoření vegetace a práce s texturami

Nanesení správných textur na místa, kde by jejich výskyt byl přirozený, je důležitý krok blíže k realističnosti dané mapy. Pro vegetaci toto pravidlo platí taktéž. Bylo by nepřirozené vytvářet palmový porost na vrcholku hor a pochopitelně nepřirozené na vrcholky hor nanést texturu pláže.

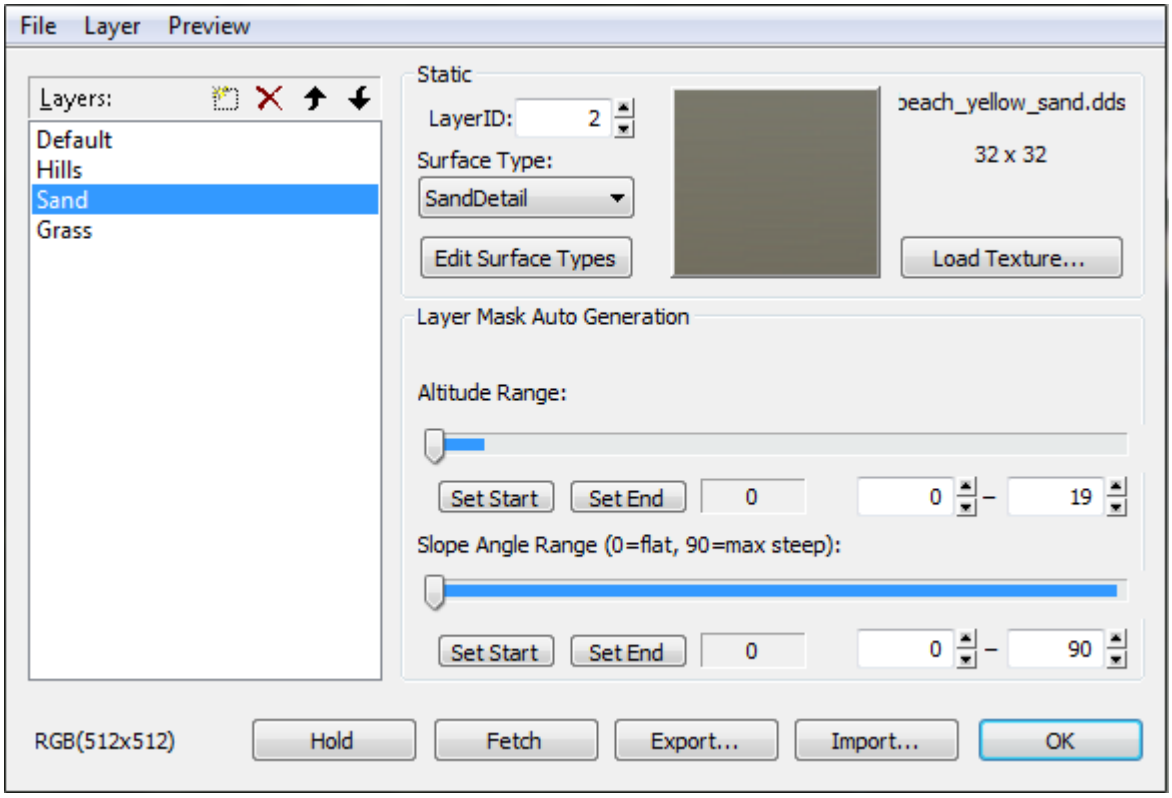
4.2.1 Práce s texturami

Pro práci s texturami existuje v editoru možnost *Texture*, kterou lze nalézt v záložce *Terrain*. Na obrázku (Obrázek 25) lze vidět okno pro práci s texturovými vrstvami. Každá vrstva má dva typy textur, jednu hrubší, druhou jemnější. Hrubší textura se vybírá za pomoci tlačítka *Load Texture*, jemnější povrchová textura se spravuje za pomoci *Edit Surface Types*, kde se vytvoří typ povrchu a vybere k němu příslušná textura. Poté již stačí přiřadit vybraný typ povrchu k hrubší textuře v roletce *Surface Type*.

Jak lze vidět na obrázku (Obrázek 25), pro účel praktické části práce byly vytvořeny tři typy textur. Textury pro písek, hory, respektive kopce, a trávu. V nabídce se nachází také výchozí textura, ta je ovšem zbytečná. Např. textura pro písek má vybranou detailnější texturu *beach_white_sand*, která se skrývá pod vytvořeným povrchovým typem *SandDetail*. Hrubá textura pro písečný povrch je *beach_yellow_sand*.

Za povšimnutí také stojí možnosti jednotlivých texturových vrstev. Možnost *Altitude Range* udává, do jaké výšky terénu se bude textura nanášet. Pro písečnou texturu byla tato hodnota zvolena od 0 do 19 bodů, což je něco málo nad hladinou vody. Tímto lze docílit, že textura písku se bude nanášet pouze na okraje terénu hraničícího s vodou a nabude tak dojmu skutečné pláže.

Další možnost *Slope Angle Range* znamená, od jakého úhlu stoupání terénu se začne textura vykreslovat. To se hodí zejména v texturové vrstvě pro hory, či kopce.



Obrázek 25 – Okno pro vytváření texturových vrstev

Zdroj: Autor

V případě, že je vše nastaveno dle představ designéra, nachází se v modulu pro terén záložka *Layer Painter*, díky které lze připravené texturové vrstvy nanášet.

Při samotném nanášení textur lze měnit nastavení šířky štětce, *Filter* a parametr *Hardness*, který plní stejnou funkci jako u funkcí na úpravu terénu, tedy čím nižší hodnota, tím menší intenzita nanášené textury. Nastavení hodnoty *Filter* určuje odstín textury. Menší hodnota reprezentuje tmavší odstín a naopak.

4.2.2 Vytváření vegetace

Pro samotnou vegetaci je v modulu *Terrain* připravena záložka *Vegetation*. Tato záložka umožňuje spravování veškeré vegetace na mapě, dělení vegetace na vlastní definované kategorie a samotné vkládání vegetace na mapu pomocí tlačítka *PaintObjects*.

Pro každou položku, či pro celou kategorii existuje soubor nastavení, jak mají být dané položky na mapě prezentovány. Nejdůležitější parametry nastavení vegetace jsou bezesporu *Density*, *RandomRotation* a *AlignToTerrain*.

Parametr *Density* udává, jak daleko budou jednotlivé vegetační objekty od sebe vzdáleny. Vypadalo by špatně, kdyby mezi jednotlivými stromy nebyla ani malá mezera. Nebylo by to podobné realitě, tudíž se tento parametr pro větší rostlinstvo dává např. 10. To ovšem neplatí pro vegetaci v podobě trávy, která je opravdu v realitě naskládána těsně vedle sebe, tím pádem jí v editoru odpovídá hodnota *Density* = 1.

Další důležitý parametr je *RandomRotation*. Nastavením tohoto parametru na true, respektive jeho zaškrtnutím lze dosáhnout toho, že každý vkládaný objekt bude jinak natočen. Podobně jako v přírodě všechny rostliny nerostou stejně, dá se tohoto dojmu docílit i v editoru právě pomocí tohoto parametru.

AlignToTerrain zajišťuje, že vkládané objekty se budou přichytávat k povrchu terénu.

Dále se nabízejí parametry na vykreslování stínu za danými vegetačními objekty, či propojení objektu s již vytvořenou texturovou vrstvou. Toto propojení se například nabízí v kombinaci textury trávy a trávy samotné. Pokud se poté nanáší textura trávy pomocí *LayerPainteru* automaticky se na tuto texturu vkládá vegetace, v tomto případě tráva.

Obrázek 26 – Příklad vegetace s nanesenými texturami

Zdroj: Autor

Každý objekt lze libovolně otáčet, přesouvat, zvětšovat či zmenšovat nebo klonovat za pomocí klávesových zkratek. Výsledný dojem z reálného přírodního prostředí se tak ještě zintenzivní.

Na obrázku (Obrázek 26) je vidět, jak vypadá část mapy po nanesení textury trávy propojené s trávou jako vegetačním objektem, texturou písku a vegetací v podobě palem.

4.3 Modelování navigace UI

V editoru se nachází rozsáhlý modul pro práci s UI postav. Designér musí vštěpit postavám, co mohou a nemají dělat v různých situacích. Kam mají jít, kde se mají schovat, kde mají hlídkovat a tak podobně. Zkrátka pomocí nástrojů editoru docílit toho, aby se UI chovala co nejvíce přirozeně.

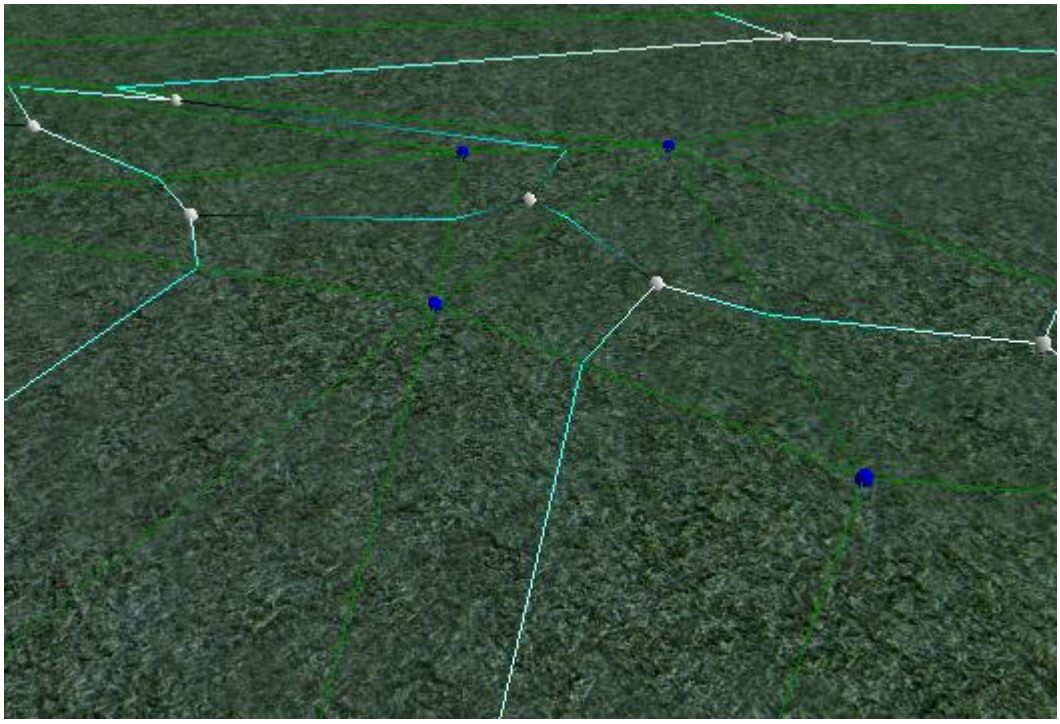
V pravé horní části editoru se vyskytuje tlačítko pro nahlédnutí do databáze objektů. Z té je možné vybrat předpřipravené postavy, které se budou po mapě pohybovat. Vedle tohoto tlačítka se nicméně nachází tlačítko pro tvorbu vzhledu vlastních postav. Postavy se dají přidávat ještě způsobem výběru záložky *Entity* v modulu pro práci s objekty. Tento způsob nabízí více možností a parametrů k úpravám dané postavy než postup skrze databázi.

Každý objekt (entita) má svoje parametry znázorněny v pravém sloupci, podobně jako tomu bylo u vegetace. Postavám se například dá měnit parametr, který jí udává, do jaké vzdálenosti spatří nepřítele. Dá se měnit také přesnost střelby, schopnost plavat ve vodě a jiné.

V případě nalezení vhodné postavy a přetažení jí do mapy se s ní dá pracovat pomocí nástrojů v nástrojové liště. Tyto nástroje umožňují její otáčení, přesouvání po osách x, y a z, či přichycení k povrchu terénu. Tímto způsobem se dají přidávat různé objekty, nejen postavy. Jako například stavby, překážky, skály.

Při přepnutí se do hry, čehož lze docílit klávesovou zkratkou CTRL + G, postava pouze stojí na místě a na hráče nereaguje. Pro vytvoření základní navigační sítě a základního chování daného programátory existuje možnost v záložce *AI,* která nese název *Generate all navigation*. Po napsání příkazu ai_DebugDraw=79 do konzole se objeví základní navigační síť, která je znázorněna na obrázku (Obrázek 27). Poté při přepnutí do hry již UI postava reaguje a pohybuje se po této síti pomocí *A** algoritmu. K reálnému chování to má ovšem daleko.

K vymodelování živoucí mapy, kde UI postavy hlídkují, pohybují se se zbraní v ruce, vědí, kde se mají schovat a nenarážejí tupě na překážky, slouží v editoru modul pro práci s objekty a v něm záložka *AI*. V ní se nacházejí prostředky, pomocí nichž je možné takto s UI pracovat. Tyto prostředky jsou *Forbidden Area*, *Forbidden Boundary*, *Tag Point*, *AI Navigation Modifier*, *AI Point* a *AI Anchor*.

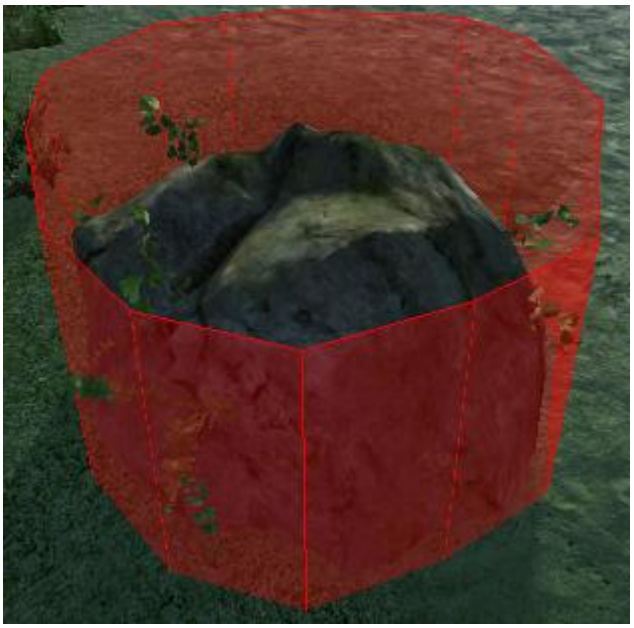


Obrázek 27 – Základní navigační síť

Zdroj: Autor

4.3.1 Forbidden Area a Forbidden Boundary

Funkce *ForbiddenArea* v záložce *AI* slouží k ohraničení objektů, do kterých by UI postavy mohly narážet, což by vypadalo značně nerealisticky. V podstatě se jedná o oblast, do které UI nemůže vstoupit.



Obrázek 28 – Forbidden area

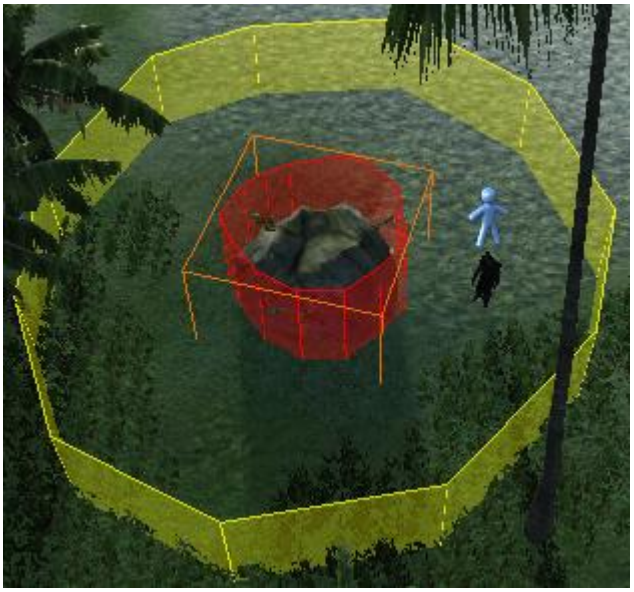
Zdroj: Autor

Tato funkce je velice důležitá a měla by se aplikovat na téměř všechny objekty na mapě, ať jsou to stromy, zátarasy, domy, skály zkrátka všude tam, kam by se UI neměla dostat. *ForbidenArea* je značena červenou barvou a měla by sahat něco málo do výšky a být aplikována mírně pod povrch terénu z hlediska preventivních opatření.

Důležité parametry této funkce jsou *Width*, který udává šířku, *Height* je již zmíněná výška a dále se může pro zviditelnění nastavit parametr *DisplayFilled*, který tuto oblast červeně vyplní.

Co se týká funkce ForbiddenBoundary, je velice podobná jako ForbiddenArea. Rozdíl je v tom, že zatímco Area ohraničuje oblast, do které se nedá vstoupit, Boundary ohraničuje oblast, ze které se nedá vystoupit.

Na obrázku (Obrázek 29) je znázorněn příklad použití těchto dvou funkcí společně. UI postava má ohraničenou oblast pohybu funkcí *ForbiddenBoundary,* kterou nemůže opustit a funkce *ForbiddenArea* ohraničuje oblast, na kterou nemůže postava vstoupit.



Obrázek 29 – Forbidden Boundary v kombinaci s Forbidden Area

Zdroj: Autor

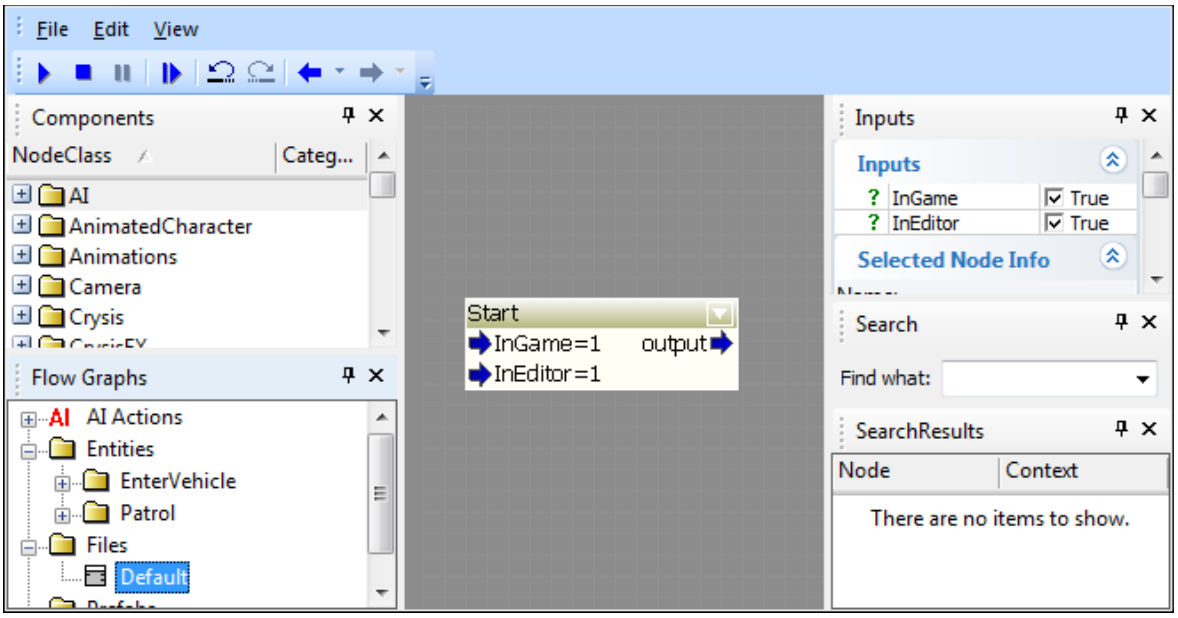
ForbiddenBoundary je označena žlutě. Opět je doporučeno nastavit parametr výšky a šířky a posunout tuto oblast kousek pod povrch terénu. Parametry jsou shodné s funkcí *ForbiddenArea*.

4.3.2 Flow Graph

Flow graf je diagram chování entit. Jedná se o skriptovací systém vložený do editoru k usnadnění práce designérů, kteří modelují určitou UI postav, aby nemuseli složitě zacházet do kódu a měnit ho podle vytoužené situace.

Rozhraní pro tvorbu grafu (Obrázek 30) se nachází v pravé horní části editoru nebo se do něho lze dostat skrze jednotlivé entity, které mají vytvoření *Flow* grafu k dispozici. V levé

části se nachází adresářová struktura již vytvořených, či rozpracovaných grafů. Nad ní jsou k dispozici nejrůznější třídy uzlů. Střední část je vyplněna pracovní plochou, na které jsou jednotlivé uzly zobrazeny. V této části lze také pochopitelně s uzly hýbat a přesouvat je, aby výsledný dojem z celkového grafu nebyl příliš chaotický. Pravá část rozhraní je určena k vyhledávání položek. Nad vyhledávací funkcí lze měnit parametry vstupních portů aktivního uzlu.



Obrázek 30 – Rozhraní pro tvorbu Flow grafů

Zdroj: Autor

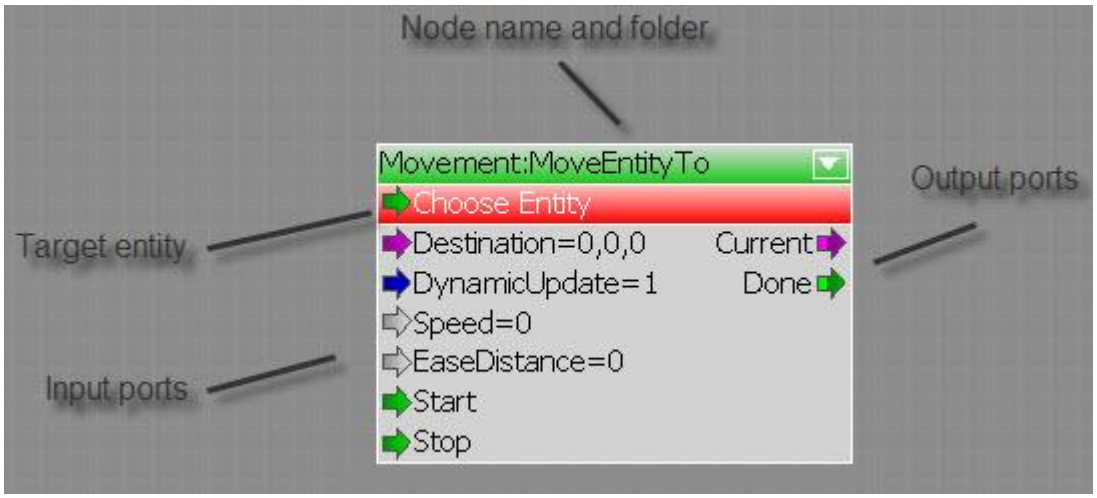
Flow graf se skládá z velké knihovny uzlů chování a akcí entit, které mají různé vstupní a výstupní porty. Většina těchto vstupů a výstupů obsahuje určité výchozí parametry, které mohou být skrze editor měněny. Pokud se na vstup jednoho uzlu připojí výstup z jiného, tento vstup se aktivuje a vrátí se hodnota jeho parametru. Vzájemným a logickým pospojováním vstupů a výstupů jednotlivých uzlů, lze docílit vytouženého chování postav.

Obrázek (Obrázek 31) popisuje konkrétní uzel Flow grafu, který má za cíl, jak již jeho název napovídá, přesunout danou entitu na určitou pozici a s určitou rychlostí.

- Vstup *Choose Entity* udává, jaká entita bude přesunována
- *Destination* je cílová pozice dána osami x, y, z
- Vstupní parametr *DynamicUpdate* říká, zda se bude dynamicky aktualizovat parametr *Destination*
- Parametr *Speed* je rychlost v metrech za sekundu, v jaké bude přesun uskutečněn
- *EaseDistance* je hodnota v metrech, po jaké bude změněna rychlost

- *Start/Stop* jsou vstupy udávající začátek a zastavení pohybu.

Každý uzel *Flow* grafu má za následek jinou akci a k této akci má k dispozici odlišnou skupinu vstupních a výstupních parametrů.



Obrázek 31 – Příklad uzlu MoveEntityTo Flow grafu

Zdroj: (Johnson, 2007)

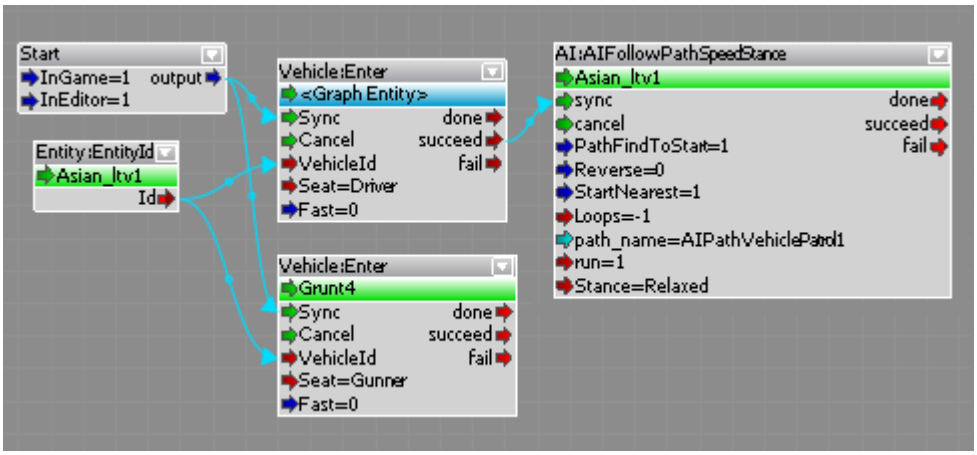
Jednotlivé vstupy a výstupy jsou různých datových typů a z hlediska toho jsou barevně odlišeny. Datové typy portů mohou být *Any*, *Boolean*, *Integer*, *Float*, *Vec3* a *String*.

- Pokud je port datového typu *Any* znamená to, že má nespecifikovaný datový typ a může tudíž obsahovat jakýkoliv. Tento typ je označen zelenou barvou
- *Boolean* je datový typ s parametry true = 1 nebo false = 0. Přísluší mu barva modrá
- *Integer* je klasické celé číslo ať záporné nebo kladné. V uzlu má červenou barvu
- *Float* je desetinné číslo s bílou barvou
- *Vec3* je datový typ skládající se ze tří desetinných hodnot, které udávají určitý vektor. Používá se pro ukládání pozice na osách x, y a z. Pro uložení hodnoty barvy RGB nebo různých úhlů. Tento datový typ má fialovou barvu
- Poslední datový typ je *String*. Ten se používá pro uložení textu a má tyrkysovou barvu

(Johnson, 2007)

Na obrázku (Obrázek 32) je příklad *Flow* grafu, který zapříčiňuje to, že dvě UI postavy nasednou do připraveného vozidla. Každý z nich má předem dané místo. Jeden je řidič, druhý střelec. Řidič potom s vozidlem hlídkuje po vyznačené trase.

Uzel *Start* je výchozí uzel. Jeho výstup se propojí se *Sync* vstupy uzlů *Enter*. Tyto uzly přikazují přiděleným entitám, tedy UI postavám nasednout do vozidla určeného vstupem *VehicleId* na pozici určenou vstupem *Seat*. K určení konkrétního vozidla, do kterého mají postavy nasednout, slouží uzel *EntityID*. Pokud nasednutí řidiče do vozidla proběhne úspěšně, propojením výstupu uzlu *Enter succeed* a vstupu *sync* uzlu *AIFollowPathSpeedStance* začne následování vytyčené trasy hlídky. Název trasy je předán jako hodnota vstupu *path_name* v uzlu *AIFollowPathSpeedStance*.



Obrázek 32 – Flow graf nasednutí a řízení vozidla

Zdroj: Autor

Jak již bylo zmíněno, uzly mají nejrůznější vstupy, které udávají parametry dané akce. Například poslední zmiňovaný uzel má vstup *Loops*, který udává počet opakování smyčky, v tomto případě hlídky. Hodnota -1 znamená nekonečně mnoho opakování, zkrátka vozidlo bude po vytyčené trase jezdit pořád, dokud nebude zničeno nepřítelem.

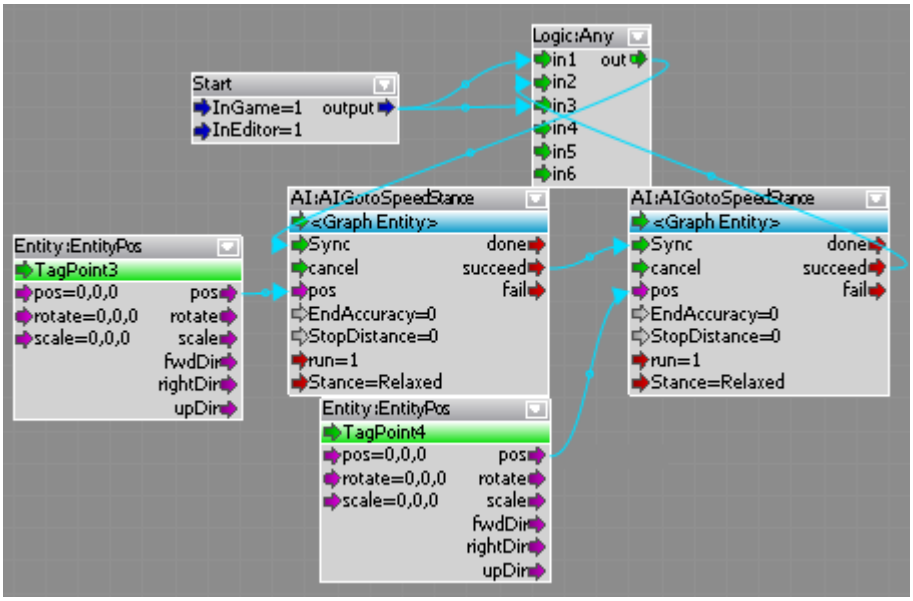
Flow grafy lze také exportovat do souboru **.xml* a následně importovat do jiného projektu v editoru, nicméně se importuje pouze graf s uzly a spojnicemi mezi nimi. Jednotlivé entity daných uzlů se do jiného projektu neimportují, tím pádem je na designérovi, aby tyto entity znovu přidal a přiřadil je konkrétním uzlům *Flow* grafu (Johnson, 2007).

4.3.3 Tag Point

Tag Point je speciální bod pro hlídkování postav. Po umístění dvou *Tag Pointů* a vytvoření vhodného *Flow* grafu může mezi nimi UI postava chodit, což navodí dojem hlídkování.

Obrázek (Obrázek 33) znázorňuje příklad takového *Flow* grafu. Uzel *Any* je použit z důvodu potřeby opakování hlídky. Jeho výstup je propojen se synchronizačním vstupem uzlu *AIGoToSpeedStance*, který určuje na jakou pozici se má postava přesunout. Pozice je určena právě zmíněnými *Tag Pointy*, které jsou předány jako parametry uzlu *EntityPos*, jehož výstup *pos* je propojen se vstupem *pos* uzlu *AIGoToSpeedStance*. V tomto případě tedy postava ví, že se má přesunout na pozici, kde se nachází první *Tag Point*. Pokud se tato akce podaří, propojí se výstup *succeed* se *sync* vstupem druhého uzlu *AIGoToSpeedStance*, který má na vstupu *pos* pozici druhého *Tag Pointu*. Tím pádem se

postava začne přesouvat na pozici druhého *Tag Pointu* a díky uzlu *Any* se tato operace opakuje pořád dokola.



Obrázek 33 – Flow graf hlídkování mezi Tag Pointy

Zdroj: Autor

4.3.4 AI Navigation Modifier a AI Point

Tyto dva prostředky se používají společně a povětšinou tam, kde se vyskytuje mnoho překážek na malém prostoru k zamezení zastavení postav. Typickým případem použití jsou např. stavby. *AI Navigation Modifier* ohraničuje oblast, ve které se dají použít *AI Pointy*. *AI Pointy* jsou body připomínající klasické Waypointy, tudíž body, po kterých se postavy uvnitř zóny ohraničené *AI Navigation Modifier* pohybují.

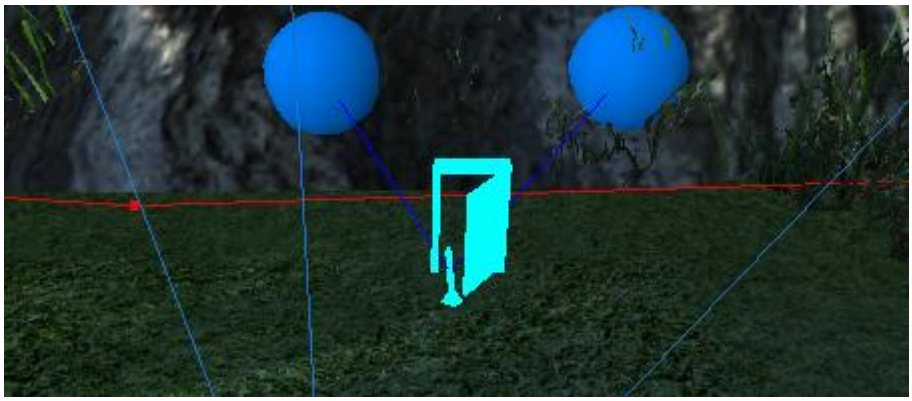
Pro první zmiňovaný prostředek platí téměř shodná pravidla jako pro *Forbidden Area*, či *Forbidden Boundary*. Mělo by se tedy počítat s určitou výškou této zóny a celou ji posunout mírně pod povrch. Parametr šířky zde nehraje důležitou roli. Na rozdíl od toho, se tu vyskytují dva důležité parametry. *NavType* určuje typ navigace uvnitř této oblasti. Typy navigace jsou například pro postavy, pro vzdušné jednotky nebo pro čluny a jednotky, které se pohybují na vodě. Druhý parametr je *WaypointConnection*, který určuje, jak se bude generovat navigace mezi *AI Pointy* v oblasti. *AI Navigation Modifier* je označen modrou barvou.

AI Pointy se pokládají tak, aby spojnice mezi nimi, nebyly přerušeny nějakou překážkou, protože by se na této spojnici mohla zastavit i UI postava a pokud by neměla jiný blízký bod k obejití překážky, její A* algoritmus by např. nedokázal nalézt cílový bod hlídky.

Existuje více typů *AI Pointů*. Např. typ *Hide* slouží pro úkryt UI postavy. Typ *Entry/Exit* znamená, že v jeho místě mohou UI postavy vstoupit nebo vystoupit z *AI Navigation Modifier* oblasti. Tento typ *AI Pointu* ignoruje oblast *Forbidden Area*. Kdyby tedy

neexistoval *Entry/Exit* bod, postava, která by se chtěla dostat do *AI Navigation Modifier* oblasti by zatuhla na hranici červené zóny *Forbidden Area*.

Na obrázku (Obrázek 34) je vidět kombinace *AI Navigation Modifier, AI Pointů* a oblasti Forbidden *Area*. Jeden z bodů je nastaven jako *Entry/Exit* jak je patrné z jeho obrázkové prezentace.

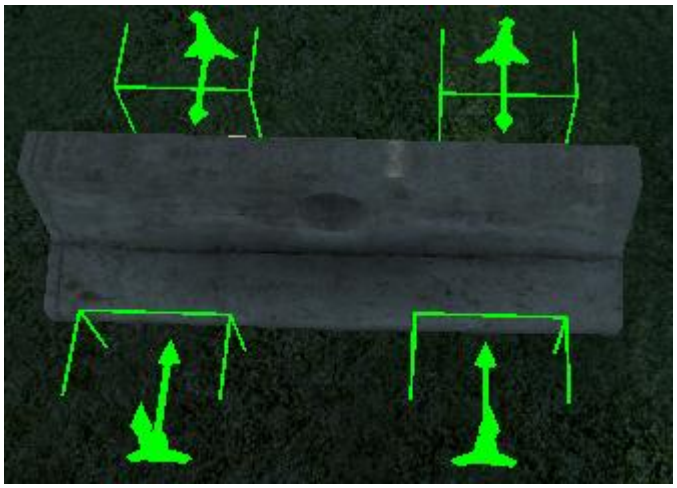


Obrázek 34 – AI Navigation Modifier s AI Pointy

Zdroj: Autor

4.3.5 AI Anchor

AI Anchor je speciální typ bodu, který se používá pro nejrůznější akce UI postav jako úkryty, přeskoky přes překážky, odstřelovací místa nebo jako shromažďovací bod pro posily.



Obrázek 35 – Příklad použití více AI Anchor k úkrytu za překážkou

Zdroj: Autor

Správnou kombinací těchto prostředků je možné docílit povedené UI postav a přiblížit tak jejich chování realitě.

Závěr

V první kapitole byly představeny základní principy UI v PC hrách. Důraz byl kladen především na schopnost pohybu jednotlivých UI postav. Rozdíl mezi waypointovou a navigační sítí a jak se aplikuje nejvyužívanější algoritmus hledání vhodné cesty, kterým je A* algoritmus. Představeno bylo také chování herní UI a prostředky jeho aplikování, kterými jsou FSM, Fuzzy logika a neuronové sítě. Dále byla věnována pozornost na další techniky herní UI, jako je rozeznání nepřítele a používání skriptů v dnešních hrách.

Druhá kapitola byla zaměřena na rozdělení herních žánrů z hlediska UI. Cílem bylo vybrat co nejzajímavější a nejpoužívanější žánry. Jednalo se o FPS a TPS hry, strategie RTS nebo tahové, sportovní a závodní hry. V této kapitole šlo o vystižení nejdůležitějších aspektů jednotlivých žánrů a tudíž ujasnění, jak se od sebe herní žánry liší.

Poslední kapitola teoretické části pojednávala o přiblížení UI v konkrétní hře F.E.A.R., která představuje legendární model herní UI. Důraz byl kladen na vysvětlení mechanismů tvorby v této hře. Pochopení těchto mechanismů je důležité, jelikož UI ve F.E.A.R. patří mezi nejvíce vydařené, tím pádem bylo cílem této kapitoly uspořádat znalosti z předešlých kapitol, do ucelenější a konkrétnější podoby.

V praktické části byl představen Crysis Sandbox editor 2, jakožto editor na modelování map, UI a postav ve hře Crysis. V tomto editoru byla vymodelována mapa s vegetací, na níž byly pomocí funkcí editoru aplikovány prostředky UI, k co nejrealističtějšímu chování postav.

Při samotném modelování v editoru se naskytly problémy zejména plynoucí z neznalosti funkcí. Hlavním problémem bylo zatuhnutí celkové UI na mapě, což bylo odstraněno tím, že byla znovu aplikována funkce na generování nové základní navigace. Toto aplikování ovšem nastavilo veškeré nanesené textury na výchozí hodnotu. Původ tohoto problému je v editoru a dá se odstranit jeho restartováním.

Práce v tomto editoru a zkoumání chování postav byla pro mě velmi zajímavou zkušeností a prohloubila moje znalosti nejen v oblasti UI, ale také v práci s editorem samotným. Obzvláště mě zaujala designérská činnost v rámci modelování dané mapy, kterou jsem toužil vždy vyzkoušet.

Celkové modelování v editoru a zkoumání chování postav v různých situacích, v kombinaci se znalostmi z předešlých kapitol, dává slušné povědomí o fungování herní UI v dnešních moderních PC hrách.

Literatura

Gonzáles-Calero, Pedro Antonio a Gómez-Martin, Marco Antonio. *Artificial Intelligence for Computer Games.* New York : Springer, 2011. 978-1-4419-8187-5.

Graham, Ross a McCabe, Hugh a Sheridan, Stephen. gamesitb.com. *Pathfinding in Computer Games.* [Online] 8. 12 2003. [Citace: 9. 3 2013.] http://gamesitb.com/pathgraham.pdf.

Hruška, František. dcgi.felk.cvut.cz. *Analýza umělé inteligence v akčních počítačových hrách a její implementace.* [Online] 19. 4 2008. [Citace: 8. 3 2013.] http://dcgi.felk.cvut.cz/home/felkepet/projekty/Hruskf1_semestr_proj.pdf.

Champandard, Alex J. aigamedev.com. *Open Challenges in First-Person Shooter (FPS) AI Technology.* [Online] 7. 4 2011. [Citace: 8. 3 2013.] http://aigamedev.com/open/editorial/open-challenges-fps/.

Johnson, Adam. sdk.crydev.net. *Importing and Exporting Flow Graphs.* [Online] 2007. [Citace: 7. 5 2013.] http://sdk.crydev.net/display/SDKDOC21/Importing+and+Exporting+Flow+Graphs.

Johnson, Adam. sdk.crydev.net. *Flow Graph Node Composition.* [Online] 2007. [Citace: 7. 5 2013.] http://sdk.crydev.net/display/SDKDOC21/Flow+Graph+Node+Composition.

Lester, Patrick. www.policyalmanac.org. *A* Pathfinding for Beginners.* [Online] 18. 7 2005. [Citace: 26. 3 2013.] http://www.policyalmanac.org/games/aStarTutorial.htm.

Millington, Ian a Funge, John. *Artificial Intelligence for Games Second Edition.* Burlington : Morgan Kaufmann Publishers, 2009. ISBN 978-0-12-374731-0.

Orkin, Jeff. web.media.mit.edu. *Three States and a Plan: The A.I. of F.E.A.R.* [Online] 2006. [Citace: 6. 4 2013.] http://web.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf.

PaulT. www.ai-blog.net. *Fixing Pathfinding Once and For All.* [Online] 26. 7 2008. [Citace: 9. 3 2013.] http://www.ai-blog.net/archives/000152.html.

TechTerms. http://www.techterms.com/. *Artificial Intelligence.* [Online] 1. 12 2010. [Citace: 3. 5 2013.] http://www.techterms.com/definition/artificial_intelligence.

Vávra, Daniel. www.games.cz. *Umělá inteligence ve hrách #1: zapomeňte na inteligenci.* [Online] 24. 7 2011. [Citace: 6. 3 2013.] http://games.tiscali.cz/tema/umela-inteligence-ve-hrach-1-zapomente-na-inteligenci-56421.

Vávra, Daniel. www.games.cz. *Umělá inteligence ve hrách #2: s AI je to peklo, bez ní ale také.* [Online] 3. 8 2011. [Citace: 28. 3 2013.] http://games.tiscali.cz/tema/umela-inteligence-ve-hrach-2-s-ai-je-to-peklo-bez-ni-ale-take-56508.

Příloha A – Přiložené CD

Přiložené CD obsahuje uložený projekt s vymodelovanou mapou a UI, spustitelný Crysis Sandbox editorem 2. Dále je na CD obsažen tento text ve formátu pdf.