ФОРМАЛІЗАЦІЯ ПРОЕКТУВАННЯ ІГРОВОЇ МЕХАНІКИ КАЗУАЛЬНИХ КОМП'ЮТЕРНИХ ІГОР

канд. техн. наук О. А. Блажко, В. В. Антонюк, Ю. Л. Трояновська

Одеський національний політехнічний університет, Україна

Розглянуто процес розробки казуальних комп'ютерних ігор, для яких формалізація такого процесу на основі парадигми автоматного програмування зменшує складність розробки. Запропоновано підхід до розробки казуальних ігор та наведено приклад у візуальному конструкторі ігор Unity3D PlayMaker.

Вступ: Вже понад 30 років існує парадигма автоматного програмування або «програмування від станів» для розробки програмного забезпечення (ПЗ) на основі кінцевих автоматів. Перевагами графічного представлення поведінки ПЗ у вигляді моделі кінцевого автомату перед звичайним алгоритмічним програмуванням є спрощення аналізу створеного програмного коду та автоматизація тестування. Основним недоліком парадигми є складність створення програмного коду мовами програмування на основі *switch*-операторів, але вона зменшується за рахунок використання спеціалізованих компіляторів, наприклад, технології *SMC* [1].

За останні 10 років підвищення інтересу до гейміфікації бізнес-процесів почало виводити розробку комп'ютерних ігор з вузькоспеціалізованого процесу розробки ПЗ до індустрії розробки ПЗ на основі інструментальних середовищ (ігрових рушіїв) з можливостями візуального автоматного програмування. Прикладами є компоненти *PlayMaker* в *Unity 3D* та *Blueprint* в *Unreal Engine*. Особливо актуальним це є для класу казуальних ігор, які характеризуються простим геймплеєм (гра-головоломка, настільна гра), який може бути повністю керований за допомогою «миші» або сенсору мобільного пристрою [2].

Але в більшості посібників зі створення ігор відсутні рекомендації з використання формалізації автоматного програмування, хоча подібні методики є в роботі [3], яка пропонує процес формалізованого перетворення сценарію гри у програмний код, та в роботі [4], яка пропонує зменшити складність трансформації довільних текстів літературних сценаріїв ігор в опис алгоритмів на основі спеціальної формалізованої мови сценаріїв, але з реалізацією для алгоритмічних мов програмування. **Тому метою роботи** є, враховуючі особливості візуального автоматного програмування сучасних рушіїв, запропонувати формалізований підхід до проектування казуальних ігор за допомогою *UML*-моделювання, яке широко використовується в об'єктно-орієнтованому аналізі та дизайні ПЗ.

Формалізація проектування ігрової механіки казуальних комп'ютерних ігор: При проектуванні ігрової механіки вхідними даними є, зазвичай, сценарій гри, поданий як довільний текстовий опис. В результаті проектування ми повинні отримати кінцевий автомат, який можна використовувати як шаблон при візуальному програмуванні гри в ігрових рушіях. Пропонується розбити процес проектування механіки гри на наступні етапи:

1) формалізація ігрового сценарію;
2) проведення аналізу сценарію використання гри;
3) визначення структури взаємодії ігрових об'єктів;
4) визначення станів ігрових об'єктів.

На етапі формалізації ігрового сценарію виконується представлення вхідного текстового сценарію гри у вигляді сценарію використання, тобто як маркованого списку, який покроково описує геймплей.

На етапі проведення аналізу сценарію використання гри виділяються ігрові об'єкти. Зазвичай в сценарії вони виступають як іменники, які мають характеристики (прикметники та іменники) та (над ними) виконують дії (дієслова). Ігрові об'єкти треба розділити на динамічні (від яких залежить геймплей) та статичні.

На етапі визначення структури взаємодії ігрових об'єктів будується діаграма класів, на якій показані класи динамічних ігрових об'єктів та статичних, з якими вони взаємодіють.

Атрибутами класів виступають характеристики ігрових об'єктів, операціями – дії, які призводять до зміни значень їх атрибутів з дозволеної множини. Класи задають шаблон ігрового об'єкта.

На етапі визначення станів ігрових об'єктів на основі розробленої діаграми класів та формалізованого ігрового сценарію визначаються стани об'єктів ігрових класів як деякі значення їх атрибутів з дозволеної множини, тригери, які призводять до зміни станів (дія користувача чи іншого ігрового об'єкта) та виконанню необхідних операції. Як результат будується кінцевий автомат у форматі діаграми станів *UML*.

Як приклад розглянемо проектування різновиду перемикальної гри Шеннона [5].

Припустимо, що наш вхідний сценарій описаний наступним чином: «Ігрове поле складається з води та островків зеленого та жовтого коліру, між якими встановлені містки. При старті гри всі містки кольору ігрового поля (води). Коли мишка над містком, він змінює колір на салатовий. Коли мишка йде, повертається колір води. Якщо кликнути на місток, він стає зеленого кольору».

Формалізуємо даний сценарій таким чином:

- за замовчуванням місток кольору ігрового поля (води);
- коли мишка над містком, він змінює колір на салатовий - міст підсвічений;
- коли мишка йде, повертається колір за замовчуванням;
- якщо кликнути на місток, він стає зеленого кольору - міст встановлено.

Виділімо ігрові об'єкти. Це вода, острівки та містки. Динамічними з них є лише містки - вони можуть змінювати свій колір, взаємодіючи з гравцем.

Визначимо структуру взаємодії ігрових об'єктів. Вочевидь, що всі містки можна визначити за допомогою єдиного класу, атрибутом якого буде колір (*Color*), а операціями – дії по його зміні, а саме встановленню в початковий стан (*SetDefaultColor*), підсвічування (*Highlight*) та встановленню в зелений колір (*SetColor*). На рисунку 1 показана отримана діаграма класів.

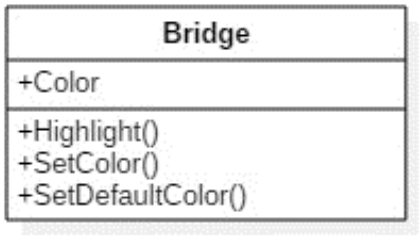
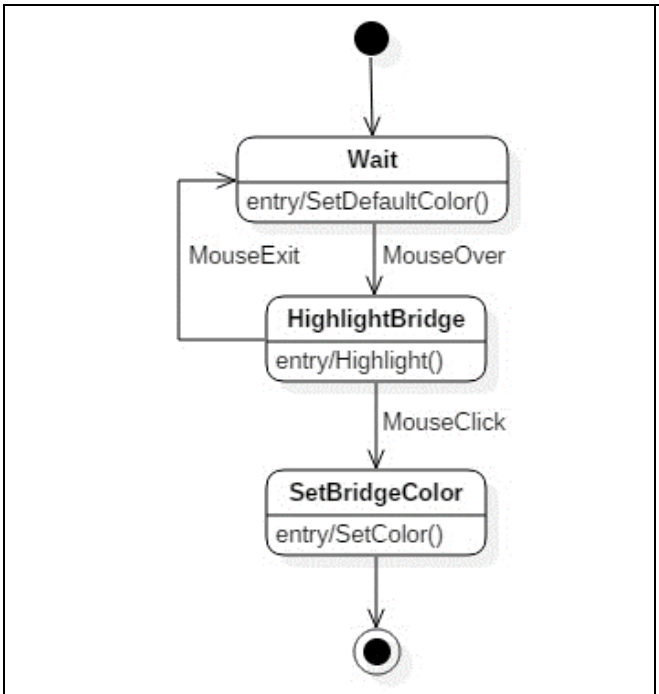


Рис. 1 – Ігровий клас місток (Bridge).

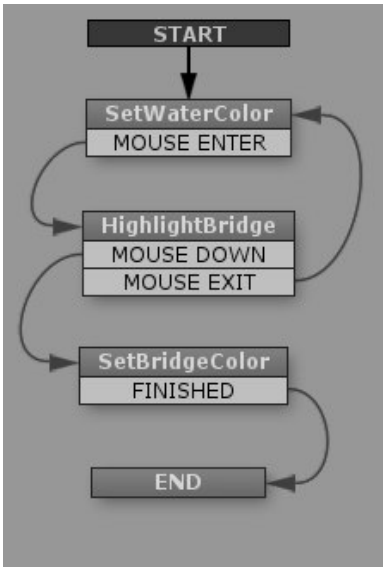
Визначимо стани ігрового об'єкта класу *Bridge*. При старті гри місток повинен бути у початковому стані, тобто його колір такий як у води, та він очікує дії гравця. Назвемо це станом очікування (*Wait*). При вході в цей стан колір містка повинен встановлюватися у колір води, тобто повинна виконуватись операція *SetDefaultColor*. При наведені мишки місток підсвічується. Назвемо цей стан *Highlight*. При вході в цей стан колір містка повинен встановлюватися у салатовий, тобто повинна виконуватись операція *Highlight*. Далі, якщо мишка йде, то виконується перехід у стан очікування *Wait*. Якщо ж натискається кнопка мишки, то відбувається перехід у стан встановлення кольору. Назвемо його *SetColor*. При вході в цей стан колір містка повинен встановлюватися у зелений, тобто повинна виконуватись операція *SetColor*. Після цього ігровий об'єкт стає статичним.

На рисунку 2 наведені приклади візуалізації автоматної моделі станів.

На рисунку 2.а показана *UML*-діаграма станів ігрового об'єкта класу *Bridge*, а на рисунку 2.б її еквівалент у формі автомата *(Finite State Machine)*, побудованого за допомогою плагіну *PlayMaker* для *Unity3D*.



(а) діаграма станів UML-стандарту



(б) FSM Unity3D PlayMaker

Рис. 2 – Приклад діаграми станів UML-стандарту (а) та його еквіваленту у Unity3D PlayMaker (б).

Висновки. Запропонована формалізація проектування ігрової механіки за допомогою *UML*-моделювання дозволяє уніфікувати процес розробки казуальних ігор на основі кінцевих автоматів, що зменшує час, та дозволяє використовувати єдиний підхід в незалежності від інструментального середовища, за допомогою якого буде реалізована гра. В той же час залишається декілька відкритих питань, а саме створення методу побудови комплексної *UML*-діаграми станів, яка складається з автоматів ігрових об'єктів та описує загалом весь геймплей гри, та можливості автоматизації запропонованого підходу. Вони потребують подальших досліджень.

Результати цієї роботи буде використано в проекті ЄС Erasmus+KA2 "GameHub: університетсько-підприємницька співпраця в ігровій індустрії в Україні".

ВИКОРИСТАНІ ДЖЕРЕЛА

1. The State Machine Compiler – Access mode: http://smc.sourceforge.net/ (last access: 01.03.17). – Title from the screen.

2. Casual game – Access mode: https://en.wikipedia.org/wiki/Casual_game (last access: 01.03.17). – Title from the screen.

3. Мазин М. А. Парфенов В. Г., Шалыто А. А. Разработка интерактивных приложений Macromedia Flash на базе автоматной технологии // Проектная документация. СПбГУ ИТМО. – Режим доступа: http://is.ifmo.ru/projects/flash/ (дата обращения: 01.03.17). – Название с экрана.

4. Кожаев В. В. Иструментарий создания игровой логики / В.В. Кожаев // Проблеми програмування. – 2012. – № 4. – С. 64 – 74. – Режим доступу: http://dspace.nbuv.gov.ua/bitstream/handle/123456789/86640/06-Kozhaev.pdf (дата звернення: 01.03.17). – Назва з екрана.

5. Shannon switching game – Access mode: https://en.wikipedia.org/wiki/Shannon_switching_game (last access: 01.03.17). – Title from the screen.

Blazhko O.A., Antoniuk V.V., Troianovska Y.L.

Computer Game Development based on Visual Automata-based Programming

There is considered casual game development by using of this process formalization with Finite State Machines (FSM) or automata-based programming to reduces complexity of development. The improved FSM-method of game development is proposed. The example of FSM transformation in visual game constructor Unity3D PlayMaker is given. The results will be used in the project EC Erasmus+KA2 – GameHub.