

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**

**FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS**

SIMULACE DAVU METODOU BOIDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

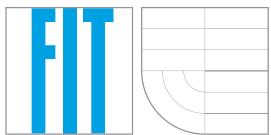
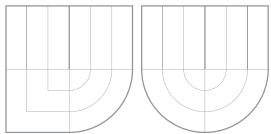
AUTOR PRÁCE
AUTHOR

RADEK BURDA

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULACE DAVU METODOU BOIDS

BOIDS METHOD FOR SWARM SIMULATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

RADEK BURDA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. FRANTIŠEK ZBORIL, Ph.D.

BRNO 2013

Abstrakt

Práce popisuje metodu simulace shlukování C.Reynoldse – BOIDS – a použije ji jako základní model pro simulaci davu. Popisuje metody vyhýbání se překážkám a jak rozsuzovat síly (pravidla shlukování, vyhýbání se překážkám a plnění cílů) v modelu BOIDS tak, aby vzájemně nekolidovaly. Mimo popis metody BOIDS zde najdete taky několik dalších přístupů, které se svými modely pokouší simulovat dav, jejich výhody a nevýhody. V závěrečné části práce je zmíněno vytvoření grafického prostředí pro simulaci pohybu boidů v nástroji Blender pro vytvoření výsledné 3D aplikace.

Abstract

This work primarily deals with C.Reynolds's model of flocking – BOIDS – and uses the model as a basis for creating a swarm simulation. It discusses methods for obstacle avoidance and principle of forces arbitration (flocking rules, obstacle avoidance and goal satisfying) to properly avoid conflicting of behaviours. Furthermore some of other approaches to flocking simulation are mentioned while their pros and cons are taken up. Last but not least proceeding of creating a graphic environment in Blender for demonstrating boids behavior in the final 3D application is described.

Klíčová slova

Simulace davu, shlukování, distribuovaný model chování, multiagentní systémy, Boids, vyhýbání se překážkám, JMonkey, Blender, XSteam.

Keywords

Swarm simulation, flocking, distributed behavioral model, multiagent systems, Boids, obstacle avoidance, JMonkey platform, Blender, XSteam.

Citace

Radek Burda: Simulace davu metodou BOIDS, bakalářská práce, Brno, FIT VUT v Brně, 2013

Simulace davu metodou BOIDS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Františka Zbořila

.....
Radek Burda
14. května 2013

Poděkování

Zde bych chtěl poděkovat především svému vedoucímu práce Ing. Františku Zbořilovi, Ph.D. za nasměrování k cílené aplikaci, protože zpočátku jsem neměl nejmenší tušení, jak by výsledný model měl vypadat, a za ceněný pozitivní přístup na všech konzultacích, kdy jsem s řešením pořád nebyl spokojený. Dále bych chtěl poděkovat vývojářům modelovacího nástroje Blender, ve kterém jsem mohl zadarmo vytvořit modely, a vývojářům platformy JMonkey, na jejichž enginu jsem 3D demonstraci mohl postavit. Velké díky patří taky mé milované přítelkyni Kláře za trpělivost a podporu v době dokončování práce, kdy jsem namísto času tráveného spolu zarytě seděl u počítače.

© Radek Burda, 2013.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1	Úvod	3
2	Distribuovaný model chování	4
2.1	Inteligence hejna	4
2.2	Multiagentní systém	4
2.3	Historie	5
2.4	BOIDS – simulace shluků	5
2.4.1	Separace	6
2.4.2	Zarovnání	6
2.4.3	Soudržnost	6
2.4.4	Tři pravidla nestačí	7
2.5	Simulace vnímání boidů	7
2.5.1	Objem vnímaného okolí	7
2.5.2	Vliv vzdálenosti okolí	8
2.5.3	Vliv odchylky souseda od směru rychlosti	8
2.6	Vyhýbání se překážkám pro model BOIDS	8
2.6.1	Zatáčení směrem od povrchu	9
2.6.2	Zatáčení pryč od středu	10
2.6.3	Zatáčení podél povrchu	10
2.6.4	Zatáčení k nejbližšímu okraji siluety	11
2.7	Rozšiřující pravidla pro BOIDS	12
2.7.1	Seek	12
2.7.2	Udržení boidů v mezích	13
2.7.3	Plnění cílů	14
2.8	Určení výslednice	14
2.8.1	Výhody přístupu BOIDS	15
2.9	Další přístupy k modelování shluků	15
2.9.1	Model shlukování ryb Tu a Terzopoulou	15
2.9.2	Vicsekův model	18
2.9.3	Model Tornera a Tu	20
2.9.4	Model Tanner a Jadbabaie	21
2.9.5	Model Ofati-Sabera	21
2.9.6	Mapy cest řízené pravidly	23
3	Návrh a Implementace	24
3.1	Použité nástroje	24
3.1.1	JMonkey engine	24
3.1.2	Blender	25

3.1.3	Blender-exporter	25
3.1.4	XStream	25
3.2	Můj postup	27
3.3	Detail implementace	28
3.4	BOIDS pseudoalgoritmus	29
3.5	Vytvoření scény	31
3.5.1	Polygonové meshe a UV texturování	31
3.5.2	Export z Blenderu a import do JMonkey	32
3.5.3	Vytvoření překážek	33
3.6	Vytvoření výsledné aplikace	34
4	Závěr	36
A	Obsah CD	40
B	Tvar konfiguračního souboru	41
B.1	Ovládání aplikace	42

Kapitola 1

Úvod

Simulace davu je dlouhá léta výzvou pro vědce po celém světě. Každý člověk jedná sám za sebe na základě omezeného vnímání, ve výsledku však tvoří ucelený systém chaosu, kde do sebe všechny části systému přirozeně zapadají. Typickým úkazem v přírodě jsou hejna letících ptáků, včelí roje, mravenčí kolonie nebo rybí školky. V každé z těchto shlukujících se entit se její účastníci pohybují nepředvídatelně, přesto jako celý shluk působí organizovaně. Bylo vytvořeno několik modelů založených na celulárních automatech a multiagentních systémech, které se snaží tento fenomén namodelovat a popsat. My se v následující práci zaměříme na jeden z prvních a v mnoha ohledech nepřekonaný model, vytvořený Craigem Reynoldsem, s názvem BOIDS (podle analogie s hejnem ptáků v reálném světě – Bird-like object).

Dokument je rozdělen do dvou velkých kapitol, první z nich nás uvede do problematiky, nastolí základní pojmy a zmíní historický vývoj simulace shlukování. Po krátkém seznámení následuje podrobný popis Reynoldsovy metody – popis chování tvořících shlukování, problematika vyhýbání se překážkám s uvedením různých přístupů k vyhýbání a návod, jak rozhodnout výsledné chování v případě několika kolidujících. Ve zbylé části první kapitoly popíšeme další přístupy a modely, které lze použít na vytvoření simulace shlukování.

Druhá kapitola se věnuje vytvoření aplikace pro demonstraci metody BOIDS. Zde popisuji použité nástroje, postup vytváření aplikace, problémy a zajímavosti, na které jsem při řešení narazil.

V závěru najdete zhodnocení zmíněných modelů pro simulaci shlukování, zhodnocení výsledné aplikace a celé práce, a návrhy na další vývoj a rozšíření stávajících modelů pro dosažení uvěřitelnějších výsledků.

Kapitola 2

Distribuovaný model chování

Modelem chování nazýváme takový matematický nebo fyzikální model, který popisuje chování daného modelu. Distribuovaným modelem chování nazýváme model, který se skládá z množiny jeho podmodelů chování. Konečný výpočet tak je složením jednotlivých výpočtů, přičemž každý jednotlivý výpočet zde zastupuje jeden konkrétní model chování.

2.1 Inteligence hejna

Inteligence hejna je odvětví umělé inteligence, které zkoumá chování hejna jako souhr chování individuálních členů hejna za cílem vytvořit chování, které, když se v množství dá dohromady, tvoří výsledné hejno. I když definice není úplně jasná a jednotná, většinou se jedná o **multiagentní systém**, ve kterém, ve kterém spolu agenty kooperují za účelem vytvoření agregovaného inteligentního chování.

Definice Gerarda Beni: *Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems.*[2]

Definice z Wikipedie: *SI systems are typically made up of a population of simple agents or boids 2.4 interacting locally with one another and with their environment. The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents. Natural examples of SI include ant colonies, bird flocking, animal herding, bacterial growth, and fish schooling.*

2.2 Multiagentní systém

Jestliže si máme vysvětlit pojem **multiagentní systém**, je potřeba rozebrat každé slovo zvlášť.

Systém je v našem chápání modelovaná abstrakce reálného světa, kde spolu pracují prvky

systému, které mění stav tohoto systému.

Slovo **agent** vychází z latinského slova *agentum*, jehož původní význam se překládá jako "ten, kdo jedná". Podobný význam má i agent v české terminologii – agent je osoba jednající v zájmu jiné osoby, společnosti, nebo státního celku. [26]

Náš **uměle vytvořený agent**¹ v našem multiagentním systému zastupuje chování abstraktního člověka pohybujícího se v simulovaném shluku lidí.

2.3 Historie

Následující informace jsou převzaty z článku [12].

Pokud zapátráme v historii a budeme hledat první zmínku simulace shluků v počítačové grafice, narazíme na práci [1], prezentovanou na konferenci **SIGGRAPH** v roce 1985. V tomto krátkém filmu vyletí hejno animovaných ptáků z věžního okna, snese se na nádvoří, kde chvíli krouží, aniž by docházelo ke vzájemným kolizím a nakonec odletí směrem dál do scény. Susan Amkarut svým modelem chování ptáků předvedla průkopnickou sílu behaviorální animace. Video je k nalezení v přílohách pod heslem **Eurythmy Motion Studies (1985)**.

Technika použitá Susan Amkarut byla konceptuálně velice zajímavá, přestože nebyla navržena přímo pro modelování shluků. Craig Reynolds ji popsal v jedné ze svých publikací [12] na základě osobní komunikace s autorkou [19].

Software se formáně nazývá "The force field animation system", tedy "Animační systém s využitím silových polí". Silová pole jsou blízká okolí objektů (jak statických, tak pohybujících se), která jsou definována **silovými maticemi**. **Silová matici** definuje odpuzující sílu působící v okolí objektu. Pokud se tyto silové pole dvou objektů setkají, jsou objekty od sebe odpuzovány – asi jako dva stejné póly magnetu. Animátor pouze definoval velikosti těchto polí, nastavil objekty do počátečních pozic, nastavil orientace a rychlosti objektů, zbytek simulace pak už běžel automaticky.

2.4 BOIDS – simulace shluků

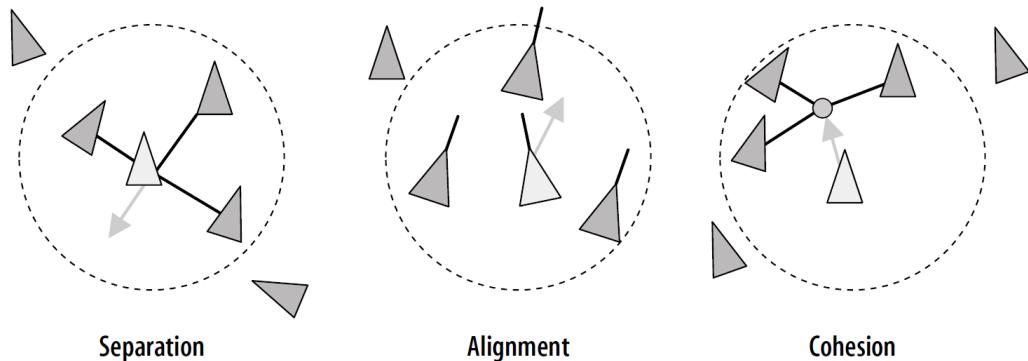
V roce 1987 vytvořil Craig Reynolds počítačovou simulaci letu ptáku v hejně. Jeho distribuovaný model byl inspirován sociálním chováním ryb a ptáků, fungoval pouze na základě vektorových výpočtů v 3D souřadnicovém prostředí. Jednotlivé agenty (prvky distribuovaného modelu) nazval podle shlukujících se ptáků, kteří ho svým chováním fascinovali - **BOIDS** [10] (Bird-OID objects).

Základní model počítá s těmito třemi pravidly, které v kombinaci a následující precedenci tvoří dojem realistického shlukování:

- Vyhýbání se vzájemným kolizím - zatáčení za účelem vyhnutí se hromadění se sousedy
- Srovnání rychlostí - zatáčení do směru stejného jako rychlosť sousedů
- Shlukování se do středu - zatáčení středu okolního hejna

¹Umělý agent je člověkem vytvořená entita, která v prostředí, do kterého je umístěna, jedná samostatně v prospěch svého klienta

Tyto tři pravidla nazval Separace (angl. Separation), Zarovnání (angl. Alignment) a Soudržnost (angl. Cohesion).



Obrázek 2.1: Reynoldsova pravidla shlukování

2.4.1 Separace

Tato síla zajišťuje, že se jednotlivé boidy nesráží, zabraňuje hromadění boidů na malé ploše a zajišťuje tak přirozenou rovnoměrnost vzdálenosti boidů od sebe.

Nevnímá rychlosť svých sousedů, zavírá pouze na jejich poloze.

V našem příkladu je tato síla spočítána jako síla směrující od bodu váženého průměru polohy sousedů. Hodnota určující váhu je nepřímá úměra vzdálenosti od souseda - jak vážit vzdálenost se dozvídáme v sekci 2.5.2.

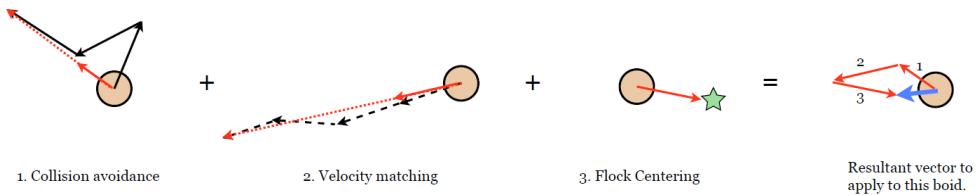
2.4.2 Zarovnání

Díky této síle mají jednotlivé boidy tendenci směrovat stejným směrem a tak vytváří hejno. Narozdíl od separace 2.4.1 je závislá pouze na rychlosti sousedů, poloha sousedních boidů ji pro výsledný výpočet nezajímá.

Zároveň je **srovnání** prediktivní metoda vyhýbání se kolizím - tzn. pokud bude boid udržovat stejný vektor rychlosti, jako jeho kolizím se vyhýbající sousedé, je dost nepravděpodobné, že by mohl v nejbližší době narazit. Udržování stejné rychlosti také zajišťuje téměř neměnnou vzdálenost sousedících boidů od sebe a hlídá plynulosť probíhající shlukové simulace.

2.4.3 Soudržnost

Boid chce být poblíž středu shluku. Protože každý boid vnímá okolní svět jinak, střed shluku tedy znamená střed poloh okolních členů hejna. Nezabráňuje tak boidům rozdělit se například před překážkou na více shluků, protože jej směruje pouze do středu **“lokálního hejna”** tvořeného jeho sousedy. Individuálnímu vnímání každého boidu je věnována sekce 2.5.



Obrázek 2.2: grafické znázornění skládání chování boidů

2.4.4 Tři pravidla nestačí

Boids popisuje 3 hlavní pravidla (viz výše). Craig Reynolds i Steven Woodcock [18] přidávají ke stávajícím chováním další sílu, kterou nazývají **Vyhýbání se překážkám**. Správné vyhýbání se překážkám utvrdí diváka o realističnosti simulace, jaké exitují přístupy a jak budeme kolize řešit my si povíme v sekci věnované vyhýbání se překážkám 2.6. Naši simulaci ale potřebujeme kontrolovat, a tak nám 3 chování tvořící náhodně se pohybující shluk a čtvrté chování uhýbající překážkám nemůže stačit. V této práci popíšeme ještě další chování 2.7, která nám umožní více ovládat simulaci, a také si popíšeme, jak zacházet s více chováními zároveň tak, abychom předcházeli vzájemným kolizím mezi chováními a nadále dostávali uspokojivé výsledky simulace v sekci 2.8.

2.5 Simulace vnímání boidů

Technika BOIDS nesimuluje přímo smysly opravdových zvířat jako čich, sluch nebo postranní čáru u vodních obratlovců² [8, 7], přesto se snaží simuloval výsledný směr pohybu, který by byl výsledkem kognitivních a vjemových procesů.

2.5.1 Objem vnímaného okolí

Pro poznání simulovaného jedince pohybujícího se ve shluku je důležité, kolik informací o svém okolí od simulačního prostředí dostane. Každý boid má přístup ke všem objektům a ostatním boidům vrámci celé simulace. Reálná shlukovaná zvířata ovšem veškeré okolí nevnímají, vnímají pouze malou (nejbližší) část, kterou jim jejich vnímací mechanismy dovolí zpracovávat.

Schopnost každého boidu mít kompletní informace o celém světě je tedy nejenom nerealistická, ale vede k fatálním chybám simulovaného modelu chování. Jedna z původních verzí BOIDS implementovala model globálního shlukování (každý boid měl tendenci směrovat do jednoho globálního středu). Toto způsobovalo nechtěné a realističnost ubírající chování, po chvíli se totiž všechny boidy nasměrovaly na globální střed, kde se setkaly, tam si jen vyměnily strany. V tomto oscilujícím průběhu se vedla celá simulace. Sérií experimentů bylo prokázáno, že distribuovaný pohyb několika objektů, který nazýváme shlukování (angl. flocking, herding) **musí implementovat pro každý boid pouze omezené lokální vnímání** jedinečné pro každý boid.

²Ryba si pomocí tlakových změn vnímaných postranní čárou vytváří imaginární obraz strukturovaného uskupení, drží si tak určitou vzdálenost od svých sousedů a zároveň udržuje stejnou rychlosť.

2.5.2 Vliv vzdálenosti okolí

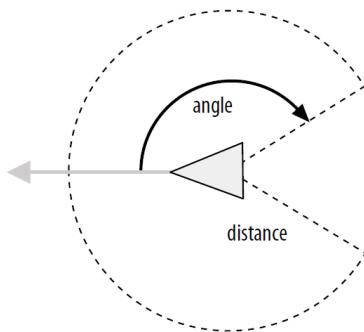
Tři základní chování 2.4, které vytváří realističnost modelu, jsou omezovány **lokálním vnímáním** každého boidu. Lokální vnímání je nepřímo závislé na vzdálenosti dvou aktuálních boid-objektů. Původní implementace vážila jednotlivé požadavky vrátce chování lineární funkcí vzdálenosti. Brian Partridge ve své studii [8] zjistil následující:

“A fish is much more strongly influenced by its near neighbors than it is by the distant members of the school. The contribution of each fish to the influence is inversely proportional to the square or the cube of the distance.”

Slova Partridge si vezmeme k srdci a tak pro zvýšení vlivu nejbližších boidů budeme používat váhu dle čtvercové vzdálenosti, namísto původní lineární.

2.5.3 Vliv odchylky souseda od směru rychlosti

Pro zvýšení důvěryhodnosti simulace můžeme při vnímání lokálního okolí vzít v úvahu i úhel, ve kterém vidí boid svého souseda ve stádu (tedy úhel mezi vektorem rychlosti a bodem polohy souseda, který uvažujeme vůči středu boidu, s jehož vektorem rychlosoti počítáme) – viz obrázek 2.3. Při tomto omezení **je správné nastavení úhlu vnímání kritické**, může simulaci jak vylepšit, tak i naprosto zruinovat. V naší aplikaci má boid rozhled 90° na každou stranu, protože rozhled 180° odpovídá zornému poli reálného člověka.



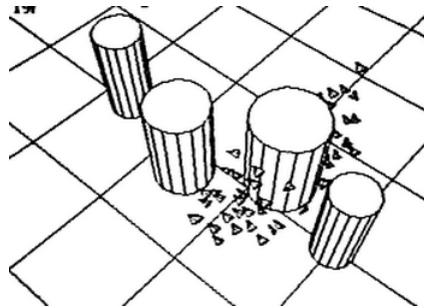
Obrázek 2.3: grafické znázornění vnímání okolí boidu

Pokud se chceme co nejbližše přiblížit simulaci vnímání, můžeme váhu jednotlivých požadavků nepřímo vážit úhlem. Tzn. že boid přímo vepředu má bětší vliv než boid sousedící napravo anebo nalevo. V naší implementaci tento dodatečný výpočet nepoužíváme.

2.6 Vyhýbání se překážkám pro model BOIDS

Základní tři pravidla fungují dobře do té doby, dokud máme úplně volnou scénu a hejno může cestovat, kam se mu zachce. Takovýto případ je jistě možný, avšak prchá od reality pryč mílovými kroky. Stejně jak jsou objekty v reálném světě neprůchozí, tak i my chceme, aby byly neprůchozí v naší simulaci. Stejně jak my, když vidíme překážku daleko před sebou, začneme s předstihem zatáčet a výsledná trajektorie pohybu je pokud možno co nejplynulejší. Pro simulaci takového plynulého a přirozeně-vypadajícího předcházení

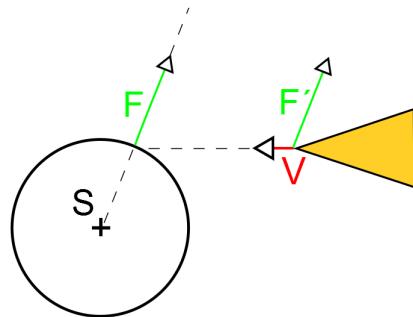
kolizím musíme zvolit správnou heuristiku.



Obrázek 2.4: boidy vyhýbající se válcovitým překážkám

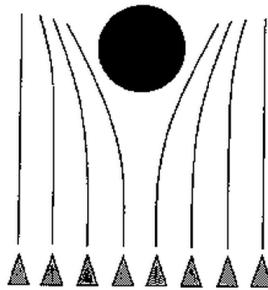
V roce 1988 vydal Craig Reynolds dodatek ke svému představení **BOIDS** s názvem **Not Bumping Into Things**^[11], kde popisuje různé přístupy předcházení kolizím. Pracuje zde s překážkami, které jsou jednoduše geometricky popsatelné - konvexní několikaúhelníky nebo válcovitá tělesa. Boid pak implementuje jednoduché rozhodování, kdy, kam a jak bude zatáčet, aby předešel kolizi. Níže uvedené přístupy počítají s tím, že každý boid má odzačátku absolutní přehled o celém překážkovém světě a každá překážka se dá geometricky vyjádřit a zároveň je po dobu simulace její poloha a popis neměnný.

2.6.1 Zatáčení směrem od povrchu

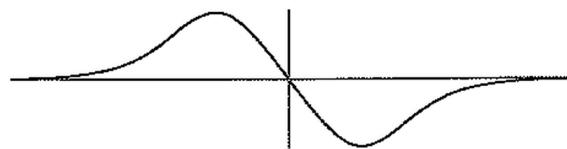


Obrázek 2.5: síla odpuzující boid od blízké překážky

První heuristika je založena na modelu Susan Amkaruth - "Force field" [1]. Kolem překážky je odpuzující pole, které tlačí bližící se objekt pryč od překážky. Na obrázku 2.5 je znázorněna odpuzující síla F směrující od povrchu překážky. Přenesená síla F' svým působením točí rychlosť boidu tak, aby předešel naražení do překážky. Nicméně **problém nastává**, pokud se boid přiblížuje k překážce v takovém úhlu, že směr odpuzující síly je přesně opačný – boid pak v lepším případě zpomalí nebo zastaví, v horším případě projde překážkou. Průběh této odpuzovací síly je znázorněn na obrázku 2.7 funkcí horizontální offsetu směru boidu vůči povrchu překážky. Hodnota na ose Y udává zrychlení působící na boida a tudíž měnící jeho směr rychlosť. V průniku os X a Y můžeme vidět **mrtvý bod**.



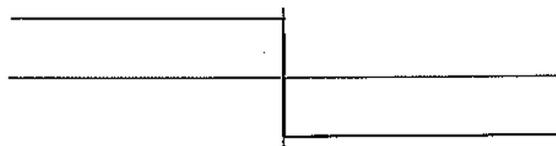
Obrázek 2.6: odklon cesty boidu od překážky díky odpuzující síle



Obrázek 2.7: průběh točivého zrychlení s mrtvým bodem uprostřed

2.6.2 Zatáčení pryč od středu

Meotda točení pryč od středu počítá s překážkou jako s bodem, od kterého se musí boid dostat co nejdál. Pokud boid spatří překážku, je tlačný zrychlením směrujícím pryč od překážky (doleva nebo doprava) do té doby, než se jeho rychlosť otočí natolik, že překážku nevidí. V případě směrování rychlosti boidu přímo na střed překážky rozhodneme explicitně, zda bude zatáčet vlevo nebo vpravo. Tato metoda přináší uspokojivé výsledky, avšak od reality se malinko vzdaluje. Pohybující se objekt totiž začne zprudka zatáčet jakmile zjistí, že před ním je překážka. Výsledný zatáčivý pohyb pak může působit trhaně. Rozložení působícího zrychlení na boid jako funkci horizontálního offsetu najdete na obrázku 2.8.



Obrázek 2.8: točivé zrychlení v případě metody **pryč od středu**

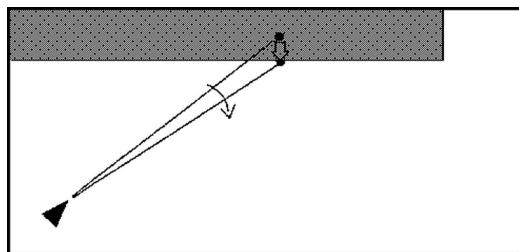
2.6.3 Zatáčení podél povrchu

Tento přístup se dá názorně vysvětlit na příkladu:

Představme si, že procházíme temnou chodbou, tak temnou, že nevidíme vůbec nic než tmu a pouze šmátráme rukama před sebou, abyhom nenarazili do stěny. Když nahmatáme stěnu, otáčíme se pryč od stěny tak, abyhom pořád pokračovali ke vztýčenému cíli – konci chodby.

V implementacích této metody nemáme ruky, tak počítáme se „sondou“, která je předsunutá boidu ve směru rychlosoti. Sonda při nárazu rozhoduje, kam a jakou sílou zatáčet pro

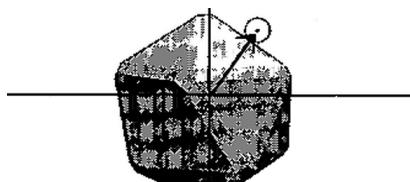
vyhnutí se kolizi. Na obrázku 2.9 je boid s předsunutou sondou, která právě zaznamenala náraz a dává boidu informaci o tom, kam by se měl směrovat v dalším kroku. Velikost točivého zrychlení může být vážena hloubkou sondy v překážkovém objektu, takže boid bude zatáčet zpočátku mírně, ostře zabere, až když bude muset. Velice záleží na vzdálenosti sondy a na správném nastavení vážení točivého zrychlení. Metoda za nízkou cenu výpočtu přináší uspokojivé výsledky.



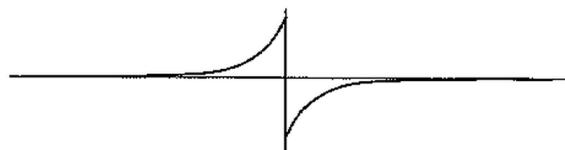
Obrázek 2.9: odklon předsunuté sondy od povrchu

2.6.4 Zatáčení k nejbližšímu okraji siluety

Zde se nezaměřujeme ani na sklon povrchu překážky vůči boidu ani na polohu středu překážky. Zajímá nás pouze, zda je překážka vidět a kde je **nejbližší hrana obrysu překážícího objektu** z pohledu přibližujícího se boidu. Tento bod, posunutý ven z překážky o velikost těla boidu a případnou další rezervu (záleží na programátorovi), bude cíl, na který budeme boid směrovat. Viz obrázek 2.10.



Obrázek 2.10: směrování za nejbližší okraj překážky



Obrázek 2.11: správný průběh chování točivého zrychlení

Jak je vidno na obrázku 2.11, pokud boid směruje na střed překážky, bude zatáčet ostře, pokud směruje na okraj, bude zatáčet mírně. Nicméně i přes různě testované koinace velikostí, tvarů, vzdáleností, rychlostí a maximální velikosti točivého zrychlení, i ten nejostřejší

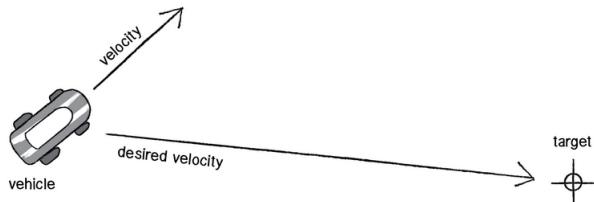
manévr nemusí stačit na to, aby boid překážku minul (pokud je blízko a přibližuje se vysokou rychlostí). Uvedená heuristika je rozhodně jedno z nejlepších, na kterých dále stavět. Složitější implementace této metody počítají s parametry "míra zatáčení" nebo "čas do nárazu" pro určení, který z bodů kolem siluety by mohl být nejvhodnější pro cílené vyhnutí se.

2.7 Rozšiřující pravidla pro BOIDS

Doposud jsme si popsali celkem čtyři chování, která zajistí shlukování boidů ve volném prostoru (zarovnání, sdružování), vyhýbání se kolizím mezi boidy samými (separace) a vyhýbání se kolizím s překážkami ve světě (vyhýbání se překážkám). V následující sekci si popíšeme, jak můžeme tuto náhodně-letící hromadu shlukujících se boidů ovládat při udržení důvěryhodnosti simulace.

2.7.1 Seek

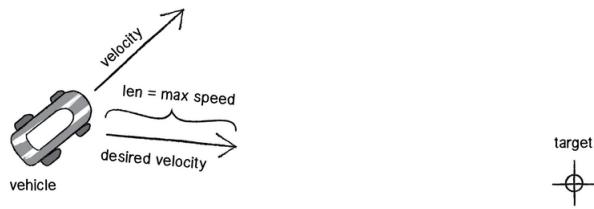
Chování, které Reynolds ve svém papíře s popisem chování pro autonomní agenty [13] nazval **seek**³, přináší nástroj směrování boidu ke stanoveném bodu v prostoru plynulým zatočením. **Seek** je v matematickém vyjádření síla (zrychlení), které svým působením způsobí zatočení boidu tak, aby jeho rychlosť směrovala na stanovený bod v prostoru. Tuto sílu spočítáme jako rozdíl rychlosti a vektoru tvořeného rozdílem mezi body polohy boidu a cíleného bodu: $steering_force = desired_velocity - current_velocity$. Tyto dva vektory jsou znázorněny na obrázku 2.12. Obrázky vztahující se k chování **seek** jsou převzaty z knihy Daniela Shiffmana [16].



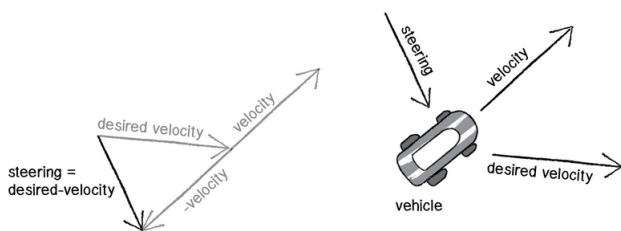
Obrázek 2.12: síla směrující od boidu ke cílovému bodu

Požadovaná rychlosť na obrázku 2.12 je přehnaně velká a slouží pouze pro demonstraci. Pokud by vzdálený bod byl milióny kilometrů vzdálený, požadovaná rychlosť by naprostě zastínila aktuální rychlosť a všechny dosavadně zmíněné chování bychom mohli zahodit. Proto je třeba požadovanou rychlosť zmenšit na maximální možnou rychlosť pohybu (obrázek 2.13), potom můžeme dostat adekvátní výsledek točivé rychlosťi. Na levé straně obrázku 2.14 vidíme graficky znázorněnou vektorovou operaci a na pravé straně výsledný náhled všech 3 sil.

³nenašel jsem odpovídající český překlad

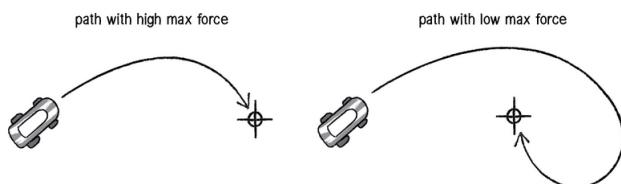


Obrázek 2.13: limitace požadované rychlosti



Obrázek 2.14: grafické znázornění vektorů použitých pro výpočet

Díky tomuto přístupu točivé zrychlení škáluje s rychlosťí boidu. Schopnost zatáčení můžeme kontrolovat nastavením velikosti výsledného točivého zrychlení, viz obrázek 2.15.



Obrázek 2.15: vliv nastavení velikosti síly **seek**

2.7.2 Udržení boidů v mezích

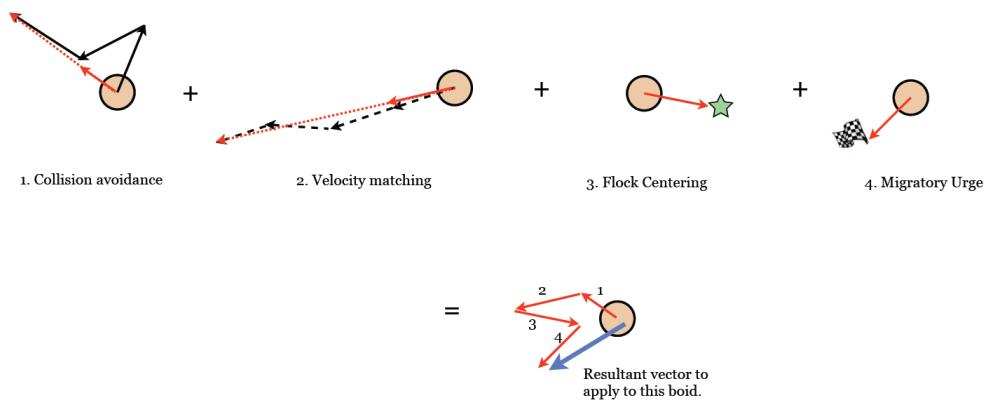
Podle výše uvedených chování mohou boidy letět kamkoliv, a pokud je s kamerou nebudeme stíhat, uletí pryč ze scény a úplně se ztratí z dohledu. Proto potřebujeme mechanismus, který udrží pohyblivý shluk ve scéně, na kterou se může dívat statická kamera. Některé implementace pro demonstrační účely pouze definují hranice - souřadnice X_{max} , Y_{max} a Z_{max} - po jejichž překlenutí je boid přesunut na opačnou stranu scény podle odpovídající překročené souřadnice. Pokud se tedy objekt pohybuje po ose X doprava, dostane se až za souřadnice (X_{max}, Y, Z) , bude přesunut na souřadnice $(-X_{max}, Y, Z)$ a odtud pokračovat zpět do scény. Vektor rychlosti přesunutého boidu nebude nijak ovlivněn hned, ale bude změněn v pár následujících krocích - kvůli přesunu se totiž úplně změnilo okolí boidu.

Uvedený přístup je velice jednoduchý, ale poněkud postrádá realističnost. My použijeme přístup stejně jednoduchý, avšak přináší uspokojivější výsledky. Budeme předpokládat stejné hranice, jako v případě výše uvedeném - body X_{max} , Y_{max} a Z_{max} . Pokud boid překročí tyto hranice, nepřesuneme ho z pozice X_{max} na $-X_{max}$, ale přidáme mu dočasné chování s vysokou prioritou, které ho bude směrovat na bod, na který by se v uvedeném případě teleportoval. Aby boid plynule zatočil, použijeme chování **seek**, které jsme si představili v minulé kapitole 2.7.1.

Až se vrátí ze **zakázaného pásma**, dočasné chování zase vypneme a boid bude pokračovat dál se svým shlukem v oblasti, ve které ho chceme držet.

2.7.3 Plnění cílů

Ještě větší moc nad ovládáním pohybu shluku nám dává síla, kterou si nazveme **plnění cílů**. Tak budeme směrovat hejno na námi zvolenou pozici. Sílu plnění cílů spočítáme jako výsledek rozdílu cílené pozice a aktuální pozice, tedy síla směřující na konkrétní cíl ve vhodné velikosti, váže a precedenci v kombinaci s ostatními silami. Směrováním každého boidu ke stanovenému cíli docílíme pohyb celého shluku k vytyčenému cíli, přičemž o zachování shlukování se postarájí ostatní pravidla s vyšší váhou.



Obrázek 2.16: zobrazení precedenze chování - collision avoidance nejdůležitější

Jestliže chceme mít vysokou kontrolu nad simulací (například chceme, aby hejno proletělo tunelem nebo prošlo konkrétní ulicí), vybereme několik kontrolních bodů, kterými chceme naše boidy vést. Těmito body interpolujeme křivku, náš cílový bod povedeme to této křivce před hejнем, které se za ním bude hnát, jak králik na mrkev.

2.8 Určení výslednice

Každé pravidlo implementuje nezávislé chování. Jakoby říkalo: *"Kdybych tomu tady velel, pokračoval bych tamtudy."* Pokud se agent například blíží k překážce, 3 pravidla říkají, aby směroval doleva, a vyhýbání se překážkám tvrdí, že je pro vyhnutí se kolizi nutné zahnout doprava, můžou se opačné síly vynulovat, agent tak bude pokračovat rovně, až narazí do překážky. Proto je potřeba požadavky správně rozsoudit a rozhodnout správnou výslednicí.

Právě v rozsuzování chování a určení výslednice má metoda velký potenciál pro zlepšování, mohou se použít různé metody umělé inteligence, jako např. **expertní systémy**⁴, Reynoldsovo řešení však pro jednoduchost implementace nabízí dvě řešení:

1. Zprůměrování sil.

Abychom alespoň částečně vyřešili problém priorit, každou sílu vynásobíme konstantní váhou, a tak dostaneme vážený průměr. Takovéto řešení je velice jednoduché a funguje "poměrně dobře". Přesto ale může dojít ke vzájemnému vylučování protichůdných sil.

2. Akumulace sil (Force accumulation).

Tento přístup je daleko uspokojivější a podle Stevena Woodcocka [18] dokonce nejlepší možný vrámci jednoduchosti implementace. Spočívá v přísné prioritizaci požadavků na zrychlení a kontrolování velikostí zrychlení. Každý požadavek je spočítán zvlášť v pořadí daném prioritou. Velikost každého požadavku na zrychlení je přidávána do akumulátoru, jehož velikost je omezena – maximální možné zrychlení v jednom kroku je parameter každého boidu. Pokud kumulovaná velikost přesáhne povolenou, nejsou další zrychlení v aktuálním kroku aplikovány. Tento přístup požívám i já v uvedené implementaci 3.4.

2.8.1 Výhody přístupu BOIDS

Boids přináší realistickou simulaci shlukování při jakémkoli počtu agentů. Můžeme hloček směrovat tam, kam chceme, navíc implementuje vyhýbání se kolizím s překážkami.

Boid pro výpočet svého dalšího kroku nepotřebuje **žádnou paměť**, stačí mu pouze pozice a rychlosti boidů sousedních. Za kvadratické složitosti algoritmu - $O(N^2)$ – si pro výpočty v každém kroku vystačíme se základníma vektorovýma operacema – nepočítáme žádné šílené rovnice na hranici matematiky a neznáma.

V případně paralelní implementace – 1 vlákno pro každý boid – se dostaneme na lineární algoritmickou náročnost - $O(N)$. Největší výhoda BOIDS tkví v jeho jednoduchosti a intuitivnosti. V poslední době se dostává čím dál více do podvědomí nadšených programátorů, protože tento jednoduchý (nazveme ho geniální) přístup nebyl vpodstatě (ano, vpodstatě, protože existují i lepší modely se složitými matematickými popisy) doposud překonán, a taky proto, že s použitím boids a neomezenou kreativitou můžeme dostat nekonečně zábavné výsledky.

2.9 Další přístupy k modelování shluků

Reynoldsův způsob simulace shluků není samozřejmě jediný, i když je ve své komplexnosti nejjednodužší. Zde se pokusím popsat pár dalších modelů, které byly vytvořeny vrámci výzkumných projektů po celém světě.

2.9.1 Model shlukování ryb Tu a Terzopoulose

Xiaoyuan Tu společně s Demetri Terzopoulos ve své výzkumné práci [24] popisují komplexní model uměle vytvořené ryby. Model definuje motorický pohyb, fyzikální vlastnosti

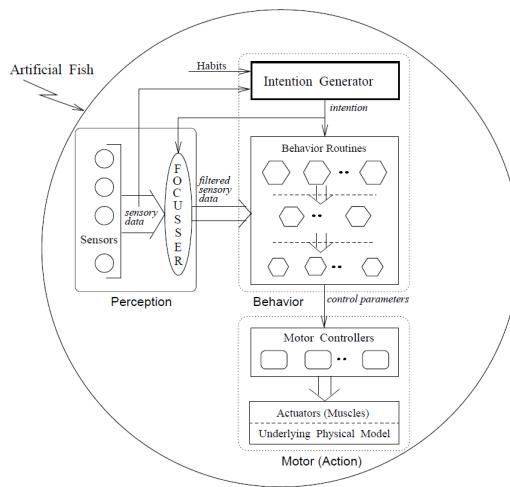
⁴Expertní systémy, ve zkratece, nazýváme odvětví umělé inteligence, které na základě implementované inteligence a známým datům pomáhá rozhodovat nejlepší z možných řešení.

(hmotnost, rychlosť, síla), vnímání okolního světa a pokročilé chování – hlad, libido, strach a individuální zvyky.

Architektura modelu

Obrázek 2.17 ilustruje podsystémy vnímání, chování a pohybu.

Pohybový systém pracuje s několika uzly rozmístěnými na povrchu rybího těla a kontrolery, které zajíšťují stahování uzel k sobě. Uzly společně s koontrolery pohybu tvoří mechanický model ryby odpovídající skutečné anatomii, tedy realistickému pohybu ryby v přírodě. Počet uzel je volen tak, aby model vypadal zároveň realisticky a zároveň výpočetní náročnost zůstala vrámci modelu výhodná. V popisovaném modelu je ryba tvořena 23 uzel.
2.19



Obrázek 2.17: řídicí a informační tok umělé ryby

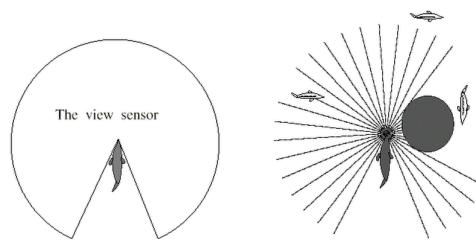
Model vnímání

Systém vnímání využívá virtuální senzory na povrchu rybího těla a dává tak modelu informace o okolním světě.

Senzory vnímání jsou dvojího typu:

- teplotní sensor – podává informace teplotě v okolí středu rybího těla
- zrakový sensor – monitoruje barvy, vzdálenosti, velikosti a identifikace okoních objektů

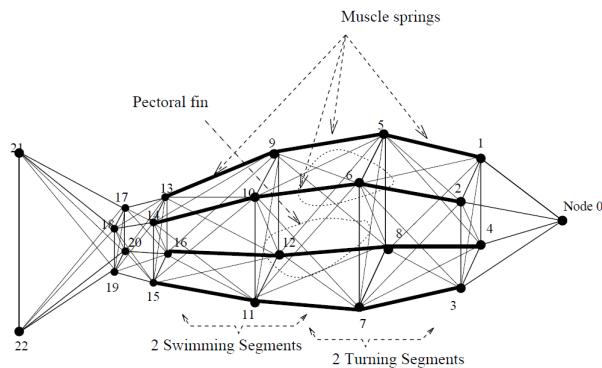
Umělá ryba má částečný přístup ke geometriím, materiálům a osvětlení scény. Zrakový senzor emuluje jedno oko s rozhledem 300° a dohledem odovídajícím dohledu v čisté nezakalené vodě.



Obrázek 2.18: ilustrace zrakového sensoru ryby - na obrázku vpravo vidí pouze rybu na levé straně vepředu

Model chování

Model chování umělé ryby je prostředníkem mezi systémem vnímání a motorickým systémem. Generátor cílů (angl. Intention generator), rybí "kognitivní" centrum vytváří pohybové požadavky, které jsou skrz motorický systém přenášeny na pohyb ryby ve světě.



Obrázek 2.19: model táhel spojujících hmotnostní body

Animátor parametricky nastaví vnitřní osobnost ryby, její zvyky, určí pohlaví, zda má ryba radši svělo nebo tmu atd. Generátor cílů kombinuje zvyky s příchozím proudem informací od senzorů a vytváří tak cíle jako hon na kořist a následné jezení nebo únik před predátorem. Generátor cílů také filtruje informace od senzorů pouze na ty, které jsou nezbytné pro splňování aktuálních cílů. Například, pokud je záměrem hledání jídla z důvodu hladu, umělá ryba se zaměřuje pouze na senzorové informace o okolní potravě. Generátor cílů spouští v každém simulačním kroku rutinu, která vyfiltruje odpovídající senzorové informace a vypočítá motorický pohyb potřebný pro posunutí ryby o krůček dál ke splnění aktuálního cíle.

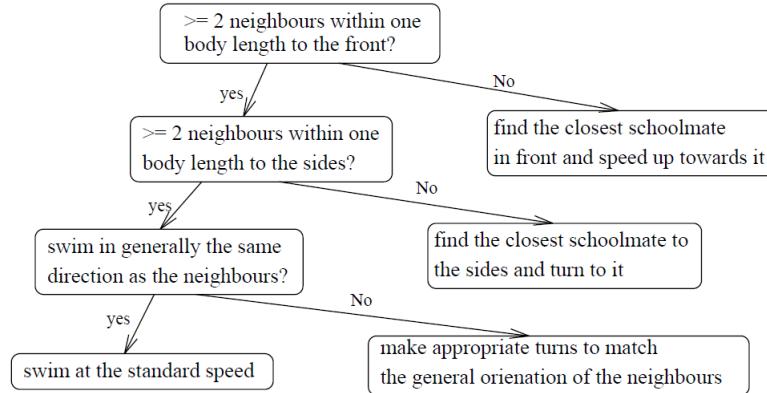
Rutina shlukování

Model Tu a Terzepulouse simuluje rybí motorické chování, vnímání, zvyky, potřeby na základě okolního vnímání. Simuluje 3 druhy rybího chování emitovaného generátorem cílů:

- Ryby predátory

- Ryby kořist
- Ryby pacifisty

Cílem této práce ale není detailní popis modelu chování Tu a Terzepulouse, takže dál zmíňme pouze pro nás nejdůležitější část, a to **rutinu shlukování ryby kořisti** - obrázek 2.20.



Obrázek 2.20: algoritmus shlukování v modelu Tu a Terzopoulose

Generátor cílů zajišťuje, že se ryby nedostanou příliš blízko sobě, protože cíl vyhýbání se kolizím má nejvyšší precedenci. Jakmile se rybí školka začne formovat, je snížena citlivost vůči kolizím, aby se mohlo dotvořit ucelené hejno a výsledný dojem tak vypadal realističtěji. Pokud velká školka narazí na překážku, samostatné chování každé ryby se snaží vyhnout překážce a školka se tak může rozdělit do dvou menších skupin. Po překonání překážky se spustí shlukovací rutina, která zajistí spojení dvou skupinek do původní velké školky.

2.9.2 Vicsekův model

Startovním bodem pro následující 3 modely mně byl článek Kishore Dutta z roku 2010 [4]. Tamás Vicsek, inspirovaný Reynoldsovým modelem BOIS, v roce 1995 ve spolupráci se svým týmem předložil jednoduchý simulační model (celulární automat) [25], podobný popisu feromagnetu, a ukázal, že je tak schopný simulovat shlukování. Částice v modelu mají tendenci posouvat se směrem, který je vypočítán **zprůměrování směrů částic v blízkém okolí**. Zároveň je směr ovlivňován **šumem prostředí**. Simulace byla postavena v 2D čtvercovitém prostředí. V čase $t = 0$ bylo N částic rozloženo na náhodných pozicích (x_i, y_i) . Velikosti rychlostí byly jednotně stanoveny konstantou v a počáteční směr – úhel θ_i – byl inicializován náhodnou hodnotou. V každém (i -tém) kroku se pozice a směr počítaly následovně:

$$x_i(t+1) = x_i(t) + v \cos \theta_i(t) \quad (2.1)$$

$$y_i(t+1) = y_i(t) + v \sin \theta_i(t) \quad (2.2)$$

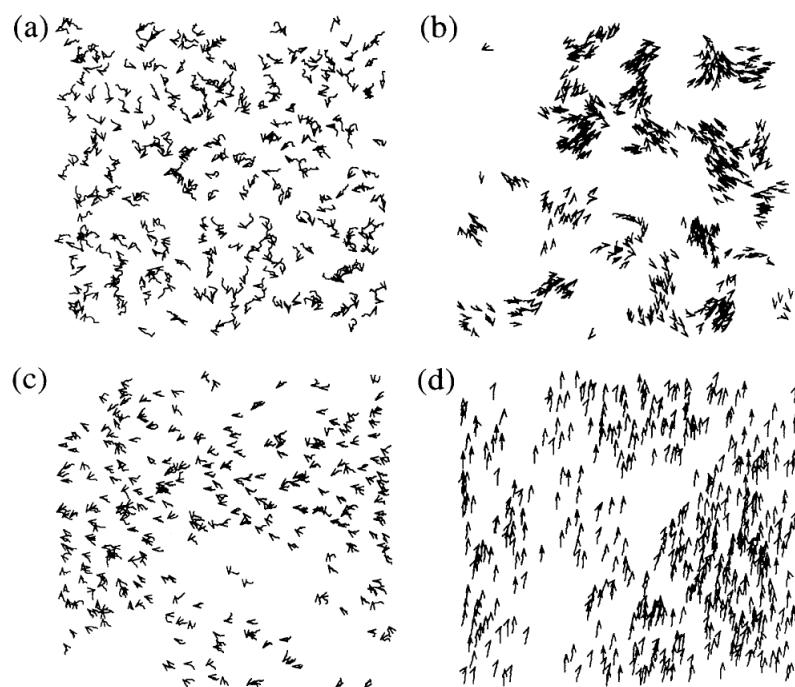
$$\theta_i(t+1) = \langle \theta_i(t) \rangle_r + \Delta\theta \quad (2.3)$$

$\langle \theta_i(t) \rangle_r$ je průměrný směr rychlostí sousedících částic do vzdálenosti r . Vzdálenější částice už nemají na výsledný pohyb vliv. Náhodné číslo $\Delta\theta$ je vytvořeno generátorem s rovnoměrným rozložením z intervalu $[-\eta/2, +\eta/2]$ a vytváří tak **šum prostředí**, který ovlivňuje

výsledný směr pohybu.

Shluk se vyvýjí na základě opakování těchto pravidel. Rovnice 2.3 říká, že se každý člen shluku sám snaží srovnat stejným směrem pohybu (první předpoklad) a zároveň je chován náhodně odchylováno (druhý předpoklad). Měněním parametrů pro citlivost (vzdálenost vlivu) a šumu prostředí byly zjištěny 4 různé chování:

1. Vysoká úroveň šumu, malá citlivost: částice se pohybovaly nezávisle
2. Nízká úroveň šumu, malá citlivost: částice tvořily skupinky, které se soudržně pohybovaly náhodnými směry
3. Vysoká úroveň šumu, veliká citlivost: částice se pohybovaly náhodně s občasnými shodami pohybu
4. Nízká úroveň šumu, veliká citlivost: všechny částice se pohybovaly ve stejném spontánně voleném směru



Obrázek 2.21: ilustrace rychlostí na závislosti šumu a citlivosti

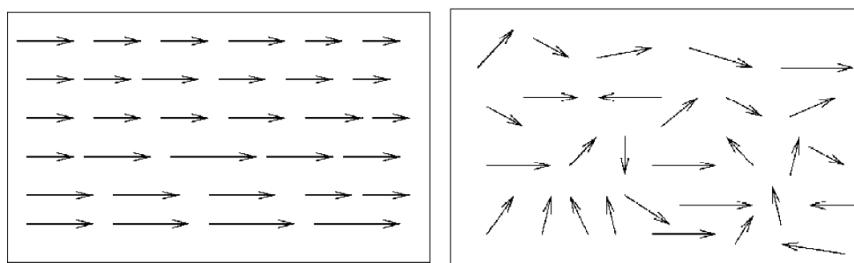
Popis k obrázku 2.21: Obrázek je rozdělen na 4 části – (a), (b), (c) a (d). Část (a) ukazuje počáteční stav v čase $t = 0$, na (b) jsou nízké hodnoty šumu a citlivosti, na (c) vysoký šum a citlivost, na (d) pak nízký šum a vysoká citlivost.

Tyto simulace vykazovaly charakteristické kolektivní chování, které je přirozené shlukům vyskytujícím se v přírodě. Šum má v organizaci shluků výraznou roli. Pro konečnou hustotu

částic v určitém prostředí je nejlepší zarovnání dosáhnuto jednoduše absencí anebo velmi nízkou úrovní šumu \Rightarrow počátek kolektivního pohybu se objevuje v určité úrovni šumu.

Vektor rychlosti individuálního ptáka v hejnu je analogií k magnetickému spinu v železném atomu feromagnetu. Všichni ptáci v hejnu se pohybují zhruba stejným směrem a vytváří tak rozsáhlý letový řád. Toto je analogie s magnetickými spiny, které spolu směrují téměř stejným směrem – obrázek 2.22. Spiny ve feromagnetu interagují pouze s nejbližšími sousedy skrz krátkodobé interakce, které jsou podobné interakcím ptáků v hejně se svým blízkým sousedským okolím.

Úroveň náhodného rušení – $\Delta\theta$ – je analogií teploty. Stejně jak teplotní výchylky vnáší náhodnost do rovnováhy feromagnetu, náhodnost chyby $\Delta\theta$ vnáší stochastický element do problému shlukování. Přechod do strukturovaného stavu (z paramagnetického do feromagnetického) znamená ztrátu symetrie, přesněji ztrátu trvající symetrie spinové rotace. Stejně tak je v modelu shluku narušována symetrie rychlostí, fungující shluk je tak rozpolcen.



Obrázek 2.22: spinové uspořádání - feromagnet vlevo, paramagnet vpravo

Navzdory těmto podobnostem je mezi dvěma modely významný rozdíl. Feromagnet je vyvážené termodynamické uspořádání, kdežto shlukování je nerovnovážný dynamický systém – struktura termodynamické rovnováhy je pevná, kdežto shluk je funkční, pořád se měnící struktura.

Přestože je Vicsekův model nejjednodužší fyzikální model shlukování, jeho jednoduché a lokální výpočty vytváří ohromující komplexní sebeorganizované shluky. Pokud se pravidla používají opakováně, vytváří rozsáhlý řád.

Přes jednoduchost simulace modelu je těžké ho analyticky popsat. Jak to, že lokální interakce tvoří rozsáhlé a soudržné vzory charakteristických velikostí? Co vysvětluje rozlehly řád, který demonstrují hejna v prostoru a čase? Vicsekův model, zdá se, vytváří přirozený systém pro řešení těchto problémů, ale detailní popis těchto procesů, dokonce i pro jednoduché organismy, pořád chybí. Řešení nachází model Tu a Tornera 2.9.3.

2.9.3 Model Tornera a Tu

Z důvodu jednoduchosti a efektivity byl Vicsekův model roky zkoumán. Podle Vicsekových pravidel vytvořili Torner a Tu následující rysy shlukování pro obecnější a matematicky lépe

popsatelný model model:

1. Každý pták se v plynoucím čase pohybuje prostorem dimenzí $d (= 2, 3, \dots)$ a po celou dobu pohybu následuje své nejbližší sousedy.
2. Každý člen hejna komunikuje jenom se svými nejbližšími sousedy pouze za pomocí krátkosáhlé interakce – to je stejné jako se spiny atomů.
3. Každý člen shluku, při své komunikaci se sousedy, dělá chyby, tzn. že “následování” není dokonalé. Chyby jsou stochasticky modelované jako šum v původním modelu.
4. Žádný organismus nemá nařízen žádný preferovaný směr, ve kterém by se měl pohybovat. Spolehlá jenom na směr definovaný jeho sousedy.

Implementováním těchto pravidel vyvinuli Torner a Tu obecnou hydrodynamickou teorii [22], aby mohli popsat chování shluku velkých měřítek. Pravděpodobnostní diferenciální rovnice této teorie pro rychlosť a pole hustot byly postaveny na analogii s magnetickými systémy a tekutými krystaly.

Hlubší detaily o různých fázích shlukování jsou popsány v článku J. Tornera, Y. Tu a S. Ramaswamy [23]. Práce přináší pár zajímavých výsledků, včetně možností rozsáhlého pohybu rádu ve 2D, kvanta predikcí a pozorovaní výchylek v seřazeném stavu.

2.9.4 Model Tannera and Jadbabaie

Model Tannera a Jadbabaie popisuje zákony řízení, které vytváří shlukování a poskytuje teoretické odůvodnění kombinací výsledků z klasické teorie řízení⁵, mechaniky a teorie grafů. Jejich publikace má dvě části. V první části [5] uvažují případ, kdy topologie řídících interakcí mezi agenty je neměnná. Každý agent reguluje svou pozici a orientaci podle stálé skupiny sousedů. Případ, ve kterém se skupina sousedů může měnit v čase na základě relativních vzdáleností mezi agenty je popsána v části 2 [20].

Jejich model využívá teorii grafů a potenční funkci, která je funkcí vzdálenosti mezi dvěma agenty. Potenční funkce je analogií separace a sdružování v modelu Reynoldse [12]. Kvůli matematickému popisu modelu, který není obsahem této práce, proto tedy čtenáře odkážu an uvedené citace. [5, 20].

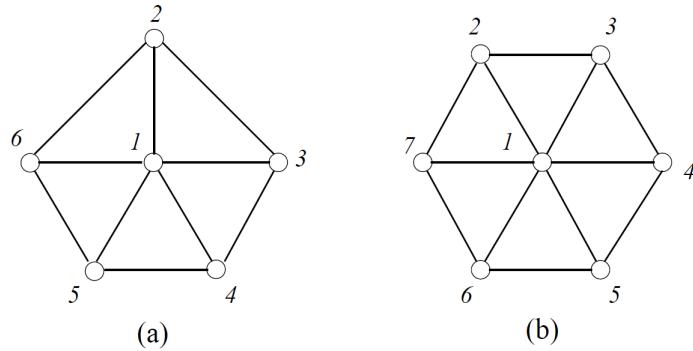
2.9.5 Model Ofati-Sabera

Ofati Saber přichází s řešením [14], které je matematicky popsatelné a navíc **počítá s vyhýbáním se kolizí s překážkami**. Jeho model staví na předpokladech shlukování Reynoldse a na teorii grafů. Popisuje shlukování jako prostorový neorientovaný graf (s_α síť), ve kterém spolu interagují 3 druhy agentů - α, β a γ pomocí vzájemných protokolů.

Agent α reprezentuje hmotný bod v podobě uzlu grafu. Má sférické zorné pole o určitém poloměru a pokud se k sobě přiblíží dva agenty α na tuto vzdálenost, vytvoří mezi sebou hranu. Protokol (α, α) pak obstarává samotné shlukování a to následovně: uzly, které jsou spojené hranou, sdílejí chování **zarovnání** (v analogii Reynoldsových boidů), na této

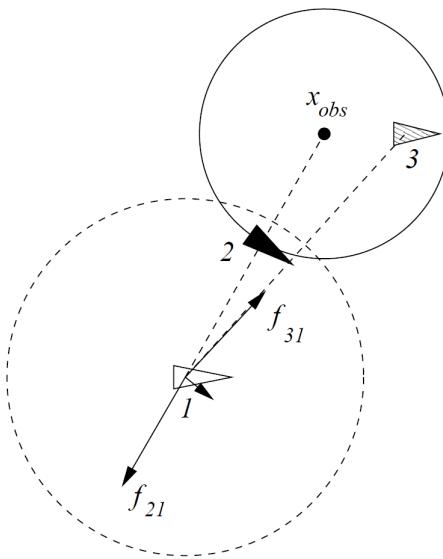
⁵Teorie řízení je naukou o řízení a popisu systému – definice z Wikipedie.

hraně je počítáno stresové napětí⁶. Protokol (α, α) se snaží napětí v celém grafu vyrovnávat ovlivňováním rychlostí (α) agentů – analogie **separace** a **soudržnosti**. Pokud je agent vytlačen za určitou vzdálenost od sousedícího agenta, rozlomí se spojení v grafu, (α, α) protokol už nemůže fungovat a tak odtržený α agent pokračuje dál svou vlastní rychlostí.



Obrázek 2.23: stresové napětí (vlevo) a vyrovnaná struktura grafu (vpravo)

Agenty β a γ slouží k vyhýbání se kolizím s překážkami. β je okrajový bod překážky nejbližší korespondujícímu agentu α a má za úkol jej odpuzovat. Agent γ slouží k vyhodnocení rychlosti pro správné vyhnutí se překážce. Oba dva typy agentů se vytváří pro svého agenta α v moment, kdy vstoupí do vzdálenosti jeho sférického vnímání. Jakmile α přestane vidět překážku, oba agenty zmizí.



Obrázek 2.24: interakce mezi α -agentem (agent 1) and překážkou v podobě agenta β (agent 2) a agenta γ (agent3)

⁶vpodstatě míra odchylky grafu od podobné symetrické struktury

2.9.6 Mapy cest řízené pravidly

Všechny doposud uvedené modely simuluji shlukování, mají ale problémy tam, kde je potřeba komplexnější navigování – např. ve městě, zaledněné místnosti nebo při drsném povrchu v prostoru. Naproti tomu algoritmy plánování tras v robotických systémech jsou plně schopny navigace v těchto komplexních prostředích. V podstatě je použita metoda trasové mapy (dále jen TM), která se většinou vytváří při postprocessingu, vytvářející síť schůdných cest v prostředí. Tento přístup se pokusila skupina výzkumníků z Texasu [6] upravit a použít na simulaci koordinovaného skupinového chování – a úspěšně.

Hlavní rysy přístupu TM jsou:

- TM poskytuje vhodnou abstraktní reprezentaci globálního prostředí.
- Dynamické TM (modifikovatelné uzly a váhy hran) umožňují komunikaci mezi agenty.
- Spojení pravidel s TM umožňují lokální přizpůsobování chování.

TM je spojitý graf, který zakódovává informace o schůdných cestách v prostředí. Každý uzel grafu je jakýsi model robota, který uspokojuje určité přijatelné požadavky. Nazveme TM, ve které jsou váhy hran upravovány podle nových informací sesbíraných členy shluku, **adaptivní** TM. Pokud se agent dostane do oblasti vlivu uzlu adaptivní TM (nebo hrany), agentovo chování je upravováno korespondujícím pravidlem. Pokud není specifikováno žádne pravidlo, pokračuje ve svém původním ochování. Metoda TM je analogicky podobná semaforům v křížovatkách, které tak řídí průjezdy aut. Pravidla mohou být jednoduchá jako “Běž na další uzel v tvé cestě” nebo komplexní jako “Počkej na pokyny, zvol vedoucího, následuj vedoucího”. Pravidla jsou uzlům adaptivní TM přidělována automaticky.

Systém tedy sestává z jednoho nebo více agentů, přičemž každý agent má páru atributů – **pozice, cíl, role a trasa**. Pokud agent narazí na uzel, začne vykonávat pravidlo uzlu. Pravidla jsou taková, že se můžou spustit jenom jednou – **triggery** – nebo se spouští v každém kroku. Prostředí pak ukládá strukturu sítě v podobě uzlů a spojujících hran. Uzly tedy obsahují pravidla, která ovlivňují chování boidů stejně, jak semafory ovlivňují auta na světelné křížovatce.

Tento model může poskytovat různá chování na základě aktuální konfigurace sítě TM. Mapy svým ovlivněním umožňují změny agentního chování – např. v otevřeném prostředí se agenty pohybují jako shluk, jakmile se ale dostanou do koridoru, přepne se chování na sledování vůdce a agenty půjdou v zástrupu. Tam, kde z uzlu vycházejí 2 hran, se stejnou váhou, může dojít k dělení shluku agentů, tam kde se sbíhají, se může naopak více shluků spojit. Počítá se s hledáním a plněním cílů – model s tím vůbec nemá problémy, protože na, temto účelu byl model postavený – pro hledání cest v grafu používá A*. V případě správně postavené TM model vykazuje výborné výsledky.

Kapitola 3

Návrh a Implementace

Cílem úlohy je vytvoření aplikace, na které demonstrujeme metodu BOIDS. Je tedy třeba vytvořit programový výpočetní model, poté vytvořit grafické prostředí (scénu) a tyto dvě entity spojit. Od začátku práce jsem byl rozhodlý pro jazyk Java a proto jsem jako implementační prostředí volil platformu JMonkey, scénu jsem vytvořil v Blenderu.

Kapitola návrh a implementace popisuje v následujícím pořadí 3 úkoly:

- Vytvoření simulačního modelu BOIDS v JMonkey
- Vytvoření scény v Blenderu a import do JMonkey
- Vytvoření výsledné aplikace

3.1 Použité nástroje

Zde jsou uvedeny nástroje, které mi výrazně pomohly s prací a s jejichž pomocí byla výsledná aplikace vytvořena.

3.1.1 JMonkey engine

JMonkey [17] je volně dostupný herní 3D engine, napsaný v Javě, uvolněný pod BSD licencí. Grafiku vykresluje za použití JOGL, což je implementace OpenGL upravená pro Java, podporuje OpenGL2 – OpenGL4. Díky spojení s OpenGL je tedy cílové vykreslování dostačující pro naš projekt.

JMonkey používá pravostranný kartézský souřadnicový systém. Implementuje operace s vícerozměrnými vektory, osvětlení, texturování a mnoha dalších nástrojů pro 3D reprezentaci. Navíc má JMonkey velkou komunitu a skvělé manuálové stránky. Nikdy se mi nestalo, že bych, po výčtení dokumentace nebo dotazů na fórech, nezvládl něco vyřešit. Můj spolubydlící měl nějaký problém s osvětlováním v uzavřené místnosti, vývojáři JMonkey mu dokonce vstřícně odpověděli, že jeho požadavek není implementovaný, že je to chyba, o které se ví a že bude opravena v další verzi.

Další věc je ta, že JMonkey obsahuje otevřený kód, až na některé předkompilované třídy, a tak si člověk může v krajině případě upravit implementaci sám.

3.1.2 Blender

Tento volně šířitelný modelovací nástroj je dostupný na adrese [3]. Je to komplexní nástroj pro vytváření modelů a animací. Nemám zkušenost s ostatními komerčními nástroji jako je Maya anebo studio 3DSMax, zastánci Blenderu ale tvrdí, že je v něm možné vytvořit úplně všechno, stejně anebo dokonce líp, jako ve dvou uvedených komerčních nástrojích. Komunita kolem Blenderu vytvořila několik filmů, aby tak demonstrovala schopnosti tohoto nástroje a přitáhla uživatele. V přiloženém CD v sekci videa najdete dvě z těchto výtvorů: Sintel – třetí otevřený film – a Tears of Steel – čtvrtý otevřený film od Blender Foundation.

Grafické uživatelské rozhraní Blenderu mně příšlo zpočátku zmatené a naprostě neintuitivní, ale bylo to asi tím, že to pro mě bylo něco úplně nového a tolik oken pohromadě jsem nikdy neviděl – a to za pocitu, že jsem nevěděl, co 90% oken vůbec znamená a jaký je jejich účel. Nějakou dobu trvá, než si člověk na ono rozhraní zvykne a tak i mně trvalo poměrně dlouho, než jsem dokázal vytvořit model scény s baráckama, silnicí a malým náměstím s dekoračním ornamentem. Blender je znám svým ovládáním přes klávesové zkratky, pomocí nichž, pokud se do nich člověk dostane, si může uživatel tvorbu v Blenderu několikanásobně urychlit. Vytváření scény je popsáno v sekci **vytváření scény 3.5**.

3.1.3 Blender-exporter

Scénu vytvořenou v Blenderu potřebujeme vyexportovat a načíst v JMonkey. Blender ani JMonkey takovýto převod nepodporují, proto jsem musel použít skript ve formě pluginu pro Blender – BlenderExporter [9] – který vyexportuje namodelovanou scénu a všechny vytvořené meshe včetně materiálů, textur a texturovacích map do formátu **dotScene**. Tento formát už JMonkey umí konvertovat do svého formátu **j3o**, který může být načten a použit ve scéně JMonkey projektu.

3.1.4 XStream

XStream [15] je open source knihovna pro Java sloužící pro serializaci datových struktur do formátu XML. Původně jsem chtěl použí XML serializer vestavěný v Javě, ale v méém případě nefungoval tak, abych ho mohl s klidem použít. Podle dokumentace jsem dělal pář pokusů ukládání a načítání, některé struktury se mi uložily, některé ne. Popravdě jsem nenašel a nepochopil vzorec, podle kterého bych mohl můj XML serializer rozchudit. Na fóru **stackowerflow** jsem se pak dočetl o knihovně XStream, která vypadala jednoduše a intuitivně. Knihovnu jsem stáhl a přidal do projektu, přeložil a spustil první pokus o serializaci a dostal uspokojivý výsledek.

Příklad použití převzatý z domovoské stránky XStream:

```
import com.thoughtworks.xstream.XStream;

// deklarace tridy Person
public class Person {
    private String firstname;
    private String lastname;
    private PhoneNumber phone;
    private PhoneNumber fax;
    // ... konstruktor a metody
}
```

```

// deklarace vnořené tridy PhoneNumber
public class PhoneNumber {
    private int code;
    private String number;
    // ... constructors and methods
}

// vytvoření instance tridy Person a inicializace atributů
Person joe = new Person("Joe", "Walnes");
joe.setPhone(new PhoneNumber(123, "1234-456"));
joe.setFax(new PhoneNumber(123, "9999-999"));

// inicializace XStreamu
XStream xstream = new XStream();

// vytvoření aliasu pro přehlednost ve výsledném XML
xstream.alias("person", Person.class);
xstream.alias("phonenumber", PhoneNumber.class);

// uložení výsledného XML do řetězce
String xml = xstream.toXML(joe);

```

Uvedený kód deklaruje dvě jednoduché třídy - třídu Person, která obsahuje dva řetězce (jméno a příjmení) a dvě instance třídy PhoneNumber (telefon a fax).

Následuje vytvoření objektu třídy Person – joe – a naplnění jeho atributů včetně dvou objektů třídy PhoneNumber.

Poté vytvoříme instanci třídy XStream a nastavíme jí dva aliasy, jinak by ve výsledném XML byla u každého elementu zmíněné třídy uvedená celá cesta včetně balíčků – to na funkčnosti nijak neubere, ale pro nás jako uživateli bude výsledné XML přehlednější.

Nakonec uložíme objekt zavoláním metody `toXml()`, které jako parametr předhodíme objekt, který chceme serializovat.

Výsledné XML bude vypadat takto:

```

<person>
    <firstname>Joe</firstname>
    <lastname>Walnes</lastname>
    <phone>
        <code>123</code>
        <number>1234-456</number>
    </phone>
    <fax>
        <code>123</code>
        <number>9999-999</number>
    </fax>
</person>

```

Deserializace potom funguje obdobně, jednoduše a intuitivně:

```

// nasledná deserializace objektu z XML
Person newJoe = (Person)xstream.fromXML(xml);

```

Co víc si může Java programátor od knihovny pro XML serializaci přát?

3.2 Můj postup

Projekt byl programován hodně inkrementálně, ze začátku jsem ani já, ani můj vedoucí neměli ponětí, jak výsledná aplikace bude vypadat. Nejdřív jsem vytvořil model BOIDů pohybujících se v 3D prostoru (obrázek 3.2), potom jsem chování přenesl na plošinu. Až jsem viděl, že shlukování funguje a dostal jsem schválení od vedoucího práce, pokračoval jsem dál implementací překážek.

Při vyhýbání se překážkám jsem narazil na tisíce problémů. Naráz přestávalo fungovat vše, co předtím bylo naprosto zřejmé. Snažil jsem se zjednodušit výpočty, protože výpočty kolizí neuvěřitelně přidávaly na době výpočtu, snažil jsem se upravovat váhy výsledných sil nebo měnil metody vyhýbání se kolizím, protože boidy pořád procházely překážkama.

Zkrátka uplynuly téměř 2 týdny a já jsem zoufale nevěděl, jak mé boidy přimět chovat se lidsky - to znamená neprocházet překážkama jako duchové a zároveň zachovat realistický dojem shlukového chování. Chtěl jsem prostě mět dokonalý model, abych ho potom mohl nasadit do scény a vše fungovalo tak, jak jsem od začátku plánoval. To je jedna z nepříjemných věcí na povolání programátora – některé věci se zkrátka neudělají samy, některé věci si musíme vysedět u počítace, až po určité době začnou fungovat a my můžeme pomalu otevírat šampaňské, protože se konečně můžeme posunout dál. Když se ale dozvím, kolik jsme onou “vítěznou” činností strávili času, rychle vystřízlivíme a pokračujeme dál (v mojem případě modelováním scény), protože termín se neúprosně blíží.

Vytvoření scény v Blendru byl daleko příjemnější úkol, než ladění překážek. Možná taky proto, že šly více vidět výsledky a jak jsem se ze začátku některé věci naučil (vytvoření domku, otexturování), vlastně jsem je používal dokola celou dobu a vytvářel jeden domek za druhým. A taky možná proto, že modelování a texturování pro mě bylo něco úplně nového a něco, co jsem odmalička ve hrách zkoumal a obdivoval.

Vytvoření scény ještě není ani polovina práce, co se scény týká, protože je potřeba scénu vyexportovat, konvertovat a načíst v JMonkey. Jakmile je scéna načtená, pořád není hotovo. Nebo možná by bylo, kdybych se rozhodl používat kolizní model vestavěný v JMonkey. Můj spolubydlící dělal pokusy s kolizním modelem a náročností výpočtů, protože chtěl dělat nějakou online hru se “zombíky” o přežití, kde právě tito “zombíci” řešili kolize s okolními objekty. Na svém stroji osazeném processorem intel i5 a 4GB paměti se s párem modely a párem zombíků dostal na 5 snímků za sekundu. Toto jsem nechtěl ve své simulaci riskovat, navíc jsem nechtěl, aby řešení kolizí bylo závislé na platformě JMonkey, proto jsem kolize řešil sám a proto jsem taky ztratil tuny trpělivosti, než jsem byl se svým kolizním modelem spokojený¹.

Překážky jsem tedy dělal zvlášť, načítají se ze souboru v XML formátu. Volil jsem XML, protože je čitelný a snadno editovatelný a výsledné překážky neodpovídají všem objektům ve scéně, překážky sem vytvořil v XML souboru manuálně, abych ušetřil na této výpočetní části. Více o překážkách v sekci věnované překážkám 3.5.3.

Máme dav ve scéně, máme fungující model překážek, je třeba inicializovat simulaci – nastavit počáteční pozice a počty lidí v davu, nastavit kamery, cíle a ostatní parametry.

¹A pořád nejsu spokojený! ale už musím odevzdat

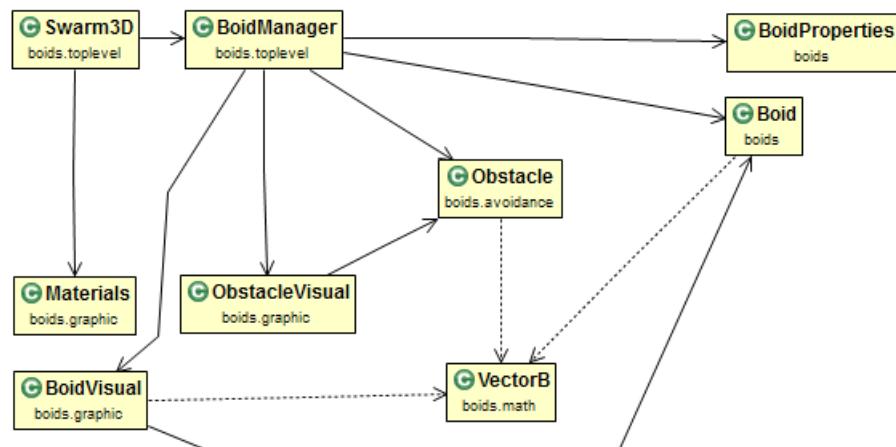
Všechny tyto údaje načítáme z konfiguračního souboru v XML.

3.3 Detail implementace

Prvním úkolem bylo vytvoření modelu BOIDS na platformě JMonkey, abych model nejdříve ve zjednodušeném prostředí naimplementoval a odladil. Tato část projektu sestávala z 9ti tříd rozdělených do celkem 5ti balíčků – obrázek 3.1.

balíček boids

- třída Boid – reprezentuje samostatný boid, počítá síly, pozice, rotace
- třída BoidProperties – chtěl jsem mít parametry boidů na jednom místě, od toho je tato třída – jedináček



Obrázek 3.1: zjednodušený diagram tříd simulačního modelu

balíček boids.toplevel

- třída Swarm3D – třída obsahující metodu main, vytvoří scénu, osvětlení a vytvoří instanci třídy BoidManager
- třída BoidManager – jak název sám vypovídá, stará se o koordinaci všech boidů, pomocí metody nextStep způsobí další krok simulace

balíček boids.avoidance

- třída Obstacle – reprezentuje válcovitou překážku v souřadnicovém prostoru, má pozici a velikost

balíček boids.graphic

- třída Materials – třída materiálů, kterými jsou obalovány geometrie ve scéně

- třída BoidVisual – grafická reprezentace boidu
- třída ObstacleVisual – grafická reprezentace překážky

balíček boids.math

- třída VectorB – moje knihovna s vektorovými výpočty

3.4 BOIDS pseudoalgoritmus

Cílem práce není popis zdrojových kódů, ten lze najít ve vygenerované dokumentaci, zde popíšu jen pseudoalgoritmus, tedy základní logiku boidu, která je k implementovaná ve třídě **Boid**.

Každý boid se pohybuje v prostoru rychlostí \vec{v} , tato rychlosť je ovlivňována zrychlením \vec{a} , které je počítáno v každém kroku výpočtu jako výslednice sil **vyhýbání se překážkám, separace, zarovnání, sdružování a plnění cílů**. Pro určení správne výslednice mezi těmito 5ti silami používám přístup akumulace velikostí aplikovaných sil v uvedeném pořadí. Vyhýbání se kolizím má nejvyšší precedenci, a tak boid, pokud bude nucen vyhnout se překážce, ignoruje zarovnání, sdružování a plnění aktuálního cíle.

Na následujících řádcích je zjednodušený pseudoalgoritmos docílení Reynoldsových pravidel. Tyto výpočty provádí každý boid nezávisle, právě díky lokálnímu vnímání 2.5 je docíleno uvěřitelného shlukování. Každé z pravidel má parametrizovaný rozhled, tím je zaručena konfigurovatelnost chování. Zvyšování rozhledu pro pravidlo separace vzdaluje boidy vzájemně od sebe, soudržnost společně ze zarovnáním pak zvětšuje shluk, co se týká počtu účastníků. Pokud budou hodnoty rozhledu soudržnosti a zarovnání malé, bude vznikat několik shluků po páru agentech.

Algoritmus 1: PRAVIDLO SEPARACE

```

1 Vector separation = Vector ZERO;
2 for every_visible_neighbouring_boid do
3     Vector direct_away_from_neighbour = neighbour_position - current_position;
4     direct_away_from_neighbour =
5         direct_away_from_neighbour / squared_distance_to_neighbour;
6     separation = separation + direct_away_from_neighbour;
7 end for
8 return separation;
```

Funkce `get_convenient_steer(nearest_obstacle)` v algoritmu 11 je implementací některého z uvedených metod vyhýbání se překážkám uvedených v sekci 2.6. V aplikaci vytvořené pro tuto práci používám metodu zatáčení směrem od středu, protože se v mém prostředí, kde několik shluků prochází koridorem obeskládaným domky, hodil nejvíce.

Boid v každém kroku vypočítá všechny síly, normalizuje je a vynásobí váhama, poté volá funkci `applyForce` 4, které předá korespondující zrychlení, pokud ještě není v aktuálním

Algoritmus 2: PRAVIDLO ZAROVÁNÍ

```
1 Vector alignment = Vector ZERO;  
2 for every_visible_neighbouring_boid do  
3     alignment = alignment + neighbour_velocity / squared_distance_to_neighbour;  
4 end for  
5 return alignment;
```

Algoritmus 3: PRAVIDLO SEEK

Input: Vector target

```
1 Vector desired = new Vector (vector_pointing_to_desired_target);  
2 Vector seek_force = vector_pointing_to_desired_target - current_velocity;  
3 return seek_force;
```

Algoritmus 4: PRAVIDLO SDRUŽOVÁNÍ

```
1 Vector cohesion = Vector ZERO;  
2 int neighbour_counter = 0;  
3 for every_visible_neighbouring_boid do  
4     neighbour_counter = neighbour_counter + 1;  
5     cohesion = cohesion + neighbour_position;  
6 end for  
7 if neighbour_counter > 0 then  
8     cohesion = cohesion / neighbour_counter;  
9     cohesion = seek(cohesion);  
10 end if  
11 return cohesion;
```

Algoritmus 5: VYHÝBÁNÍ SE PŘEKÁŽKÁM

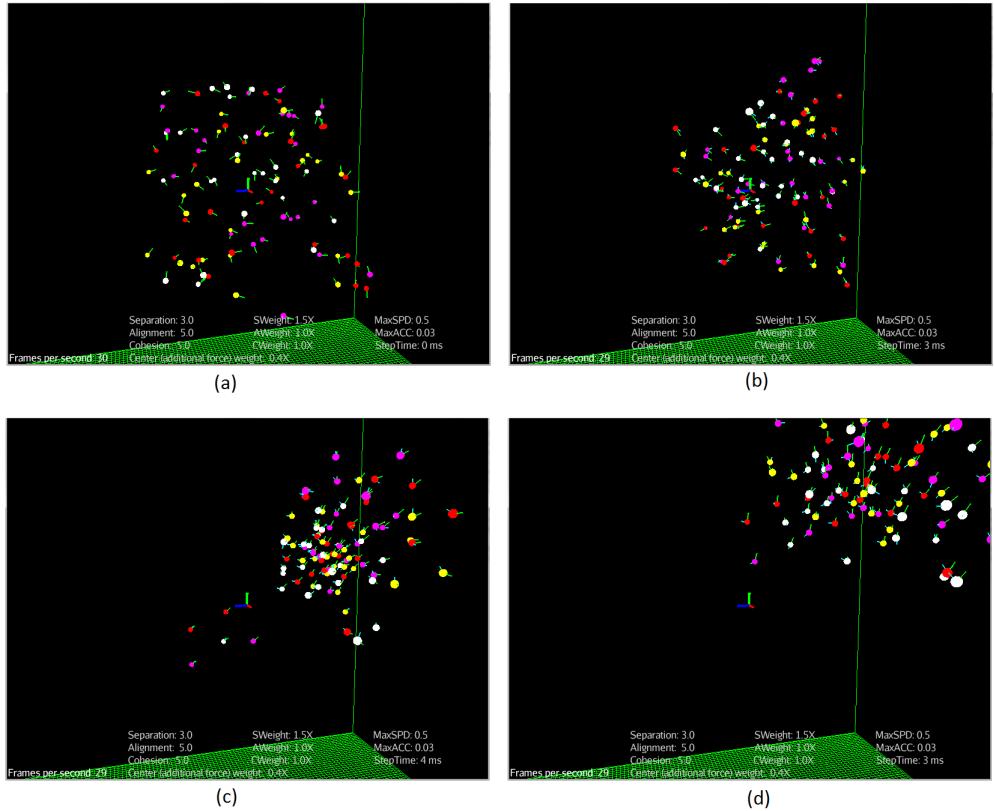
```
1 Vector obstacles = Vector ZERO;  
2 Obstacle nearest_obstacle = find_nearest_visible_obstacle();  
3 Vector steer_direction = get_convenient_steer(nearest_obstacle);  
4 return steer_direction;
```

Algoritmus 6: AKUMULACE SIL

Input: Vector force

```
1 global float accumulator_limit ;  
2 global float accumulated_size ;  
3 float acceleration_magnitude = force.get_size;  
4 accumulated_size = accumulated_size + acceleration_magnitude;  
5 if accumulated_size <= accumulator_limit then  
6     add_acceleration_to_current_velocity(force);  
7 end if  
8 return;
```

kroku naplněný akumulátor, který se nuluje na začátku každého nového kroku. Provedením těchto výpočtů je vypočítána nová rychlosť. Posunutí boidu na novou pozici provedeme prostým přičtením rychlosti k aktuální pozici boidu.



Obrázek 3.2: Jedna z prvních verzí aplikace - implementace chování separace, zarovnání sdružování. Na (a) jsou boidy ve stavu ihned po incicializaci, mají náhodné směry a rychlosti. Na (b) se začínají srovnávat díky pravidlu zarovnání, na (c) boidy tvoří zhuštěný shluk díky sdružování. Na (d) se díky separaci shluk formuje tvořením konstantních vzdáleností mezi boidy. Posléze hromada boidů odléta jako shluk ze zorného pole.

3.5 Vytvoření scény

Pro demonstraci fungování jsem potřeboval vytvořit scénu, kde bych demonstroval fungování metody BOIDS. Pro splnění tohoto úkolu jsem potřeboval 3D modelovací nástroj, jak je už z dřívějšího popisu asi zřejmé, rozhodl jsem se pro Blender. Na následujících řádcích popíšu, jak jsem scénu vytvořil a následně ji dostal do prostředí JMonkey.

3.5.1 Polygonové meshe a UV texturování

Polygonový mesh je souhrnný název pro grafický objekt složený z polygonů. Nebudu psát o modelování v Blenderu, osvětlování a dalších detailech, popíšu jen to nejnuttnejší, co bylo

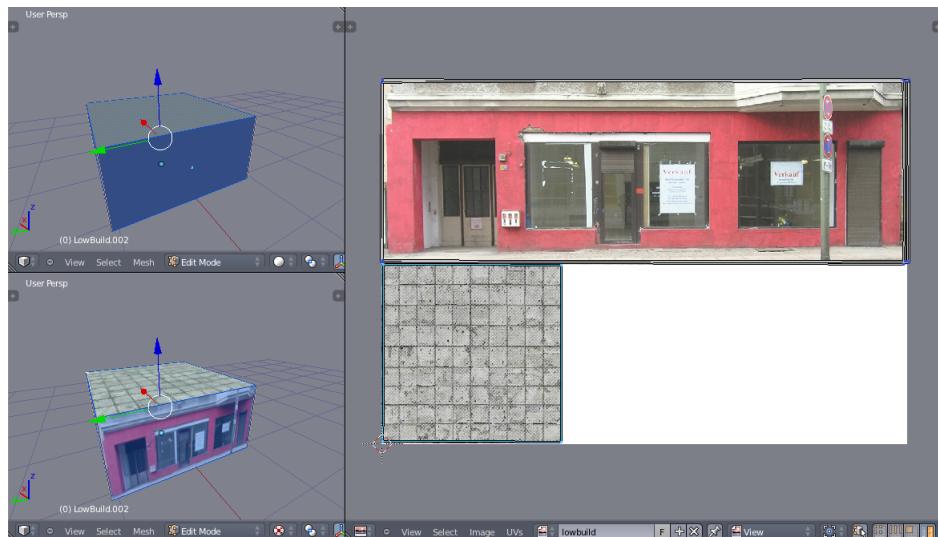
potřeba udělat pro vytvoření pár jednoduchých objektů a vytvoření prostředí pro boidy z těchto objektů.

V podstatě bylo potřeba udělat následující:

- Vytvořit mesh - dům nebo plošinu
- Vytvořit materiál pro mesh - povrchové vlastnosti
- Vytvořit texturu a namapovat ji na mesh

U materiálu jsem pouze definoval difuzní a spekulární odraz, pak už nanášel texturu. Zde jsem se poprvé seznámil s UV texturováním – funguje to tak, že se polygonový mesh rozloží jakoby se rozstříhl a měl položit na papír. Pak se vybere textura, která se pod rozložený mesh položí a polygony nad texturou jsou v UV mapovacím prostředí posouvány a upravovány do té doby, než jsme spokojeni s výsledným namapováním. Následně se vytvoří **UV mapa**, která se použije pro správné namapování textury na náš polygonový mesh.

Obrázek 3.3 je vyfocen v Blenderu, znázorňuje UV mapování textury svislé stěny a střechy domu na mesh. Na levé straně nahoře je k vidění původní mesh, na pravé straně textura s UV mapou, na levé straně dole pak otexturovaný mesh.



Obrázek 3.3: UV mapování v Blenderu

3.5.2 Export z Blenderu a import do JMonkey

Pravotčivá a levotočivá soustava prostorových kartézských souřadnic

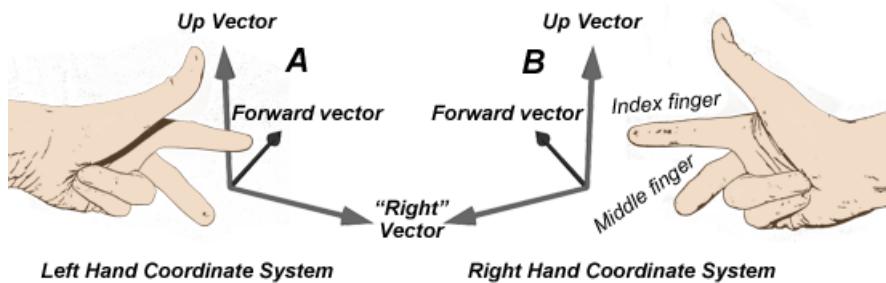
Blender a JMonkey používají odlišné uspořádání souřadnic v prostorovém kartézském systému souřadnic.²

Pravotočivá soustava - JMonkey: Představme si, že stojíme v počátku prostorové

²Kartézská soustava souřadnic je taková soustava souřadnic, u které jsou souřadné osy vzájemně kolmé a protínají se v jednom bodě - počátku soustavy souřadnic. (Wikipedia)

kartézské soustavy. Osa x nechť směřuje přímo vpřed, osa y směřuje kolmo vlevo a osa z směřuje přímo vzhůru. Rozpoznání této soustavy usnadňuje mnemotechnická pomůcka **pravidlo pravé ruky**: *Pokud rozevřenou dlaň pravé ruky natočíme tak, aby konečky prstů směřovaly ve směru osy Z, palec ukazuje nahoru ve směru osy Y, z dlaně ven pak vystupuje osa X.*

Naproti tomu **Blender** používá **levotočivou** soustavu souřadnic, kterou z pravočivé dostaneme tak, že zaměníme osy X a Y.



Obrázek 3.4: ilustrace levotočivé a pravotčivé soustavy souřadnic

Poprvadě jsem dříve ani nevěděl, že existuje levotočivá a pravotočivá soustava. Zjistil jsem to až v okamžiku, kdy jsem vyexportoval model z Blenderu a načetl ho v prostředí JMonkey s hrůzným zjištěným, že je všechno převrácené. Proto je potřeba při exportu z Blenderu volit možnost **převrátit souřadnicové osy**, pokud nám to exportér dovoluje.

Ogre mesh a Ogre dotScene format

Ogre mesh je formát reprezentace polygonového meshe v prostředí Ogre3D [21]. Ogre mesh se vyskytuje ve dvojím formátu:

- *.mesh - binární formát používaný v Ogre3D aplikacích
- *.mesh.xml - XML reprezentace meshe, kterou používá právě JMonkey

Ogre dotScene formát (*.scene) je standardizovaný formát reprezentace scény. Obsahuje informace o meshích ve scéně, osvětlení a použitých materiálech. Velkou výhodou je, že se informace o scéně ukládaní ve formátu XML, tak je pro člověka dobře čitelný.

Do těchto dvou formátů lze vyexportovat meshe a celou scénu z Blnderu, za použití nástroje [9] a následně výsledný ***.scene** soubor načít v JMonkey. Přesně tedy výsledný ***.j3o³** soubor, který bude z původního ***.scene** převeden vestavěnými nástroji v prostředí JMonkey.

3.5.3 Vytvoření překážek

Z důvodů uvedených v sekci 3.2 jsem se rozhodl kolize počítat sám a nespolehat se v tomto na JMonkey.

³formát reprezentace objektů v prostředí JMonkey

Všechny překážky jsem vytvořil jako válcovité tělesa, abych sjednotil a ulehčil výpočty. Překážky jsou uloženy v externím souboru – XML dokumentu – který jsem ručně vytvořil podle souřadnic a velikostí překážek předem namodelovaných v Blenderu.



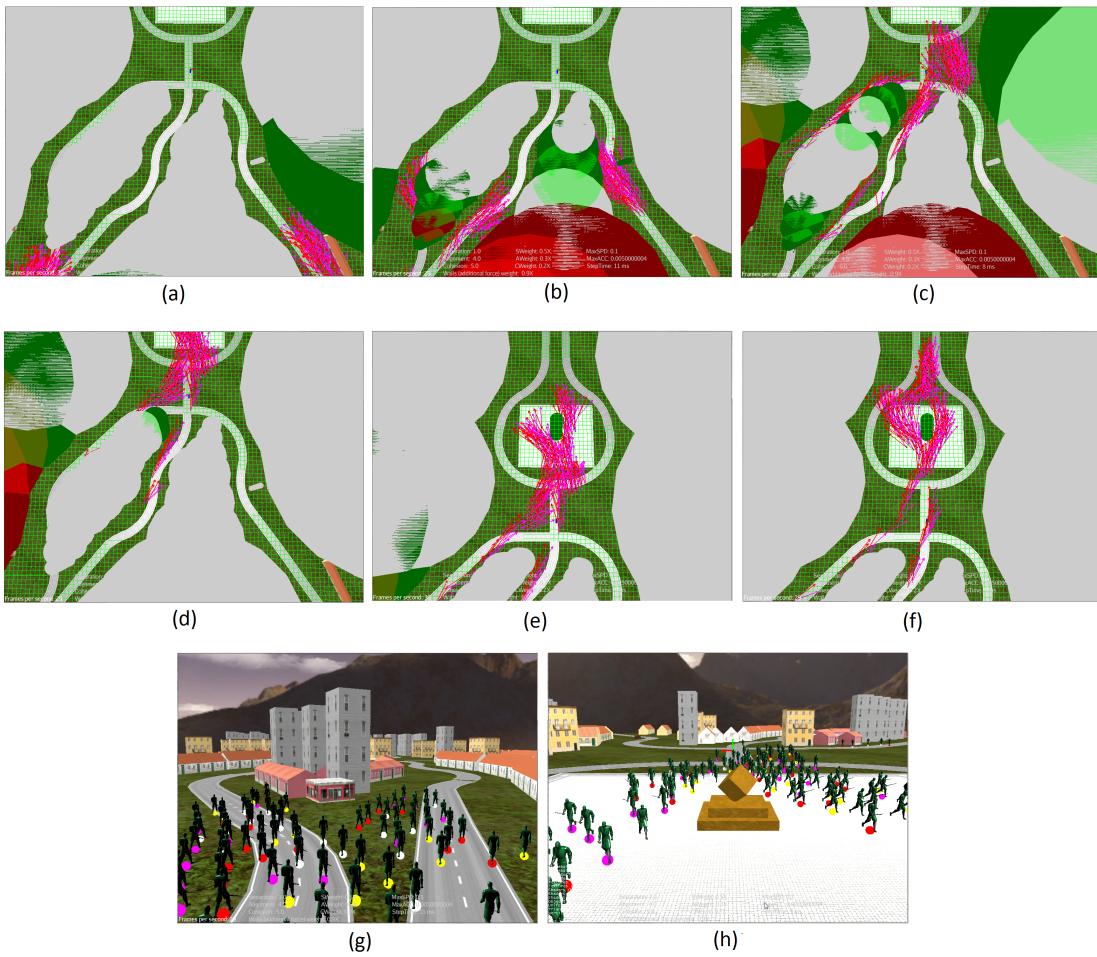
Obrázek 3.5: pohled na scénu (vlevo) a vymezení prostoru pro boidy (vpravo)

3.6 Vytvoření výsledné aplikace

Vytvořený model BOIDS byl vložen do prostředí vymodelovaného v Blenderu. Prostředí tvoří 3 koridory lemované domkama, koridory se sbíhají ve středu scény na ploše, která má znázorňovat náměstí. Uprostřed náměstí je ornament, který působí jako překážka a má shluk zase rozdělit. Po obejítí překážky (ornamentu) se boidy zase spojují v jeden shluk.

Při použití BOIDS v komplexnějších prostředích, jako např. námi použité zjednodušené město, jsou boidy velmi citlivé na nastavení jejich parametrů, zejména parametr určující **viditelnost překážek**. Protože vyhýbání se kolizím má nejvyšší precedenci, může vytlačit ostatní chování a žádné ze 3 pravidel zajišťujících shlukování se nebude provádět.

Rozhodovací logika vyhýbání se překážek dle Reynoldse je velice jednoduchá, a tak může občas dojít ke kolizím nebo nepředvídatelnému chování právě díky privilegované snaze boidu za každou cenu uhnout překážce. Na obrázku 3.6 jsou snímky z průběhu simulace výsledné aplikace.



Obrázek 3.6: Části (a)-(f) zobrazují pohyb boidů v prostředí s překážkami. Překážky jsou šedé válcovité objekty v okolí silnic, pokud je překážka spatřena některým boidem, svítí zeleně, pokud se boid dostane do prostoru překážky, patřičná překážka svítí červeně. Pro vizualizaci boidů je použit model s animací přímo z prostředí JMonkey, na obrázcích v takto malém formátu jsou vidět pouze růžové a červené šipky vycházející z každého boidu, které reprezentují výhled boidu na překážky. (a) je počáteční stav po inicializaci, boidy se nacházejí na pozicích v levé a pravé dolní části obrazovky ve dvou shlucích. (b) pokračuje rozdelením boidů na levé části do dvou koridorů, v (c) se boidy blíží ke křížovatce, za kterou se spojují v jeden shluk (d). Boidy pokračují dál za svým cílem a dochází na náměstí, kde jsou rozděleny překážkou uprostřed náměstí (e). Na (f) se zase shlukují po překonání překážky. Obrázky (g) a (h) ukazují pohled na simulaci bez vizualizace rychlostí, zrychlení, překážkového rozhledu a překážek ze startovní a cílové kamery.

Kapitola 4

Závěr

V první části práce byl představen distribuovaný model shlukování BOIDS “otce shlukování” Craiga Reynoldse, byla popsána a odůvodněna základní pravidla shlukování, byly popsány metody vyhýbání se překážkám pro tento model a sily, kterými můžeme ovlivňovat chování shluku. BOIDS funguje perfektně v prostředí, pro které byl navržený – simulace hejna ptáků nebo rybích školek v otevřených prostorech s občasnými překážkami. Když jsme ale model použili na námi modelované město, vyhýbání se překážkám způsobovalo nechtěné chování, které ubíralo na dojmu realističnosti. BOIDS ve svém modelu implementuje pouze jednoduché rozhodování sil prioritizací, a tak se boid ve složitějším prostředí zaobírá pouze vyhýbáním se kolizí, protože má nejvyšší prioritu, a “zapomíná” se tak shlukovat. Pokud by každý boid implementoval chytřejší rozhodovací logiku, určitě by se dalo dosáhnout lepších výsledků – například v kombinaci s neuronovými sítěmi nebo expertními systémy. Takovéto řešení by rozhodně stálo za pokus a vytvoření sofistikovaných boidů by mohlo být dobrým námětem třeba pro diplomovou práci.

Dále byly, pro porovnání a případnou inspiraci pro vytvoření dalšího modelu, zmíněny další existující přístupy pro simulovalní shluky. Model Tu a Terzepulouse je pokročilým modelem simulace ryby v jejím přirozeném prostředí, implementuje výrazně lepší rozhodování o dalších krocích, než Reynoldsův algoritmus, za užití generátoru cílů, který filtruje informace o svém okolí a zaměřuje se pouze na ty informace, které rybu přiblíží k uspokojení aktuálního cíle.

Vicsekův model je fyzikálním popisem Reynoldsova shlukování a analogí s magnetickými spiny magnetu, který pro lepší matematický popis rozšířili Torner a Tu. Dalším matematickým popisem je model Tannera a Jadbabaie, ti vytváří teorii shlukování, kterou popisují teorií řízení a teorií grafů. Přes rozsáhlé publikace jsem ale nenašel žádný přístup k řešení vyhýbání se kolizím – jak v modelu Tannera a Jadbabaie, tak v modelu Vicseka a jeho matematictější variaci.

Velice se mně líbil model Ofati-Sabera, který využívá teorie grafů a matematicky popsané protokoly, které řídí jeho α , β a γ agenty. Model je schopný shlukování, rozdělování a spojování shluků a velice dobře si stojí ve vyhýbání překážkám. Nepřeše zde nic o plánování nebo směrování shluků za určitým cílem nebo o jakémkoli lepším kontrolování simulace. Jeho další publikace jsem ale nesledoval, a tak je možné, že model dávno vylepšil. Navíc si myslím, že rozšíření tomto modelu za účelem uspokojování cílů by neměl být problém – třeba přidáním dalších agentů a protokolů.

Poslední z uvedených modelů byly trasové mapy, které si vzaly za vzor robotické systémy. Od ostatních přístupů se odlišují tím, že agent není řízen sám svým rozhodováním (to v podstatě není agent), ale je dirigován sítí, která organizuje veškerý pohyb (zde roli agentů zastupují uzly sítě). Model má velice silnou stránku právě v plánování, díky použitému hledání optimální cesty pomocí A*.

V druhé části dokumentu byl popsán postup vytvoření 3D aplikace, která demonstrovala metodu BOIDS na pohybu agentů v částečně uzavřeném prostředí. Aplikace je konfigurovatelná a nastavení boidů je parametrickatelné, takže zde lze s chováním experimentovat. S výsledkem ale úplně spokojený nejsem. Byl totiž použit původní algoritmus BOIDS, tak jak ho Reynolds popsál, tedy s "hloupým" rozhodováním, a tak výsledný dojem nemusí vždy vypadat ve strukturovanějším překážkovitém prostředí uspokojivě.

Největší přínos pro mě byl v získání přehledu o existujících modelech shlukového chování a ve vytváření 3D aplikace, což pro mě byla výzva odzačátku, protože jsem nikdy nic podobného neprogramoval. K JMonkey enginu a k modelování v Blenderu se určitě budu vracet, protože to pro mě byl určitým způsobem krok dál v programování takovým způsobem, jak jsem si ho odmalička představoval.

Literatura

- [1] Amkraut. S. Girard. M., K. G.: Eurythmy. in SIGGRAPH Video Review. Issue 21 (second item, time code 3:58 to 7:35), 1985, motion studies for a work in progress entitled ‘Eurythmy’ produced at the Computer Graphics Research Group. Ohio State University. Columbus, Ohio.
- [2] Beni, W. J., G.: *Robots and Biological Systems: Towards a New Bionics*, kapitola Swarm Intelligence in Cellular Robotic Systems. Tuscany, Italy: Springer Berlin Heidelberg, Červen 1993, ISBN 978-3-642-58069-7, s. 703–712.
- [3] community, B.: Home of the Blender project - free and open 3D creation software. <http://www.blender.org/>.
- [4] Dutta, K.: How Birds Fly Together: The Dynamics of Flocking. *Resonance*, dec 2010.
- [5] Herbert, F. T.; Tanner, H. G.; Jadbabaie, A.: Stable Flocking of Mobile Agents, Part I:. In *In IEEE Conference on decision and control*, 2003, s. 2010–2015.
- [6] O. Burchan Bayazit, J.-M. L.; Amato, N. M.: Better Group Behaviors using Rule-Based Roadmaps. Technická zpráva, Texas A&M University, College Station TX 77843, USA, 2004.
- [7] Partridge, B. L.: A blind fish can school. *Science*, ročník 194, č. 4268, Listopad 1976: str. 964.
- [8] Partridge, B. L.: The Structure and Function of Fish Schools. *Scientific American*, ročník 246, č. 6, Červen 1982: s. 114–123.
- [9] Reimpell, M.: BlenderExporter.
<http://www.ogre3d.org/tikiwiki/Blender+Exporter>, 2013.
- [10] Reynolds, C.: Boids - background and update. <http://www.red3d.com/cwr/boids/>.
- [11] Reynolds, C.: Not Bumping Into Things: A Distributed Behavioral Model. In *Flocks, Herds, and Schools: A Distributed Behavioral Model*, 1988, notes on obstacle avoidance for the course on Physically Based Modeling at SIGGRAPH 88, August 1 through 5 in Atlanta, Georgia.
- [12] Reynolds, C. W.: Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, New York, NY, USA: ACM, 1987, ISBN 0-89791-227-6, s. 25–34, doi:10.1145/37401.37406.
URL <http://doi.acm.org/10.1145/37401.37406>

- [13] Reynolds, C. W.: Steering Behaviors For Autonomous Characters. In *Game Developers Conference*, San Jose, California: Miller Freeman Game Group, 1999, s. 763–782.
- [14] Saber, R.; Murray, R.: Flocking with obstacle avoidance: cooperation with limited communication in mobile networks. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, ročník 2, 2003, ISSN 0191-2216, s. 2022–2028 Vol.2, doi:10.1109/CDC.2003.1272912.
- [15] Schaible, J.: XSteam homepage. <http://xstream.codehaus.org/index.html>, 2013.
- [16] SHIFFMAN, D. (editor): *The Nature of Code: Simulating Natural Systems with Processing*. The Nature of Code, 2012, ISBN 978-0985930806.
- [17] source, O.: JMONKEYENGINE 3.0 - JAVA OPENGL GAME ENGINE. <http://jmonkeyengine.com/>, 2013.
- [18] Steven, W.: *Game programming gems*, kapitola Flocking: A Simple Technique for Simulating Group Behavior. Hingham: Charles River Media, 2000, ISBN 1-58450-049-2, s. 305–318.
- [19] Susan, A.: Osobní komunikace, Leden 1987, the force field animation system.
- [20] Tanner, H. G.; Jadbabaie, A.; Pappas, G. J.: Stable Flocking of Mobile Agents, Part II: Dynamic Topology. In *In IEEE Conference on Decision and Control*, 2003, s. 2016–2021.
- [21] team, O.: Ogre 3D – Object-Oriented Graphics Rendering Engine. <http://www.ogre3d.org/>, 2013.
- [22] Toner, J.; Tu, Y.: Flocks, herds, and schools: A quantitative theory of flocking. *Phys. Rev. E*, ročník 58, Oct 1998: s. 4828–4858, doi:10.1103/PhysRevE.58.4828.
URL <http://link.aps.org/doi/10.1103/PhysRevE.58.4828>
- [23] Toner, J.; Tu, Y.; Ramaswamy, S.: Hydrodynamics and phases of flocks. *Annals of Physics*, ročník 318, č. 1, 2005: s. 170 – 244, ISSN 0003-4916, doi:10.1016/j.aop.2005.04.011.
URL <http://www.sciencedirect.com/science/article/pii/S0003491605000540>
- [24] Tu, X.; Terzopoulos, D.: Artificial fishes: physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, New York, NY, USA: ACM, 1994, ISBN 0-89791-667-0, s. 43–50, doi:10.1145/192161.192170.
URL <http://doi.acm.org/10.1145/192161.192170>
- [25] Vicsek, T.; Czirók, A.; Ben-Jacob, E.; aj.: Novel Type of Phase Transition in a System of Self-Driven Particles. *Physical Review Letters*, ročník 75, Srpen 1995: s. 1226–1229, doi:10.1103/PhysRevLett.75.1226, [arXiv:cond-mat/0611743](https://arxiv.org/abs/cond-mat/0611743).
- [26] Zbořil, F.: *Agentní a multiagentní systémy*. FIT VUT, 2006, studijní opora.

Příloha A

Obsah CD

1. Zdrojové texty programu
2. Javadoc
3. Spustitelná verze aplikace pro Windows
4. Spustitelná verze aplikace pro Linux
5. Videá vztahující se k práci

Příloha B

Tvar konfiguračního souboru

Aplikace používá 2 externí soubory ve formátu XML. První nese název **obstacles.xml** (jeho struktura je jednoduchá, je to pouze seznam s překážkama, je lehce čitelný).

Druhé, ale důležitější XML se jmenuje **config.xml**. Struktura je popsána v následujícím XML s vloženými komentáři:

```
<config>
    // definice vytvorených shluku pro inicializaci
    <crowds>
        // elementu crowdGroup zde muze byt nekolik
        <crowdGroup>
            <crowdSize>80</crowdSize>
            // pozice vyvorení a směr shluku
            <spawnLocDir>
                <location>
                    <x>26.7711</x>
                    <y>0.0</y>
                    <z>39.9295</z>
                </location>
                <direction>
                    <x>-0.52747834</x>
                    <y>-0.21986791</y>
                    <z>-0.8206247</z>
                </direction>
            </spawnLocDir>
            // seznam cílu cesty, kterými má shluk projít
            <checkpoints>
                <vector>
                    <x>0.0</x>
                    <y>0.0</y>
                    <z>2.0</z>
                </vector>
                <vector>
                    <x>0.0</x>
                    <y>0.0</y>
                    <z>-30.0</z>
                </vector>
                <vector>
```

```
<x>0.0</x>
<y>0.0</y>
<z>-80.0</z>
</vector>
</checkpoints>
</crowdGroup>
</crowds>
// seznam kamer - kamera ma pozici a smer, je ukladana v elementu vectorPair
<cameras>
    <vectorPair>
        <location>
            <x>31.602345</x>
            <y>3.2666392</y>
            <z>51.225876</z>
        </location>
        <direction>
            <x>-0.52664393</x>
            <y>-0.33381882</y>
            <z>-0.7817998</z>
        </direction>
    </vectorPair>
</cameras>
// pouzity soubor s prekazkama
<obstacles>obstacles.xml</obstacles>
// pouzita scena pro simulaci
<scene>Scenes/smallCity/small_city.j3o</scene>
// pouzite pozadi pro simulaci
<sky>Textures/Sky/Bright/BrightSky.dds</sky>
</config>
```

B.1 Ovládání aplikace

Klávesa	Akce
SPACE	Spuštění a pozastavení simulace
N	Další výpočetní krok simulace
X	Ladící zobrazení
E	Nastavení defaultních hodnot nastavení
R	Restartování simulace
T	Zvětšení poloměru separace
G	Zmenšení poloměru separace
Y	Zvětšení poloměru zarovnání
H	Zmenšení poloměru zarovnání
U	Zvětšení poloměru sdružování
J	Zmenšení poloměru sdružování
I	Zvětšení maximální rychlosti
K	Zmenšení maximální rychlosti
O	Zvětšení maximálního zrychlení
L	Zmenšení maximálního zrychlení
F1 - F9	Pohledy z předchystaných kamer

Tabulka B.1: Popis kláves