УДК 004.434+004.4'41: 004.896

ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЙ ЯЗЫК ДЛЯ ОПИСАНИЯ МОДЕЛИ ПОВЕДЕНИЯ КОМПЬЮТЕРНОГО ОППОНЕНТА (БОТА)

Статья поступила в редакцию 16.11. 2014, в окончательном варианте 07.03. 2015

Ильин Дмитрий Юрьевич, аспирант, Московский государственный университет приборостроения и информатики, 107996, Российская Федерация, г. Москва, ул. Стромынка, 20, e-mail: serendine@gmail.com

Корягин Сергей Викторович, кандидат технических наук, доцент, Московский государственный университет приборостроения и информатики, 107996, Российская Федерация, г. Москва, ул. Стромынка, 20, e-mail: dongenealog2003@mail.ru

Рассмотрены существующие варианты реализации проблемно-ориентированных транслирующих средств в рамках разработки моделей поведения компьютерных оппонентов в игровых приложениях. По теме статьи представлен сравнительный анализ существующих программных решений. На основе анализа их достоинств и недостатков сформированы требования к разработке программного средства и определена его целевая аудитория. Описаны предложенные авторами проблемно-ориентированный язык и транслирующее средство для него. С использованием этого языка, позволяющего в формализованном виде описать модели поведения компьютерного оппонента, программно реализуется его взаимодействие с игровым процессом и игроком. Язык предусматривает возможность использования как готовых алгоритмов, так и написания собственных – для исключения шаблонности поведения компьютерного оппонента в процессе игры. Рассмотрен набор встроенных в разработанный язык алгоритмов, а также математический аппарат, обеспечивающий их реализацию. Рассмотрены примеры построения пользовательских алгоритмов, иллюстрирующих функциональные возможности предложенного авторами проблемно-ориентированного языка.

Ключевые слова: игровые приложения, компьютерные оппоненты, методы трансляции, проблемно-ориентированные языки программирования, алгоритмы поиска кратчайшего пути, баллистика, искусственный интеллект, игры для программистов

PROBLEM-ORIENTED PROGRAMMING LANGUAGE FOR ARTIFICIAL GAME OPPONENT BEHAVIOR MODELING

Ilin Dmitriy Yu., post-graduate student, Moscow State University of Instrument Engineering and Computer Science, 20 Stromynka St., Moscow, 107996, Russian Federation, e-mail: serendine@gmail.com

Koryagin Sergey V., C.Sc. (Engineering), Associate Professor, Moscow State University of Instrument Engineering and Computer Science, 20 Stromynka St., Moscow, 107996, Russian Federation, e-mail: dongenealog2003@mail.ru

This article discusses the options for implementing problem-oriented translation media for artificial game opponent behavior modeling in the computer game applications. Existing software solutions have been reviewed and analyzed according to the topic. Solution requirements have been formed based on the analysis results, as well as target audience has been chosen. In order to achieve the point, problem-oriented programming language has been developed as well as interpreting software for this language within picked technology stack. According to the formal description of the computer opponent behavior the software performs interaction between computer opponent, game process and end user. The software provides possibility of usage of predefined algorithms and of user-defined as well, to keep variety of computer opponent behavior cases. List of available features has been shown in the article along with encapsulated mathematical part.

Examples of user-defined algorithms were developed based on these features and reviewed in case of better illustration of problem-oriented language usage.

Keywords: gaming applications, artificial computer opponents, translation methods, problem-oriented programming languages, shortest path search algorithms, ballistics, artificial intelligence, games for programmers

Введение. Игровые приложения (ИП) имеют достаточно высокую популярность на рынке программного обеспечения и коммерческую значимость. Однако только малая доля ИП предоставляет возможности внесения изменений в поведение игровых единиц. Поэтому целесообразна разработка специальной программной составляющей, которая бы позволила конечному пользователю вносить свои изменения в модель поведения компьютерного оппонента. Такой поход позволит охватить новый сегмент целевой аудитории, в т.ч. для целей обучения элементам программирования и конструирования интерактивных игровых подходов. В свою очередь, это облегчит ход обучения, снизит рутинность его процесса, позволит обучающимся проявлять творческий подход. Питер Норвиг пишет в своей статье: «Заинтересуйтесь программированием и занимайтесь им ради развлечения. Старайтесь сделать эти занятия достаточно привлекательными для того, чтобы не хотелось их бросать в течение десяти лет» [13]. Его высказывание удачно иллюстрирует тот факт, что предлагаемая разработка может быть эффективно использована в образовательных целях.

Таким образом, целью работы является создание специализированного «событийного» проблемно-ориентированного языка (ПОЯ) для управления поведением компьютерного оппонента, а также интерпретирующего (транслирующего) средства для этого языка.

Общая характеристика проблематики статьи. Сфера создания ИП является достаточно разносторонней и требует от разработчиков решения ряда задач, которые бы позволили конечному пользователю получить должное удовлетворение от игрового процесса [7]. В большинстве случаев перед разработчиками игровых приложений стоит задача реализации искусственного интеллекта (ИИ) в той или иной его форме.

Для компьютерных ботов ИИ в общем случае можно классифицировать на три основных категории: скриптовый ИИ (СИИ); системы, основанные на правилах; рассуждения, основанные на прецедентах.

В данной статье рассматривается только СИИ, т.к. он предпочтителен с точки зрения объемов потребления вычислительных ресурсов и достаточно прост в реализации. Чтобы расширить его функциональность, в ИП имеет смысл добавить встроенный ПОЯ – для возможности написания программистом своей собственной модели ИИ игровых ботов или изменения уже существующего готового варианта (алгоритма).

Сравнительная характеристика существующих решений. На текущий момент существует ряд ИП, целевой аудиторией которых являются программисты или лица, обучающиеся программированию. Для того чтобы сузить рассматриваемую область, положим, что ИП должно быть в жанре боевых ботов. Таким образом, круг актуальных решений на рынке ссужается, по мнению авторов статьи, до двух: Robocode [11] (рис. 1) и Fightcode [6] (рис. 2).

ИП Robocode является кроссплатформенным десктоп-приложением, что расширяет аудиторию потенциальных пользователей за счет возможности использования различных операционных систем (ОС).

Как видно из рисунка 2, ИП Fightcode – это онлайновый сервис, что имеет свои плюсы и минусы. К плюсам можно отнести доступность этого сервиса на любой ОС, в которой присутствует браузер. К минусам – вероятность того, что ИП может потерять доступность в связи с нарушениями в работе сервиса (сайта в Интернете) или канала связи пользователя с сетью Интернет.

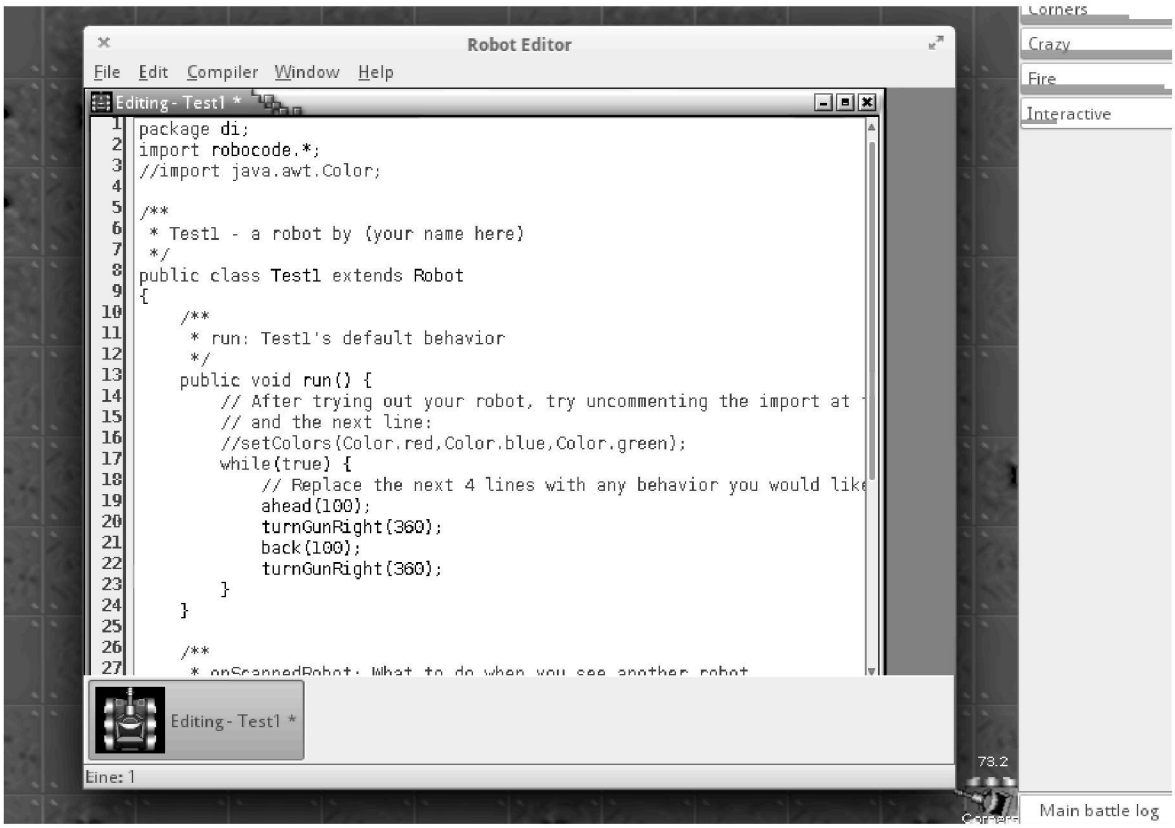

Рис. 1. Пример экрана редактирования кода в ИП Robocode

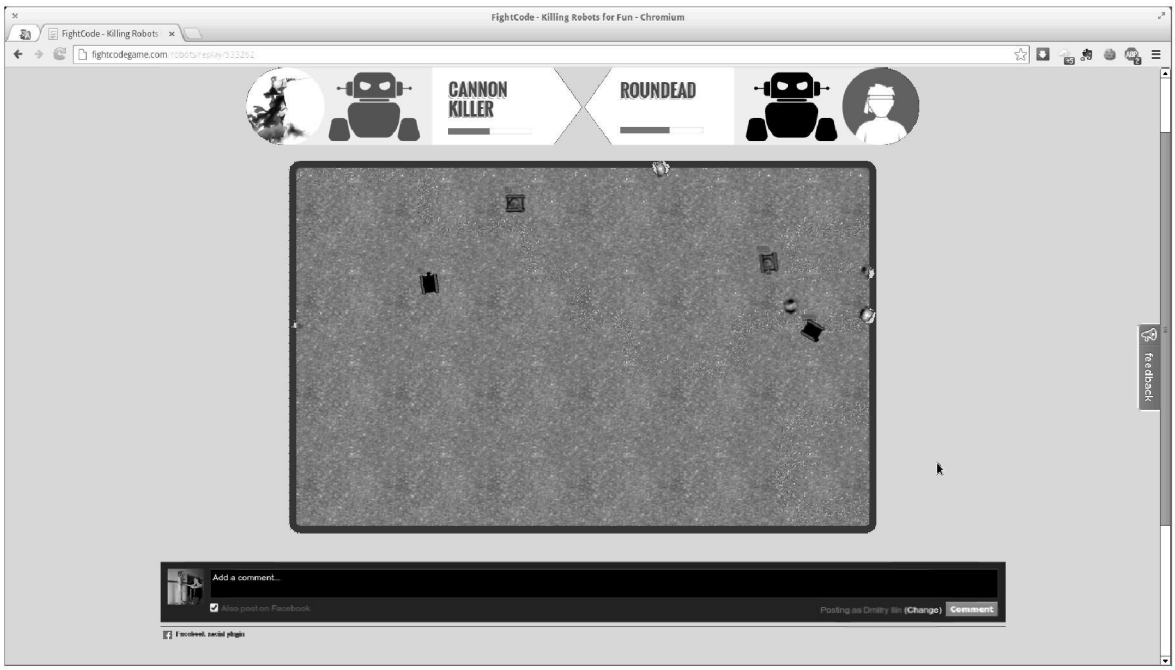

Рис. 2. Пример экрана игрового процесса в приложении Fightcode

Оба ИП предоставляют возможности только 2D визуализации. Поэтому программирование перемещений объектов происходит только на плоскости, а не в пространстве. Это упрощает разработку в рамках ИП, но заметно снижает качество их визуальной составляющей. Такая ситуация ограничивает целевую аудиторию для ИП. Так как сторонники качественных визуальных эффектов не смогут воспринять идею игр в упрощенном визуальном дизайне. Это также негативно сказывается на разнообразии возможных действий, доступных при описании модели поведения компьютерных оппонентов. Данные ИП можно отнести к разряду так называемых Zero-игр. Этим понятием обозначают ИП, участие игрока в которых сведено к минимуму. То есть сам игровой процесс напрямую не зависит от действий игрока. В данном случае все взаимодействие игрока с ИП ограничено написанием исходного кода для игровых ботов.

Оба ИП используют языки общего назначения – это имеет и плюсы, и минусы. Robocode использует синтаксис Java (см. рис. 1), который не проектировался под конкретную предметную область. Как следствие, возможности языка в ряде случаев оказываются избыточными. В ИП Fightcode используется синтаксис JavaScript. При этом среда исполнения ограничивается веб-браузерами либо замещающими веб-контейнерами в приложении для ОС [6]. Помимо этого, оба языка имеют низкий уровень абстракции по отношению к предметной области.

Событийный подход в ИП Fightcode реализован в виде методов прототипа объекта, представляющего бота. Это в определенной степени неудобно, потому что события никак не отличаются визуально и технически от каких-либо других методов, которые могут присутствовать в объекте, а различие осуществляется только на уровне договоренности.

Минусы Java в применении для ИП в том, что необходима компиляция исходного кода программы. Это может требовать дополнительных временны*х* расходов, которые будут мешать игровому процессу на компьютерах с низким быстродействием. Необходимость в больших вычислительных ресурсах в контексте сферы применения, рассматриваемой в настоящей статье, возникает не столь часто. Поэтому наиболее целесообразно использовать интерпретируемый язык.

Характеристика предложенного решения. Рассмотрев существующие программные решения, сравнив их достоинства и недостатки, авторы сделали вывод о целесообразности разработки нового (собственного) решения, не имеющего описанных выше проблем, но сохраняющего положительные качества существующих ИП. Такое решение должно быть сравнительно компактным с точки зрения синтаксиса; учитывать специфику предметной области – чтобы снизить «порог вхождения» для конечных пользователей-программистов.

При этом авторы статьи ориентируются на «парные боевые игры» («человек-игрок» против компьютерного оппонента) и не рассматривают групповые игры, в которых группа управляемых объектов должна взаимодействовать друг с другом. Это также отличает предложенное решение от конкурирующих ИП, в которых человек-игрок создает «программу» для управления игровым объектом, а затем эта программа «сражается» с компьютерным оппонентом без участия игрока.

Игровой процесс для «парной боевой игры» разделен на две основные составляющие: разработку игроком модели поведения компьютерного оппонента и непосредственное состязание с ним человека-игрока. Критерием победы при этом будет являться нанесение «повреждений» оппоненту равных или больших по сумме количеству «брони» у его игровой единицы. Возможность изменения модели поведения бота через редактирование его исходного кода позволит гибко настроить сложность состязания.

«Игровой движок» Unity3D целесообразно использовать как основную платформу [9]. Данный «движок» позволит добиться кроссплатформенности, а в будущем – портировать разработанное приложение на мобильные платформы. Целевыми платформами в рамках рас-

сматриваемых задач будут ОС Windows, Linux и Mac OS X, поддержка которых уже заложена в указанный игровой движок.

Для реализации поставленной задачи необходимо было разработать ПОЯ [3] и специализированное транслирующее средство [4] – с целью упрощения процесса написания (разработки) модели поведения компьютерного оппонента. Язык был разработан для ИП, основными управляемыми игровыми единицами (объектами) которого являются танки. В случае необходимости, имеется возможность применить предлагаемый подход и языковое средство для иных аналогичных приложений - с доработкой функциональности в зависимости от требований к конечному программному продукту.

Формальное описание предлагаемого ПОЯ, характеризующего поведение одного конкретного компьютерного оппонента (бота), имеет такой вид.

```
tank <название танка>;
begin
   var:
      {<имя переменной> = <выражение>;}
   info:
      {<имя параметра танка> = <выражение>;}
   functions:
      {<имя функции>(<параметры функции>):
         <тело функции>
      next;}
   events:
      {<имя события>:
         <набор вызываемых функций>
      next;}
   advices:
      {advice:
         joinPoint: <момент выполнения>;
         pointcut: <список названий функций>;
         actions:
            <набор вызываемых функций и выражений>
      next;}
        end.
```

Синтаксис рассматриваемого ПОЯ использует элементы, характерные для таких распространенных языков как «Си» и «Паскаль». Это снижает трудозатраты при освоении языка, уменьшает количество потенциальных ошибок при написании кода.

Предлагаемая структура использует следующие терминальные символы:

tank – начало программы, где пользователь задает название танка;

begin и **end**– начало и конец описательного раздела;

var – блок глобальных переменных;

info – блок параметров танка;

functions – блок функций, их входные параметры и набор входящих в них операций;

events – блок обработчиков событий;

advices – блок советов, применяемых при выполнении функций;

advice – начало совета;

joinPoint – блок, где указывается, в какой момент будет применен совет;

pointcut – блок, где перечисляются функции, для которых будет применен совет;
actions – блок, содержащий набор действий, выполняемых в рамках совета;
next – конец повторяющегося составного блока, такого как событие, совет, функция.

Каждый раздел и составной блок отделяется от содержимого символом «:». Правая и левая части отделяются знаком «=». Используются только целые числа, латинский алфавит и десятичная система счисления.

Рассматриваемый ПОЯ предоставляет возможности выполнения разработки с применением следующих парадигм программирования: императивная; процедурная; событийно-ориентированная; аспектно-ориентированная.

Реализация транслирующего средства для созданных описаний компьютерных оппонентов предполагается в соответствии с принципами построения однопроходных трансляторов [1, 2, 5].

Рассмотрим предоставляемый конечному пользователю функционал более подробно.

В игровом процессе имеет место следующий набор событий, на которые можно объявлять обработчики.

1. Start – срабатывает сразу после начала игры.
2. EachSecond – срабатывает каждую секунду.
3. Hit – срабатывает после попадания снаряда в танк.
4. Aimed – срабатывает, как только орудие танка было наведено на цель с помощью функции Aim.

Список функций, отвечающих за движение компьютерного оппонента.

1. MoveForward – двигаться вперед (на входе –«время в секундах»).
2. MoveBackward – двигаться назад (на входе – «время в секундах»).
3. TurnLeft – поворот влево (на входе – «время в секундах»); угловая скорость поворота зависит напрямую от параметра TankRotationSpeed.
4. TurnRight – поворот вправо (на входе – «время в секундах»); угловая скорость поворота зависит напрямую от параметра TankRotationSpeed.
5. Aim – навести орудие на цель (нет входных параметров).
6. Fire – произвести выстрел из орудия (нет входных параметров).
7. MoveToPoint – перемещение в точку на карте (на входе координаты X/Y, в пределах от 0 до 79).
8. IntellectualMoveToPoint – перемещение в точку на карте с учетом наличия препятствий на виртуальной местности (на входе - координаты X/Y, в пределах от 0 до 79).

Список функций для обзора текущей ситуации.

1. CheckPathAhead – проверить наличие препятствий впереди (на входе – дистанция; на выходе «0» если препятствий нет или «1» если препятствия в пределах дистанции есть).
2. CheckPathBehind – проверить наличие препятствий сзади (на входе – дистанция; на выходе «0» если препятствий нет или «1» если препятствия в пределах дистанции есть).
3. CurrentArmor – текущее состояние брони танка (на входе нет параметров).

Список общих функций.

1. Print – печать списка значений (неограниченное количество параметров на входе).
2. Time – количество секунд, с момента начала игры (на входе нет параметров).

Список параметров танка, который доступен конечному пользователю ИП.

1. TankMaxSpeed – максимальная скорость, которую может развить танк при прямолинейном движении.
2. TankAcceleration – максимальное ускорение, которое доступно танку при прямолинейном движении.
3. TankRotationSpeed – максимальная скорость поворота корпуса танка.

4. TankArmor – начальное количество единиц брони танка, определяющее количество попаданий, которое танк может выдержать (это примерно соответствует количеству слоев «пассивной защиты»). Различие в толщине брони для различных частей корпуса танка мы не учитываем.

5. TankDamage – урон, который наносит попадание в танк.

6. TurretRotationSpeed – максимальная скорость поворота башни танка по горизонтали.

7. CannonMoveSpeed – максимальная скорость перемещения ствола орудия танка по вертикали.

Отсутствие объявления одного или нескольких из перечисленных параметров не приведет к сбоям в работе программы. Так как в этом случае будут использоваться значения по умолчанию.

Помимо этого, в ПОЯ встроен набор наиболее распространенных математических функций, необходимых для выполнения расчетов по описываемым действиям. В частности применяются функции для получения остатка от деления (встроенная функция **MOD**); вычисления sin и arcsin.

По мнению авторов статьи, приведенный набор функций обладает свойством «необходимости и достаточности».

В отличие от рассмотренных аналогов, предложенное решение имеет три измерения вместо двух. В связи с этим возникает необходимость наведения орудия танка на цель с учетом траектории полета снаряда в трехмерном пространстве. Для наиболее правдоподобной имитации компьютерным оппонентом поведения экипажа танка-противника в функцию Aim заложен алгоритм наведения на цель, работающий по аналогии с человеческим мышлением. Для этого вместо точных математических формул, которые бы позволили идеально вычислить угол, необходимый для попадания в танк оппонента, используются приближенные значения, полученные с помощью функции «угла достижения» [8]:

$$\varphi = 0{,}5\arcsin\left(gd \div v^2\right),$$

где φ – рекомендуемый угол наклона относительно горизонтальной плоскости, вычисляемый с допустимой погрешностью; d – дистанция между целью и стартовой позицией полета снаряда; g – ускорение свободного падения, равное 9.81 м/с^2 (в рамках описываемой задачи); v – скорость полета снаряда (м/с).

Таким образом, сопротивление воздуха при полете снаряда не учитывается. Это позволяет обойтись при расчете траектории без использования сложных формул или численных методов, реализующих конечно-разностные представления дифференциальных уравнений движения.

Функция «угла достижения» является обратной к функции для расчета расстояния полета снаряда, выпущенного по баллистической траектории на плоскости. Такое алгоритмическое решение следует отнести к задаче внешней баллистики. Под ней понимается наука о движении тел в воздушном и безвоздушном пространстве под действием только внешних сил. Слово «внешний» в данном термине происходит от рассмотрения движения артиллерийского снаряда вне орудийного ствола. Существует также понятие терминальной (конечной) баллистики, имеющей отношение к взаимодействию снаряда и объекта, в которое он попадает; движению снаряда после попадания, в т.ч. и в случае рикошетов. Однако в контексте данной задачи вопросы терминальной баллистики рассматриваться не будут.

Стоит обратить внимание, что в язык встроены две функции, отвечающие за перемещение по координатной сетке: MoveToPoint; IntellectualMoveToPoint.

Первая функция представляет из себя обычное перемещение по кратчайшему расстоянию, вторая – учитывает наличие препятствий и выстраивает кратчайший путь с учетом «непроходимых» клеток на координатной сетке. Так как в рамках игрового приложения присутст-

вуют препятствия, например между координатами, как показано на рисунке 3 (20, 60) и (20, 70), то разумно применение функции **IntellectualMoveToPoint**. Она не требует дополнительных усилий от конечного пользователя, предоставляя готовый алгоритм решения класса задач по поиску кратчайшего расстояния перемещения с учетом препятствий на неориентированном графе единой стоимости [12]. В текущей реализации языка в данную функцию заложен алгоритм Jump Point Search, являющийся разновидностью алгоритма A* [10]. Он является оптимальным на данный момент для ИП в жанре боевых ботов [14]. В случае необходимости, заложенный алгоритм может быть заменен любым другим, сохраняя при этом полную совместимость написанного ранее пользователем исходного кода игрового оппонента.

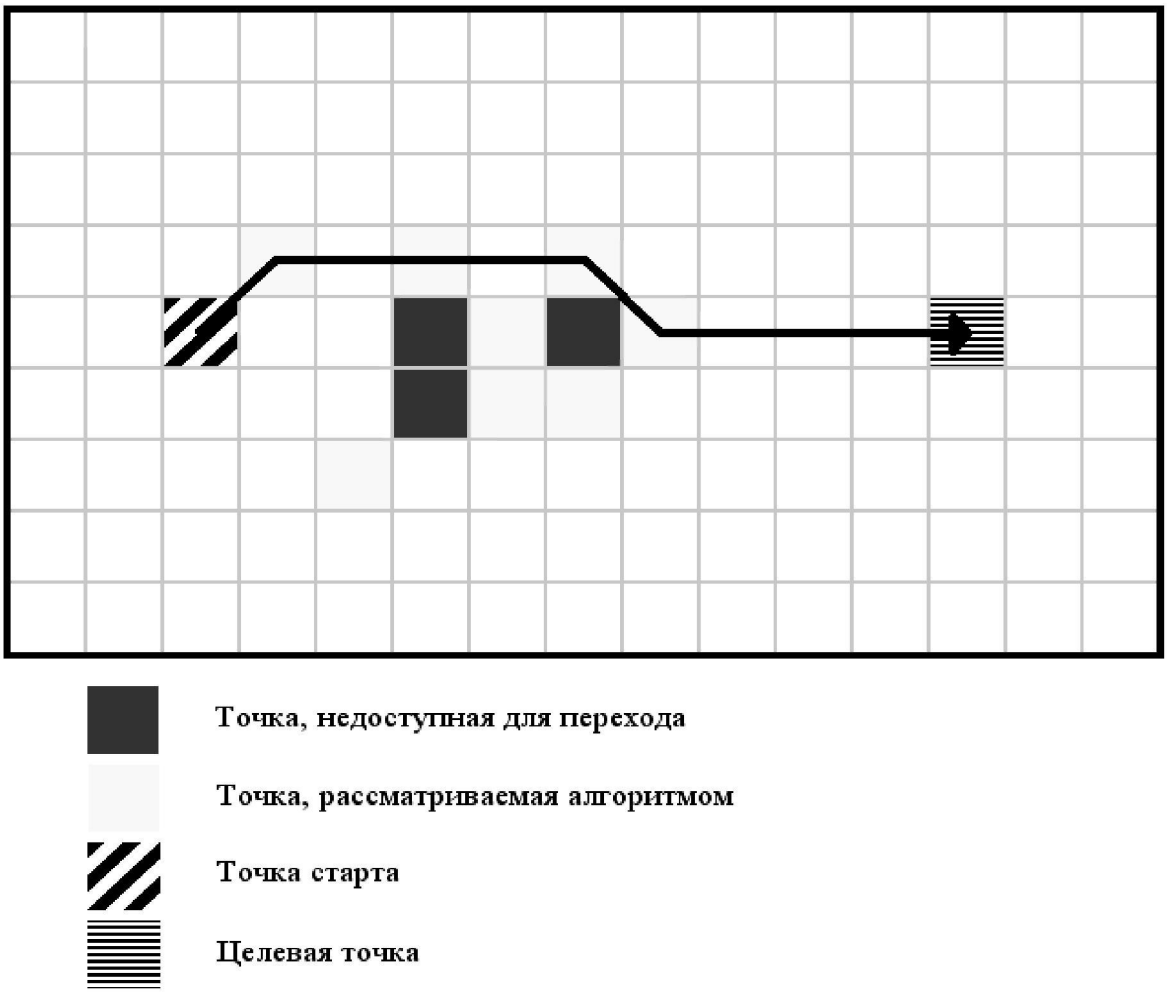

Рис. 3. Схематичное представление работы алгоритма Jump Point Search

Следует пояснить, какие именно точки рассматриваются алгоритмом, как указано в легенде на рисунке 3, что и отличает его от алгоритма A*. Это так называемые «прыжковые точки» (ПТ), которые позволяют ускорить алгоритм поиска пути, за счет рассмотрения только «необходимых» точек. ПТ могут быть описаны двумя правилами выбора соседей при рекурсивном поиске: одно правило для прямолинейного движения и другое – для диагонального. В обоих случаях необходимо доказать, что при исключении из набора ближайших

соседей вокруг ПТ, найдется оптимальный путь из предка ПТ до каждого из соседей, и этот путь не будет содержать в себе посещенную точку [10].

Предложенный ПОЯ является регистрозависимым, а встроенные математические функции записываются в верхнем регистре. Именование прочих встроенных функций придерживается стандарта Upper Camel Case. То есть первая буква каждого слова, входящего в название функции, записывается в верхнем регистре, а все остальные – в нижнем. Параметры игровой единицы (в данном случае – танка) записываются также в соответствии со стандартом Upper Camel Case.

Модификаторы области видимости переменных или же маркировка статичности с целью сохранения состояния между вызовами функции в предложенном ПОЯ не предусмотрены. Присутствуют две основные области видимости: глобальная и локальная (в рамках одной функции). Первая применяется для хранения состояний, вторая – для локальных вычислений. Этого достаточно для того, чтобы охватить весь спектр задач. Такой подход позволяет сохранить простоту синтаксиса – за счет снижения количества альтернатив реализации задач.

В ПОЯ присутствует только целочисленный тип данных, так как этого достаточно для работы с основными параметрами игровых единиц. Булевы значения эквивалентны значениям «1» и «0» соответственно. Подобную реализацию булевых значений можно встретить в языке «С», однако в нем также существуют константы, соответствующие булевым значениям. Предлагаемый ПОЯ не предоставляет возможности объявления констант, так как сложность результирующих программ на нем предполагается небольшой.

Транслирующее средство реализовано с использованием нисходящего метода разбора, с последовательностью разбора слева-направо и с применением возвратов [7]. Механизм работы распознавателя строится на основе подбора альтернатив. Интерпретатор разработан с применением метода рекурсивного спуска. Такой метод позволяет наиболее удачно реализовать отладочные сообщения об ошибках для конечного пользователя, так как разбор будет идти строго последовательно. Подобный парсер может потребовать экспоненциального времени работы и не всегда гарантирует ее завершение (что зависит от грамматики), но в контексте данной предметной области это не окажет критического влияния на конечный программный продукт [9].

Процесс трансляции исходного кода разработанного игрового оппонента разделен на два этапа.

1. Разбор входной строки, инициализация глобальных переменных и выделение функций, советов и обработчиков событий в отдельные цепочки лексем.

2. Разбор отдельных цепочек лексем – по наступлению тех или иных событий в рамках игрового процесса.

Данный подход позволяет обращаться к каждой из цепочек в отдельности, сокращая тем самым время, требуемое для исполнения. Такой подход может снизить производительность при больших объемах исходного кода, но в отношении возможностей для предметной области он соответствует желаемым результатам.

Таким образом, предлагаемое решение (ПОЯ + средство трансляции) имеет ряд преимуществ перед существующими решениями.

1. Отсутствие требования по наличию сетевого подключения, т.е. ИП может использоваться на автономной ПЭВМ.

2. Наличие 3D графики и алгоритмов, соответствующих трехмерному пространству.

3. Сниженный «порог вхождения» при освоении предложенного ПОЯ.

4. Потенциальная возможность к портированию разработанного программного средства на мобильные устройства.

Для «человека-игрока» предусматриваются те же функциональные возможности управления игровыми объектами в реальном времени, что и для «компьютерного оппонента». При этом «органами управления» могут быть клавиатура, манипулятор мышь, джойстик и пр.

Примеры кода, написанного на разработанном ПОЯ. В качестве первого примера рассмотрим элементы, относящиеся к описанию перемещений компьютерного бота. Приложение (игровая программа) подразумевает перемещение игровых единиц на плоскости, содержащие некоторые непреодолимые препятствия, которые танк может обойти. Игровое поле разделено на квадраты и представляет собой сетку размерностью 80x80 квадратов. Масштаб задается ИП и его характеристиками. Основная задача языкового средства в данном аспекте – облегчить реализацию перемещений объекта на игровом поле с учетом препятствий, предоставив конечному пользователю, имеющему начальный уровень знаний в программировании, должный уровень абстракции. Пример текста исходного кода показан на рисунке 4

```
tank Mover;
begin
   var:
      step = 5;
   info:
      TankMaxSpeed = 40;
      TankAcceleration = 40;
   functions:
      func():
         step = - step;
         way = 65;
         x = 20;
         y = 65 + step;
         Print(y);
         IntellectualMoveToPoint(x, y);
      next;
   events:
      Start:
         func();
      next;
      PointReached:
         func();
      next;
      Aimed:
         Fire();
      next;
   advices:
      advice:
         joinPoint: before;
         pointcut: func;
         actions:
            Aim();
         next;
end.
```

Рис. 4. Пример записи кода программы

В тексте программы данного примера описано следующее:

2. Объявляется название бота для целей приложения.

3. Задается глобальная переменная step.

4. Задаются параметры игрового оппонента, такие как максимальная скорость и ускорение.

5. Объявляется функция **func**, в теле которой и описывается перемещение игровой единицы.

6. Объявляются обработчики событий начала игрового процесса и на событие достижения точки на карте.

7. Объявляется обработчик на событие успешного наведения на цель.

8. Объявляется совет, который вызывает функцию наведения орудия на цель после выполнения функции **func.**

Подробнее стоит остановиться на теле функции **func**, чтобы рассмотреть каким образом осуществляется перемещение. После каждого вызова этой функции происходит изменение знака у глобальной переменной **step.** Затем пользователем, на основании произвольного выбора в рамках ограничений координатной сетки, выбирается точка достижения, имеющая координату *X* всегда равную 20 и координату *Y,* постоянно изменяющую свое значение на 60 и 70 (на основе знака глобальной переменной **step**). Таким образом, игровой единице попеременно задаются начальная и конечная точки на координатной сетке – по аналогии с тем, как показано на рисунке 3.

Благодаря использованию рассмотренной ранее функции **IntellectualMoveToPoint** компьютерный оппонент обходит присутствующие на его пути препятствия.

Данный пример иллюстрирует как общие принципы языкового средства, так и заложенный в него механизм перемещения компьютерного оппонента по виртуальной местности. Описанный механизм значительно облегчает решение типовых задач по поиску оптимального пути для перемещения между двумя точками, которые бы пришлось решать конечному пользователю.

В качестве второго примера рассмотрим элементы, касающиеся уже непосредственно взаимодействия игрока и смоделированного им компьютерного оппонента. На примере ИП, основными единицами которого являются танки, основное взаимодействие будет построено вокруг стремления попасть снарядом в танк своего оппонента и не дать оппоненту попасть в собственный танк. На основе этой идеи возможна разработка различных моделей поведения, например описываемых алгоритмом типа показанного на рисунке 5.

```
tank MainTestBot;

begin
   var:
      tankDamage = 300;
      tankArmor = 2000;
   info:
      TankDamage = tankDamage;
      TankArmor = tankArmor;
      TankMaxSpeed = 20;
      TankAcceleration = 14;
      TurretRotationSpeed = 100;
   functions:
      Overrun():
```

```
        check = CheckPathAhead(10);
        true = 1;
        if (check ! true)
           MoveForward(12);
        endif;
        TurnRight(8);
   next;
   Counter():
      time = Time();
      mod = MOD(time, 10);
      zero = 0;
      if (mod = zero)
         Aim();
      endif;
      mod = MOD(time, 25);
      if (mod = zero)
         Overrun();
      endif;
   next;
   events:
      Start:
         Overrun();
      next;
      Aimed:
         Fire();
         MoveBackward(4);
      next;
      EachSecond:
         Counter();
      next;
end.
```

Рис. 5. Пример записи алгоритма

В данном примере описывается следующее:

1. Объявляется название бота для целей приложения.
2. Задаются глобальные переменные **tankDamage** и **tankArmor.**
3. Задаются параметры игрового оппонента, такие как урон, наносимый выстрелами танка-противника, единицы жизнеспособности, максимальная скорость, ускорение и скорость вращения башни.
4. Объявляется функция **Overrun**, в теле которой описан процесс движения танка-противника вперед с поворотом направо, при отсутствии препятствий в направлении движения.
5. Объявляется функция **Counter**, используемая как контейнер для вызова действий с определенными промежутками времени.
6. Объявляются обработчики событий начала игрового процесса, на событие успешного наведения орудия на цель и на ежесекундный отсчет таймера.

Приведем «скриншоты», соответствующие последовательным этапам игры, сделанные на основе записанного на видео игрового процесса (рис. 6).

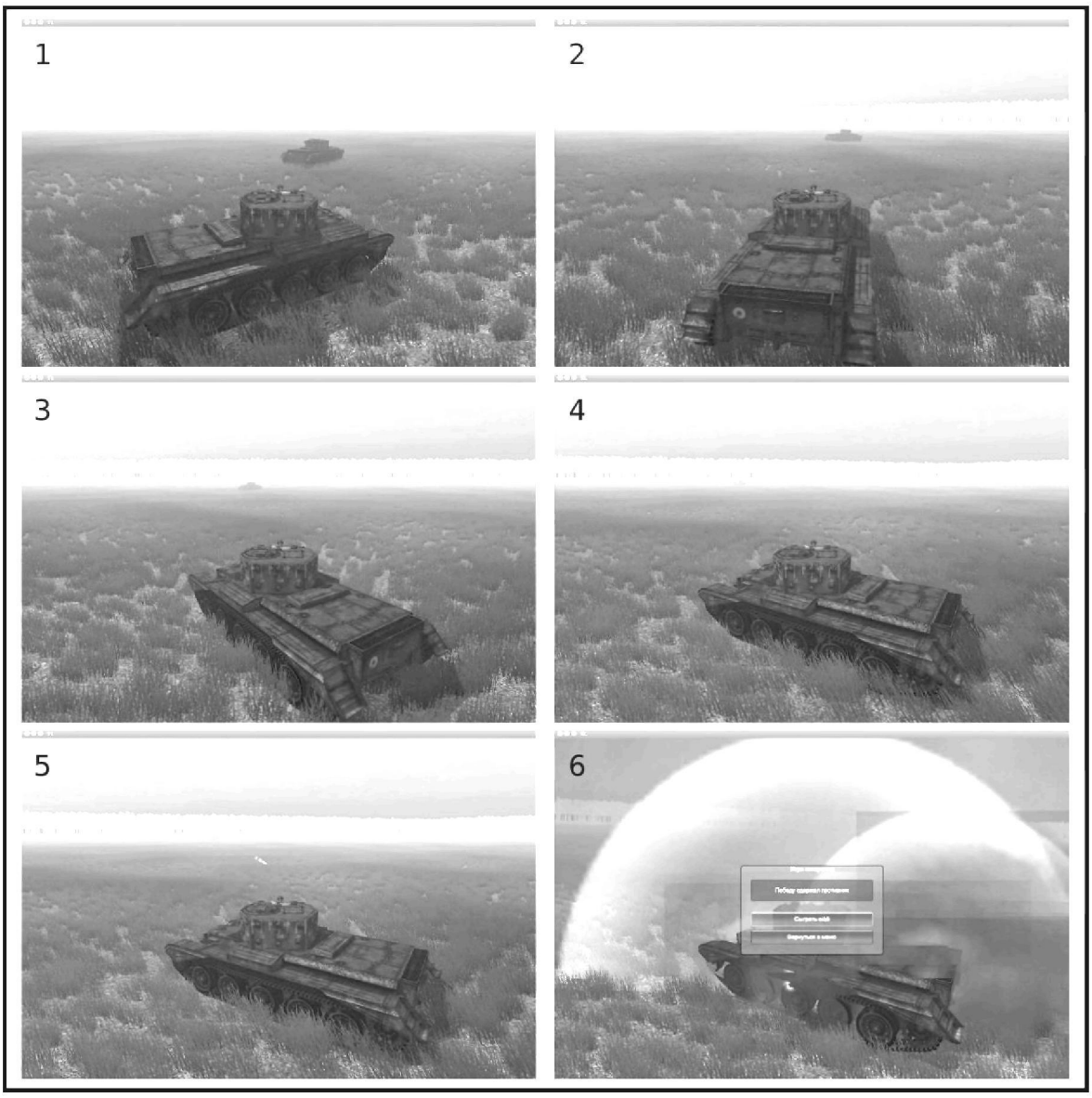

Рис. 6. Последовательность этапов выполнения функции Counter

Как можно видеть на рисунке 6, компьютерный оппонент выполняет маневр, увеличивая расстояние до игрока, после чего осуществляет выстрел из орудия; выстрел достигает игрока и попадание оказывается достаточным, чтобы снизить количество единиц «брони» игрока до 0. Это иллюстрирует то, как модель игрового оппонента может взаимодействовать с человеком, реализуя тем самым состязательный этап игрового процесса.

Для более детального понимания принципов работы ПОЯ рассмотрим каждую из функций приведенного примера в отдельности.

Функция **Counter** в данном примере выполняет два основных действия: наведение на цель и запуск функции **Overrun**. Эти действия осуществляются на основании свободного выбора пользователя, разрабатывающего ИИ игрового оппонента, с промежутками в 10 и 25 сек.

соответственно. В отношении тела функции стоит отметить, что в ПОЯ реализована функция **Time**, отражающая количество секунд с момента запуска игрового процесса.

В рамках функции **Overrun** осуществляется проверка наличия препятствий непосредственно перед игровой единицей – при этом дистанция проверки ограничивается 10 условными единицами. В зависимости от целей, поставленных конечным пользователем, это значение может быть изменено в б*о*льшую или меньшую сторону. Дальнейшие действия в рамках функции сравнительно тривиальны, а их продолжительность задана пользователем на основании свободного выбора:

1. Проверка условия наступления каждой 25-ой секунды на истинность.
2. Запуск движения вперед продолжительностью 12 секунд.
3. Поворот корпуса игровой единицы направо продолжительностью 8 секунд.

Прочие функции, отвечающие за перемещение танка, работают аналогичным образом, принимая в качестве аргумента продолжительность действий в секундах. Перемещение компьютерного оппонента осуществляется по аналогии с возможностями, предоставленными пользователю.

Все переменные, объявленные в рамках функции **Overrun**, а также аналогичным образом в других функциях, являются локальными и не будут доступны извне этой функции.

Выводы. В результате проведенной работы были созданы ПОЯ программирования и специализированное интерпретирующее средство, соответствующее этому языку. В ПОЯ заложен необходимый функционал, требуемый для игрового процесса, а также встроены функции, отсутствующие у разработок-конкурентов.

За счет исключения объектно-ориентированной парадигмы программирования, снижения числа возможных конструкций в языке, а также уменьшения числа областей видимости удалось добиться значительно более простого синтаксиса ПОЯ. Конечное приложение, для которого был разработан интерпретатор, благодаря адекватному выбору технологий сохранило качество кроссплатформенности, при этом возможность его использования не зависит от наличия сетевого подключения. Имеется возможность дальнейшего развития ИП под мобильные платформы – с доработками, соответствующими техническим особенностям портативных устройств.

За счет достаточно низкого «порога вхождения» применение описанной разработки возможно в качестве инструмента для интерактивного обучения программированию, а также в качестве ИП для программистов.

Список литературы

1. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технология и инструменты / А. Ахо, Р. Сети, Д. Ульман. – Москва : Издательский дом «Вильямс», 2011. – 768 с.

2. Ахо А. Структуры данных и алгоритмы / А. Ахо, Д. Хопкрофт, Д. Ульман.– Москва : Вильямс, 2010. – 400 с.

3. Корягин С. В. Перекодировщик языков непрерывного моделирования / С. В. Корягин // Промышленные АСУ и контроллеры. – 2013. – № 10. – С. 52–56.

4. Молчанов А. Ю. Системное программное обеспечение : учебник для вузов / А. Ю. Молчанов. – 3-е изд. – Санкт-Петербург : Издательский дом «Питер», 2010. – 400 с.

5. Свердлов С. З. Языки программирования и методы трансляции : учебное пособие / С. З. Свердлов. – Санкт-Петербург : Издательский дом «Питер», 2007. – 638 с.

6. Сименко И. FightCode: Танковые войны на JavaScript / И. Сименко. – Режим доступа: http://habrahabr.ru/post/174461/, свободный. – Заглавие с экрана. – Яз. рус. (дата обращения: 03.09.2014).

7. Шампандар А. Искусственный интеллект в компьютерных играх: как обучить виртуальные персонажи реагировать на внешние воздействия / А. Шампандар.– Москва : Издательский дом «Вильямс», 2007. – 768 с.

8. Akcay M. Development of Universal Flight Trajectory Calculation Method for Unguided Projectiles / M. Akcay // Turkish Journal of Engineering & Environmental Sciences. – 2004. – No. 28. – Pp. 369–376. – Available at: http://journals.tubitak.gov.tr/engineering/issues/muh-04-28-6/muh-28-6-3-0404-5.pdf (accessed: 27.05.2014).

9. Gibson J. Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C# / J. Gibson. – Addison-Wesley Professional, 2014. – 994 p.

10. Harabor D. Online Graph Pruning for Pathfinding on Grid Maps / D. Harabor, A. Grastien. – Available at: http://grastien.net/ban/articles/hg-aaai11.pdf (accessed: 28.05.2014).

11. Heijenk G. Robocode: programming or game / G. Heijenk // ICT BlogICT Blog. – Available at: http://blog.ict.eu/2013/03/robocode-programming-or-game/ (accessed: 03.09.2014).

12. LaValle S. Planning Algorithms / S. LaValle. – Cambridge University Press, 2006. – 842 p.

13. Norvig P. Teach Yourself Programming in Ten Years / P. Norvig. – Available at: http://norvig.com/21-days.html (accessed: 12.06.2014).

14. Tanner B. Jump Point Search Analysis / B. Tanner. – Available at: http://www.cs.fsu.edu/~cop4601p/project/students/bryan-tanner/JumpPointSearch_tanner.pdf (accessed: 05.09.2014).

References

1. Akho A., Seti R., Ulman D. *Compilyatory: printsipy, tekhnologiya i instrumenty* [Compilers: Principles, Techniques, and Tools], Moscow, Williams Publ. House, 2011. 768 p.

2. Akho A., Khopkroft D., Ulman D. *Struktury dannykh i algoritmy* [Data Structures and Algorithms], Moscow, Williams Publ. House, 2010. 400 p.

3. Koryagin S. V. Perekodirovshchik yazykov nepreryvnogo modelirovaniya [Transcoder of continuous modeling languages]. *Promyshlennye ASU i kontrollery* [Industrial AMS and Controllers], 2013, no. 10, pp. 52–56.

4. Molchanov A. Yu. *Sistemnoe programmnoe obespechenie* [System software], 3rd ed. Saint Petersburg, Piter Publ. House, 2010. 400 p.

5. Sverdlov S. Z. *Yazyki programmirovaniya i metody translyatsii* [Programming languages and translation methods], Saint Petersburg, Piter Publ. House, 2007. 638 p.

6. Simenko I. *FightCode: Tankovye voiny na JavaScript* [FightCode: Tank battles on JavaScript]. Available at: http://habrahabr.ru/post/174461/ (accessed: 03.09.2014).

7. Shampandar A. *Iskusstvennyy intellekt v kompyuternykh igrakh: kak obuchit virtualnye personazhy reagirovat na vneshnie vozdeystviya* [Artifical Intelligence in computer games: how to teach vitrual characters to react on external actions], Moscow, Williams Publ. House, 2007. 768 p.

8. Akcay M. Development of Universal Flight Trajectory Calculation Method for Unguided Projectiles. *Turkish Journal of Engineering & Environmental Sciences*, 2004, no. 28, pp. 369–376. – Available at: http://journals.tubitak.gov.tr/engineering/issues/muh-04-28-6/muh-28-6-3-0404-5.pdf (accessed: 27.05.2014).

9. Gibson J. *Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C#*, Addison-Wesley Professional Publ., 2014. 994 p.

10. Harabor D., Grastien A. *Online Graph Pruning for Pathfinding on Grid Maps*. Available at: http://grastien.net/ban/articles/hg-aaai11.pdf (accessed: 28.05.2014).

11. Heijenk G. Robocode: programming or game. *ICT BlogICT Blog*. Available at: http://blog.ict.eu/2013/03/robocode-programming-or-game/ (accessed: 03.09.2014).

12. LaValle S. *Planning Algorithms*, Cambridge University Press, 2006. 842 p.

13. Norvig P. *Teach Yourself Programming in Ten Years*. Available at: http://norvig.com/21-days.html (accessed: 12.06.2014).

14. Tanner B. *Jump Point Search Analysis*. Available at: http://www.cs.fsu.edu/~cop4601p/project/students/bryan-tanner/JumpPointSearch_tanner.pdf (accessed: 05.09.2014).