Игровое поле вычислительной дидактики

Патаракин Евгений Дмитриевич,
доктор педагогических наук,
профессор Института Цифрового Образования МГПУ

Появление новых словосочетаний «цифровая дидактика» [1], «STEM-дидактика»[2] «объектно-дидактика» [3] связано с освоением дидактикой новых областей обучения. Серьёзные вызовы, которые встают перед современной дидактикой, связаны с тем, что в процесс обучения, взаимодействия и построения знаний в современном мире вовлекаются не только люди, но и вычислительные машины. К привычным дидактическим отношениям, когда обучающий человек учит обучаемого человека, добавляются новые дидактические отношения, когда обучающая машина учит другую обучаемую машину, когда обучающая машина учит человека или когда обучающий человек учит обучаемую программу или обучаемого агента. В связи с этим педагогический дизайн в цифровой среде предполагает проектирование и поддержку ситуаций, возможные варианты которых представлены в таблице 1.

В данной главе рассматривается та область образовательного ландшафта, где дизайнер проектирует для обучаемого

Таблица 1. Области вычислительной дидактики

	Человек	Компьютерный агент
Человек	Традиционный дизайн, когда обучающий человек учит обучаемого человека, дополненный возможностями вычислительной техники.	Конструкционизм, когда человек учится в процессе обучения программных агентов правилам поведения в цифровой среде.
Компьютерный агент	Искусственные обучающие системы, когда компьютерная программа выступает в роли учителя для обучаемого человека.	Метапрограммирование и машинное обучение, когда одна компьютерная программа обучает другую программу.

человека цифровую среду, в которых тот выступает обучающим демиургом для обучаемых программных агентов. В зоне пересечения обучающего ученика и обучаемого программного агента перед педагогическим дизайном открывается поле вычислительной дидактики, на котором ученики обучают цифровых исполнителей правилам поведения. Это поле вычислительной дидактики развивается достаточно давно и предполагает формирование у обучаемого вычислительного мышления, как способности человека использовать возможности вычислительных систем. Особенность этой области знаний подчеркивает простое и компактное определение вычислительного мышления, данное А. Репеннингом — «синтез способностей человека и возможностей вычислительных систем» [4]. При этом вычислительная грамотность предпола-

гает способность человека говорить с компьютерными сущностями на языке, который они понимают, обращаться к ним с запросами, которые они способны выполнить, учить их тому, чему они способны научиться. Традицию использовать прилагательное «вычислительный» по отношению к области знаний впервые в 1969 году предложили С.Пейперт и М.Минский в своей книге о персептронах и вычислительной геометрии [5]. В дальнейшем термин использовался по отношению к различным областям знаний, но всякий раз прилагательное «вычислительный» подчеркивало, что формулировка задач, подготовки решений и способы их проверки должны строиться таким образом, чтобы в этих процессах могли принимать участие и люди, и компьютеры. Союз Минского и Пейперта в области искусственного интеллекта и вычислительного мышления в компьютерной среде был очень плодотворным. В этой статье хотелось бы подчеркнуть роль Минского в становлении конструкционизма и формировании вычислительного мышления как условия успешного сосуществования людей и машин в гибридной среде. Компьютерные программы для Минского это – среда, в которой ученики могут читать и рассказывать чудесные истории, собирая их из цифровых элементов как из цифровых кубиков. Как отмечал Минский, магия арифметики состоит в том, что для неё безразлично, что именно считать - цветы, деревья, автомобили или динозавров. А компьютерные программы делают ровно обратное и позволяют вещам возникать там, где их раньше не было. Для Минского компьютерный мир является источником бесконечных детских кубиков, из которых ребёнок может собирать свой собственный город или собственный искусственный мир - вселенную, в которой с её

обитателями могут происходили различные чудесные истории, сюжет и правила которых направлялись бы ребёнком и где ребёнок может эти кубики научить как себя вести и организовать их поведение так, что они имитировали бы для зрителя поведение живых существ. Наглядным примером могут служить среды блочного визуального программирования, в которых со сфабрикованными из цифровых кубиков существами будут происходить чудесные истории. И тут важно даже не то, что компьютерная среда бесконечно богата объектами, а то, что эти объекты принадлежат ребенку, который является субъектом деятельности и творцом искусственного мира. Эта позиция программиста как творца искусственного мира была очень близка А.П. Ершову, который писал том, что программист ощущает себя и отцом-создателем программы, и сыном-братом этой машины, и носителем святого духа — вложенного в нее интеллекта [6]. В статье о переоценке значимости конкретного мышления Тёркл и Пейперт указывали, что компьютер расположен между мирами формальных систем и физических вещей, поэтому он способен сделать абстрактное конкретным, а назначение компьютера состоит в том, чтобы спустить философию на землю и обеспечить контекст для развития конкретного мышления [7]. Пейперт обсуждал возможность конкретизации абстрактного, когда движущийся на экране компьютера объект ограничен формальными правилами и является абстракцией, но в то же время он видим, осязаем и мы можем манипулировать этим объектом. И далее Пейперт приводил простой пример управления таким объектом в стиле конкретного программирования, которое начинается, когда мы определяем команды *REPEAT 4 [FORWARD 100 RIGHT 90]* как процедуру *TO SQUARE*.

Внимание к конкретным примерам реализации того или иного принципа, той или иной идеи в компьютерной среде с указанием кода программы или конкретной видео-игры, которая реализует мощную идею, является отличительной чертой Пейперта как учёного. И дело здесь не в том, что он был одним из ведущих разработчиков языка Logo и для него было естественно думать при помощи конструкций этого языка [8]. Использование кода конкретных программ внутри текстов книг и статей является отличительной чертой большинства публикаций Пейперта и скрытым примером его собственного вычислительного мышления. Э. Акерман в ряде работ отмечает, что хотя Пейперт и считает формальное и абстрактное мышление мощными инструментами, для него в отличие от Пиаже и Выготского важнейшее значение имеет погружение в конкретную ситуацию в компьютерной среде, где учащиеся сами изобретают для себя необходимые инструменты и средства деятельности [9,10].

В переводе на русский язык книга Пейперта вышла под названием «Переворот в сознании: дети, компьютеры и плодотворные идеи», хотя в исходном тексте использовалось словосочетание powerful ideas – «мощные идеи». Позднее Пейперт не раз отмечал, что читатели и критики обратили внимание на детей и на компьютеры, а ключевое положение, что в компьютерной среде ученики получают доступ к мощным идеям, осталось незамеченным. Впоследствии Пейперт неоднократно возвращался понятию мощных идей. Так, в статье о больших идеях Пейперт отмечал, что почти сорок лет тому назад он влюбился в идею, что технологически насыщенная среда может дать

тем детям, которые любят идеи, возможность работать с идеями в среде богатой учебными возможностями, а тем детям, которые меньше любят идеи, возможность побольше полюбить идеи [11]. Принцип мощности является ключевым среди принципов обучения, представленных Пейпертом в работе о пространстве обучения, в рамках которой Пейперт обсуждает следующие принципы:

1) сначала использование, потом понимание принципов
2) сначала проект, а потом проблема
3) сначала объект (вещь), а потом операция над объектом
4) сначала динамика, потом статика [12].

Для каждого из принципов Пейперт всегда приводит конкретные примеры реализации в компьютерной среде. Для принципа вещественности Пейперт приводит пример, связанный со значением вероятности и тем, как ученики используют простую функцию *SETCOLOR RANDOM [blue green]* и начинают воспринимать её как рабочий контейнер, куда можно загружать значения и получать на выходе полезные результаты. Пейперт обращает внимание на вещественность функций, которые можно перемещать, которым можно давать имена, из которых можно что-то строить в таких объектно-ориентированных средах подобных Scratch или Snap. Важно, что вещественность функций на экране компьютера позволяет осваивать эти идеи даже в дошкольном возрасте. Другой пример для принципа вещественности связан с конкретной компьютерной игрой аркадой - платформером Mario Bros, где герой постоянно бегает и прыгает. Ученики, пытающиеся создать игру в стиле Mario Bros пытаются научить своего персонажа прыгать. Но,

что такое прыгать для агента? И ученики начинают с простого изменения положения агента на экране *SETY YCOR + 10* и затем *SETY YCOR -10*, а потом приходят к необходимости более плавного изменения траектории с имитации гравитации. Таким образом, гравитация может служить примером одной из мощных вычислительных идей: объект, изначально наделён способностью иметь свойства. В данном примере у агента исходно есть способность испытывать на себе силу тяжести.

В следующем примере Пейперт использует возможности многоагентной среды StarLogo, но сам код вполне может быть реализован с использованием неподвижных агентов-пятен NetLogo

```
let A list -70 0
let B list 0 50
ask patches [ let a1 towardsxy first A last A
let b1 towardsxy first B last B
set pcolor( a1 - b1 ) ]
```

В книге «Переворот в сознании» Пейперт писал о необходимости «*объекта, при помощи которого можно лучше думать*». В поведении такого объекта ученик может увидеть свое собственное поведение. Впервые этот подход был воплощен в языке Лого, когда ученик получил в свое распоряжение агента - Черепашку, которой можно было давать простые команды: вперед, назад, направо, налево, повтори и т.д. Из этих простых кирпичиков постепенно складываются значительно более сложные микромиры. Основной методологический принцип, который развивал С. Пейперт и его последователи, состоит в том, что: «*Обучение происходит наиболее эффек-*

тивно, если учащийся вовлечен в создание общественного объекта (public entity), будь то замок из песка, машина, книга или компьютерная программа» [13]. Этот принцип использовался в качестве исходного принципа для многочисленных ремиксов. Например, среда Scratch с библиотекой проектов, которые можно читать, копировать, видоизменять и повторно использовать, создавалась как дополнение методологического принципа условием, что объект может обсуждаться, оцениваться и использоваться другими участниками для создания новых объектов [14]. В дальнейшем это условие было дополнено возможностями учебной аналитики и принцип был переформулирован в следующей редакции – «*обучение происходит наиболее эффективно, если субъект образования вовлечён в создание продукта деятельности, который может обсуждаться, оцениваться и использоваться другими участниками для создания новых объектов, а данные о взаимодействиях субъектов образования могут быть представлены в виде карты*» [15].

Другое направление развитие конструкционистского принципа было связано с тем, что для создания продукта деятельности учащийся должен обучить программных агентов алгоритмам поведения в цифровой среде. И в этом направлении последователи Пейперта наибольшее внимание уделяли созданию условий, в которых обучающиеся могли бы создавать компьютерные программы, управляющие поведением компьютерных агентов или роботов. Развитие идей Пейперта в обучении вычислительному мышлению привело к тому, что интерес ученика должен состоять не в том, чтобы удовлетворять требованиям учебной

программы, а в том, чтобы учить агентов в искусственном мире вести себя так, как будто этот искусственный мир воспроизводит естественный. Такое обучение черепахи использованию рекурсивных паттернов поведение продемонстрировали Абельсон и диСесса в книге по геометрии черепахи [16]. В дальнейшем авторы перенесли этот конструктивистский подход в свои книги по обучение вычислительному мышлению и в такие среды образовательные среды как Boxer [17] и App Inventor [18]. Другая ветвь развития микромиров была связана с созданием программ, которые могут создавать тексты книг и других программ. Положение, что компьютерные программы одинаково хорошо могут работать и с графикой и текстом и со своим собственным текстом является ключевым для трехтомника Брайна Харви «Стиль программирования Лого», где автор показывает, что возможности Лого и без черепашьей графики огромны и здесь можно понять и имитировать различные подходы к программированию, использовать тексты программ в качестве входного параметра процедур [19]. В дальнейшем этот подход Харви воплотился в таких образовательных средах как Snap!, NetsBlox и GP [20–22].

Эволюционное древо обучающих языков, роботов и микромиров, ведущих свою историю от Черепашки Лого, включает сотни видов. На рисунке 1 представлен лишь небольшой фрагмент таксономии.

Потенциальные возможности для творчества, которые скрываются в различных учебных микромирах и языках программирования, с одной стороны чудесны сами по себе, но, с другой стороны они сами по себе эти возможности не приводят

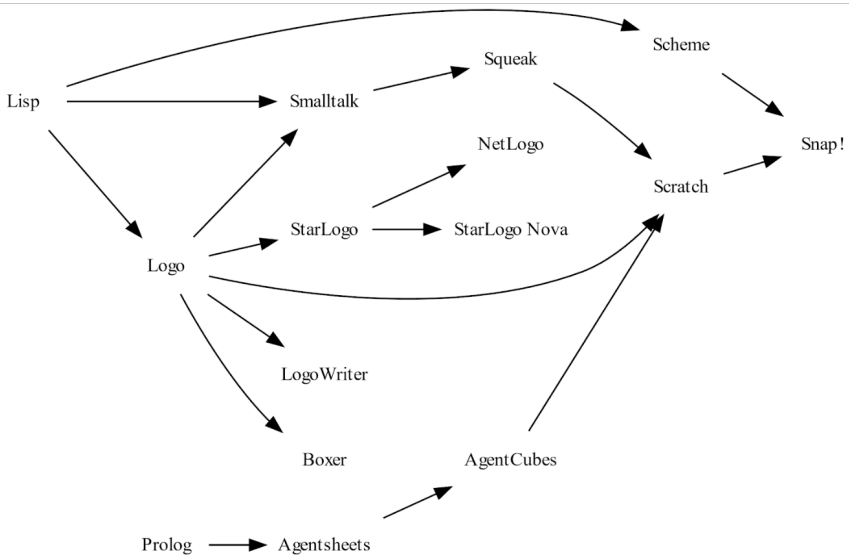


Рисунок 1. Фрагмент таксономии образовательных микромиров

к тому, что учащийся сам начнёт активно их использовать. И тут вряд ли может помочь обучение правилам и возможностям языка, поскольку важно не только дать ученику в руки удочку, но и взять его с собой на рыбалку и показать, как могут выглядеть результаты деятельности. Важное значение имела критическая статья Минского про обучение языку Лого [23], которая начиналась со слов о том, что освоение языка не происходит через изучение его правил, язык всегда осваивается через чудесные истории, которые люди хотят читать и рассказывать. Педагогический дизайнер обучающих микромиров должен заранее предусматривать создание или использование существующих коллекций чудесных историй, которые могут происходить с агентами-исполнителями в цифровой среде. Такими коллекциями примеров, с которыми ученик может

быть знаком до того, как он попадает в микромир творчества, могут быть коллекции видеоигр, в которых уже реализовано поведение агентов-исполнителей в рамках заданного игрой сценария. Даже если ученик не видит, за счёт чего это поведение реализовано, и какие алгоритмы использовали разработчики видеоигры, он знакомится с основными паттернами поведения исполнителей, которые потом можно использовать в своих собственных историях. мало создать среду, богатую возможностями для конструирования, нужно показать примеры того, что может быть сконструировано. Иными словами, мало дать детям в распоряжение учебную типографию, надо заранее пустить их в школьную библиотеку, где бы они могли посмотреть на примеры чудесных историй, чтобы им захотелось похожие истории создавать в своей типографии. Если продолжать этот пример, то современная цифровая школьная библиотека — это коллекция видеоигр, которые используются в образовании для формирования у ученика цифровой грамотности, начитанности, насмотренности и наигранности, чтобы в дальнейшем, когда ученик попадал бы в микромиры агентов и роботов, он бы представлял возможные варианты того, что здесь может быть собрано и сконструировано из этих цифровых или физических кирпичиков.

В рамках данной главы мир видеоигр рассматривается только с точки зрения коллекции примеров цифровых историй, побывав внутри которых, учащийся узнаёт возможные действия компьютерных исполнителей в цифровой среде. При этом собственные дидактические возможности видеоигр остаются вне рамок рассмотрения. Для знакомства с этими

возможностями следует обратиться в первую очередь к работам Джи [24,25] и Ривза [26]. Кроме того, в этой книге есть глава Ольги Максименковой о том, чем видеоигры могут обогатить цифровую дидактику. В данной главе хотелось бы зафиксировать связь, которая существует между областью видеоигр, где ученики знакомятся с возможными паттернами поведения исполнителей, и областью микромиров, где ученики осваивают навыки управления поведением исполнителей, обучая их через создание алгоритмов (Таблица 2).

Удивительный учебный мир возможных историй с объектами, которые собираются из цифровых кирпичиков в видеоиграх и обучающих микромирах, несомненно испытал влияние работ Джеймса Гибсона об экологическом подходе к зрительному восприятию окружающего мира, когда части ландшафта и отдельные вещи описываются с точки зрения тех возможностей, которые они открывают перед персонажем, и это описание выглядит как прообраз инструкции к компьютерной игре типа квест - перед тобой пещера, в которой можно укрыться, а тут лежит камень, который можно взять [27,28]. Так Гибсон описывает обрыв, как место, откуда

Таблица 2. Связь микромиров и видеоигр

	Видеоигры	**Микромиры**
Агенты исполнители	Ученики знакомятся с возможными паттернами поведения	Ученики реализуют паттерны поведения через создание алгоритмов, управляющих поведением агентов.

можно упасть; склон, как деталь местности, которая позволяет или не позволяет идти пешком в зависимости от текстуры и угла наклона; хижину, как объект, где кровля защищает от дождя, снега и прямых солнечных лучей, а стены защищают от ветра и предотвращают утечку тепла. Сегодня это описание выглядит как перечень объектов, свойства которых надо запрограммировать внутри компьютерной видеоигры. И этот же экологический подход к действиям объектов в естественном или искусственном мире, где они прячутся или толкают друг друга, считывается создателями существ, состоящих из цифровых кубиков, мы можем обнаружить в работах Мишот [29]. В дальнейшем экологический подход получил развитие в исследованиях Ривза и Нааса о том, что люди воспринимаем медиа и компьютерные сущности так, как будто они являются живыми существами, с которыми нужно быть вежливыми и нужно уметь договариваться [30]. Во многом этот же экологический подход в образовании реализован в работах Репеннинга и его коллег в области дизайна образовательных игр, где они сумели выделить перечень простых поведенческих паттернов, которым ученик в искусственном мире может научить существа, состоящие из цифровых кирпичиков [31]. В поведении агентов есть простые базовые правила - как что-то подсчитать или нарисовать. А есть более сложное поведение, направленное на то, чтобы ввести наблюдателей за поведением исполнителей в заблуждение, так чтобы они поверили, что исполнители на экране толкают друг друга, убегают, ищут, преследуют, рождаются и умирают. Ниже приведён перечень двенадцати паттернов, наиболее часто встречающихся в видеоиграх.

1. Рождать или генерировать: - агент порождает поток других агентов - например, в экологических моделях рождение используется для создания потомков, в играх из пистолета вылетают пули, которые он порождает, в историях Scratch капли дождя клонируются и падают из тучи на землю.

2. Уничтожать или поглощать: обратный генерации процесс, когда агент не порождает, а поглощает других агентов. Например, земля поглощает падающие с неба капли воды, хищник съедает жертву, с которой он встречается.

3. Столкновение: паттерн определяет поведение агентов в случае их физического столкновения с другими агентами. Например, столкновение пули с мишенью. В игре Frogger, если грузовик сталкивается с лягушкой, лягушку нужно «раздавить».

4. Транспортировка: Транспортировка представляет собой ситуацию, когда один агент перевозит на себе другого агента. Например, черепаха в Frogger несёт лягушку, пересекая реку. В экологических симуляциях процесс транспортировки можно использовать, например, для перевозки пыльцы пчёлами.

5. Толкать — это паттерн, который мы видим во многих играх, например, в Sokoban или Digger, где агент должен толкать ящики или мешки с золотом. Когда игрок толкает коробку или мешок, эти объекты движутся в том направлении (вверх, вниз, вправо или влево), в котором их толкнули.

6. Тащить — противоположен по значению паттерну толкать. Агент может тащить за собой другого агента или агентов. Например, поезд тащит вагоны.

7. Диффузия: паттерн распространения, когда вы можете распространять определённое количество агента на других агентов, расположенных по близости за счёт процесса диффу-

зии. Например, в модели муравьёв муравьи выделяют вещества феромоны, которые диффундируют на ближайшие поля игрового поля.

8. Перемещаться случайным образом. Агент или агенты перемещаются по экрану случайным образом - стандартная для многих игр и симуляций ситуация, которую надо уметь воспроизводить - термиты, птицы и рыбы при формировании стай изначально перемещаются случайным образом.

9. Подчиняться клавишам клавиатуры: нажатия кнопок клавиатуры управляют движением агента. И здесь могут быть паттерны типа «Прыгать».

10. Преследовать другого агента. Один агент преследует другого агента - поворачивается в его сторону или поворачивается в том же направлении, куда движется другой агент.

11. Превращаться в другого агента, когда один агент превращается в другого агента. Возможный вариант - агент одной породы превращается в агента другой породы.

12. Поиск восхождением к вершине - алгоритм поиска в компьютерных науках, когда агент просматривает значения переменных на ближайших полях и на поле с максимальным значением переменной. Использование алгоритма поиск восхождением к вершине можно наблюдать в таких играх как Sims или Pac-man, когда призраки преследуют Pac-man, следуя наивысшему значению запаха Pac-man, который распространяется по всему полю.

Для каждого поведенческого паттерна можно определить видеоигры, где ученики могут посмотреть на примеры поведения исполнителей, которые этот паттерн реализуют. Кроме того,

в пространстве языков программирования и образовательных микромиров постепенно накапливаются алгоритмы – рецепты того, как тот или иной паттерн поведения может быть реализован в цифровой среде. Для каждого образовательного микромира мы можем найти или написать программы алгоритмы, реализующие базовые паттерны поведения, с тем чтобы ученики могли их использовать в своих работах в качестве основы. В данной главе мы использовали в качестве среды для реализации алгоритмов язык NetLogo, поскольку у языка очень простая нотация и он является современным стандартом при создании как научных, так и учебных симуляций [32,33].

В таблице 3 названия видеоигр представлены во втором столбце, а в третьем столбце представлен листинг алгоритма, которые позволяет получить необходимый поведенческий паттерн в среде NetLogo.

Итак, к этому моменту на поле образовательной дидактики у нас могут быть представлены: образовательные видеоигры, в которых ученики знакомятся с возможными паттернами поведения программных агентов-исполнителей; обучающие среды (языки программирования), где учащиеся могут создавать свои собственные истории и игры; примеры - рецепты приготовления алгоритмов, управляющих поведением программных агентов. К этим взаимосвязанным классам сущностей следует добавить класс основных понятий поля вычислительной дидактики, класс авторов, которые вводят новые понятия, создают видеоигр и обучающие микромиры; класс роботов исполнителей в окружающем физическом мире вещей; класс сетевых сообществ,

Таблица 3. Паттерны, игры и алгоритмы

Описание поведения	Образец	Код (NetLogo)
Рождать	Frogger, Centipede	crt 10 ask turtles [hatch-sheep 1] ask patches [sprout 10]
Уничтожать	Frogger, Centipede	ask turtles with [color = red] [die]
Сталкиваться	Frogger, Pac-man	ask turtle 1 [if other turtles-here [die]] ask turtle 2 [if other turtles-here [hatch 2]]
Превращаться в другого агента	Frogger, Ants	if shape = “ant” [set shape “ant-has-food”]
Перемещаться случайным образом	Mario Bros	to wiggle fd 1 rt random 50 lt random 50 end
Преследовать другого агента	Pac-man	face agent move-to agent set heading
Перевозить другого агента на себе	Frogger	ask turtle 0 [move-to turtle 1 set heading [heading] of turtle 1]
Толкать (push)	Sokoban, Digger	ask turtle 0 [move-to turtle 1 set heading [heading] of turtle 1 fd 2]
Тащить (pull)	Sims, Termites	ifelse pcolor = black [set pcolor yellow get-away]
Распространяться (диффузия)	Ants	diffuse chemical (diffusion-rate / 100)

Продолжение таблица 3.

Искать методом восхождения к вершине	Sims, Pac-man, HillClimbingExample	uphill patch-variable to move uphill elevation end
Подчиняться (клавишам клавиатуры)	Mario Bros, Frogger, Tetris	to rotate-right if rotate-right-clear? and not game-over? [ask pieces [rotate-me-right]] end

в которых может происходить обмен рецептами приготовления алгоритмов; класс знаний и умений, которых требуют образовательные стандарты в сфере вычислительного мышления в разных странах. Объединение и наращивание многообразия сущностей на поле вычислительной дидактики множеством участников предполагает создание разделяемой онтологии, которая была реализована в среде Semantic MediaWiki через создание категорий - классов объектов. В качестве инструмента коллективного сбора и объединения знаний о поле вычислительной дидактики была использована вики-платформа MediaWiki. В течение нескольких лет мы используем вики как среду для совместной работы со знаниями в образовании [34,35]. Для изучения области вычислительной дидактики, в которой для обучаемого создаются возможности выступить в роли обучающего для обучаемого программного агента или робота, мы предложили использовать площадку MediaWiki с расширением Semantic MediaWiki [15,36]. Расширение Semantic MediaWiki позволяет добавлять семантические аннотации к вики-страницам, тем самым пре-

вращая вики в семантическую вики и существенно расширяя возможности совместной работы не только над текстами статей, но и над совместной онтологией знаний [37]. Экспериментальная площадка использует технологию Semantic MediaWiki и главная особенность заключается в том, что мы думаем сразу о классах статей объектов, которые относятся к определенной категории и обладают определёнными свойствами. Вики площадка или поле вычислительной дидактики открыто для совместного использования преподавателей и студентов по адресу http://digida.mgpu.ru

Интерактивный сбор и представление знаний при помощи форм, шаблонов и свойств системы совместной работы со знаниями Semantic Media Wiki, что обеспечивает единообразный сбор данных о сущностях, представляющих новую область знаний и возможность поиска и формирования отчётов по запросам, сформулированных на языке ASK. Мы не просто создаем отдельные статьи, например, про понятие «рекурсия» или «большие данные» или про языки обучения «Scratch» или «Snap!», но исходно создаем классы, к которым относятся эти статьи.

В своей работе по конструированию поля вычислительной дидактики мы использовали класс объектов паттернов внутри небольшой, но очень востребованной категории паттернов поведения. Объекты этой категории связаны с объектами из категории языки программирования и объектами из категории рецепты. Мы можем обращаться к объектам из любой категории и просить их вывести на экран ту или иную информацию. Например, если для категории паттернов мы хотим получить только

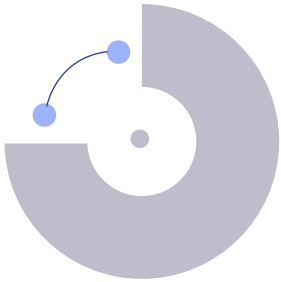
список с описанием поведения, которое должен имитировать программный агент, то мы обращаемся к системе с запросом:
{{#ask: [[Категория:HowTo]] [[Description_of_problem::+]] | ?Description_of_problem | format = ol}}

В результате выполнения запроса мы получаем на странице нумерованный список с описанием паттернов.

У объектов из класса паттернов есть так же свойства, связывающие их с образовательными видеоиграми, языками программирования и рецептами - алгоритмами решения той или иной проблемы. Мы можем использовать категорию «Видеоигры» и узнать в какой видеоигре можно посмотреть на реализацию того или иного паттерна или получить информацию из категории «Рецепты» о том, как реализуется то или иное поведение в конкретной среде программирования (категория «Язык программирования).

Категория языков программирования и сред конструирования была первой и достаточно простой категорией объектов, которые мы начали собирать на поле вычислительной дидактики. Как правило, у языков есть год создания, авторы, языки – предшественники и языки потомки. Даже такой ограниченный перечень параметров позволяет внутри экспериментальной площадке получить историю языков программирования оформленную как ленту времени (рисунок) в ответ на запрос:

{{#ask:
[[Category:Язык программирования]]
[[launch year::+]]



|format=moderntimeline
|?launch year
|? Ancestors
|?Descendants
}}

Кроме того, из окружающего мира статей из других категорий мы можем узнать о языках дополнительную информацию о том, для освоения каких понятий (категория «Понятия») или для имитации каких паттернов (категория «Паттерны») поведения может быть использован тот или иной язык. Для всех классов объектов, представленных на экспериментальной площадке совместной работы со знаниями, на странице категории представлено краткое описания и диаграмма класса, которая позволяет увидеть основные свойства объектов из данной категории. Читатель может посмотреть на возможности организации динамического поиска на страницах экспериментальной площадки, используя примеры из пространства «Концепты». Концепты или Концепции Semantic MediaWiki представляют собой простейшие динамические категории, которые возникают в ответ на запросы. Например: Концепт «Язык с родословной» задаётся при помощи следующих условий:

{{#concept:
[[Категория:Язык программирования]]
[[Ancestors::+]]
}}

В переводе на человеческий язык поисковый запрос выглядит так: покажи нам все слова из категории «Язык програм-

мирования» для которых свойство «Предшественник» имеет не пустое значение.

Следует обратить внимание, что и страницы, и свойства Semantic MediaWiki служат строительными блоками-кирпичиками, из которых можно собирать разнообразное содержание. В этот перечень строительных блоков, которые можно использовать на страницах экспериментальной площадки, мы добавим блоки визуального программирования на таких языках как Scratch и Snap! Например, если на странице вики следующая последовательность текстовых блоков, то она превратится в визуальные блоки (Рисунок 2)

<scratchblocks>
когда спрайт нажат
сказать [«Привет!»]
идти (выдать случайное от (20) до (40)) шагов
играть звук [мяу]
если <касается [край v]>, то
повернуться к [указатель мышки v]
конец
</scratchblocks>

Блоки визуального программирования на страницах экспериментальной площадки можно легко переносить на другие страницы, видоизменять и работать с ними как со строительными блоками. Следующий тип объектов, представленных на экспериментальной площадке, относится к классу и категории обучающих видео игр. В категории DEG – digital educational games пока только около 25 игр, но шаблон и форма описа-

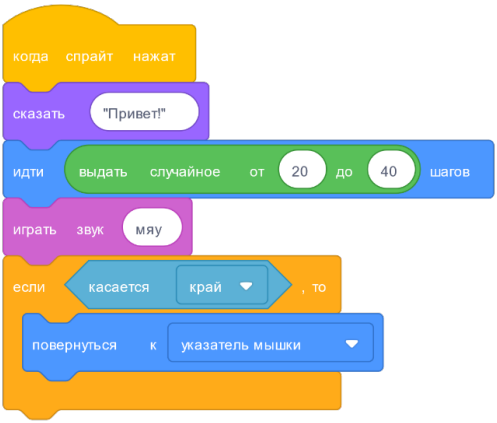


Рисунок 2. Блоки визуального программирования на странице вики

ния объектов этого класса сделаны очень подробно. Полный перечень свойств мы можем увидеть в шаблоне описания обучающих видео игр на площадке совместной деятельности:

{{DEG

|Description= Описание (сценарий) игры

|Website= Веб-сайт, где можно запустить или скачать или скачать игру

|Running_on= Операционные системы, под которыми запускается игра

|game_genre= Жанр игры

|Ages= Возраст, с которого доступна (рекомендована) игра

|Affective_tasks= Аффективные цели, которые стоят перед героем игры

|learning_cases= Примеры использования

|game_resource= Дополнительная информация, помогающая игроку

|Language_Ru_Eng= Поддерживается ли русский язык

|Assistance tools= Какие есть инструменты поддержки?
|Developer= Разработчик игры
|launch year= Год запуска
|closing year= Год закрытия (если поддержка закрыта)
|framework= Теоретическая Рамка Игры
|Screenshot= Скриншот экрана игры
|Clarifying_video= Поясняющее видео
|Community= Сообщество игроков (как отдельная статья)
|Competences= Формируемые игрой компетенции
|Ancestors= Предки игры или пререквизиты, помогающие в освоении
|Descendants= Потомки игры
|Difficult= Трудности использования
|distant_collab= Совместное сетевое использование
|License= Тип лицензии, по которой предлагается игра
|Визуальная реализация= Способы управления героем
|FieldActivity= Сфера деятельности
|Dimension= 2 или 3 D
}}

Вся база знаний носит открытый характер и может дополняться участниками. После того как свойства статей в категории были определены, мы продолжаем поиск и добавление новых свойств, характеризующих образовательные игры. Например, мы добавили разделение образовательных их по теориям обучения, которые лежат в основании создания этих игр. Это разделение было предложено Э. Брукман в статье «Может ли образование быть весёлым» [38], где она выделяла пять теорий обучения и приводила примеры образовательных игр,

созданных в русле каждого из подходов. Важно, что пользователи могут формировать к базе запросы и получать выборки игр с конкретными характеристиками. Например, если мы получить перечень доступных игр с указаний аффективных целей, то запрос будет выглядеть следующим образом:
{{#ask: [[Категория:DEG]] [[?Affective_tasks::+]] | ?Affective_tasks | }}

Если мы хотим получить перечень игр доступных через веб-интерфейс, то запрос выглядит следующим образом:
{{#ask: [[Категория:DEG]] [[Running_on::on-line]] }}

Рассмотрим в качестве следующего примера класс объектов из категории «Понятие». У каждой статьи из этого класса есть перечень предопределённых свойств. У понятия есть предметная область, к которой оно относится, автор понятия, средства освоения (тут могут быть среды обучения программированию, видеоигры, конструкторы или среды управления роботами), возраст учащегося, когда он готов к тому, чтобы освоить данное понятие. Мы можем в дальнейшем существенно расширить перечень свойств, которые описывают понятие, но пока короткий перечень позволяет использовать эту категорию в качестве примера. Итак, у понятия есть свойство «Предметная область». Для этого свойства мы выбрали тип данных «Страница». Это означает, что свойство и само будет ссылкой на страницу, которая об этом свойстве рассказывает. Вторая особенность свойства «Предметная область» заключается в том, что мы заранее предопределяем возможный перечень его возможных значений. Это ограничение допустимых значений существенно

упрощает работу участников, поскольку при создании новых понятий или при редактировании уже существующих понятий, они выбирают из перечня допустимых значений. Перечень допустимых значений правится и дополняется участниками совместной деятельности. В описании свойства мы видим, что у для свойства определён тип «Страница» и мы работает со значениями этого свойства именно как со страницами. При этом система позволяет нам осуществлять для понятия множественный выбор значений. Например, понятие «Агент» относится и к информатике и робототехнике, понятие «большие данные» относится и к информатике, и к социологии.

Использование свойств на страницах вики позволяет существенно упростить процедуру создания статей и задать формат, в котором участники вносят данные. Это существенно отличает семантические вики от обычных вики-площадок, где авторы в принципе не ограничены формальными рамками. Но, достоинства и возможности Semantic MediaWiki состоят не столько в ограничениях, сколько в новых возможностях. Например, для понятия есть свойства. Одно из свойств понятия – предметная область. И наличие этого свойства позволяет представить распределение всех опубликованных на экспериментальной площадке понятий по свойству. Например, рассмотрим запрос

```
{{#ask:[[Категория:Понятие]] [[Field_of_knowledge::+]]
|?Field_of_knowledge
|mainlabel=-
|format=jqplotchart
|charttype=bar
|charttitle= Предметная область
```

|distribution=yes
|width=80%
|datalabels=percent
}}

Этот запрос позволяет получить распределение всех понятий, для которых указаны предметные области, в виде диаграммы – см. рисунок 6. Обратите внимание, что мы сформулировали в вики запрос на языке #ask для объектов из категории «Понятия», у которых определено значение свойства «*Field_of_knowledge::*» (область знаний) представить распределение значения свойства «область знаний» в формате format=jqplotchart – это просто один из многочисленных форматов представления результатов семантических запросов. И в результате мы получили диаграмму распределения, встроенную непосредственно в текст вики страницы (Рисунок 3).

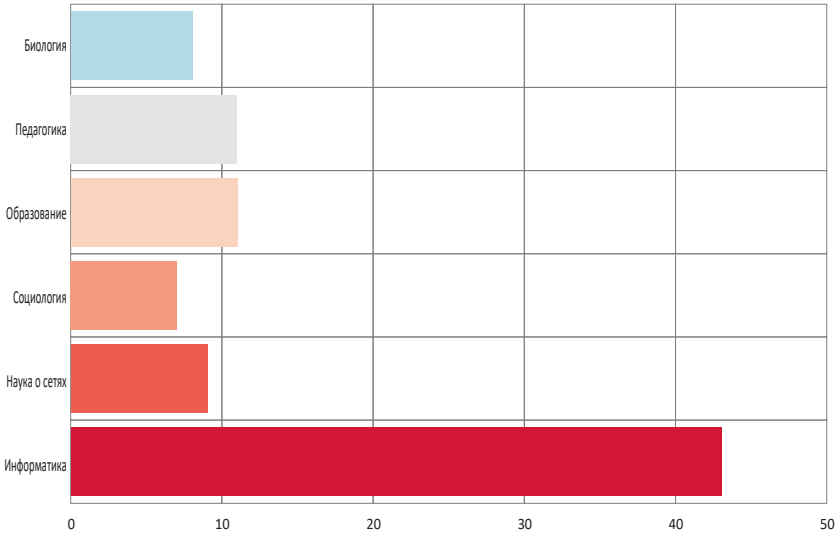


Рисунок 3. Представление диаграммы в вики

Точно так же, как мы формулировали запрос по распределению понятий по областям знаний, мы можем сформулировать запрос о том «какие среды чаще всего используются для освоения того или иного понятия» и получить графическое представления результатов распределения – см. рисунок 4.

{{#ask: [[Category:Понятие]] [[Environment::+]]
|?Environment
|mainlabel=-
|format=jqplotchart
|charttype=bar
|charttitle= Среды освоения
|distribution=yes
|height=800
|width=100%

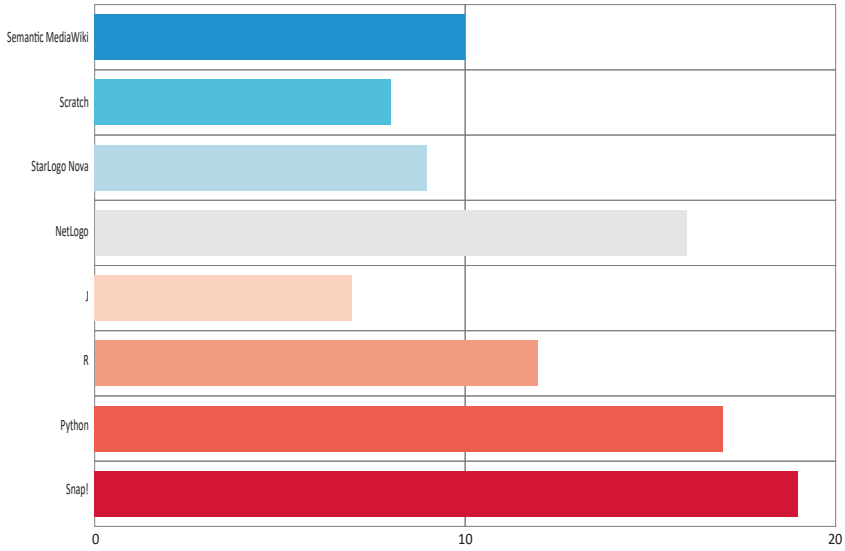


Рисунок 4. Распределение сред, в которых осваиваются понятия

|datalabels=percent
|direction=horizontal
}}

Поле вычислительной дидактики, представленное в данной работе, структурировано как множество разноцветных книг - категорий. В каждой категории собраны статьи одного класса с заранее определёнными свойствами и типами данных. Наиболее распространённым свойством является свойства типа страница, что подразумевает создание ссылки либо уже на существующую страницу, либо на возможное действие по созданию страницы, которой пока ещё нет. И тут необходимо отметить существование чудесных красных ссылок на несуществующие пока статьи. Для многих свойств возможность дальнейшего создания и редактирования статьи заранее предопределена внутри форм создания статей. Например, если участник добавляет текст в статью из класса человек (Person) при помощи формы Person в свойство «ученики этого человека», то система понимает, что и ученики данного человека должны описываться с использованием формы Person

| {{#arraymap:{{{Descendants|}}}|,|x|{{#formredlink:target=x|form=Person}}}}

Если же участник добавляет текст в свойство «среды и средства, на которые повлиял этот человек, то система понимает, что языки программирования и цифровые образовательные игры должны описываться с использованием соответствующих форм.

В начале работы мы определяли вычислительное мышление и вычислительную грамотность как способность говорить с компьютерными сущностями на языке, который они понимают, обращаться к ним с запросами, которые они способны выполнить. Теперь мы можем добавить к этому перечню способность обучать искусственные сущности, собранные из цифровых кирпичиков, навыкам поведения, чтобы они могли принимать участие в чудесных историях искусственного мира.

В заключении главы хочется подчеркнуть, что наши обращения к компьютерным сущностям выглядят сходно вне зависимости от того в какой среде конструирования мы к ним обращаемся. Например, когда мы обращаемся к исполнителям-черепашкам в среде NetLogo и просим некоторых из них показаться и сообщить какую-то информацию о себе, то запрос будет выглядеть следующим образом:
Ask turtles with [color = blue] [set label who] ;; просьба ко всем синим черепашкам написать на себе свой номер.

Если же мы обращаемся к страницам внутри Semantic Media Wiki и просим некоторых из них появиться на экране и и сообщить некоторую информацию о себе, то запрос будет выглядеть сходным образом:
{{#ask: [[Категория:DEG]] [[Assistance_tools::+]] |?Assistance_tools }} ;; просьба ко всем страницам из категории цифровых образовательных игр, для которых определены средства поддержки, выйти на экран и показать средства своей поддержки.

Возможен и обратный перенос знаний и умений из среды MediaWiki в микромиры программирования компьютерных агентов. Например, мы можем реализовать привычную для вики среды практику создания новых категорий и объектов внутри этих категорий в среде программирования Snap!

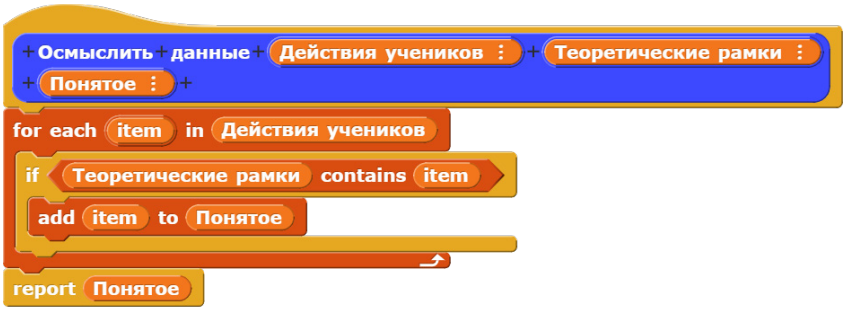
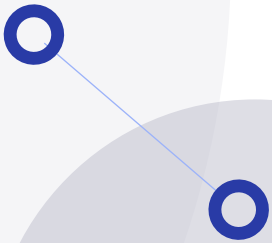


Рисунок 5. Создание блоков в собственной категории «Дидактика»

Мы можем вернуться к утверждению Пейперта о о необходимости «объекта, при помощи которого можно лучше думать» и посмотреть на многообразие доступных нам в настоящее время объектов, как на различные классы объектов, по-разному помогающих нам думать. Общее правило, которое действует внутри сформированного на http://digida.mgpu.ru поля вычислительной дидактики, состоит в том, что все представленные здесь сущности являются объектами - представителями того или иного класса, к которым мы можем обращаться за помощью и просить их что-то для нас сделать. Это такой волшебный мир, где каждая сущность будет вам послушна, как дракон из средиземноморских сказок Ле-Гуин, если только вы обратитесь к нему по верному имени и сформулируете понятный ему запрос. Мы можем обращаться к страницам, агентам-испол-

нителям, внутренним для площадки или удалённым наборам данных, используя сходные фразы запросов. И эта постоянная возможность обращения с запросом, просьбой или командой представляется мне сейчас главным отличительным свойством игрового поля вычислительной дидактики. Есть известная история о ребёнке, который впервые сталкивается с телевизором, после опыта использования детского планшета, и не понимает, зачем нужно устройство, которое никак не реагирует на его действия. Мне кажется, что это привычка взаимодействовать и разговаривать с цифровыми сущностями будет стремительно распространяться, и те цифровые объекты, с которыми ученики и учителя не смогут разговаривать, обречены на вымирание.



Список литературы

1. Jahnke I. et al. Digital Didactical Designs as research framework: iPad integration in Nordic schools // Computers & Education. 2017. Vol. 113. P. 1–15.

2. Martínez-Valdés J.A., Martínez-Ijají N.A. An Experience with the App Inventor in CS0 for the Development of the STEM Didactics // Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality. New York, NY, USA: Association for Computing Machinery, 2018. P. 51–56.

3. Bengtsson S.L., Lysgaard J., Jord-Laugesen N. Object-Oriented Didactics. 2019.

4. Repenning A., Basawapatna A.R., Escherle N.A. Principles of Computational Thinking Tools // Emerging Research, Practice, and Policy on Computational Thinking / ed. Rich P.J., Hodges C.B. Cham: Springer International Publishing, 2017. P. 291–305.

5. Minsky M., Papert S.A. Perceptrons: An Introduction to Computational Geometry. MIT Press, 2017. 317 p.

6. Ershov A.P. Aesthetics and the Human Factor in Programming // Commun. ACM. 1972. Vol. 15, № 7. P. 501–505.

7. Turkle S., Papert S. Epistemological pluralism and the revaluation of the concrete // Journal of Mathematical Behavior. 1992. Vol. 11, № 1. P. 3–33.

8. Solomon C. et al. History of Logo // Proc. ACM Program. Lang. 2020. Vol. 4, № HOPL. P. 79:1-79:66.

9. Ackermann E.K. The “Agency” Model of Transactions: Toward an Understanding of Children’s Theory of Control. P. 17.

10. Ackermann E. Piaget's Constructivism, Papert's Constructionism: What's the difference? // Future of learning group publication. 2011. Vol. 5, № 3. P. 1–11.

11. Papert S. What's the big idea? Toward a pedagogy of idea power // IBM Systems Journal. 2000. Vol. 39, № 3.4. P. 720–729.

12. Papert S. An exploration in the space of mathematics educations // Int. J. Comput. Math. Learn. Citeseer, 1996. Vol. 1, № 1. P. 95–123.

13. Harel I., Papert S. Constructionism: research reports and essays, 1985-1990. Ablex Pub. Corp., 1991. 540 p.

14. Resnick M. et al. Scratch: Programming for All // Commun. ACM. 2009. Vol. 52, № 11. P. 60–67.

15. Patarakin E., Burov V., Yarmakhov B. Computational Pedagogy: Thinking, Participation, Reflection // Digital Turn in Schools—Research, Policy, Practice / ed. Väljataga T., Laanpere M. Singapore: Springer, 2019. P. 123–137.

16. Abelson H., diSessa A. Turtle Geometry: The Computer as a Medium for Exploring. 1st U.S, 1st Printing. Cambridge, Mass: The MIT Press, 1981. 497 p.

17. diSessa A.A., Abelson H. Boxer: a reconstructible computational medium // Commun. ACM. 1986. Vol. 29, № 9. P. 859–868.

18. Wolber D., Abelson H., Friedman M. Democratizing Computing with App Inventor // GetMobile: Mobile Comp. and Comm. 2015. Vol. 18, № 4. P. 53–58.

19. Harvey B. Computer Science Logo Style 2/e, Vol. 1: Symbolic Computing. second edition. Cambridge, Mass: The MIT Press, 1997. 344 p.

20. Stein G. et al. Engaging Female High School Students in the Frontiers of Computing. 2022.

21. Djelil F., Sanchez E. Game design and didactic transposition of knowledge. The case of progo, a game dedicated to learning object-oriented programming // Educ Inf Technol. 2022.

22. Broll B., Grover S., Babb D. Beyond Black-Boxing: Building Intuitions of Complex Machine Learning Ideas Through Interactives and Levels of Abstraction // Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2. New York, NY, USA: Association for Computing Machinery, 2022. P. 21–23.

23. Minsky M. Introduction to LogoWorks // LogoWorks: Challenging Programs in Logo / ed. Solomon C., Minsky M., Harvey B. McGraw-Hill Osborne Media, 1986. P. 388.

24. Gee J.P. What video games have to teach us about learning and literacy. Palgrave Macmillan, 2004.

25. Gee J.P. Good video games + good learning: collected essays on video games, learning, and literacy. Peter Lang, 2007. 210 p.

26. Reeves B., Read J.L. Total Engagement: Using Games and Virtual Worlds to Change the Way People Work and Businesses Compete. Harvard Business Press, 2009. 285 p.

27. Gibson J.J. The Ecological Approach to Visual Perception. New York: Psychology Press, 2014. 346 p.

28. Greeno J. Gibson's affordances // Psychological Review. 1994. № 101(2). P. 336–342.

29. Michotte A. The Perception of Causality. London: Routledge, 2017. 452 p.

30. Reeves B., Nass C. The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places. Center for the Study of Language and Inf, 2003.

31. Basawapatna A.R. et al. The zones of proximal flow: guiding students through a space of computational thinking skills and

challenges // Proceedings of the ninth annual international ACM conference on International computing education research. New York, NY, USA: Association for Computing Machinery, 2013. P. 67–74.

32. Wilensky U., Rand W. An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo. MIT Press, 2015.

33. Railsback S.F., Grimm V. Agent-Based and Individual-Based Modeling: A Practical Introduction, Second Edition. Princeton University Press, 2019. 359 p.

34. Patarakin E., Visser L. New Tools for Learning - The Use of Wiki's // Trends and Issues in Distance Education 2nd Edition International Perspectives / ed. Visser L. et al. Information Age Publishing, 2012. P. 287–299.

35. Patarakin E.D. Wikigrams-Based Social Inquiry // Digital Tools and Solutions for Inquiry-Based STEM Learning. IGI Global, 2017. Vol. 1. P. 112–138.

36. Parandekar S.D., Patarakin E., Yayla G. Children Learning to Code: Essential for 21st Century Human Capital. Washington, DC: World Bank, 2019. P. 151.

37. Díaz-Nafría J.M. et al. glossaLAB: Co-creating Interdisciplinary Knowledge // Applied Informatics / ed. Florez H. et al. Cham: Springer International Publishing, 2019. P. 423–437.

38. Bruckman A. Can educational be fun // Game developers conference. 1999. Vol. 99. P. 75–79.