УДК 004.94

ОБЗОР СИСТЕМ ПРОЦЕДУРНОЙ ГЕНЕРАЦИИ ИГР

М.Г. Меженин

Процедурная генерация контента (ПГК) является одной из наиболее актуальных задач в индустрии видеоигр. Под ПГК понимают автоматическое создание различных составляющих частей игр; особый интерес представляет проблема автоматического создания игровых правил и целых игр. В статье представлен обзор исследований, посвященных данной проблеме; описываются алгоритмы генерации игровых правил и игр различных жанров, в том числе алгоритмы на основе парадигм эволюционного моделирования и логического программирования, а также способы автоматической оценки генерируемых игр. Кратко рассмотрены первые системы генерации игр, реализованные в универсальных игровых программах. Также описаны различные форматы представления и специализированные языки для описания игровых правил в подходящем для алгоритмической обработки виде.

Ключевые слова: процедурная генерация контента, игровой дизайн, универсальные игровые программы, искусственный интеллект.

Введение

Процедурная генерация контента является одним из наиболее актуальных и активно развивающихся направлений исследований в сфере мультимедиа, в частности в индустрии видеоигр. Под *процедурной генерацией контента (ПГК)* понимают автоматическое и полуавтоматическое создание и динамическое изменение различных составляющих частей игр, в том числе игровых объектов и уровней, двумерной и трехмерной графики, эффектов, звуков, музыки, персонажей, сюжетов и др. [10]. Использование ПГК позволяет не только значительно понизить стоимость создания контента, но и решает проблему персонализации, приобретающую большую значимость в связи с увеличением количества потенциальных игроков.

В исследованиях алгоритмов ПГК особенный интерес представляет проблема процедурной генерации игровых правил, которые лежат в основе всех, в том числе и неэлектронных, игр. Решение данной задачи обладает практической значимостью, поскольку процедурная генерация правил позволит не только существенно разнообразить видеоигры за счет динамического изменения правил в процессе игры (в том числе, адаптируя игровой процесс под желания игрока), но и, возможно, создать совершенно новые игровые механики и жанры [24]. Кроме того, системы генерации игровых правил могут быть полезны исследователям алгоритмов искусственного интеллекта в сфере универсальных игровых программ, а также игровым дизайнерам для быстрого создания и тестирования различных игровых прототипов [15].

Процедурная генерация игр невозможна без языка описания игровых правил, то есть некоторого строго-определенного формата их представления в подходящем для алгоритмической обработки виде. Разработка такого языка описания обладает теоретической значимостью, поскольку ее результатом будет строгий научно-терминологический аппарат, который позволит формализовать и описать пространство возможных игровых правил [24, 25].

Процедурная генерация игровых правил является достаточно новой областью исследований. В данной статье рассмотрены ключевые работы в этой сфере, посвященные разработке систем генерации игр и языков их описания.

Статья организована следующим образом. В первом разделе рассмотрены исследования в области универсальных игровых программ. Во втором разделе описаны современные системы процедурной генерации видеоигр. В третьем разделе кратко рассмотрены различные языки описания игр. В заключении суммируются сделанные на основе данного обзора выводы и рассматриваются направления дальнейших исследований.

1. Универсальные игровые программы

В конце 1980-х годов одной из основных и наиболее актуальных задач в области искусственного интеллекта являлась задача создания компьютерной программы для игры в шахматы. С каждым годом шахматные программы становились всё умнее, и уже в 1989 году компьютерная система Deep Thought выиграла в демонстрационном матче у международного мастера по шахматам Дэвида Леви. Вместе с тем, данные системы были настолько сильно специализированы на игру в шахматы, что, по мнению некоторых ученых, достижения в данной области уже не могли использоваться для решения более общих задач искусственного интеллекта (ИИ) [24].

Для решения данной проблемы была предложена концепция *универсальной игровой программы* (УИП), то есть такой системы ИИ, которая была бы способна играть в различные игры, не обладая какими-либо знаниями о конкретной игре до ее начала. Одной из первых работ в данной области является система *METAGAME* [17], состоявшая из двух основных подсистем: модуля ИИ, способного играть в различные настольные игры, и *генератора игр*, назначение которого заключалось в генерации игр для тестирования алгоритмов ИИ.

Таким образом, METAGAME генерирует игры из многочисленного, но конечного множества «симметричных шахмато-подобных игр», удовлетворяющих следующим основным критериям:

— игры происходят на прямоугольной доске с квадратными клетками;
— в игре принимает участие двое игроков;
— начальные позиции фигур и правила симметричны для обоих игроков.

Множество возможных игр в METAGAME задается с помощью формальной *игровой грамматики*, которая определяет начальное расположение фигур, правила их движения и взятия, условия победы и др. Пример описания конкретной игры в формате METAGAME приведен на рис. 1.

Конкретные игры генерируются с помощью случайной выборки из данной грамматики, без проверки на их проходимость, поскольку в общем случае данная проблема является NP-полной. Для отсечения тривиальных игр, авторы реализовали несколько простых проверок, например, не допускающих игры, цель которых заключается в достижении фигурами своей начальной позиции.

Некоторые из параметров генератора также могут задаваться пользователем перед запуском системы. В частности, такими параметрами являются:

— сложность правил (максимальная длина описания правил движения и взятия для каждой фигуры);

— сложность предоставляемого игроку выбора (например, при генерации более простой версии шахмат игроку может не предоставляться выбор, на какую фигуру заменяется пешка при достижении последней горизонтали);
— глубина поиска (например, при меньшей глубине поиска могут рассматриваться игры с меньшим размером доски);
— локальность (максимальное расстояние, преодолеваемое фигурой за один ход).

```
DEFINE man
 MOVING
  MOVEMENT
   LEAP
   <1,1> SYMMETRY {side}
  END MOVEMENT
 END MOVING
 CAPTURING
  CAPTURE
   BY {hop}
   TYPE [{opponent} any_piece]
   EFFECT remove
                                    MOVEMENT
                                     HOP BEFORE [X = 0]
                                         OVER [X = 1]
                                         AFTER [X = 0]
                                     HOP OVER [{opponent} any_piece]
                                     <1,1> SYMMETRY {side}
                                    END MOVEMENT
                                   END CAPTURE
                                  END CAPTURING
                                  PROMOTING
                                   PROMOTE TO king
                                  END PROMOTING
                                  CONSTRAINTS continue_captures
                                 END DEFINE
```

Рис. 1. Частичное описание игры «американские шашки» на языке METAGAME

Несмотря на то, что процедурная генерация игровых правил не была основной целью проекта METAGAME, эта работа оказала большое влияние на последующие исследования в данной области. Многие последующие языки описания и системы генерации игр во многом основываются именно на результатах METAGAME.

Так, в проекте *Gala* [12] было предложено развитие концепций METAGAME с целью создания универсальной игровой программы для игр с неполной информацией. В рамках проекта был предложен одноименный декларативный язык описания игр на основе языка Prolog и алгоритмы поиска оптимальных стратегий для игр, определенных с помощью этого языка.

```
game(blind_tic_tac_toe,
 [players : [a, b],
  objects : [grid_board : array('$size', '$size')],
  params : [size],
  flow : (take_turns(mark,unless(full),until(win))),
  mark : (choose('$player', (X, Y, Mark),
             (empty(X, Y), member(Mark, [x, o]))),
          reveal('$opponent', (X, Y)),
          place((X, Y), Mark)),
  full : (\+(empty(_, _)) ->
  outcome(draw)),
  win : (straight_line(_, _, length = 3, contains(Mark)) ->
           outcome(wins('$player')))]).
```

Рис. 2. Описание игры «слепые крестики-нолики» на языке Gala

В отличие от языка METAGAME, с помощью языка Gala можно описать на порядок большее множество игр, а именно настольные игры на прямоугольной доске с полной и неполной информацией для одного, двух и более игроков. По мнению авторов, их язык описания игр более лаконичен и прост для понимания. На рис. 2 приведен пример

описания на языке Gala игры «слепые крестики-нолики», отличающейся от обычной версии тем, что во время своего хода игрок сообщает лишь клетку, на которую он сходил, не указывая, поставил ли он в нее «крестик» или «нолик».

Язык Gala позволяет задавать некоторые параметры игры в виде переменных; такими параметрами могут быть в том числе и правила движения фигур. Поскольку генерация игровых правил не была целью проекта Gala, его авторы предлагают простой программный интерфейс, благодаря которому для генерации правил на языке Gala можно использовать систему METAGAME.

2. Процедурная генерация игр

2.1. Генерация настольных игр

Исследования в области универсальных игровых программ были в первую очередь ориентированы на изучение алгоритмов ИИ, и генерация игр в них либо не рассматривалась вовсе, либо была основана на простейших алгоритмах.

В работе В. Хома и Д. Маркса [11] задача автоматического создания игр впервые была рассмотрена как отдельная актуальная проблема в области ИИ. Целью работы было создание алгоритма для генерации *сбалансированных* игр, то есть игр, в которых все игроки находятся в равных условиях и имеют одинаковые шансы выиграть. Отметим, что симметричность игры не гарантирует ее сбалансированность, поскольку даже при идентичности правил для обоих игроков игрок, ходящий первым, может обладать преимуществами над вторым игроком (и, в редких случаях, наоборот).

Авторами работы был предложен термин *автоматический игровой дизайн*, под которым понималась автоматическая корректировка баланса игры путем изменения ее правил. Отметим, что идея динамического изменения игрового баланса рассматривалась и в более ранних исследованиях, однако в них корректировка баланса достигалась путем оптимизации заранее определенных параметров игры, а не самих игровых правил.

Работа Хома и Маркса основывалась на коммерчески успешной универсальной игровой программе Zillions of Games (Zillions Development Corp., 1998), игры для которой описывались пользователями вручную на специальном декларативном языке ZRF. Система Zillions of Games предназначалась для визуализации любых настольных игр на доске, описанных на языке ZRF, с возможностью игр в них с УИП.

Предложенный алгоритм автоматического изменения баланса основан на генетическом алгоритме и заключается в следующем. На каждой итерации алгоритма из некоторого набора игр путем изменения и комбинирования их правил генерируются новые игры, в каждую из которых УИП Zillions of Games играет сама с собой 100 матчей. Каждая игра оценивается с помощью функции оценки по шкале баланса (большую оценку получают игры с минимальной разницей между количеством побед у двух игроков) и шкале разнообразия (большую оценку получают игры, наименее похожие на ранее сгенерированные результаты), после чего формируется новый набор из старых и новых игр с наибольшей оценкой и начинается следующая итерация.

Использование генетических алгоритмов и оценка результатов некоторой симуляции, а не самих особей, являются достаточно распространенными методами процедурной генерации контента; в данной работе было продемонстрировано, что эти методы могут быть успешно использованы и для динамической генерации игр.

В дальнейшем данные методы были развиты в проекте *Ludi* [1], одним из главных достижений которого является игра Yavalath — первая в мире коммерчески изданная игра, полностью сгенерированная программой в автоматическом режиме.

В рамках проекта Ludi был разработан специальный функциональный язык для описания настольных игр на доске, названный Ludi GDL. Описания на данном языке состоят из абстракций высокого уровня, что позволяет легко и лаконично описывать различные игры. Ludi GDL также является расширяемым, но для успешной работы любые дополнительные абстракции необходимо реализовывать и на низком уровне. Пример описания игры «крестики-нолики» на языке Ludi GDL приведен на рис. 3.

```
(game Tic-Tac-Toe
  (players White Black)
  (board (tiling square i-nbors)(shape square)(size 3 3))
  (end (Allwin(in-a-row 3)))
)
```

Рис. 3. Описание игры «крестики-нолики» на языке Ludi GDL

При оценивании игр в системе Ludi акцент делается на поиск наиболее интересных игр. Для этой цели по каждой сгенерированной игре в Ludi проводится некоторое количество автоматических матчей с двумя компьютерными игроками, после чего игра оценивается по 57 критериям, разделенным на три основные группы:

— *объективные* критерии для оценки качеств самих правил независимо от тестовых матчей;
— критерии *играбельности (playability)* для оценки результатов тестовых матчей;
— *качественные* критерии для оценки динамики тестовых матчей.

Авторами отмечается, что объективные критерии были наименее эффективны в поиске интересных игр. Критерии играбельности, характеризующие интересность и базовые свойства игрового процесса, в Ludi повторяют описанные выше критерии сбалансированности игр с добавлением критерия средней длины матча.

Особый интерес представляют качественные критерии, основанные на анализе *историй лидирования* в тестовых матчах, то есть данных о степени лидирования одного игрока над другим в каждый из ходов. Пример истории лидирования приведен на рис. 4.

Рис. 4. Пример истории лидирования

В данном примере перед своей победой на 27-м ходу белый игрок находится в проигрышной позиции с 18 по 22 ход. По мнению авторов, если при некотором наборе правил во многих матчах в процессе игры происходит смена лидера, данная игра является более «драматичной» и интересной.

Генерация игр в Ludi реализована на основе стандартных методов генетического программирования. В рамках тестирования системы с помощью Ludi было сгенерировано 1389 игр, из которых 19 были признаны авторами достаточно интересными, а еще 2, названные Yavalath и Pentalath, — крайне интересными. Описание игры Yavalath на языке Ludi GDL приведено на рис. 5. Правила Yavalath схожи с «крестиками-ноликами» на гексагональной доске; главное отличие — игрок побеждает, поставив четыре фигуры в ряд, но проигрывает, поставив в ряд только три фигуры.

```
(game Yavalath
  (players White Black)
  (board (tiling hex)(shape hex)(size 5))
  (end (Allwin(in-a-row 4)) (Alllose (in-a-row 3)))
)
```

Рис. 5. Описание игры Yavalath на языке Ludi GDL

Дополнительное правило может показаться излишним, но на самом деле оно добавляет игре стратегическую глубину. Например, на рис. 6 приведена позиция, в которой ход #1 белого игрока вынуждает черного игрока сделать блокирующий ход #2, и тем самым проиграть из-за нового правила. Таким образом, игроки могут в некоторой степени управлять ходами противника с помощью нетривиальных последовательностей ходов – подобное появление сложных стратегий из достаточно простых правил подтверждает высокое качество и интересность данной игры.

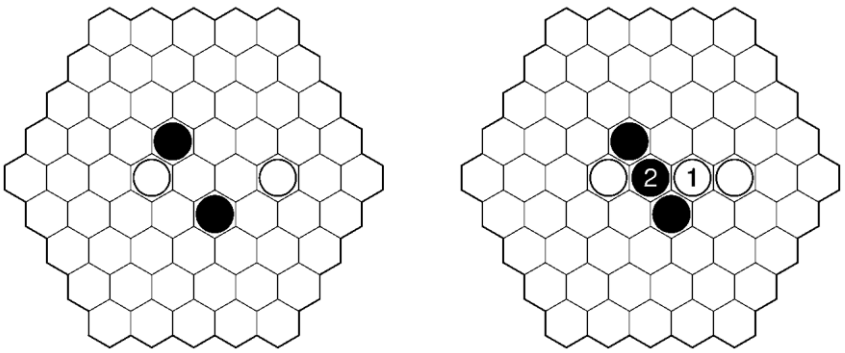

Рис. 6. Выигрышная последовательность ходов белых в Yavalath

Yavalath и Pentalath были впоследствии изданы в коммерческом виде издательством Nestorgames и являются одними из наиболее успешных игр в их каталоге.

2.2. Генерация аркадных видеоигр

В конце 2000-х годов несколько исследователей параллельно начали работу над системами процедурной генерации правил для аркадных видеоигр. Под аркадными играми мы будем понимать двумерные и трехмерные игры, игровой процесс которых заключается в движении некоторых объектов, их столкновении, появлении и исчезновении, и так далее. Данный набор действий характеризует достаточно базовые понятия, из кото-

рых, однако, состоит множество классических игр, таких как Pac-Man (Namco, 1980), Super Mario Bros. (Nintendo, 1985) и Space Invaders (Taito, 1978).

В отличие от шахмато-подобных игр, в аркадных видеоиграх игрок обычно принимает роль некоторого персонажа, движениями которого он управляет в реальном времени. Другими распространенными элементами данного жанра являются управляемые компьютером противники, изменяющие характеристики игрока бонусы, и карта-уровень с преградами. Эти и другие отличия аркадных видеоигр от настольных представляют особый интерес в исследовании алгоритмов их процедурной генерации.

Один из первых опытов в данном направлении описан в работе Дж. Тогелиуса [22], в которой предложен способ генерации простых двумерных игр, подобных знаменитой игре Pac-Man. Все игры в разработанной авторами системе происходят на дискретном поле размером 15×15 клеток; поле состоит из свободного пространства и стен и никогда не меняется. На игровом поле также расположен игрок и некоторые объекты разных цветов, поведение и характеристики которых определяются правилами, едиными для всех объектов одного цвета. Игра происходит в реальном времени, разделенном на отдельные промежутки; в каждый дискретный отрезок времени игрок и объекты могут двигаться на одну клетку в любом свободном направлении. Конец игры наступает по истечении определенного количества отрезков времени; игра считается «выигранной» по достижении некоторого числа очков.

Для кодирования подобных игр в системе используется набор переменных и две матрицы. Переменные задают базовые параметры игры — длительность игры, максимальное количество очков и число объектов, а также шаблон их движения. В первой матрице задаются эффекты от столкновений (то есть расположения на одной клетке) объектов разных цветов друг с другом и игроком. Список возможных эффектов включает в себя: исчезновение, перемещение на случайную клетку, а также пустой эффект. Например, в матрице может быть указано, что при столкновении красного и зеленого объекта красный объект исчезает, а зеленый объект перемещается на случайную клетку. Подобным образом во второй матрице описывается положительное или отрицательное изменение очков игрока от столкновения объектов и игрока.

Как и в случае описанной ранее системы Ludi, авторы ставили перед собой задачу генерации интересных игр с помощью методов генетического программирования. В системе использовался достаточно тривиальный генетический алгоритм, на каждой итерации которого копия текущей игры подвергалась операции мутации (то есть случайному изменению некоторых значений переменных и элементов матриц), после чего оригинальная игра заменялась на копию, если значение функции оценки новой игры было выше.

Особый интерес в данной работе представляет реализация функции оценки. По мнению психологов и игровых дизайнеров, одной из ключевых составляющих любой игры является изучение игроком ее правил; чем интереснее игроку осваивать игру и учиться лучше в нее играть, тем интереснее игра в целом. Основываясь на данной теории, авторы работы предложили оценивать интересность игр на основе их *изучаемости*. Таким образом, высоко оцениваются игры, которые, с одной стороны, достаточно сложны для новых игроков, но, с другой стороны, достаточно легки для освоения. Для автоматической оценки изучаемости игр авторы реализовали игрового агента на основе нейронной сети, обучение которого происходит с помощью алгоритма эволюционной

стратегии. Каждая новая игра несколько раз тестируется с помощью двух агентов со случайными стартовыми параметрами. Игра оценивается тем выше, чем больше очков было набрано агентами; если любым из агентов было набрано большое количество очков уже на первых тестах, игра считается слишком легкой и оценивается отрицательно.

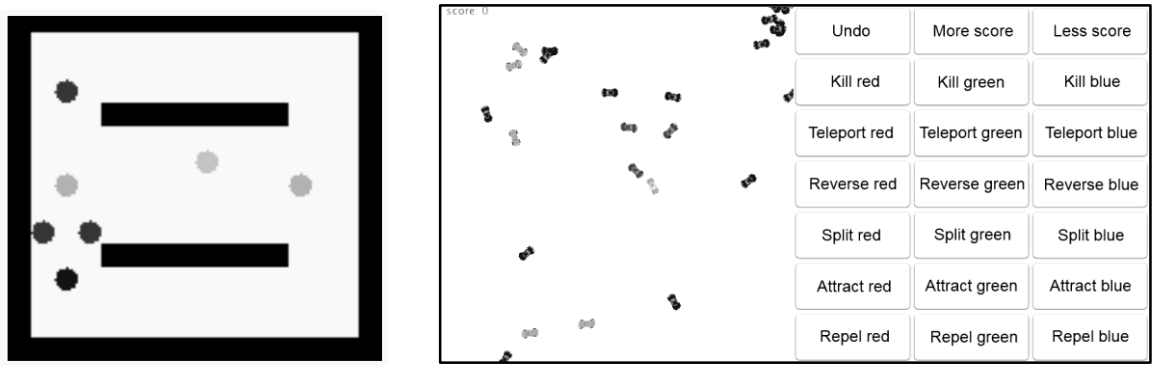

Рис. 7. Скриншоты простых аркадных игр, сгенерированных в рамках экспериментов Дж. Тогелиуса

Дальнейшие исследования авторы проводили в области интерактивной полуавтоматической генерации игр [21]. В реализованной ими системе моделируется простая двумерная игра, в которой игрок в реальном времени управляет автомобилем с простой физикой движения и может сталкиваться с управляемыми компьютером машинами разных цветов. На старте системы при столкновении игрока с другими машинами ничего не происходит; однако, после каждого столкновения игрок может нажать одну из кнопок, отвечающих за разные эффекты («добавить очки», «уничтожить объект», и так далее). После нажатия кнопки в игре происходит выбранный эффект, а система с помощью алгоритмов машинного обучения пытается определить, почему игрок нажал на эту кнопку, и сгенерировать соответствующие игровые правила. Скриншоты игр, сгенерированных в описанных выше экспериментах, изображены на рис. 7.

Во всех описанных в данной главе работах для генерации игровых правил в той или иной мере использовались генетические алгоритмы и методы эволюционного программирования. Недостатком данного подхода является необходимость генерировать и оценивать большое количество игр, из которых лишь единицы удовлетворяют заданным критериям качества.

Альтернативный подход был предложен в рамках проекта *Variations Forever* [19], в котором описание множества генерируемых игр реализовано на основе парадигмы *Answer Set Programming* (ASP, программирование стабильных моделей). В данном подходе множество всех возможных игр задается с помощью набора логических ограничений.

```
pushes(A,B) :- on_collide(A,B,bounce) , on_collide(B,A,bounce).
kills(A,B)  :- on_collide(A,B,kill).
indirectly_pushes(A,B) :- pushes(A,B).
indirectly_pushes(A,C) :- pushes(A,B) , indirectly_pushes(B,C).
winnable_via(indirect_push_kill(A,C)) :-
  indirectly_pushes(A,B),kills(B,C).
compute {
  player_agent(A) , goal(kill_all(B)),
  winnable_via(indirect_push_kill(A,B)) }.
```

Рис. 8. Набор логических ограничений для генерации игр в Variations Forever

Каждый такой набор задает некоторое множество игр, схожих по некоторому критерию. На рис. 8 приведен пример такого набора, описывающего все игры, выиграть в которых можно, толкая объекты друг в друга. Таким образом, вместо перебора и модификации конкретных игр, в данном подходе изменяется исходный набор ограничений. Добавление одного ограничения позволяет исключить из рассмотрения не отдельную игру, а целый класс неинтересных игр.

Конкретные игры генерируются из набора ограничений и представляются в виде *логических термов*; термами являются константы, переменные и функторы, аргументами которых являются другие логические термы. Игровое правило, представленное в виде логического терма, может иметь следующий вид:

$$\text{scripted\_event}(\text{spawn}(\text{boss\_creature},\ \text{temple}),\ 120) \qquad (1)$$

и означать событие «через две минуты после начала игры поместить в храм существо класса boss_creature». Подобным образом в Variations Forever представлены все параметры генерируемых игр: характеристики и поведение игровых объектов, игровое поле с препятствиями и эффекты взаимодействия с ними игрока. Пример полного ASP-описания простой игры и скриншот ее визуализации приведены на рис. 9.

space_resolution(32,24).
space_topology(spherical).
background(grids; stars).
active_agent(red; yellow; white; cyan).
agent_movement(red,asteroids; white,asteroids;
yellow,roguelike; cyan,pacman).
agent_population(red,many; white,singleton;
yellow,singleton; cyan,many).
agent_collide_effect(red,white,kill;
cyan,yellow,kill).
player_agent(white).
obstacle_distribution(enclosure; random_walls;
random_blocks).
obstacle_collide_effect(red,kill; white,kill).
goal(kill_all(red)).

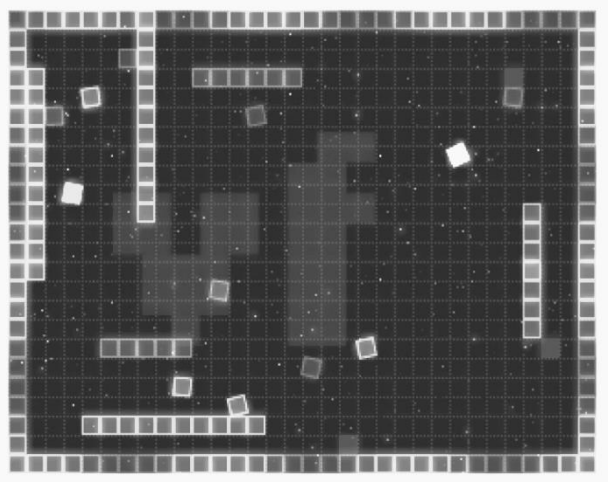

Рис. 9. Описание и скриншот игры, сгенерированной в системе Variations Forever

Отметим, что на данном этапе в системе отсутствует возможность автоматической оценки качества игр и корректировки набора ограничений; обе этих задачи должны решаться вручную игровым дизайнером.

Наиболее успешных результатов в области процедурной генерации видеоигр добился М. Кук в рамках проекта *ANGELINA*, каждая новая версия которого решает новую исследовательскую задачу. Скриншоты игр, сгенерированных разными версиями системы ANGELINA, представлены на рис. 10.

Первая версия системы ANGELINA генерировала простые двумерные игры с помощью эволюционного моделирования [7] и во многом была схожа с описанной выше работой Дж. Тогелиуса. Главным отличием данной системы является представление игровых правил, расположения объектов и карты уровня в виде трех раздельных сущностей, генерируемых с помощью методов *кооперативной коэволюции*. Под кооперативной коэволюцией в эволюционном моделировании понимают раздельное эволюционирование

особей из популяций различных классов, при котором, однако, оценка результатов происходит совместно. При совместной оценке принимается во внимание, насколько хорошо особи из разных популяций подходят друг другу. Например, в системе ANGELINA может быть сгенерирована интересная карта уровня и оригинальные игровые правила, которые будут оценены отрицательно вследствие того, что с такими правилами уровень становится непроходимым.

Рис. 10. Скриншоты игр, сгенерированных разными версиями системы ANGELINA (в хронологическом порядке)

Во второй версии системы ANGELINA авторы сфокусировались на генерации игр в так называемом жанре «Metroidvania» [4]. Данный жанр игр характеризуется сложной структурой уровней и наличием особых «предметов-усилителей», которые позволяют нашедшему их игроку открывать все большее число ранее недоступных ему областей карты. Примером усилителя может служить предмет, увеличивающий высоту прыжка игрока и тем самым позволяющий ему допрыгнуть до ранее недоступных участков уровня. Авторы добавили в систему кооперативной коэволюции отдельную популяцию с характеристиками усилителей, положительно оценивая такие наборы усилителей, которые позволяют проходить игру разными путями. Оценка игр в ANGELINA проводится автоматически, путем моделирования действий игрока.

Данный подход был развит в системе *Mechanic Miner*, в которой при генерации усилителей во внимание принимался исходный код игры [6]. Теперь, при генерации усилителя изменяемую им характеристику система Mechanic Miner выбирала не из заранее заданного списка, а из всех переменных, найденных с помощью рефлексии в исходном коде игры. Например, в процессе генерации одной из игр система Mechanic Miner обнаружила у объекта *player* переменную *acceleration,* и в качестве эффекта усилителя инвертировала значение ее y-компоненты:

$$\text{player.acceleration.y} \mathrel{*}= -1 \tag{2}$$

что в терминах игры означает, что после нахождения данного усилителя игрок буквально начнет ходить по потолку уровня, попросту игнорируя ранее мешавшие ему препятствия. Найденная с помощью Mechanic Miner в автоматическом режиме, данная игровая механика является крайне интересной, хотя и не новой — она лежит в основе популярной коммерческой игры *VVVVVV* (Cavanagh, 2010).

Отметим, что автоматическое изменение кода программы конечно же не может не приводить к ошибкам исполнения. Для решения этой проблемы, в процессе оценивания каждой игры Mechanic Miner перехватывает все возможные исключения и дает отрицательную оценку тем усилителям, что привели к появлению ошибок. В случае отсутствия

ошибок Mechanic Miner использует комплексную функцию оценки, принимающую в расчет *полезность* усилителя; усилитель считается полезным, если он позволяет игроку получить доступ к новым участкам карты и помогает пройти игру. Так, увеличение высоты прыжка на один пиксель скорее всего не даст игроку преимуществ, поэтому такой усилитель будет оценен отрицательно. У данного подхода есть несколько недостатков; во-первых, он малоприменим на уровнях большого размера из-за сложности расчета доступности различных участков карты; во-вторых, в текущей реализации отсутствует обработка «слишком полезных» усилителей, делающих игру неинтересной и тривиальной: например, усилитель, который моментально переносит игрока к концу уровня, получит высокую оценку несмотря на то, что он сводит к нулю все остальные составляющие игры.

В последней версии ANGELINA авторы впервые реализовали процедурную генерацию трехмерных игр [3, 5]. Несмотря на смену архитектуры системы при ее портировании на платформу Unity, алгоритмы генерации игр практически не изменились.

Отдельный интерес представляют разработанные в рамках проекта ANGELINA методы генерации игр на заданную тему [2]. Принимая на входе одно слово или целый текст, ANGELINA выделяет ключевые слова и подбирает к ним различные ассоциации. Далее система анализирует семантику полученных слов с помощью нескольких источников и на основе полученных результатов производит поиск свободно распространяемых графических материалов и музыки, подходящих по смыслу и настроению к ключевым словам и ассоциациям.

Отметим, что система ANGELINA не лишена недостатков. Авторы указывают, что реализованная ими автоматическая оценка игр дает недостаточно качественные результаты, в то время как живые игроки затрудняются давать сравнительные оценки множеству похожих игр.

3. Языки описания игр

Одним из главных факторов успешности системы процедурной генерации является выбор способа представления контента, поскольку именно от него во многом зависит разнообразие генерируемого контента [25]. При выборе способа представления контента нужно найти баланс между слишком общим и слишком узким представлением. Чем шире класс контента, который можно описать с помощью выбранного способа представления, тем сложнее найти среди экземпляров этого класса подходящие качественные решения; в случае выбора слишком узконаправленного способа представления, экземпляры генерируемого контента будут отличаться лишь деталями.

Данная проблема особенно актуальна в случае с процедурной генерацией игр, ведь многие игры не похожи друг на друга – например, шахматы и Mario не имеют между собой ничего общего. Достаточно общим способом представления, подходящим для всех игр, является, например, язык программирования C++. Действительно, любую игру можно представить в виде программы на C++. Однако, данный способ представления задает чрезвычайно обширный класс контента: подавляющее большинство программ, которые можно написать на языке C++, не будут являться играми; более того, лишь малая часть из них не будет являться случайным набором инструкций.

Несмотря на то, что во всех работах, описанных во втором разделе данной статьи, авторы создавали свой способ представления игровых правил, многие ученые исследуют

возможность разработки специализированного языка для описания игровых правил. Создание подобного языка позволит не только отделить описание игры от используемых средств визуализации, оценки и генерации игр, но и упростит задачу разработки и взаимодействия универсальных игровых программ [8].

Одним из первых и наиболее известных языков описания игр является Game Description Language (GDL), разработанный в стэндфордском университете [13]. Данный язык был разработан в рамках исследований УИП и может использоваться для описания широкого класса пошаговых игр с полной информацией; большинство описанных на нем игр схожи с шахматами и другими настольными играми на доске. GDL основан на логике первого порядка и крайне подробен. Описание даже простейших игр на GDL занимает несколько страниц; так, для описания правил «крестиков-ноликов» потребуется около трех страниц. Это является одной из причин, по которой данный язык не используется в системах процедурной генерации, поскольку автоматическое генерация и оценка правил на столь сложном языке затруднительны. Существуют различные модификации данного языка, в том числе для поддержки игр с неполной информацией [20].

Одной из первых попыток создания языка для описания видеоигр является проект *Extensible Graphical Game Generator* [16], в рамках которого был реализован язык для описания двумерных графических игр. Основной целью данного проекта являлась разработка системы, позволяющей создавать игры, не прибегая к программированию. С помощью данной системы, пользователь может сгенерировать полноценную игру на языке Perl, просто описав ее правила. Отметим, что в данной работе не идет речь о процедурной генерации игр, поскольку все правила генерируемой игры должны быть заранее заданы пользователем.

Язык Video Game Description Language является попыткой реализовать универсальный язык для описания видеоигр, который бы мог использоваться как в УИП, так и в процедурной генерации игр [8]. Как и в работах, описанных во втором разделе этой статьи, на данном этапе создатели VGDL делают акцент на описании простых аркадных игр. Описание игр на VGDL состоит из пяти частей:

— описания карты на основе двумерного массива ASCII-символов;
— списка сопоставлений ASCII-символов с графическими спрайтами;
— набора спрайтов с описаниями их схем движения;
— набора правил взаимодействия игровых объектов друг с другом;
— набора условий конца игры.

На данный момент ведется активная разработка языка, его спецификации изменяются и улучшаются. Кроме того, для генерации описанных на данном языке игр была реализована система PyVGDL [18], генерирующая готовые игры по их описанию.

Примером более узконаправленного подхода к представлению правил является язык Strategy Game Description Language [14], предназначенный для описания стратегических видеоигр. Авторы языка предлагают использовать для описания различных игровых объектов и правил представление на основе деревьев, в котором сложность правил увеличивается с добавлением новых вершин. Данный язык находится на начальном уровне разработки и в данный момент поддерживает только описание простых характеристик игровых объектов.

Другим примером языка для описания одного жанра игр является язык Card Game Description Language [9], предназначенный для описания карточных игр. Данный язык

описывает три основных характеристики карточных игр: стадии игры, каждая из которых характеризуется своим набором правил, ранжирование карт и их комбинаций, а также условия победы. Описание правил игр основано на контекстно-свободной грамматике; допустимые правила включают в себя скрытие и раскрытие карт, передвижение карт между разными участками игрового стола, а также действия со ставками и так далее. Авторы CGDL планируют реализовать на его основе систему процедурной генерации карточных игр.

Заключение

Данная статья посвящена обзору исследований в области процедурной генерации игровых правил и игр различных жанров. На основе данного обзора можно сделать вывод, что задача процедурной генерации игр является на сегодняшний день актуальной. Существующие экспериментальные решения ориентированы на генерацию игр нескольких отдельных жанров, а созданные с их помощью игры зачастую крайне просты и не могут конкурировать с играми, созданными человеком.

Таким образом, перспективным направлением для дальнейших исследований является улучшение существующих и создание новых алгоритмов генерации с целью повышения качества и разнообразия генерируемых игр. В дальнейшем нами планируется работа над созданием системы процедурной генерации игр, не ограниченной рамками конкретных жанров.

Литература

1. Browne C. Evolutionary Game Design / C. Browne — Berlin: Springer, 2011. — 122 p.
2. Cook, M. Aesthetic Considerations for Automated Platformer Design / M. Cook, S. Colton, A. Pease // AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. — 2012. — P. 56–63.
3. Cook, M. Automating Game Design In Three Dimensions / M. Cook, S. Colton, J. Gow // Proceedings of the AISB Symposium on AI and Games. — 2014. — P. 20–24.
4. Cook, M. Initial Results from Co-operative Co-evolution for Automated Platformer Design / M. Cook, S. Colton, J. Gow // European Conference on Applications of Evolutionary Computing. — 2012. — P. 194–203.
5. Cook, M. Ludus Ex Machina: Building A 3D Game Designer That Competes Alongside Humans / M. Cook, S. Colton // Proceedings of Fifth International Conference on Computational Creativity. — 2014. — P. 54–62.
6. Cook, M. Mechanic Miner: Reflection-Driven Game Mechanic Discovery and Level Design / M. Cook, S. Colton, A. Raad, J. Gow // Applications of Evolutionary Computation. — 2013. — P. 284–293.
7. Cook, M. Multi-Faceted Evolution Of Simple Arcade Games / M. Cook, S. Colton // IEEE Conference on Computational Intelligence and Games. — 2011. — P. 289–296.
8. Ebner, M. Towards a Video Game Description Language / M. Ebner, J. Levine, S. Lucas, T. Schaul, T. Thompson, J. Togelius // Dagstuhl Seminar on Artificial and Computational Intelligence in Games. — 2013. — P. 1–17.
9. Font, J. A Card Game Description Language / J. Font, T. Mahlmann, D. Manrique, J. Togelius // Applications of Evolutionary Computation. — 2013. — P. 254–263.

10. Hendrikx, M. Procedural Content Generation for Games: A Survey / M. Hendrikx, S. Meijer, J. Van Der Velden, A. Iosup // ACM Transactions on Multimedia Computing, Communications, and Applications. — 2013. — Vol. 9, No 1. — P. 1–22.
11. Hom, V. Automatic Design of Balanced Board Games / V. Hom, J. Marks // Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference. — 2007. — P. 25–30.
12. Koller, D. Generating and Solving Imperfect Information Games / D. Koller, A. Pfeffer // Proceedings of the 14th international joint conference on Artificial intelligence. — 1995. — P. 1185–1192.
13. Love, N. General Game Playing: Game Description Language Specification / N. Love, T. Hinrichs, M. Genesereth — Stanford: Stanford University, 2006. — 25 p.
14. Mahlmann, T. Towards Procedural Strategy Game Generation: Evolving Complementary Unit Types / T. Mahlmann, J. Togelius, G. N. Yannakakis // Applications of Evolutionary Computation. — 2011. — P. 93–102.
15. Nelson, M. Recombinable Game Mechanics for Automated Design Support / M. Nelson, M. Mateas // Proceedings of the Fourth AIIDE Conference. — 2008. — P. 84–89.
16. Orwant, J. EGGG: Automated Programming for Game Generation / J. Orwant // IBM Systems Journal — 2000. — Vol. 39, No 3. — P. 782–794.
17. Pell, B. METAGAME in Symmetric Chess-Like Games / B. Pell // Heuristic Programming in Artificial Intelligence 3: The Third Computer Olympiad. — 1992. — P. 1277–1307.
18. Shaul, T. A Video Game Description Language for Model-based or Interactive Learning / T. Shaul // Proceedings of IEEE Conference on Computational Intelligence in Games. — 2013. — P. 1–8.
19. Smith, A. Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games / A. Smith, M. Mateas // Proceedings of the IEEE Conference on Computational Intelligence and Games. — 2010. — P. 273–280.
20. Thielscher, M. A General Game Description Language for Incomplete Information Games / M. Thielscher // Proceedings of the AAAI Conference on Artificial Intelligence. — 2010. — P. 994–999.
21. Togelius, J. A Procedural Critique of Deontological Reasoning / J. Togelius // Proceedings of the 2011 DIGRA International Conference. — 2011. — P. 110–123.
22. Togelius, J. An Experiment in Automatic Game Design / J. Togelius, J. Schmidhuber // IEEE Symposium on Computational Intelligence and Games. — 2008. — P. 111–118.
23. Togelius, J. Characteristics of Generatable Games / J. Togelius, M. Nelson, A. Liapis // Proceedings of the Fifth Workshop on Procedural Content Generation in Games. — 2014. — P. 63–68.
24. Togelius, J. Procedural Content Generation in Games: A Textbook and an Overview of Current Research / J. Togelius, N. Shaker, M. Nelson — Berlin: Springer, 2014. — 142 p.
25. Togelius, J. Search-Based Procedural Content Generation: A Taxonomy and Survey / J. Togelius, G.N. Yannakakis, K.O. Stanley, C. Browne // IEEE Transactions on Computational Intelligence and AI in Games. — 2011. — Vol. 3, No 3. — P. 172–186.

Меженин Михаил Григорьевич, аспирант, преподаватель кафедры системного программирования Южно-Уральского государственного университета (Челябинск, Российская Федерация), m.mezhenin@gmail.com.

SURVEY ON PROCEDURAL GAME GENERATION

M.G. Mezhenin, South Ural State University (Chelyabinsk, Russian Federation)

Procedural content generation (PCG) is one of the most important fields of research in videogame industry. PCG allows creating different parts of games automatically; an interesting task of PCG is generating game rules and even whole games. In this paper, we present an overview of research in this field; algorithms from evolutionary computation and logic programming fields that can be used to generate game rules and games in different genres are described, as well as methods of evaluating the resulting games. We briefly describe general game-playing programs and the PCG systems that were used in them. Several representations and description languages utilized in PCG systems to encode game rules are also described.

Keywords: procedural content generation, game design, general game playing, artificial intelligence.

References

1. Browne C. Evolutionary Game Design. Berlin: Springer, 2011. 122 P.
2. Cook M. Aesthetic Considerations for Automated Platformer Design // AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 2012. P. 56–63.
3. Cook M. Automating Game Design In Three Dimensions // Proceedings of the AISB Symposium on AI and Games. 2014. P. 20–24.
4. Cook M. Initial Results from Co-operative Co-evolution for Automated Platformer Design // European Conference on Applications of Evolutionary Computing. 2012. P. 194–203.
5. Cook M. Ludus Ex Machina: Building A 3D Game Designer That Competes Alongside Humans // Proceedings of Fifth International Conference on Computational Creativity. 2014. P. 54–62.
6. Cook M. Mechanic Miner: Reflection-Driven Game Mechanic Discovery and Level Design // Applications of Evolutionary Computation. 2013. P. 284–293.
7. Cook M. Multi-Faceted Evolution of Simple Arcade Games // IEEE Conference on Computational Intelligence and Games. 2011. P. 289–296.
8. Ebner M. Towards a Video Game Description Language // Dagstuhl Seminar on Artificial and Computational Intelligence in Games. 2013. P. 1–17.
9. Font J. A Card Game Description Language // Applications of Evolutionary Computation. 2013. P. 254–263.
10. Hendrikx M. Procedural Content Generation for Games: A Survey // ACM Transactions on Multimedia Computing, Communications, and Applications. 2013. Vol. 9, No 1. P. 1–22.
11. Hom V. Automatic Design of Balanced Board Games // Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference. 2007. P. 25–30.
12. Koller D. Generating and Solving Imperfect Information Games // Proceedings of the 14th international joint conference on Artificial intelligence. 1995. P. 1185–1192.
13. Love N. General Game Playing: Game Description Language Specification / N. Love T. Hinrichs M. Genesereth Stanford: Stanford University, 2006. 25 P.
14. Mahlmann T. Towards Procedural Strategy Game Generation: Evolving Complementary Unit Types // Applications of Evolutionary Computation. 2011. P. 93–102.
15. Nelson M. Recombinable Game Mechanics for Automated Design Support // Proceedings of the Fourth AIIDE Conference. 2008. P. 84–89.

16. Orwant J. EGGG: Automated Programming for Game Generation // IBM Systems Journal. 2000. Vol. 39, No 3. P. 782–794.
17. Pell B. METAGAME in Symmetric Chess-Like Games // Heuristic Programming in Artificial Intelligence 3: The Third Computer Olympiad. 1992. P. 1277–1307.
18. Shaul T. A Video Game Description Language for Model-based or Interactive Learning // Proceedings of IEEE Conference on Computational Intelligence in Games. 2013. P. 1–8.
19. Smith A. Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games // Proceedings of the IEEE Conference on Computational Intelligence and Games. 2010. P. 273–280.
20. Thielscher M. A General Game Description Language for Incomplete Information Games // Proceedings of the AAAI Conference on Artificial Intelligence. 2010. P. 994–999.
21. Togelius J. A Procedural Critique of Deontological Reasoning // Proceedings of the 2011 DIGRA International Conference. 2011. P. 110–123.
22. Togelius J. An Experiment in Automatic Game Design // IEEE Symposium on Computational Intelligence and Games. 2008. P. 111–118.
23. Togelius J. Characteristics of Generatable Games // Proceedings of the Fifth Workshop on Procedural Content Generation in Games. 2014. P. 63–68.
24. Togelius J. Procedural Content Generation in Games: A Textbook and an Overview of Current Research / J. Togelius N. Shaker M. Nelson Berlin: Springer, 2014. 142 P.
25. Togelius J. Search-Based Procedural Content Generation: A Taxonomy and Survey // IEEE Transactions on Computational Intelligence and AI in Games. 2011. Vol. 3, No 3. P. 172–186.