ISSN 2078–5097. Вісн. Львів. ун-ту. Сер. пр. матем. та інф. 2012. Вип. 18. С. 208–221
Visnyk of the Lviv University. Series Appl. Math. and Informatics. Issue 18. P. 208–221

І Н Ф О Р М А Т И К А

УДК 004.4

ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОДЕЛЮВАННЯ ПОВЕДІНКИ АГЕНТІВ У ЗАДАЧІ НАВІГАЦІЇ

О. Годич, П. Кушнір, Ю. Щербина

Львівський національний університет імені Івана Франка,
вул. Університетська, 1, Львів, 79000, e-mail: dais@franko.lviv.ua

Розглянуто задачу побудови програмного забезпечення для моделювання поведінки інтелектуальних агентів у віртуальному середовищі. Проведено аналіз існуючих рішень, окреслено їхні переваги та недоліки. Сформульовано основні вимоги до систем такого типу. Розглянуто розроблене програмне забезпечення, яке якісно реалізує поставлені вимоги. Наведено опис отриманої системи, а також обґрунтовано її ефективність і конкурентоспроможність.

Ключові слова: програмне забезпечення, моделювання, віртуальне середовище, інтелектуальний агент, задача навігації.

1. ВСТУП

Із розвитком технологічних досягнень людства робототехніка зробила великий крок вперед. Алгоритми, які використовують для розв'язання задач позиціонування і навігації, в сучасній робототехніці стають дедалі складнішими. Це стосується аспектів пов'язаних з реалізацією алгоритмів на обчислювальній машині, аспектів, які пов'язані з вивченням і розумінням цих алгоритмів. Із таким ускладненням алгоритмів пов'язана необхідність у можливості тестового запуску цих алгоритмів, щоб надалі відлагодити і вдосконалити. Очевидно, що для таких цілей використання апаратних рішень (з використанням агента (робота) як виконавця алгоритму) недоцільне. Це спричинено обмеженістю технічних можливостей агента та складністю у проведенні тестових запусків.

У зв'язку з цим за останні десятиліття було розроблено велику кількість програмних рішень здатних моделювати поведінку агентів у навколишньому середовищі. Проте через орієнтованість лише на окремі тестові запуски, для підготовки яких потрібен досить тривалий час, ці системи практично не можливо використовувати для алгоритмів, які потребують навчання.

Ми розглянули задачу побудови середовища моделювання поведінки агентів у задачі навігації. Проаналізували існуючі рішення, сформулювали основні вимоги до середовища моделювання. Подали опис програмного забезпечення, що відповідає зазначеним вимогам, з'ясували переваги його використання.

2. ОСНОВНІ ПОНЯТТЯ ТА ОГЛЯД СТАНУ ПРОБЛЕМИ

2.1 *ОСНОВНІ ПОНЯТТЯ*

Концепцію мобільного агента детально описано в [1]. У цьому дослідженні під терміном *агент* розуміємо деяку сутність, що отримує інформацію про навколишнє середовище за допомогою сенсорів і діє у ньому за допомогою маніпуляторів. Кажуть, що поведінка агента раціональна, якщо він здатен до аналізу і його дії завжди спрямовані на досягнення певної мети. Такий агент може бути і роботом, і вбудованою програмною системою. Про інтелектуальність агента можна говорити

тоді, коли він взаємодіє з навколишнім середовищем приблизно так само, як взаємодіяла би людина.

Мобільним агентом є автономний об'єкт, який може пересуватись у навколишньому середовищі, використовуючи обчислювальні ресурси комп'ютера й дані сенсорів, що дають зазвичай не повну і не точну інформацію про навколишнє середовище.

Модель – це подання об'єкта, системи чи поняття у формі, яка відрізняється від реальної, але яка наближена до неї, враховуючи й набір даних, якими характеризується такий об'єкт (система), і динаміку зміни цих даних у часі [4].

Віртуальне середовище у нашому випадку є комп'ютерною моделлю деякого приміщення / відкритого простору, яка містить об'єкти-перешкоди і в якій може пересуватись мобільний агент. У віртуальному середовищі задано ціль, яку потрібно досягнути мобільному агенту в процесі своєї роботи – це частина приміщення/простору, досягнувши якої робота агента вважається успішною.

Задача навігації (задача пошуку шляху) мобільного агента полягає в автоматичному генеруванні шляху, який би пролягав між кількома наперед заданими точками. Цю задачу можна звести до задачі пошуку шляху до деякої наперед заданої точки на карті, який би оминав перешкоди, що розміщені в навколишньому середовищі.

Надалі під поняттям *алгоритм* будемо розуміти алгоритм керування мобільним інтелектуальним агентом у віртуальному середовищі, що розв'язує задачу навігації.

2.2. *ОГЛЯД СТАНУ ПРОБЛЕМИ*

Описані концепції стають все популярнішими серед розробників і науковців. Це зумовлено практичною важливістю отриманих результатів, адже інтелектуальні роботи можуть бути корисними для оптимізації процесу виробництва, для поліпшення умов побуту. Тому створено декілька систем комп'ютерного моделювання віртуального середовища.

Розглянемо детальніше деякі існуючі системи

- Microsoft Robotics Developer Studio [7] – комерційний проект, що забезпечує можливість створювати програмне забезпечення для агентів. Підтримує широкий спектр існуючих апаратних рішень. Дає змогу відлагоджувати алгоритми та спостерігати за їхньою роботою у власноруч створеному середовищі.
  Переваги: можливість відлагодження алгоритмів; підтримка широкого спектра апаратних рішень; широкий спектр мов програмування, які використовують для написання алгоритмів.
  Недоліки: складність у використанні; погана пристосованість для алгоритмів, що потребують навчання; недостатня підтримка порівняння ефективності алгоритмів; комерційний проект.
- Gazebo [8] – проект з відкритим вихідним кодом, що забезпечує можливість моделювання поведінки агентів у віртуальному середовищі.
  Переваги: вільне програмне забезпечення; широка підтримка апаратних рішень; реалістичність при симуляції сенсорів.
  Недоліки: необхідність компіляції всього проекту при зміні алгоритму; погана пристосованість до алгоритмів, що потребують навчання; недостатня підтримка порівняння ефективності алгоритмів.

- Stage [8], [9] – схожий до Gazebo програмний пакет, який переважає його своєю швидкістю, проте завдяки цьому втрачається точність моделювання.
  Переваги: вільне програмне забезпечення; висока швидкість роботи; підтримка одночасної роботи великої кількості агентів; підключення алгоритму без компіляції всього проекту.
  Недоліки: складність відлагодження алгоритмів; погана пристосованість до алгоритмів, що потребують навчання; недостатня підтримка порівняння ефективності алгоритмів; обмеженість у виборі мови програмування (тільки C++).
- Moby [10] – це відкрите програмне забезпечення для максимально реалістичного моделювання фізичної взаємодії агентів.
  Переваги: відкрите програмне забезпечення; точність моделювання; можливість зручного відлагодження.
  Недоліки: вузька спеціалізація; низька швидкість роботи; погана пристосованість до алгоритмів, що потребують навчання; недостатня підтримка порівняння ефективності алгоритмів; обмеженість у виборі мови програмування (C++).

Як бачимо, більшість створених сьогодні систем досить вузькоспеціалізовані і недостатньо гнучкі для розробки всього спектра алгоритмів для інтелектуальних агентів, зокрема алгоритмів, що потребують навчання. Вузька спеціалізація таких систем також не дає змоги використовувати їх при подальшому розширенні досліджень у бік порівняння ефективності алгоритмів, а також отримання статистичних даних щодо результатів роботи алгоритму.

Ще одним недоліком більшості існуючих систем є погана пристосованість до тестування і відлагодження алгоритмів, які використовують цю систему. Фактично, вони створені для демонстрації роботи алгоритму, а не для відлагодження і тестування його роботи. Зауважимо, що автори не знають вітчизняних напрацювань у цьому напрямі.

Отже, для визначеної мети треба створити нову платформу, яка б забезпечила можливість ефективно проводити тестування і відлагодження алгоритмів, використовуючи комп'ютерну модель віртуального середовища, з можливістю проведення порівняльного аналізу ефективності отриманих алгоритмів.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1. *ВИМОГИ ДО СИСТЕМИ*

Для успішного використання середовища було сформульовано такі вимоги.

1. Середовище повинно підтримувати динамічне підключення алгоритмів до системи. Це забезпечить економію часу, потрібного для початку використання системи.
2. Середовище повинно надавати функціональність зручного відлагодження алгоритмів. Це необхідна умова для успішного створення алгоритмів.
3. Середовище повинно забезпечувати автоматизацію процесу навчання алгоритмів.
4. Середовище повинно давати змогу проводити порівняльний аналіз ефективності алгоритмів.
5. Середовище повинно давати змогу використовувати для написання алгоритмів мову програмування високого рівня. Це допоможе писати короткий і зрозумілий програмний код, що полегшить розробку алгоритмів.

6. Середовище повинно бути максимально гнучким і масштабованим. Це дасть змогу розширювати його для роботи з ширшим набором алгоритмів і агентів.
7. Для створення середовища треба використовувати концепції відкритого програмного забезпечення. Це дасть підстави вести розроблення проекту з використанням мінімальних затрат і зацікавити більшу кількість науковців і студентів до його використання.

3.2. *СТВОРЕННЯ ОБ'ЄКТНОЇ МОДЕЛІ СИСТЕМИ*

Для розв'язання задачі створення інформаційної системи моделювання поведінки інтелектуальних агентів обрано об'єктно-орієнтований підхід. Це означає, що вся логіка, яка є в цьому програмному забезпеченні, розбивається на дрібніші частини, які взаємодіють між собою.

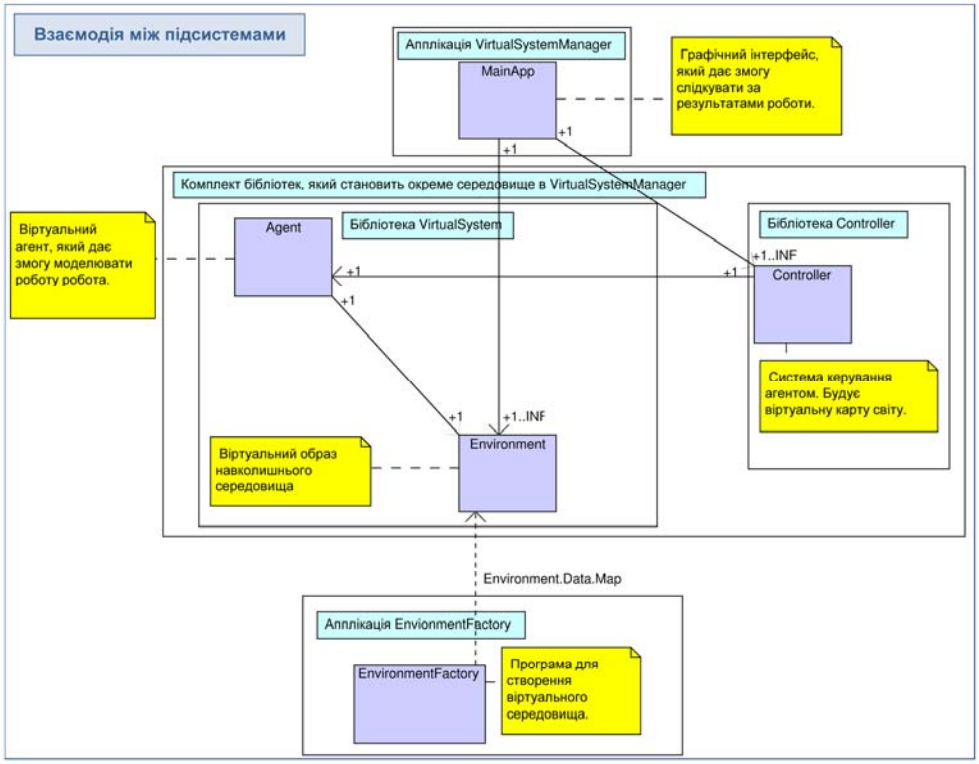

Рис. 1. Компонентна діаграма розробленого програмного забезпечення

Проте при застосуванні цього підходу виникають деякі труднощі. Здебільшого вони пов'язані з необхідністю розподілу обов'язків між об'єктами. Коли розглядають окремо взятий об'єкт, задача розв'язання поставлених перед ним обов'язків повинна бути досить тривіальною, оскільки саме для цього і проводили декомпозицію програмного забезпечення на об'єкти. З іншого боку, необхідною є також можливість розширення існуючого програмного рішення, додавання нових можливостей, застосування нових технологій для розв'язання сформульованої задачі. Саме тому правильна декомпозиція на об'єкти (компоненти) є запорукою успішного розвитку системи в подальшому [3].

Для ліпшого розуміння і наочності при виконанні декомпозиції використовують графічне подання типів (класів) об'єктів. Це інформація про деяку сукупність однотипних об'єктів. Для створення таких графічних зображень за роки розвитку об'єктного-орієнтованого підходу створено багато різних методів і підходів. Згодом ці методи і підходи були злиті в єдине ціле і, у підсумку створено мову моделювання UML. UML (Unified Modeling Language) – це мова моделювання, яка на сучасному етапі розвитку є сукупністю нотацій і метамоделей. Нотація – це сукупність графічних зображень, які використовують у моделях; вона є синтаксисом цієї мови. З іншого боку, метамодель – це правила, яких треба дотримуватись при створенні такої нотації. UML став de facto стандартом у створенні подібних зображень [2]. Саме тому його й було вибрано як мову моделювання при створенні цього проекту.

Розглянемо за допомогою UML-діаграм як зроблено розподіл обов'язків у створеній інформаційній системі моделювання поведінки інтелектуальних агентів. На рис. 1 зображено компоненти програмного забезпечення. Як бачимо, весь проект поділено на компоненти та бібліотеки.

Зокрема, до компонентів належать:

1) EnvironmentFactory – допоміжна аплікація, яка забезпечує зручне створення віртуального середовища, в якому працюватиме агент;
2) Agent – модель агента. Зокрема, це стосується внесення похибок до даних, які насправді обчислюються середовищем (наприклад, у реальному житті радар агента не може дати абсолютно точні дані щодо розміщення навколишніх об'єктів, тому віртуальний агент повинен внести деяку штучну похибку до цих даних);
3) Environment – інкапсулює модель навколишнього середовища (перешкоди, які треба обходити, ціль яку потрібно досягнути тощо);
4) MainApp – адміністративний модуль, який дає змогу керувати і стежити за роботою всієї системи. Цей модуль реалізований у вигляді окремої аплікації;
5) Controller – інкапсулює алгоритм, який треба протестувати і/або відлагодити. Також містить інтерфейс для роботи з даними, які отримують від інших компонент аплікації;
6) зовнішній пакет класів містить ті сутності, які використовують відразу кілька компонент системи. Зокрема, це структури даних, за допомогою яких відбувається передача інформації між компонентами системи.

Ці компоненти поділено на відповідні бібліотеки для ефективного динамічного завантаження алгоритму без потреби компіляції всієї системи.

Ефективність використаного поділу підтверджується Sequence-діаграмою, яка зображена на рис. 2. З неї видно, як відбувається обмін інформацією між основними компонентами програмного забезпечення. Кількість зв'язків між цими компонентами зведена до мінімуму, що свідчить про ефективну декомпозицію.

3.3. ФУНКЦІОНАЛЬНІСТЬ ОТРИМАНОЇ СИСТЕМИ

Для відлагодження роботи алгоритму система дає змогу покроково тестувати і допомагає відстежити роботу алгоритму після кожної ітерації, і коректувати її в разі потреби.

Для навчання алгоритму система пропонує генерування випадкових карт з наперед заданими параметрами і послідовний запуск алгоритму на цих картах. Також є можливість навчання на наборі наперед заданих карт. За такої конфігурації карта

буде вибиратися випадково з заданого списку. Зокрема, такий підхід використовують в алгоритмах навчання з підсиленням [5].

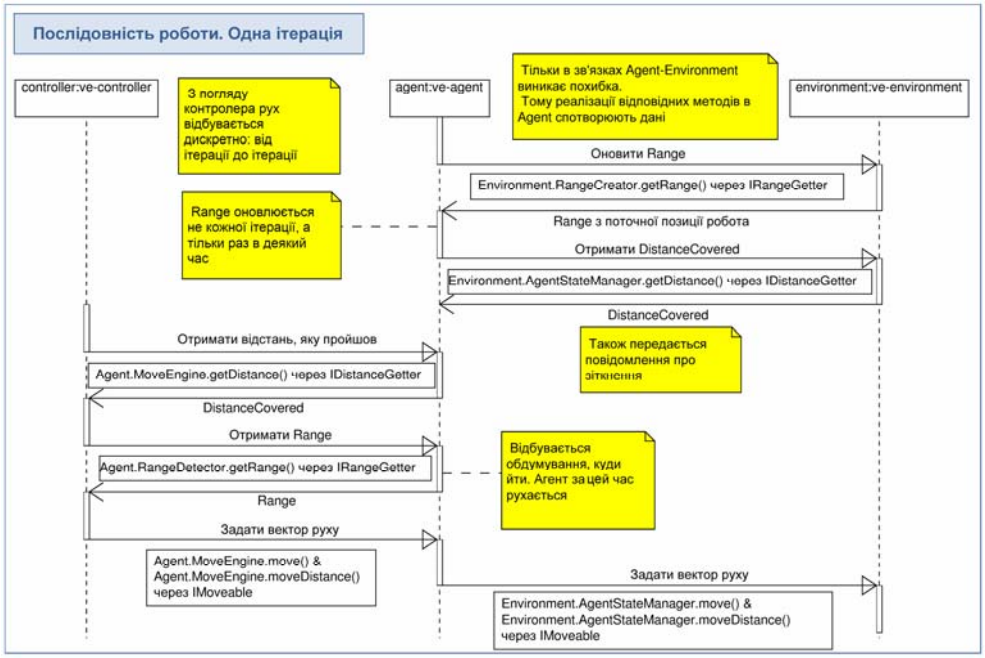

Рис. 2. Sequence-діаграма роботи віртуального середовища

Для проведення порівняльного аналізу ефективності роботи використовують підхід, схожий до процесу навчання: два алгоритми послідовно виконуються на випадкових картах, що допомагає проаналізувати їхні недоліки та переваги.

Впродовж моделювання створюються умови максимально наближені до реальних. Це виконують за допомогою широкого спектра налаштувань, доступних користувачеві. Зокрема, сенсори агента можуть, як і в реальних умовах, надавати неточну інформацію про відстань до навколишніх об’єктів і про пройдену агентом відстань. Також можливе створення різноманітних середовищ: замкненого та відкритого типів. Це дає змогу якнайкраще підготувати алгоритм до роботи в реальних умовах.

Чим ліпше умови наближаються до реальних, тим складнішою стає розробка відповідних алгоритмів. Тому завжди треба шукати компроміс між точністю моделі і зручністю розроблення алгоритмів у середовищі моделювання. Таким компромісом стало використання покрокового характеру тестування, що дає можливість найзручніше відлагодити алгоритм.

Проект реалізований за допомогою мови програмування Java. Вибір саме цієї мови програмування зумовлений кількома чинниками. Одним з них є вимога використання концепцій відкритого програмного забезпечення. Іншою важливою перевагою є незалежність від платформи. Також Java повністю об’єктно-орієнтована мова, а отже, для використання об’єктно-орієнтованого проектування є ідеальним варіантом [6]. Окрім Java, для розроблення алгоритму можна використовувати й інші мови програмування [11, 12, 13], які підтримуються JVM.

3.3.1. *АПЛІКАЦІЯ ENVIRONMENT FACTORY*

Для проведення наперед спланованих випробувань, або навчання, ретельного тестування і відлагодження алгоритму, а також для отримання порівняльних характеристик кількох алгоритмів часто потрібне створення віртуальної карти з наперед заданими властивостями. Аплікація Environment Factory дає змогу користувачеві розробити віртуальну карту самому, або згенерувати її автоматично, задавши характеристики, якими вона повинна володіти і у разі потреби відредагувати її за допомогою розробленого редактора карт.

У головному вікні аплікації (рис. 3) відображається загальний вигляд карти, у цьому разі надається можливість її генерування та редагування. Вона складається з панелі меню (6) та робочої панелі, на якій відображається поточний стан віртуальної карти. На ній є такі елементи:

1) початок координат; місце, де буде починати свій шлях агент;
2) ціль, яку повинен досягнути агент; досягнувши будь-якої точки цілі завдання задачі навігації вважається виконаним;
3) активна перешкода, тобто така, яку можна переміщати за допомогою мишки і копіювати/об'єднувати з іншими перешкодами;
4) неактивні перешкоди, тобто такі, які стаціонарно розташовані на карті;
5) необов'язкова границя карти; вона також є перешкодою, проте для зручності в програмі передбачено окремий спосіб її створення.

Отже, процес створення перешкоди вручну складається з таких кроків: створення многокутника, багаторазове його копіювання/об'єднання; створення наступного многокутника, його копіювання / об'єднання тощо. Повністю карту можна отримати послідовним створенням усіх перешкод.

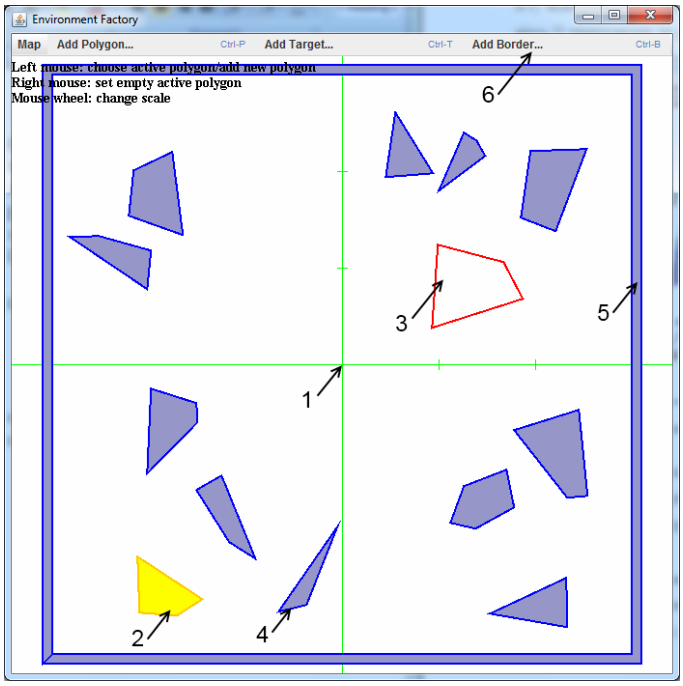

Рис. 3. Аплікація Environment Factory. Головне вікно

Для створення нового многокутника, який моделює перешкоду на шляху агента, треба вибрати команду меню “Add Polygon...” (або натиснути комбінацію клавіш Ctrl+P). Відкриється вікно, зображене на рис. 4.

Інтерфейс цього вікна інтуїтивно зрозумілий – за допомогою маніпулятора мишки додають нові точки, доки не буде досягнуто бажаного результату. Можна також згенерувати випадкову множину точок (за допомогою кнопки “Generate Random”). Результуючою перешкодою буде многокутник, що задається випуклою оболонкою заданої множини точок.

Для задання границі карти потрібно вибрати команду меню “Add Border...” (або натиснути комбінацію клавіш Ctrl+B). Границя карти – це перешкода прямокутної форми. З нею теж можна працювати, як і з іншими перешкодами (а саме, копіювати, об’єднувати з іншими перешкодами тощо). Це дає змогу створювати кімнати довільної форми, не обмежуючись прямокутними. Зокрема, можливим є використання карти, що не має границі, це забезпечує можливість моделювання поведінки агента не тільки у замкнених приміщеннях, а й у середовищах відкритого типу.

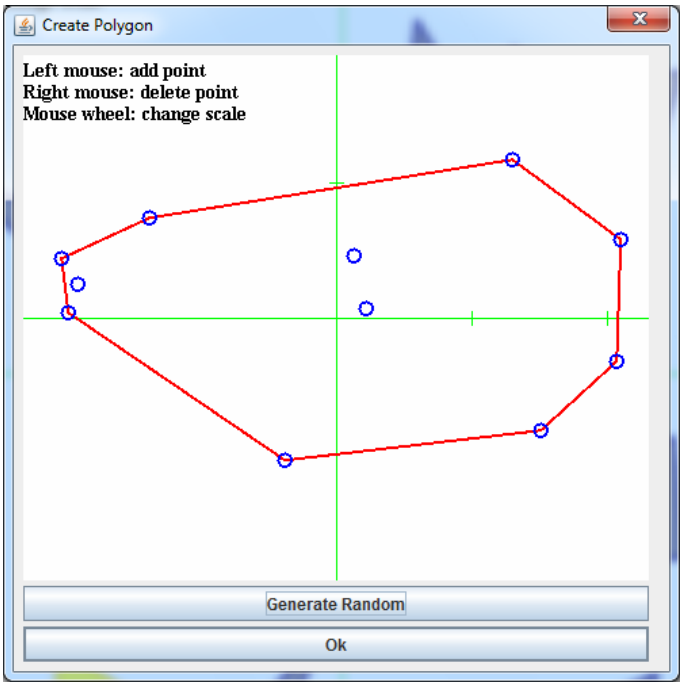

Рис. 4. Аплікація Environment Factory. Вікно створення перешкоди

Альтернативою до ручного створення віртуальної карти є генерування псевдовипадкової карти з наперед заданими параметрами. Для цього треба вибрати команду меню Map → Create Random Map... (або натиснути комбінацію клавіш Ctrl+R). Після цього відкриється вікно зображене на рис. 5. В цьому вікні можна задати характеристики карти, яка буде створена, зокрема кількість і розмір перешкод на карті; наявність в карти границі; розмір карти; складність створених перешкод тощо. Також можна виконати випадковий вибір з наперед заданого списку карт, що є корисною можливістю при розробці алгоритму, який ґрунтується на підході навчання

з підсиленням [14]. Звісно, після генерації карти її можна редагувати вручну, методами описаними раніше.

Аналогічне до зображеного на рис. 5 вікно використовують не тільки під час створення карти за допомогою аплікації Environment Factory, а й під час налаштування процесів навчання і порівняння ефективності алгоритмів в аплікації Virtual System Manager. Це стало можливим завдяки використанню об'єктно-орієнтованого підходу, що ще раз підтверджує його доцільність у створенні програмного забезпечення такого типу.

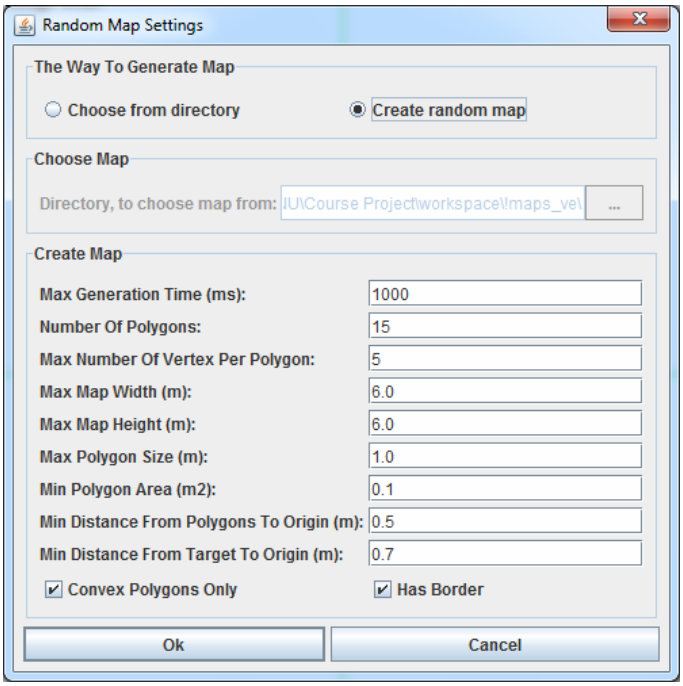

Рис. 5. Аплікація Environment Factory. Вікно створення випадкової карти

3.3.2. *АПЛІКАЦІЯ VIRTUAL SYSTEM MANAGER*

За допомогою цієї аплікації користувач може виконати тестування та відлагодження алгоритму, навчання алгоритму, а також провести порівняльний аналіз ефективності алгоритмів. Розглянемо детальніше інтерфейс програми.

На рис. 6, *а* зображено процес керування тестуванням і відлагодженням алгоритму. У полях (1) задається інформація про розміщення бібліотеки, що містить алгоритм, а також розташування файлів налаштувань агента та середовища.

За допомогою панелі керування (2) відбувається ініціалізація нового тестового запуску, а також послідовно відбуваються ітерації алгоритму. У цьому разі можливе коригування руху агента вручну за допомогою елементів керування (4).

На панелі (3) відображається процес руху агента та сприймання ним середовища. Карта, побудована алгоритмом (вона ґрунтується на даних, які отримали з сенсорів агента), передається градаціями сірого: чим вища ймовірність знаходження в цій частині навколишнього середовища перешкоди, тим темніший колір. Зокрема, алгоритм вважає світлу область (5) вільною від перешкод, а темну (6) – зайнятою перешкодою.

На рис. 6, *б* зображено вікно, яке дає користувачеві додаткову інформацію про хід тестового запуску. Зокрема, точні координати агента, поточний час, інформацію про внутрішній стан алгоритму тощо.

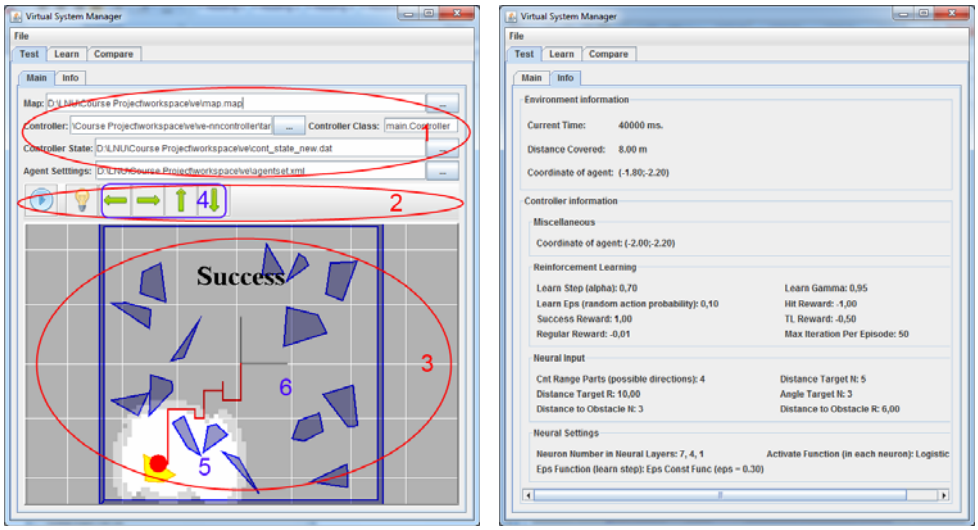

Рис. 6. Аплікація Virtual System Manager. Тестування роботи алгоритму:
а – керування; *б* – інформація

Процес навчання зображено на рис. 7. Аналогічно, як і під час тестування, поля (1) задають розміщення бібліотеки, що містить алгоритм, а також розташування файлів налаштувань агента. За допомогою панелі керування (2) виконується ініціалізація нового навчального процесу, а також його зупинка/запуск.

Процес навчання полягає в послідовному виконанні алгоритму у випадково згенерованих середовищах, параметри яких задають за допомогою кнопки (3). Також є можливість навчання на наборі наперед заданих карт. У такому випадку карта буде вибиратись випадково з заданого списку. Це відбувається за допомогою вікна, аналогічного до зображеного на рис. 5.

Додаткову інформацію про процес навчання можна переглянути, натиснувши кнопку (4). Зберегти алгоритм після навчання можна за допомогою кнопки (5). Це дасть змогу використовувати отримані результати для тестування, подальшого навчання та порівняння роботи алгоритму з іншими.

Панелі (6) і (7) забезпечують можливість інтерактивного спостереження за процесом навчання. На панелі (6) показується загальна успішність алгоритму під час навчання, а на панелі (7) поточна (за останні 100 виконань).

Панелі (8) дають змогу переглянути роботу алгоритму впродовж навчання в кількох останніх згенерованих середовищах. Це уможливлює спостереження за тим, як змінюється поведінка алгоритму впродовж навчання, а також вчасного внесення коректив у процес навчання (наприклад, змінити характер середовищ, в яких тестується алгоритм).

Процес порівняння алгоритмів зображено на рис. 8. Поля (1), (2) задають розміщення бібліотек, що містять алгоритми, які будуть порівнюватись. Поле (3) задає розташування файлів налаштувань агента. За допомогою панелі керування (4) відбувається ініціалізація нового порівняння, а також його зупинка/запуск. Процес порівняння полягає у послідовному запуску алгоритмів у випадково згенерованих

середовищах, параметри яких задаються за допомогою кнопки (5). Це відбувається за допомогою вікна, аналогічного до зображеного на рис. 5. Додаткову інформацію про процес порівняння можна переглянути, натиснувши кнопки (6) і (7).

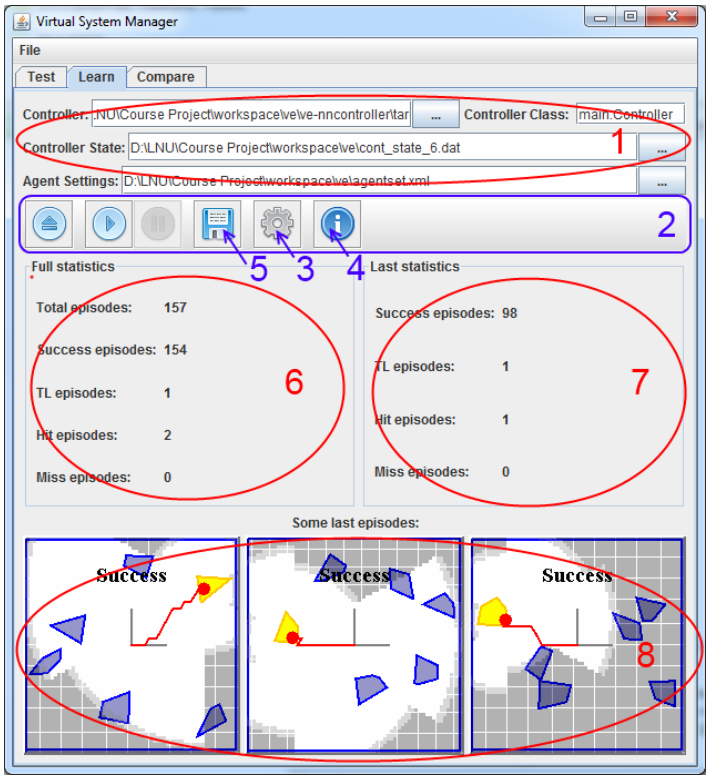

Рис. 7. Аплікація Virtual System Manager. Процес навчання алгоритму

Панелі (8) і (9) дають змогу інтерактивно спостерігати за процесом порівняння. На панелі (8) відображаємо загальну успішність першого алгоритму впродовж порівняння, а на панелі (9) загальну успішність другого алгоритму впродовж порівняння.

Панелі (10) дають змогу переглянути роботу алгоритмів впродовж роботи в одному з останніх середовищ. Це уможливлює аналіз того, як відрізняється поведінка алгоритмів, а отже, ліпше дослідити їхні переваги та недоліки.

3.4. РЕЗУЛЬТАТИ ЗАСТОСУВАННЯ

За допомогою системи моделювання віртуального середовища було відлагоджено і протестовано алгоритм побудови карти навколишнього середовища SLAM (Simultaneous Localization and Mapping) [15].

У процесі створення робочої версії алгоритму побудови уточненої карти SLAM систему моделювання навколишнього середовища застосовували як зручне середовище. З допомогою цієї системи легко можна було спостерігати за роботою алгоритму, відлагоджувати основні кроки роботи алгоритму, проводити тестові запуски для відшукання помилок у реалізації.

Це дало змогу за короткий час отримати робочу версію алгоритму. Надалі проводили налаштування параметрів алгоритму для поліпшення роботи в умовах

отримання даних із сенсорів з високою похибкою. З цією метою теж використовували систему моделювання віртуального середовища. Її використання суттєво спростило отримання даних про роботу алгоритму з конкретними налаштуваннями і дало змогу вибрати ті, які найліпше себе зарекомендували.

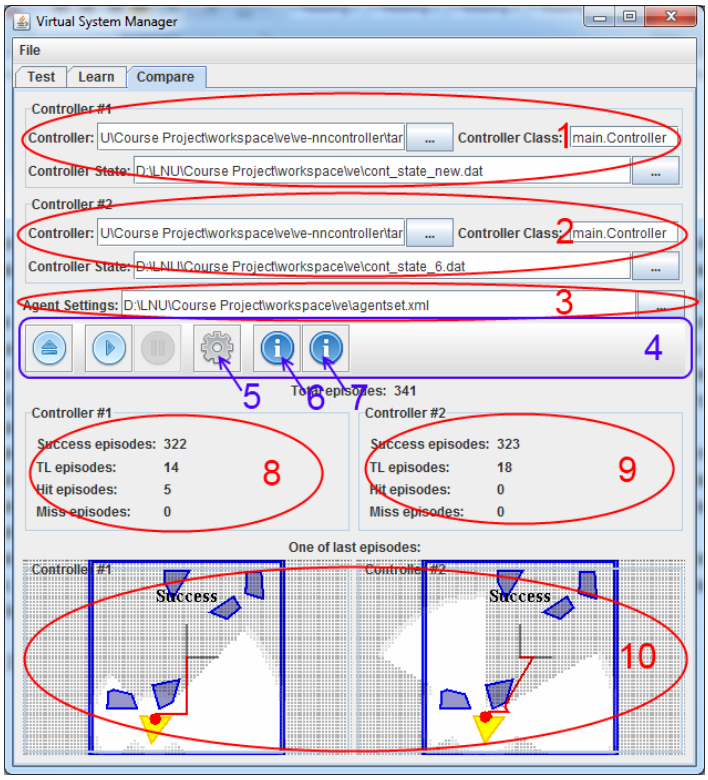

Рис. 8. Аплікація Virtual System Manager. Процес порівняння алгоритмів

Також система є повноцінною платформою створення інших алгоритмів навігації та позиціонування, оскільки корисна на кожному етапі створення алгоритму: від відлагодження, до виконання підбору параметрів і аналізу ефективності алгоритму.

4. ВИСНОВКИ

Розв’язано актуальну науково-прикладну задачу створення інформаційної системи моделювання поведінки інтелектуальних агентів у задачі навігації. Для цього створили гнучке програмне забезпечення, яке дає змогу відлагоджувати, тестувати, навчати та порівнювати ефективність алгоритмів. За допомогою цієї системи було відлагоджено і протестовано алгоритм побудови карти навколишнього середовища SLAM, що на практиці підтверджує його дієздатність.

Отримано такі результати.

1. Сформульовано основні вимоги до систем такого типу.
2. Проаналізовано існуючі рішення, розглянуто їхні переваги та недоліки.
3. Створено систему для розробки, відлагодження, тестування та порівняння алгоритмів розв’язання задачі навігації, які працюють із мобільним агентом.

4. Ефективність системи було підтверджено шляхом відлагодження з її допомогою алгоритму побудови карти навколишнього середовища SLAM. Проведено експерименти з поліпшення налаштувань алгоритму, що допомогло отримати високоточний алгоритм побудови карти навколишнього світу в умовах отримання даних з високою похибкою.

Наступним кроком у вдосконаленні розробленої інформаційної системи моделювання поведінки агентів є забезпечення можливості збору статистичної інформації про роботу алгоритму, яка б забезпечила можливість автоматично визначати його слабкі та сильні моменти. Також пріоритетним є моделювання ширшого класу агентів. Планується збільшити спектр алгоритмів, які б могли використовувати цю систему. Розроблена інформаційна система ґрунтується на гнучкому каркасі незалежних компонент, які уможливлюють легкість змін і масштабованості.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. *Russel S.* Artificial Intelligence: A Modern Approach. Pearson / S. Russel, P. Norvig 2010. – 1152 p.
2. *Pilone D.* UML 2.0 in a Nutshell. O'Reily / D. Pilone, N. Pitman. – 2005. – 234 p.
3. *Avinash C.Kak.* Programming with Objects: A Comparative Presentation of Object-Oriented Programming with C++ and Java / C.Kak. Avinash. – John Wiley & Sons, 2003. – 1144 p.
4. *Томашевський В.М.* Моделювання систем / В.М. Томашевський. – Видавнича група BHV, 2005. – 352 с.
5. *Rummery G.A.* Problem Solving with Reinforcement Learning / G.A. Rummery. – Ph.D. thesis. Cambridge University Engineering Department, 1995.
6. *Шилдт Г.* Полный справочник по Java / Г. Шилдт. – Вильямс, 2007. – 1040 с.
7. *Morgan S.* Programming Microsoft Robotics Studio / S. Morgan. – Microsoft Press, 2008. – 288 p.
8. *Tippenhauer N.* Introduction to Player / Stage / Gazebo / N. Tippenhauer. – http://www.pami.uwaterloo.ca/groups/asrtdd/psg.pdf
9. *Owen J.* How to use Player / Stage / J. Owen. – http://www-users.cs.york.ac.uk/jowen/player/playerstage-tutorial-manual-2.1.pdf
10. *Drumwright E.* Moby / E. Drumwright. – http://physsim.sourceforge.net
11. *Pollak D.* Beginning Scala. Apress / D. Pollak. – 2009. – 320 p.
12. *Nutter C.O.* Using JRuby: Bringing Ruby to Java / C.O. Nutter, T. Enebo, N. Sieger, O. Bini, I. Dees. – Pragmatic Bookshelf, 2011. –300 p.
13. *Robert B.* Jython for Java Programmers. Sams / B. Robert. 2001. – 496 p.
14. *Sutton R.S.* Reinforcement Learning: An Introduction. A Bradford Book / R.S. Sutton, A.G. Barto. – 1998. – 551 p.
15. *Ozkucur N.E.* Multi-Agent Visual-SLAM Algorithms on Autonomous Robots / N.E. Ozkucur. – LAP LAMBERT Academic Publishing, 2010. – 120 p.

Стаття: надійшла до редколегії 14.04.2011
доопрацьована 12.05.2011
прийнята до друку 26.05.2011

ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА МОДЕЛИРОВАНИЯ ПОВЕДЕНИЯ АГЕНТОВ В ЗАДАЧИ НАВИГАЦИИ

О. Годыч, П. Кушнир, Ю. Щербина

Львовский национальный университет имени Ивана Франко,
ул. Университетская, 1, Львов, 79000, e-mail: dais@franko.lviv.ua

Рассмотрено задачу построения программного обеспечения для моделирования поведения интеллектуальных агентов в виртуальной среде. Проведено анализ существующих решений, очерчиваются их преимущества и недостатки. Сформулировано основные требования к системам такого типа. Рассмотрено программное обеспечение, качественно реализирующее поставленные требования. Приведено описание полученной системы, а также обосновано её эффективность и конкурентоспособность.

Ключевые слова: программное обеспечение, моделирование, виртуальная среда, интеллектуальный агент, задача навигации

INTELLIGENT SYSTEM FOR MODELING THE BEHAVIOR OF AGENTS IN THE PROBLEM OF NAVIGATION

O. Hodych, P. Kushnir, Y. Shcherbyna

Ivan Franko National University of Lviv,
Universytetska str, 1, Lviv, 79000, e-mail: dais@franko.lviv.ua

This article discusses the problem of creating the virtual environment for modeling behavior of intelligent agents. Some existing solutions are analyzed, and description of their pros and cons is given. The principle requirements for such systems are formulated. The article considers creation of the software, which comply with the specified requirements. The developed system is presented together with the analysis of its efficiency and competitiveness.

Key words: software, modeling, virtual environment, intelligent agent, navigation problem.