УДК 004
DOI https://doi.org/10.32838/2663-5941/2022.1/15

Завгородній В.В.
Державний університет інфраструктури та технологій

Завгородня Г.А.
Державний університет інфраструктури та технологій

Валявська Н.О.
Державний університет інфраструктури та технологій

Адаменко В.С.
Державний університет інфраструктури та технологій

Дороговцев Є.В.
Державний університет інфраструктури та технологій

Несмачний П.В.
Державний університет інфраструктури та технологій

МЕТОД АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ КОНТЕНТУ НА ОСНОВІ ПРОЦЕДУРНИХ АЛГОРИТМІВ

Робота присвячена розробці методу генерації ігрового контенту, який заснований на використанні ланцюгів Маркова. Система задається простором станів, що описують усі можливі її конфігурації. У такому випадку переходи між станами описуються як дискретні кроки, які характеризуються їх імовірностями.

Розглянуто алгоритм Монте-Карло для ланцюгів Маркова як методику отримання вибірки з простору станів. Вибірка означає вибір стану виходячи з імовірності вибору з урахуванням внутрішнього розподілу. Імовірність знаходження в стані пропорційна певній функції витрат, яка дає оцінку поточному стану, у якому знаходиться система. Показано, що навіть при надзвичайно великому просторі станів незалежно від складності системи цей алгоритм знаходитиме рішення з низькими витратами, якщо дати йому достатньо часу для сходження.

Також розглянуто розподіл Гіббса як розподіл з найменшою чутливістю до варіацій в обмеженнях. Для обчислення ймовірності переходу між станами в розподілі Гіббса використана функція витрат, яка є єдиним обмеженням, що накладається на розподіл станів.

Розглянуто алгоритм для процедурної генерації рівнів, який можна розбити на етапи: генерування та розміщення кімнат (створюється ігрове поле й розміщуються на ньому кілька випадкових кімнат, що не повинні перетинатися), з'єднання кімнат (усі кімнати треба зв'язати між собою коридорами) і створення стін (метод пройдеться по всій мапі та додасть стіни скрізь, де «порожні» клітини межують із клітинами кімнат і коридорів).

Застосування розглянутих алгоритмів дає змогу визначати метод для генерації будь-якого можливого результату, усвідомлено обирати оптимальний на підставі його оцінки.

Ключові слова: *метод генерації контенту, марковський процес, процедурні алгоритми, метод Монте-Карло, розподіл Гіббса, процедурна генерація рівнів.*

Постановка проблеми. Ігрова індустрія з'явилася порівняно недавно, проте вже встигла стати величезною галуззю з колосальним прибутком у десятки мільярдів доларів США на рік [1–3]. Зрозуміти таку раптову популярність, що зростає, віртуальних ігор дуже легко: усе це пов'язано з широким розповсюдженням комп'ютерної техніки, включаючи появу Інтернету. Комп'ютерні ігри, на відміну від інших розваг, є доступнішими для кінцевих користувачів [4; 5].

Зараз, щоб пограти, гравець повинен мати лише комп'ютер або ігрову приставку, копію самої гри з широким доступом до Інтернету, а щоб отримати копію гри, не потрібно виходити з дому. Крім того, для вибору відповідної гри споживачеві не потрібно мати особливу освіту, а більшість інших

розваг вимагають як мінімум розуміння необхідного обладнання [6; 7].

Останнім часом комп'ютерні ігри перестали розглядатися як програми, призначені виключно для відпочинку чи розваг. Завдяки ігровим технологіям створено спеціальні симуляційні комплекси, призначені для підготовки фахівців із різних сфер діяльності: управлінської, організаційної, транспортної, економічної тощо [8–10]. Таким чином, розвиток технології може бути одним із найперспективніших напрямів.

Аналіз останніх досліджень і публікацій. Терміном «комп'ютерна гра» класифікується комп'ютерна програма, яка працює для організації ігрового процесу (геймплея), зв'язку з партнером по цій грі або сама виступає як партнер. Комп'ютерні ігри часто формуються на основі сторонніх джерел, таких як кінострічки або книги, але останнім часом стали помічати зворотні випадки, коли по знайомій ігровій серії починають видавати допоміжні матеріали, що розширюють усесвіт гри [11–12].

Більш того, спеціально створені ігри можуть виступати як навчальний матеріал або дають змогу гравцям застосувати їх у науково-дослідних цілях. Ці ігри рідко випускаються в широкі маси. За деякими іграми ведуться аматорські та професійні змагання, які називаються кіберспортом.

Комп'ютерні ігри здійснили настільки значний вплив на інформаційні технології, що останнім часом виникла стійка тенденція до гейміфікації неігрового прикладного програмного забезпечення. Таким чином, у деяких європейських школах почали використовувати відому гру Minecraft для навчання, а для армій стали робити особливі симулятори для занять боєм. Комп'ютерні ігри з 2011 року офіційно визнані урядом Сполучених Штатів Америки й американським Державним фондом окремим видом мистецтва, в одному ряду з театром і кінотеатром, а в Україні кіберспорт офіційно зарахований до видів спорту.

Постановка завдання. Мета дослідження – розглянути методи генерації контенту з використанням ланцюгів Маркова, а саме метод Монте-Карло, розподіл Гіббса та процедурну генерації рівнів.

Виклад основного матеріалу дослідження. Метод генерації ігрового світу складається з кількох кроків. На початку генерується сама мапа, яка складається із сукупності тайлів різного типу ландшафту. Потім на локації розміщуються різноманітні ігрові об'єкти, наприклад, пастки, схованки, скрині, нейтральні чи дружні неігрові персонажі.

Насамперед необхідно згенерувати локації заздалегідь підготовленого розміру, які складаються з ділянок землі певного типу зі списку можливих типів ландшафту. Будь-який тип такого ландшафту має можливість появи й із іншими типами, включаючи самого себе. Генерація ландшафту в такому разі ґрунтується на генерації випадкових чисел.

Ланцюг Маркова – це випадковий процес, що задовольняє властивості Маркова та приймає кінцеве значення станів. Тобто це черговість випадкових заходів із кінцевою кількістю підсумків, що характеризується властивістю незалежності майбутніх станів від минулого. На майбутній стан процесу впливає лише поточний стан, а не черговість дій у минулому [13].

Ланцюги Маркова мають велику кількість застосувань як статистичні моделі процесів у реальному світі, таких як дослідження систем круїзного контролю в автотранспортних засобах, черг або ліній клієнтів, що прибувають в аеропорт, курсів обміну грошових одиниць і динаміка популяції тварин.

Марковські процеси вважаються основою для способів стохастичного моделювання, відомих як ланцюг Маркова Монте-Карло, які застосовуються для моделювання вибірки зі складних розподілів імовірностей, і знайшли використання в байєсівській статистиці та штучному інтелекті. Існують два види ланцюгів Маркова – ланцюги Маркова дискретного часу й ланцюги Маркова нескінченного часу [14; 15].

Ланцюги Маркова – це послідовність станів, якою рухається система, що описується переходами в часі. Переходи між станами стохастичні, тобто описуються ймовірностями, що є характеристикою системи. Система задається простором станів, що є простором для всіх можливих конфігурацій системи. Якщо система описана правильно, ми також можемо описати переходи між станами як дискретні кроки. Треба врахувати, що зі стану системи часто буває кілька можливих дискретних переходів, кожен із яких веде до різного стану системи. Імовірність переходу зі стану *x* до стану *y* дорівнює *Pxy*.

Марковський процес – це процес дослідження цього простору станів за допомогою ймовірностей, що передаються йому. Важливим є те, що марковські процеси «не мають пам'яті». Це означає, що ймовірності переходу з поточного стану до нового залежать тільки від поточного стану й не залежать від інших станів, відвіданих раніше, тобто $Pxy = P(x,y)$.

Метод Монте-Карло для ланцюгів Маркова. Оскільки переходи між станами визначаються ймовірностями, то можна задати ймовірність «стійкого» перебування в стані (чи, якщо час прагне до нескінченності, середній час, що ми перебуватимемо в конкретному стані). Це внутрішній розподіл станів.

Тоді алгоритм Монте-Карло для ланцюгів Маркова (*Markov-Chain Monte-Carlo, MCMC*) – це методика отримання вибірки з простору станів. Вибірка означає вибір стану виходячи з ймовірності вибору з урахуванням внутрішнього розподілу. Імовірність перебування в стані пропорційна певній функції витрат, яка дає «оцінку» поточному стану, у якому знаходиться система. Уважається, що якщо витрати низькі, то ймовірність перебування в цьому стані висока, і це співвідношення монотонне. Функція витрат задається як *R(x)*.

Навіть при надзвичайно великому просторі станів (можливо нескінченному, але «зліченно нескінченному») незалежно від складності системи алгоритм *MCMC* знаходитиме рішення з низькими витратами, якщо дати йому достатньо часу для сходження.

Подібне виконання дослідження простору станів є стандартною технікою стохастичної оптимізації та має безліч застосувань у таких галузях, як машинне навчання.

Розподіл Гіббса. Розподіл Гіббса – це розподіл максимальної ентропії при заданій множині обмежень. По суті, це означає, що якщо задати множину обмежень для ймовірностей системи, то розподіл Гіббса створить найменшу кількість припущень про форму розподілу. Розподіл Гіббса також є розподілом з найменшою чутливістю до варіацій в обмеженнях (за метрикою розбіжності Кульбака-Лейблера). Єдине обмеження, яке ми накладаємо на розподіл станів, – це функція витрат, тому ми використовуємо її в розподілі Гіббса для обчислення ймовірності переходу між станами:

$$Pxy = \exp(-\frac{R(y) - R(x)}{T}) * \frac{1}{Gx}, \qquad (1)$$

де G – це функція розбиття множини переходів зі стану x. Це нормуючий множник, який гарантує, що сума ймовірностей переходів із будь-якого стану дорівнює 1.

$$Gx = \sum_{y} (Pxy). \qquad (2)$$

Зауважимо, що якщо ми вирішимо, що наступний стан буде тим самим станом, то відносні витрати дорівнюють нулю, тобто ймовірність після нормалізації буде ненульовою (через форму розподілу з показником). Це означає, що в множину переходів необхідно включити можливість незміни станів.

Варто також зауважити, що розподіл Гіббса параметризується обчислювальною температурою *T*. Одна з ключових переваг використання ймовірностей при дослідженні простору станів полягає в тому, що система може виконувати переходи до більш витратних станів (оскільки вони мають ненульову ймовірність переходу), перетворюючи алгоритм на «нежадібний» метод оптимізації.

При прагненні температури до нескінченності ймовірність будь-якого окремого переходу прагне одиниці таким чином, що при нормалізації множини ймовірностей усіх переходів зі стану вони стають рівноймовірними (або розподіл Гіббса наближається до нормального розподілу), незважаючи на те, що їх витрати більші!

При наближенні обчислювальної температури до нуля ймовірнішими стають переходи з меншими витратами, тобто можливість кращих переходів підвищується.

За виконання дослідження/оптимізації простору станів ми поступово знижуємо температуру. Цей процес називається «імітацією відпалу». Завдяки цьому ми спочатку можемо легко вийти з локального мінімуму, а в кінці вибирати найкращі рішення.

Коли температура досить мала, то всі ймовірності прагнуть до нуля, за винятком імовірності відсутності переходу. Так відбувається тому, що відсутність переходу має нульову різницю витрат, тобто перебування в тому ж стані не залежить від температури. Через форму показової функції при *T=0* це виявляється єдиною можливістю з ненульовим значенням, тобто після нормалізації вона перетворюється на одиницю. Отже, наша система зійдеться до стійкої точки й подальше охолодження більше не буде потрібно. Це невід'ємна властивість створення ймовірностей за допомогою розподілу Гіббса.

Процес сходження системи можна налаштовувати за допомогою зміни швидкості охолодження. Якщо охолодження відбувається повільніше, то в результаті ми зазвичай приходимо до вирішення з меншими витратами (певною мірою), але ціною більшої кількості кроків збіжності. Якщо охолодження відбувається швидше, то вища ймовірність того, що система на ранніх етапах потрапить у пастку підділянки з більшими витратами, тобто ми отримаємо «менш оптимальні» результати.

Отже, марковський процес не просто генерує випадкові результати, а намагається згенерувати «хороші» результати й із певною ймовірністю йому це вдасться.

За визначенням довільних функцій витрат унікальний оптимум не повинен існувати. Цей метод імовірнісної оптимізації генерує лише наближення до оптимуму, намагаючись мінімізувати функцію витрат, і через виконання вибірки щоразу генеруватиме різні результати.

Сам процес вибірки може виконуватися за допомогою методу зворотного перетворення над функцією розподілу дискретної множини переходів.

Процедурна генерація рівнів. У деяких системах часто буває складно визначити простий алгоритм, що генерує гарні результати, особливо в разі складних систем. Завдання довільних правил генерації не тільки складне, а й обмежене лише нашою уявою та обробкою граничних випадків.

Якщо система задовольняє певному набору вимог, то застосування *MCMC* дає змогу нам не хвилюватися про підбір алгоритму чи правил. Натомість ми визначаємо метод для генерації будь-якого можливого результату й усвідомлено вибираємо хороший на підставі його «оцінки».

Розглянемо алгоритм для процедурної генерації мап, який можна розбити на три етапи:

1. Генерування та розміщення кімнат.
2. З'єднання кімнат коридорами.
3. Створення стін.

Генерування та розміщення кімнат. Створимо ігрове поле й розмістимо на ньому кілька випадкових кімнат, які стануть основою нашої мапи.

Для початку заведемо структури для мапи та кімнат у ній (відразу задаємо розміри мапи):

```
class Chart {
public:
struct Location {
int x, y, w, h;
};
Chart(int width, int height): chart_width(width), chart_height(height) {
chart_data.resize(width * height, 0);
}
private:
int chart_width, chart_height; // розміри мапи
std::vector<int> chart_data; // фінальні дані мапи
std::vector<Location> chart_locations; // кімнати
};
```

Одна з вимог для нової кімнати: вона не повинна перетинатися з наявними, але, щоб кімнати стояли щільно одна до одної, необхідно додати в *Location* таку функцію для перевірки перетину:

```
bool Location::intersect(const Location &r) const {
return !(r.x >= (x + w)
|| x >= (r.x + r.w) ||
}
```

Тепер можна згенерувати кілька кімнат із випадковим розташуванням і розмірами, для цього необхідно додати новий метод:

```
void Chart::generate(int locationCount) {
chart_locations.clear();
// другий цикл запобігає залипанню, якщо на мапу вже не поміщається жодна кімната
for (int i = 0; i < locationCount; ++i)
for (int j = 0; j < 1000; ++j) {
// ширина та висота кімнати в межах [10,40]
const int w = 10 + rand() % 31;
h = 10 + rand() % 31;
// уникаємо «прилипання» кімнати до краю мапи
const Location location = {3 + rand() % (chart_width - w - 6), 3 + rand() % (chart_height - h - 6), w, h};
// знайдемо першу кімнату, з уже наявних, яка перетинається з новою
auto intersect = std::find_if(std::begin(chart_locations), std::end(chart_locations), [&location](const Location &r){
return location.intersect(r);
});
// якщо нова кімната не має перетинів - додаємо її
if (intersect == std::end(chart_locations)) {
chart_locations.push_back(location);
break;
}
}
}
```

Тепер є набір випадкових кімнат, який досить легко можна перетворити на *2D*-масив мапи:

```
// занулюємо мапу індексом 0
chart_data.assign(chart_width * chart_height, 0);
// простір кімнат заповнюємо індексом 1
for (const Location &locations : chart_locations) {
for (int x = 0; x < locations.w; ++x) for (int y = 0; y < locations.h; ++y) {
chart_data[(locations.x + x) + (locations.y + y) * chart_width] = 1;
}
}
```

З'єднання кімнат коридорами. Тепер усі ці кімнати треба зв'язати між собою коридорами, інакше як герой переміщатиметься мапою? І знову ідея досить проста: послідовно переберемо кімнати на мапі та шукаємо шлях від середини однієї кімнати до середини наступної в списку, так отримуючи гарантований прохід від першої до останньої кімнати.

Для пошуку шляху використовуємо базовий алгоритм *A* (A star)*:

```
struct Dot {
```

```
int x, y, price;
bool operator==(const Dot &p) const {
return x == p.x && y == p.y;
}
bool operator<(const Dot &p) const {
return price > p.price;
}
};
void Chart::generatePassage(const Dot &start, const Dot &finish) {
// для зберігання напряму на «батьківську» клітину
std::vector<int> parents(chart_width * chart_height, -1);
// пріоритетна черга доступних клітин, відсортована за «вартістю»
std::priority_queue<Dot> active;
active.push(start);
// Напрями можливих переміщень
static const int directions[4][2] = {{1,0}, {0,1}, {-1,0}, {0,-1}};
while (!active.empty()) {
// беремо «найдешевшу» клітину зі списку доступних
const Dot = active.top();
active.pop();
if (dot == finish)
break;
// продовжуємо пошук у доступних напрямках
for (int i = 0; i < 4; ++i) {
Dot d = {dot.x – directions[i][0], dot.y – directions[i][1], 0};
if (d.x < 0 || d.y < 0 || d.x >= chart_width || d.y >= chart_height)
continue;
// якщо ми ще відвідували задану клітину
if (parents[d.x + d.y * chart_width] < 0) {
// обчислюємо «вартість» зазначеної клітини
d.price = calcPrice (d, finish);
active.push(d);
parents[d.x + d.y * chart_width] = i;
}
}
}
// шлях знайдено – тепер прокладаємо його на мапі, починаючи з кінця
Dot dot = finish;
while (!(dot == start)) {
chart_data[dot.x + dot.y * chart_width] = 1;
const int *directon = directions[parents[dot.x + dot.y * chart_width]];
dot.x + = directon [0];
dot.y + = directon [1];
}
}
```

Функція *calcPrice()* повертає «вартість» клітини, що обчислюється так, що нам вигідніше переміщатися вже наявними кімнатами й коридорами, ніж створювати нові. Плюс невелика евристика, яка дає змогу нам цілеспрямовано йти до кінцевої точки. Із цими параметрами можна грати самостійно.

Створення стін. Для повноти не вистачає ще однієї невеликої деталі – стін. Звичайно, можна обійтися й без них, але, як правило, стіни можна красиво намалювати, візуально розмежовуючи простір мапи. Тому додамо новий метод, який пройдеться по всій мапі й додасть стіни скрізь, де «порожні» клітини межують із клітинами кімнат та коридорів

```
void Chart::generateSide() {
// Зміщення для сусідніх клітин
static const int offsets[8][2] = {
{-1,-1}, {0,-1}, {1,-1}, {1, 0},
{1, 1}, {0, 1}, {-1, 1}, {-1, 0},
};
// Ігноруємо границю мапи, щоб не перевіряти граничні умови
for (int x = 1; x < chart_width - 1; ++x)
for (int y = 1; y < chart_height - 1; ++y) {
if (chart_data[x + y * chart_width] == 0)
for (int i = 0; i < 8; ++i) {
// якщо по сусідству є хоч одна клітина кімнати чи коридору – розміщуємо стіну (індекс 2)
if (chart_data[(x + offsets[i][0]) + (y + offsets[i][1]) * chart_width] == 1) {
chart_data[x + y * chart_width] = 2;
break;
}
}
}
}
```

Висновки. У статті досліджено техніку автоматичної генерації контенту на основі процедурних алгоритмів. Проаналізовано варіанти основної структури даних для генератора ігрових мап і наведено остаточну реалізацію гнучкої, універсальної та розширюваної структури даних. Детально описано алгоритми процедурної генерації контенту, які використовуються для побудови ігрових рівнів. Наведено список класів і методів, що описує реалізацію цих алгоритмів. На основі представленої гнучкої структури даних може бути розроблено програму-генератор двовимірних ігрових мап, що забезпечить можливість її розширення з подальшим інтегруванням не тільки в комп'ютерних або мобільних іграх, а й у тренувальних симуляторах для підготовки фахівців різних галузей діяльності людини, у тому числі транспортної сфери.

Список літератури:

1. Jeong K., Lee J., Kim J. A Study on New Virtual Reality System in Maze Terrain. *Int. J. Hum. Comput. Interact.* 2018. № 34. P. 129–145. URL: https://doi.org/10.10-80/10447318.2017.1331535.
2. Lambie R. Build Your Own First-Person Shooter in Unity. Raspberry Pi Press, 2020. 140 p.
3. Hocking J. Unity in Action: Multiplatform Game Development in C#. 2nd edition. Manning Publications Co, 2018. 400 p.
4. Baron D. Hands-on Game Development Patterns with Unity 2019. *Packt* Publishing, 2019. 242 p.
5. Kok B. Beginning Unity Editor Scripting: Create and Publish Your Game Tools. Apress, 2021. 274 p.
6. Engelbert R. Word Games With Unity. *Engelbert Publishing,* 2018. 457 p.
7. Bangs E. Unity. Tachyon Publications LLC, 2021. 304 p.
8. Murray J.W. C# Game Programming Cookbook for Unity 3D. CRC Press, 2021. 316 p.
9. Peterson K. Unity Game Development: Programming C# in Unity Engine, a guide book for beginners. Independently published, 2021. 151 p.
10. Moore A., Aronowitz A. C# for Unity: Beginning C# Programming with Unity. NLN lnc, 2021. 232 p.
11. Alves C. Unity 3d: Build, customize, and optimize professional games using unity 3d. Independently published, 2021. 232 p.
12. Linowes J. Unity Virtual Reality Projects (1st ed.). Packt Publishing, 2015. 288 p. URL: https://www.perlego.com/book/4096/unity-virtual-reality-projects-pdf
13. Methods and models for assessment of reliability of structural-complex systems / A. Zavgorodnya, V. Zavgorodnii, V. Maiko, V. Malikov, D. Zhuk. *World Science.* 2018. № 11 (39). P. 5–14. URL: https://doi.org/10.31435/rsglobal_ws/30-112018/6227.
14. Завгородній В.В., Завгородня Г.А. Метод кількісної оцінки ризику технічних систем. *Транспортні системи і технології.* 2018. Вип. 32–33. С. 87–95. URL: https://doi.org/10.32703/2617-9040-2018-32-2-87-95.
15. Завгородня Г.А., Завгородній В.В. Аналіз методів виробки рішень при виникненні техногенних аварій у системах управління реального часу. *Транспортні системи і технології.* 2019. Вип. 34. С. 175–181. URL: https://doi.org/1-0.32703/2617-9040-2019-34-2-1.

Zavgorodnii V.V., Zavgorodnya A.A., Valyavska N.O., Adamenko V.S., Dorogovtsev E.V., Nesmachny P.V. **METHOD OF AUTOMATIC CONTENT GENERATION BASED ON PROCEDURAL ALGORITHMS**

The work is devoted to the development of the method of game content generation based on the use of Markov chains. The system is defined by the space of states describing all possible configurations. In this case transitions between states are described as discrete steps characterized by their probabilities.

We consider the Monte Carlo algorithm for Markov chains as a technique for obtaining sampling from the state space. Sampling means selecting states based on the probability of selection based on the internal distribution. The probability of being in a state is proportional to a certain cost function, which gives an estimate of the current state the system is in. It is shown that even with an extremely large space of states, regardless of the complexity of the system, this algorithm will find a low-cost solution if given enough time to climb.

We also consider the Gibbs distribution as the distribution with the lowest sensitivity to variations in constraints. To calculate the transition probability between states in the Gibbs distribution, a cost function was used, which is the only constraint imposed on the distribution of states.

An algorithm for procedural generation of levels is considered, which can be divided into stages: generation and placement of rooms (a playing field is created and several random rooms are placed on it, which should not intersect), connection of rooms (all rooms should be connected by corridors) and creating walls (the method will run across the map and add walls wherever "empty" cells border cells in rooms and corridors).

The application of the considered algorithms makes it possible to define a method for generating any possible result and consciously choose the optimal one based on its evaluation.

Key words: *content generation method, Markov process, procedural algorithms, Monte Carlo method, Gibbs distribution, procedural level generation.*