УДК 004.415.2.22

© 2010 р. В.С.Семенякін, Т.М.Заболотня

Національний технічний університет України «Київський політехнічний інститут», м.Київ, Україна

МОДИФІКОВАНИЙ ОБ'ЄКТНООРІЄНТОВАНИЙ ПІДХІД ДО ПОБУДОВИ ПРОГРАМНИХ ЗАСОБІВ МОДЕЛЮВАННЯ ФІЗИЧНИХ ПРОЦЕСІВ

У статті розглядається питання побудови архітектури програмних засобів моделювання фізичних процесів. Пропонується новий *SIM* підхід до організації об'єктів простору моделювання, орієнтований на максимальне абстрагування останніх. Визначаються основні кроки створення програмного забезпечення у межах описаного підходу. Розробляється *SIM* архітектура фізичного движка та проводиться її модифікація з метою підвищення ефективності алгоритмів обробки фізичних об'єктів за критерієм швидкодії. На основі отриманої архітектури здійснена програмна реалізація основних класів фізичного движка.

This article is devoted to the question of development of the software architecture for physical processes modeling. The new *SIM* approach to the organization of simulation space's objects is proposed. It is oriented to the maximum abstraction of the latter. Basic steps of software creating according to the described approach are defined. *SIM* architecture of the physics engine is implemented and its modification for enhance the effectiveness of physical objects processing algorithms on the criteria of speed performance is considered. The software implementation of major classes of the physical engine is done based on the *SIM* architecture.

Вступ

У галузі створення сучасної анімації і розробки комп'ютерних ігор все частіше виникає необхідність у побудові програмних систем, здатних реалістично відтворювати різні фізичні явища. Використання таких систем сприяє значному прискоренню процесу створення комп'ютерної мультиплікації, оскільки автоматизована обробка «фізики» знімає необхідність відтворення останньої вручну. В комп'ютерних іграх програмне моделювання фізичних процесів дозволяє зробити графічне зображення віртуального світу максимально подібним до реального світу.

Існуючі у даній галузі програмні рішення здатні відтворювати закони динаміки, статики, гідродинаміки, фізики газів, тканин тощо. Всі вони мають оптимізований щодо часу виконання код і, як правило, можуть виконувати моделювання фізичних законів в реальному часі.

Їх спільним недоліком є архітектура, що дозволяє працювати з описами лише заданого заздалегідь спектра законів фізики. Робота з даними, які стосуються інших областей фізики, або неможлива, або потребує перебудови движка. Тому пошук нових рішень у галузі програмного моделювання фізичних процесів є актуальною задачею.

З огляду на вищезазначене метою даної роботи є створення відкритого до розширення функціонального ядра програмних засобів для моделювання фізичних процесів.

Відповідно до вказаної мети в роботі поставлені і розв'язані такі задачі:

- визначення підходу до побудови та розробка базової архітектури фізичного движка;
- модифікація отриманої архітектури для забезпечення реалізації ефективних за критерієм швидкодії алгоритмів обробки фізичних об'єктів.

1. Огляд існуючих рішень

Програмний комплекс, що реалізує симуляцію фізичних законів реального світу з деякою ступінню апроксимації, називається ***фізичним движком*** [1]. Під ***симуляцією*** в даному випадку будемо розуміти програмну імітацію фізичного процесу або об'єкта [2].

Поняття ***архітектура*** програмного забезпечення включає в себе:

1. *Набір значущих рішень* з приводу організації системи програмного забезпечення.

2. Набір *структурних елементів* та їх інтерфейсів, за допомогою яких компонується система, разом з їх поведінкою, що визначається взаємодією між цими елементами.

3. *Компоновку* елементів у підсистеми, що поступово розширюються.

4. *Стиль архітектури*, котрий спрямовує цю організацію – елементи та їх інтерфейси, взаємодії та компоновку [3].

Статичною будемо називати таку архітектуру фізичного движка, що не підтримує (або робить складною) можливість підключення до нього бібліотек обробки даних щодо фізичних законів або колекцій законів з цілих розділів фізики.

Проаналізуємо існуючі фізичні движки з точки зору їх архітектури:

- ***Box2D*** – зручний у використані та ефективний з точки зору швидкодії движок з відкритим кодом. Існують його реалізації різними мовами, у тому числі *ActionScript*. Вважається одним з кращих двовимірних движків [4, 5].
- ***NAPE*** [6, 7] – новий відкритий движок; зараз закінчується розробка, але вже йде широке застосування. Автор движка анонсував реалізацію широкого спектру фізичних законів – не лише механіки, зокрема йдеться про симуляцію води [8].
- ***ODE*** **(***Open Dynamics Engine***)** [9] – відкритий, частково безкоштовний тривимірний фізичний движок, орієнтований на реалістичне відтворення складних з'єднань (шарнірів, пружин, тощо).

Побудова вищезгаданих движків. Головний контейнер містить у собі менеджери, наприклад: менеджер обробки колізій, розподільник пам'яті движка (потрібен для реалізацій движків на нескриптових мовах програмування), а також список об'єктів та список осей, що їх фіксують. Вищезгадані менеджери описані в окремих класах. Програмні об'єкти, що зберігають динамічні властивості, містять велику кількість даних щодо фізичних властивостей тіла і одночасно виконують роль об'єктів віртуального простору. Це робить опис нових для движка властивостей незручним у використанні.

Відтворення фізичного впливу на об'єкт моделювання у коді здійснюється за допомогою виклику спеціальних функцій додавання сил та імпульсів.

Слід помітити також, що в системі класів усіх движків, що розглядаються, існує поняття *зв'язків* (*joints*), які описують взаємодію між двома об'єктами. Але для жодного з движків можливість створення користувацьких *joints*-ів не розглядалася як головна ідея побудови архітектури движка.

Таким чином, можна зробити висновок, що за ідеєю розробників ***архітектура*** вищезгаданих движків ***є статичною***.

- ***APE*** [10] – відкритий, дещо застарілий фізичний движок.

Побудова: Движок не має структури, що описує об'єкт фізичного простору. Є набір алгоритмів і класів для фізичного моделювання, та пошук колізій між формами, що описуються багатокутниками. З метою спрощення роботи у *APE* введене поняття груп, у які розробник може об'єднувати об'єкти.

Архітектура движка є незакінченою**:** взагалі не описане поняття фізичного об'єкту. На основі движка можлива реалізація гнучкої до розширення архітектури, але на досліджуваному етапі розробки *APE* фактично є лише калькулятором фізичних законів.

Примітка 1: У даному огляді не наведена інформація про комерційні движки через те, що їх програмний код є закритим, а, отже, неможливо детально розглянути їх архітектуру. Однак, слід згадати такі движки як ***PhysX*** і ***Havok***, що є одними з найкращих комерційних рішень в області моделювання фізичних об'єктів (вони використовують апаратну оптимізацію і паралельне програмування для прискорення обчислень).

Судячи з прикладів коду програмних продуктів, що їх використовують [11, 12], архітектура цих движків також є ***статичною***.

Примітка 2: Також слід згадати програмний продукт ***Phun*** та його комерційну версію – ***ALGODOO*** [13]. Останній здатен моделювати процеси, напевне, найбільшого спектру розділів фізики серед усіх сучасних систем програмного моделювання: динаміки, статики, гідродинаміки та оптики. Мінусом є закритість системи. ***ALGODOO*** не є бібліотекою моделювання «фізики», це закінчений програмний продукт з реалізованим графічним інтерфейсом.

Таким чином, доцільним є проведення дослідження можливості створення програмної системи моделювання фізичних процесів з об'єктами, що є максимально абстрагованими.

2. Обґрунтування ідеї побудови движка

Об'єкт віртуального простору доцільно визначати, включаючи до його структури екземпляри класів-контролерів, що описують певні ідейно різні сутності об'єкту - ***зони відповідальності***. Така побудова проекту робить код проекту більш структурованим і зручним.

У нашому випадку об'єкт віртуального простору включатиме в себе екземпляр графічного об'єкту, зоною відповідальності якого є обробка зовнішнього вигляду об'єкту (графічний об'єкт не буде розглянутий детально у даній статті), і екземпляр фізичного об'єкту, що буде здійснювати контроль фізичної складової об'єкту.

Фізичним об'єктом будемо називати сукупність двох складових: набору властивостей об'єкту (положення у просторі, маса, швидкість тощо), і законів (правил), що впливають на об'єкт (закон тяжіння, правило пружного зіткнення твердих тіл тощо).

Найбільш поширеним підходом щодо реалізації фізичних об'єктів є об'єктноорієнтований підхід (див. архітектуру будь-якого фізичного движка), відповідно до якого програмний опис об'єкту постає у вигляді класу. Поля такого класу є фізичними властивостями об'єкту, а методи – функціями моделювання впливу на останні (приклад: вплив на об'єкт силою або імпульсом).

Перевагою використання вказаної структури програмного об'єкту є швидке створення архітектури проекту, мінусами – відсутність гнучкості розробки і реалізація лише фіксованого, визначеного заздалегідь спектру способів впливу на фізичні властивості. Така реалізація можлива в програмних комплексах, що не потребують відтворення складної поведінки об'єктів або покладають її реалізацію на користувача.

Реалізація більш складної поведінки об'єктів базується на врахуванні законів з різних розділів фізики та (або) відтворенні більш складних відношень між об'єктами. Вона потребує гнучкої, здатної до розширення архітектури движка, а отже, постає необхідність максимального абстрагування фізичного об'єкту. Для цього:

1) фізичний об'єкт не може мати однозначного набору властивостей: користувач сам повинен визначити цей набір, включаючи їх у об'єкт за необхідності;

2) фізичний об'єкт не може мати однозначного набору правил, що будуть впливати на нього: правила можна додавати до об'єкту, але їх набір не може бути фіксованим.

Поставлені задачі не можуть бути реалізовані з використанням лінійних відношень між класами. Вони потребують розробки більш розгалуженої, ієрархічної архітектури програмних засобів.

3. SIM підхід

У даній статті пропонується підхід до побудови архітектури програмних комплексів, реалізація якого дозволить гнучко редагувати набори використовуваних властивостей та методів об'єктів. Новий підхід – ***SIM підхід (Shell-Influence-Model)*** – побудований на основі об'єктноорієнтованої парадигми програмування, ідеї так званого предметноорієнтованого підходу до програмування (subject-oriented programming) [14] та використанні шаблону проектування *стратегія* [15]. В межах *SIM* поведінка об'єктів і групи властивостей (принцип об'єднання вказаний нижче) розглядаються як окремі сутності.

Підхід має такі особливості:

1. Відокремлені групи властивостей забезпечують збереження стану об'єкту, описуючи параметри останнього з точки зору певного підрозділу предметної області.

2. Відокремлена поведінка визначає вплив на включені до об'єкта властивості та може реалізовувати одну або декілька функцій впливу на властивості - *функцій збудження*. Інтерфейси цих функцій є спільними для усіх об'єктів відокремленої поведінки.

Головна ідея підходу полягає у:

1) мінімізації розмірів коду за рахунок внесення засобів обробки складного впливу на об'єкт (систему об'єктів) безпосередньо до структури об'єкта – включенням відокремленої поведінки у об'єкт;

2) універсалізації опису об'єктів за рахунок можливості включення до них будь-якої комбінації властивостей без необхідності подвійного (або множинного) успадкування.

Центральним елементом нового підходу є головний клас-оболонка – *shell-class* (у нашому випадку – фізичний об'єкт), що виступає у ролі контейнеру, який зберігає списки класів-методів (*influence-class*) і класів-властивостей (*model-class*), які будемо надалі називати *наповненням shell-class-у* (або просто ***наповненням***). При цьому *shell-class* здійснює мінімальний контроль свого *наповнення*: реалізовує фу-

нкції включення та вилучення екземплярів *influence* та *model –class*-ів з контейнеру, а також методи виклику спільних для класів-методів функцій збудження. Останній слугує для ініціалізації процесу обробки стану об'єкту включеними до нього екземплярами *influence-class*-ів.

Розглянемо основні кроки створення проекту відповідно до *SIM* підходу:

1. Описати абстрактні класи: базова властивість і базовий метод (base *influence-class* і base *model-class*). Вони мають описувати функції роботи з ієрархією проекту (наприклад, з посиланнями на батьківський контейнер *shell-class*), а базовий метод повинен містити ще як мінімум одну *функцію збудження*.
2. Виділити логічно відокремлені підрозділи в обраній предметній галузі.
3. Формалізувати виділені підрозділи, розглядаючи властивості (характеристики, параметри) кожного з них як поля *model-class*-у, що його описують, а закони – як функції збудження *influence-class*-ів. Вищезазначені компоненти реалізують відповідні абстракції, описані у пункті 1. Слід відмітити, що *model-class* може включати деякі елементарні операції впливу на власні властивості, які будуть викликатись для реалізації складної поведінки в тілі функцій збудження *influence-class*-ів.
4. Описати *shell -class*, керуючись його визначенням (див. вище).
5. Розробити системи перевірки валідності включень (див. розділ 4.1), орієнтовану на обрану предметну галузь.

Архітектура програмних засобів, побудованих відповідно до *SIM* підходу, є гнучкою і відкритою для розширень: фактично, в основі архітектури лежить реалізація *динамічного класу*, до якого можна у будь-який час під'єднувати методи і властивості, а також легко виключати їх. Така архітектура є зручною у роботі:

- для користувача: не потрібно бути обізнаним у предметній області для створення складних взаємодій між об'єктами, адже достатньо включити у проект бібліотеки їх реалізацій;
- для розробника системи: нові закони потребують реалізації лише конструктору, функції збудження й опису властивостей закону. Відсутні проблеми з читанням величезної документації до продукту. Достатньо знати основні принципи організації і функціонування движка, а також структуру *model-class*-ів та – за необхідності – методів, з якими повинен взаємодіяти модуль, що розробляється.

SIM підхід має сенс використовувати у випадках, коли програмний комплекс, що розробляється, розрахований на:

- роботу з предметною областю, що має багато (більше трьох) підрозділів;
- здійснення стратегічного контролю користувачем програмного забезпечення - користувач задає (описує сам) блоки, що будуть впливати на стан комплексу, й після цього здатен викликати лише функції обробки усіх блоків, не маючи безпосереднього впливу на окремі складові об'єкту;
- реалізацію великого спектру складних алгоритмів у кожній окремо визначеній предметній області.

Розглянемо побудову архітектури фізичного движка відповідно до запропонованого *SIM* підходу.

4. Розробка базової архітектури движка

Відповідно до *SIM* підходу роль *model-class*-ів в архітектурі виконують описи згрупованих за певними розділами (динаміка, гідродинаміка, електрика, оптика тощо) фізичних властивостей об'єкту (далі будемо називати їх ***пакетами властивостей***); *influence–class*-си описують фізичні закони, які моделюються (закон пружності, закон Стокса, закон Ома, закон відбивання світла. Далі називаються ***законами***). *Shell-class*-ом буде слугувати *фізичний об'єкт*.

4.1. Перевірка валідності включень

Побудована відповідно до запропонованого підходу архітектура движка є досить гнучкою, але, водночас, вразливою до помилок. Для її коректної роботи необхідна система перевірки валідності включення нових *законів* до *фізичного об'єкту*.

Владним включенням будемо називати дію включення до *фізичного об'єкту* такого нового *наповнення*, що не протирічить зробленим раніше включенням (наприклад, подвійне включення опису однакових областей), й має у *фізичному об'єкті* необхідні для коректної роботи компоненти. Прикладом, коли може виникати помилка, є така ситуація: користувач намагається додати у фізичний об'єкт *закон*, що реалізує силу тяжіння, не додавши перед цим пакет властивостей динаміки.

У статті пропонується два способи перевірки валідності включень:

1. Використання класів-заголовків. Кожен *пакет властивостей* і кожне *правило* оснащуються заголовком, що описує умови та інформацію, необхідні для валідного включення даного *пакету властивостей* / *правила*.

Переваги: Швидка реалізація класів-заголовків. Також останні можна використовувати для уточнення режиму обробки правил у фізичному об'єкті. Приклад: необхідною є обробка сумарної швидкості після обчислення сили тертя - остання повинна зменшити швидкість у декілька разів.

Недоліки: Фіксованість полів у заголовку внаслідок того, що алгоритм обробки полів є статичним, призводить до необхідності розробки складної системи додавання надбудованих алгоритмів перевірки валідності включення.

2. Використання спеціальної функції *setParent( )* у пакетах властивостей і правилах. За допомогою даної функції можна перевірити валідність включення. Функція повертає булеве значення в залежності від результату перевірки.

Переваги: Як завгодно гнучка перевірка валідності включення – її реалізація покладається на розробника *наповнення*.

Недоліки: Необхідність реалізації додаткової функції для розробника бібліотек движка.

На думку авторів, найбільш зручним є комбінований метод перевірки. У такому випадку заголовок зберігає інформацію щодо особливостей включення і обробки законів, а метод *setParent( )* – перевіряє *валідність включення*.

Розглянемо можливість розширення отриманої базової архітектури движка з метою забезпечення ефективної за критерієм часу виконання глобальної обробки фізичних об'єктів.

5. Модифікована SIM архітектура фізичного движка

5.1. Складність обробки загальних правил

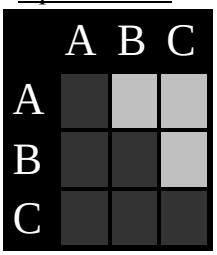
Нехай є динамічна система твердих пружних тіл. Дана система має обмеження: об'єкти не можуть проходити один через одного. Для цього потрібен алгоритм, що буде обробляти зіткнення. В чистій *SIM* системі подібна модель може бути реалізована за допомогою включення правил обробки зіткнень у кожний об'єкт системи. Внаслідок цього:

1) порушується принцип інкапсуляції, оскільки буде мати місце звертання до списку об'єктів безпосередньо із правила, що є зайвим посиланням до області відповідальності іншого класу;

2) підвищується загальна складність системи внаслідок наявності великої кількості додаткових зв'язків у ієрархії об'єктів;

3) виникає необхідність у виконанні зайвих обчислень або в ускладненні ієрархії (див. пояснення у Примітці 3).

Примітка 3: Перевірка колізій шляхом повного перебору потребує $N*(N-1)$ операцій пошуку (матриця, без головної діагоналі). Пошук можна оптимізувати, спираючись на факт, що якщо *A* зіткнувся з *B* то й *B* зіткнувся з *A* (світліша область). За рахунок цього час пошук зіткнень буде зменшено у два рази.



При використанні звичайного *SIM* підходу у даному випадку подібну оптимізацію зробити дуже складно. Можна, наприклад, ввести прапорець, що буде зберігати статус обробки зіткнення і не оброблятиме надалі колізії між об'єктами, що зіткнулися, однак таке рішення є очевидно громіздким. Подібні труднощі будуть виникати з усіма правилами, що залежать від не тільки від стану об'єкту, на який впливає дане правило, але від стану всієї системи в цілому.

Таким чином, реалізація простору моделювання у вигляді лише списку фізичних об'єктів не є оптимальною. Потрібен додатковий механізм – обгортка навколо списку об'єктів – що буде здійснювати менеджмент і оптимізацію роботи з простором моделювання.

З цього слідує необхідність введення поняття *фізичної системи*.

5.2. Фізична система

Фізичною системою будемо називати клас-контейнер, що включає список фізичних об'єктів, список служб та глобальних законів. ***Службою*** будемо називати деякий математичний (геометричний) алгоритм, обгорнутий класом на зразок правила, що здійснює обчислення допоміжних параметрів системи. Результат роботи служби може бути використаний як правилом, так і деяким зовнішнім для фізичної моделі об'єктом. В останньому випадку службу можна розглядати як диспетчер подій, а результат його роботи – як подію.

Глобальний закон – це *influence*-клас, функція збудження якого реалізує відтворення фізичних законів, що впливають одразу на всі об'єкти фізичної системи.

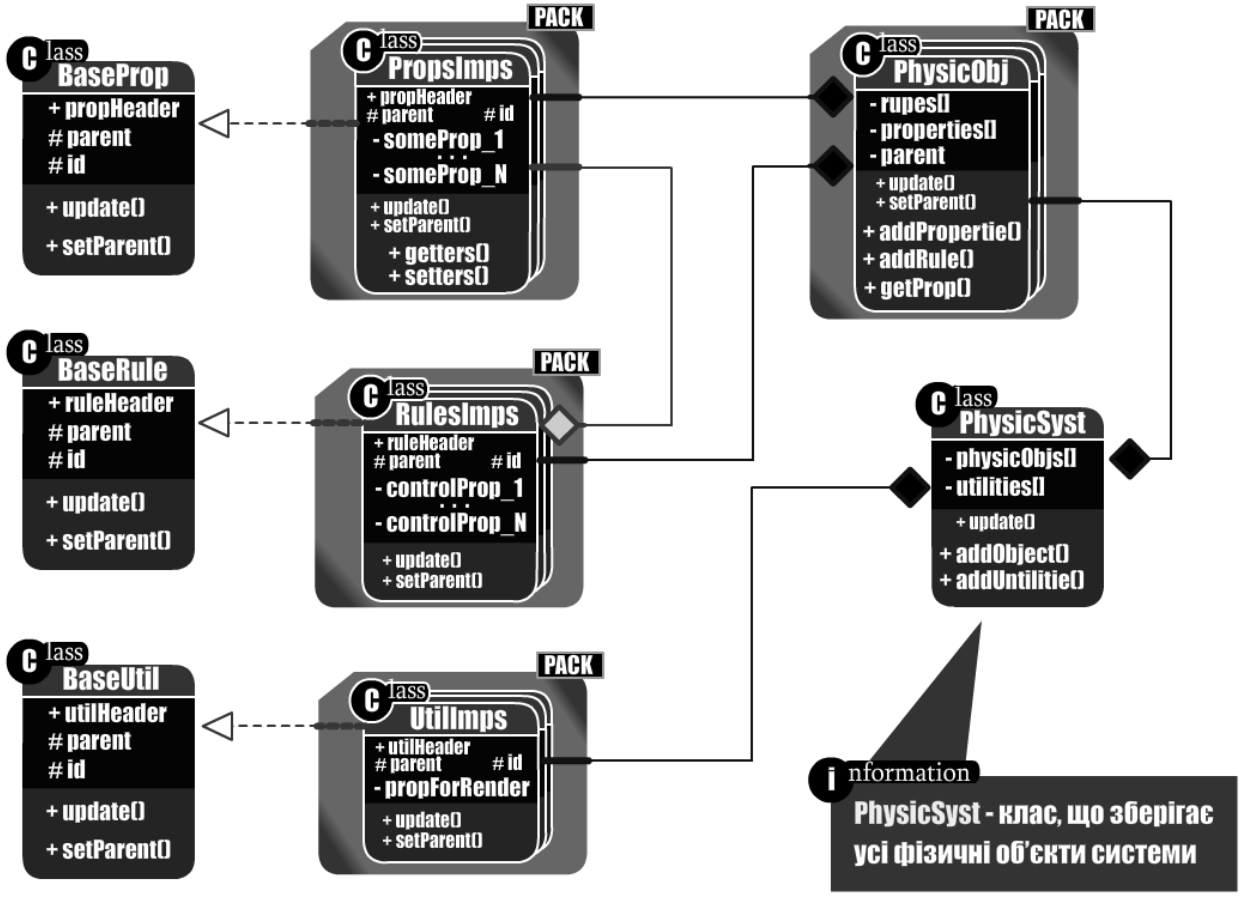


Рис.1. Загальна *UML*-діаграма архітектури *SIM* движка

Примітка 4: Прикладом *служби* може бути алгоритм обробки колізій. Даний алгоритм передбачає пошук геометричних перетинів тіл, описаних в пакеті динамічних властивостей кожного об'єкту системи. Однак результат роботи цієї *служби* – список перетинів тіл - є дуже важливим для реалізації будь-якої динаміки твердих тіл. *Закони*, що обробляють дії об'єктів моделі при зіткненні, можуть включати в свою структуру посилання на цю *службу*.

Приклад глобального закону: обробка результату зіткнень твердих тіл. Обробка повинна виконуватись для всіх об'єктів системи, що включають в свої описи пакети динамічних властивостей.

Висновки

На основі дослідження існуючих програмних засобів моделювання фізичних процесів аргументовано доцільність модифікації загальновживаного об'єктноорієнтованого підходу до створення такого роду ПЗ з метою реалізації відкритого до розширення фізичного движка.

Запропоновано новий *SIM* підхід до створення програмних комплексів опису складних систем фізичних об'єктів, розроблено базову *SIM* архітектуру фізичного движка, дано визначення основних складових архітектури. Отримана архітектура дозволяє спростити додавання до фізичного движка елементів, що описують різні розділи фізики.

Проведено першу модифікацію *SIM* архітектури з метою забезпечення реалізації ефективних за критерієм швидкодії алгоритмів обробки фізичних об'єктів, а також даних щодо загального стану системи, що моделюється. Результатом модифікації стало введення додаткових елементів до архітектури: фізичної системи, служби, глобального закону.

Серед напрямків подальшого вивчення та розвитку запропонованого підходу можна виділити такі:

- створення програмних бібліотек законів і властивостей для моделювання конкретних областей фізики;
- аналіз можливості побудови та використання додаткових контролерів, що описують

закони взаємодії між набором об'єктів і контролеру макросів фізичної імітації, що реалізовують послідовну генерацію сил протягом деякого часу (приклад: м'язи людини під час бігу);

- подальше удосконалення архітектури з точки зору зручності розробки коду на її основі.

Програмна реалізація движка виконана на базі мови ActionScript3.

СПИСОК ЛІТЕРАТУРИ

1. Физический движок [Електронний ресурс / http://ru.wikipedia.org/wiki/Физический_движок], дата візиту 27.10.2010
2. *Е.М.Пройдаков, Л.А.Теплицький* Англо-український тлумачний словник з обчислювальної техніки, Інтернету і програмування. – Вид.1 – К.: Видавничий дім «СофтПрес», 2005. – 552с.
3. Что такое архитектура программного обеспечения? [Електронний ресурс / http://www.ibm.com/developerworks/ru/library/eeles/#notes], дата візиту 30.11.2010
4. Физика на Flash. Box2D Engine. [Електронний ресурс / http://habrahabr.ru/blogs/flash/76611/], дата візиту 15.11.2010
5. Документація до відкритого движка Box2D (Box2D v2.1.0 User Manual) [Електронний ресурс / http://www.box2d.org/manual.html] , дата візиту 20.11.2010
6. Физика на Flash. Создание Ragdoll в Nape на AS3 [Електронний ресурс / http://habrahabr.ru/blogs/Flash_Platform/104176/], дата візиту 26.11.2010
7. Документація до движка Nape на AS3. Chapter 2. Spaces and Objects [Електронний ресурс / http://www.deltaluca.me.uk/doc/docs/Chapter%202.%20Spaces%20and%20Objects.html], дата візиту 21.11.2010
8. Polygon water [Електронний ресурс/ http://www.newgrounds.com/dump/item/8e04592be7f7a7d4262a20483734f582], дата візиту 10.11.2010
9. Документація до відкритого движка ODE [Електронний ресурс / http://opende.sourceforge.net/wiki/index.php/Manual_(All)], дата візиту 17.11.2010
10. Документація до відкритого движка APE (Actionscript Physic Engine), [Електронний ресурс /http://www.cove.org/ape/docs/api/org/cove/ape/package-detail.html], дата візиту 2.11.2010
11. Irrlicht + PhysX example, [Електронний ресурс / http://irrlicht.sourceforge.net/phpBB2/viewtopic.php?t=19002&sid=98ad59f05dcebf57d0b5207a3dae75a2], дата візиту 23.11.2010
12. Havok code example [Електронний ресурс / http://director-online.com/havok/demos/demo-marble.html], дата візиту 23.11.2010.
13. ALGODOO wiki [Електронний ресурс/ http://www.algodoo.com/wiki/Home], дата візиту 17.11.2010
14. Subject-oriented programming [Електронний ресурс/ http://en.wikipedia.org/wiki/Subject-oriented_programming], дата візиту 7.11.2010
15. Pattern Strategy [Електронний ресурс http://ru.wikipedia.org/wiki/Strategy], дата візиту 7.11.2010
16. *Семенякін В.С, Заболотня Т.М.* Узагальнений алгоритм моделювання фізичних процесів на базі мови програмування C# // "Прикладна математика та комп"ютинг ПМК-2010": Зб.тез доповідей. - К.:Просвіта, 2010. - с.353-356.
17. Non-core joints [Електронний ресурс / http://www.box2d.org/forum/viewtopic.php?f=4&t=3970], дата візиту 1.12.2010

V.Semenyakin, T.Zabolotnia **Modified object-oriented approach to building software tools for modeling physical processes.**

В.С.Семенякин, Т.Н.Заболотняя **Модифицированный объектноориентированный подход к построению программных средств моделирования физических процессов**

В статье рассматривается вопрос построения архитектуры программных средств моделирования физических процессов. Предлагается новый *SIM* подход к организации объектов пространства моделирования, ориентированный на максимальное абстрагирование последних. Определяются основные шаги создания программного обеспечения в рамках описанного подхода. Разрабатывается *SIM* архитектура физического движка и проводится её модификация с целью повышения эффективности алгоритмов обработки физических объектов по критерию быстродействия. На основе полученной архитектуры выполнена программная реализация основных классов физического движка.