Списание „Математика, компютърни науки и образование“
том 2, брой 2, 2019

Journal “Mathematics, Computer Science and Education”
vol. 2, No. 2, 2019

Реализация на objects-early подход в училищния курс по информатика

Габриела Чотова, Ивайло Дончев

Objects-Early Approach to Teaching Informatics at School Level

Gabriela Chotova, Ivaylo Donchev

Abstract: *The technology development and the changing requirements of the labor market lead to changes in education. When we talk about computer science and information technology these changes need to be more frequent and extensive. We need not only a change in the curricula content to reach our education aims, but a thorough approach pursuant to the curriculum and the psychological characteristics of the contemporary students as well. In this article we will outline the main approaches used to teach CS at school and present our ideas based on the combination of the best classical approaches practice and experience from the application of innovative didactic methods and tools.*

Keywords: *teaching, informatics, programming, didactic approaches, programming paradigms.*

ВЪВЕДЕНИЕ

Общообразователната подготовка в училищното образование претърпя сериозна реформа през последните години. Особено силно това се отрази на групата учебни предмети, развиващи дигитална компетентност и компетентности в областта на технологиите – компютърно моделиране, информатика и информационни технологии. Акцент в обучението по информатика вече е използването на визуално програмиране за създаване на несложни приложни софтуерни продукти със съвременен интерфейс на управление [1]. Това доведе до необходимост от коренна промяна в подхода на преподаване, изразяваща се не само в проста замяна на използвания език за програмиране с чист обектно ориентиран, а и в цялостна реконструкция на учебния курс, така че да осигурява придобиването на нужните знания и умения за правилното прилагане на обектно ориентираната парадигма в разработването на софтуер.

РАЗВИТИЕ НА УЧИЛИЩНИЯ КУРС ПО ИНФОРМАТИКА

Нормативно регламентираните промени, които настъпиха в училищния курс по информатика чисто формално се изразяват в следното:

– курсът се премества от IX в VIII клас и не е задължителен за всички училища. Намален е и предвиденият хорариум. За да се компенсира това, учителите трябва да разчитат на допълнителни избираеми учебни часове;

– преобладаващата част от учебния материал е посветена на програмирането. Затова и дидактическият подход за преподаване на информатика трябва да е съобразен с подходите за преподаване на програмиране;

– в новата учебна програма [2] темата за двузначната логика – съждения и булеви функции – вече не е самостоятелна, а е част от темата „Програмни конструкции за реализация на разклонен алгоритъм“. Редуцирано е и времето, предвидено за нея – по-малко от един учебен час;

– намален е хорариумът, предвиден за създаване на алгоритми. Темата за описанието на алгоритми чрез блок-схеми отпада от учебното съдържание по Информатика, но се добавя към учебното съдържание по Информационни технологии в X клас (две години по-късно);

– намален хорариум има и при темата за позиционни бройни системи. Силно ограничено е времето, заложено за преобразуване в шестнадесетична бройна система;

– разделите „Компютърни системи“ и „Операционни системи“ са предвидени за изучаване само по предмета информационни технологии в VIII клас.

Като положителна промяна в учебните планове можем да считаме ранното запознаване с програмирането. От новия предмет компютърно моделиране, който се изучава в III и IV клас се очаква да формира основите на алгоритмичното мислене у учениците. Там те се запознават неформално с понятията „алгоритъм“, „обект“ (под формата на герои), „функция“, „променлива“ и други. Учениците придобиват нужните знания и умения като създават прости игри и анимации в олекотена и съобразена с възрастовите им особености среда за разработка. Това е чудесен дидактически инструмент, резултатите от прилагането на който очакваме да оценим в часовете по информатика, когато сегашните четвъртокласници станат ученици в VIII клас (през учебната 2023/2024 г.), тъй като обучението по програмиране продължава чак в VIII клас по предмета информатика.

Друга съществена промяна в училищния курс е замяната на езика C++ с чист обектно ориентиран – Java или C#. Това налага и радикална промяна в подхода на преподаване. До началото на учебната 2017/2018 година учениците изучаваха процедурно програмиране по класическия подход *imperative-first* чрез хибридния език C++. Този език е универсален индустриален език и позволява програмиране в различни стилове (парадигми) – процедурно, функционално, обектно ориентирано и генерично програмиране. Но това е тежък език за начинаещи и сложният му синтаксис отклонява фокуса от важните програмни концепции, които трябва да се изучат за краткото предвидено време. Замяната на C++ със C# или Java предполага съществена промяна и в дидактическия подход, тъй като класическият вариант на *imperative-first* не дава желаните резултати с използването на чист обектно ориентиран език. Акцентът трябва да се измести върху усвояването на обектно ориентираните концепции и механизми. Това означава и коренна промяна в системата от примери и задачи.

В наши дни е проблем да се задържи продължително време вниманието на учениците и за целта е нужен прецизен подбор на примери и задачи, които са близки до проблемите, които ги вълнуват. Силно мотивиращо влияят задачите, свързани с изграждане на графичен интерфейс, и почти задължително условие е учениците да виждат незабавно ползата от практическото прилагане на разработения от тях продукт. Затова в новата учебна програма по информатика са предвидени елементи на дидактическия подход с ранно въвеждане на графичния интерфейс (*GUI-first*). Учениците имат няколко часа, в които изграждат десктоп приложения чрез подреждане и настройване на готови компоненти (контроли), преди те да бъдат запознати с фундаментални понятия като типовете данни, променливи, изрази, функции и др.

Очакваме учениците, които в III и IV клас са изучавали компютърно моделиране, да се справят още по-добре в тази част от курса, тъй като много от понятията ще им бъдат познати, макар и в друга среда за разработка.

ЗНАНИЯ И УМЕНИЯ, СВЪРЗАНИ С ПРОГРАМИРАНЕТО

Безспорна е ползата от програмирането за насърчаване на креативността, логическата мисъл, прецизността при решаване на проблеми/задачи [6]. То дава нагледна представа за понятието „абстракция“ и доказва ползата от нея за моделиране на реалността. В информатиката абстракцията е начин на разглеждане на обект, при който се вземат предвид само тези негови характеристики, които са важни в конкретната ситуация. Тя е изключително мощна техника за справяне със сложността – ако не можем да обхванем целия сложен обект, избираме да игнорираме неговите несъществени детайли и работим с обобщен, идеализиран модел на обекта [9]. Използването на абстракции прави възможно конструирането на сложни програми, въпреки ограничените възможности на човешкия мозък. Абстракцията фокусира върху външната гледна точка към обекта и така служи за отделяне на характерното за обекта поведение от неговата реализация. Изборът на точния набор от абстракции за дадената предметна област е централен проблем на обектно ориентирания дизайн [5].

Изчислителното мислене (*computational thinking*) е подход към решаването на задачи, обединяващ в едно логически умения и основни концепции от компютърните науки – абстракция, капсулиране, модулност, йерархичност, алгоритми, ефективност и др. То включва набор от умствени инструменти, които отразяват широтата на областта на компютърните науки [8]. Добре развитите умения за изчислително мислене дават основата на уменията за решаване на проблеми [7].

Изучавайки програмиране, учениците трябва да придобият базови знания и умения за проектиране и реализиране (имплементиране) на софтуерни системи. Това включва и откриване и отстраняване на грешки в кода, проследяване на състоянието на програмата при изпълнение – работата с дебъгера.

Важно е да могат да разбират и обясняват логиката на управление в програмата, а не чрез налучкване да достигат до работеща програма, но с неразбираем код. Съществени умения, свързани с програмния дизайн, са да се анализира условието на конкретна задача, да се открият правилните абстракции в предметната област, да се конструират класовете и обектите, да се моделира взаимодействието между тях и не на последно място – да се създаде ефективен алгоритъм за решението. Можем да кажем, че програмирането формира алгоритмична култура и развива познавателните способности на учениците – абстрактно, логическо и алгоритмично мислене. Също така формира умения за работа с различни видове информация и данни.

ПОДХОДИ ЗА ОБУЧЕНИЕ ПО ИНФОРМАТИКА

Необходимите знания и умения могат да бъдат постигнати чрез прилагането на различни дидактически подходи. Голяма част от тези подходи разчитат на въвеждане на компютърните концепции чрез програмиране (моделът *programming-first*). Затова в новата учебна програма по информатика за VIII клас [2] и учебниците, изготвени по нея, програмирането заема значителна част. Например в [10] пет от шестте раздела са посветени изцяло на програмиране.

В програмирането са известни множество парадигми (стилове на програмиране), но в обучението място намират най-често само четири основни – императивно, обектно ориентирано, функционално и логическо програмиране. Всяка от тези парадигми насърчава различен подход към моделиране на задачите и техните програмни решения, използва различен набор от концепции и механизми и се поддържа от различни езици за програмиране. Популярните съвременни езици за програмиране позволяват прилагането на техники от поне две парадигми.

В училищния курс по информатика се придобиват знания за императивно и обектно ориентирано програмиране (ООП), като в [2] акцентът е изместен към второто. Визуалното програмиране също се класифицира като отделна програмна парадигма и на него се отделя сериозно внимание в учебния курс, но тази парадигма не е сред основните и има силна връзка с ООП.

В обзорния доклад CC2001 [11], даващ препоръки за организирането на обучението по компютърни науки, са описани подробно шест класически реализации на модели на уводни курсове. Три от тях следват подхода *programming-first*, а останалите три – алтернативни подходи.

В основата на педагогическия подход *programming-first* стои убеждението, че програмирането трябва да се включи на ранен етап от обучението. Така учениците (или студентите) ще придобият рано ключови за областта знания и умения. Имплементации на този модел са подходите *imperative-first*, *objects-first* и *functional-first*, особеностите на които ще коментираме накратко:

– *imperative-first* реализира увода в програмирането, следвайки историческото развитие на парадигмите и езиците за програмиране – започва с основните понятия от процедурното програмиране, след което преминава към обектите. Изучаваните теми в уводния курс включват типове данни, управляващи конструкции, функции, масиви, файлове, както и техники за тестване и дебъгване на програмите. Тук проблемът е, че обучаемите имат по-малко време за запознаване с обектно ориентираните концепции;

– *objects-first* подходът акцентира на програмирането в началото на компютърното обучение, но препоръчва възможно най-ранно въвеждане на принципите на обектно ориентирания дизайн и програмиране. Основни понятия са „обект" и „клас". Ранно се въвеждат и ключовите механизми наследяване и полиморфизъм. За да могат обучаемите да експериментират на този ранен етап, те трябва да използват готови или полуготови интерактивни програми. Традиционните процедурни конструкции се въвеждат паралелно на обектните, като допълващи основните идеи. Предимство на този подход е това, че обектите в програмата кореспондират директно с обектите от предметната област и това помага, особено на учениците, да разберат по-добре какво точно правят техните програми. Подходът има и недостатъци – основните елементи, чрез които се реализират алгоритмите (променливи, изрази, условна логика, цикли и др.), са засенчени от допълнителните нива на абстракция, които внасят класовете и обектите;

– *functional-first:* използва се функционален език за програмиране в уводния курс, след което се прави преход към обектно ориентиран език. Предимство на подхода е олекотеният синтаксис. Обучаемите могат да създават самостоятелни, работещи и разбираеми програми от самото

начало. Понятия като „свързани списъци“, дървета, „функции“ и „рекурсия“ се въвеждат на доста по-ранен етап от обучението. Недостатък на този подход е проблемът с мотивацията на обучаемите, защото се изучава език, който е подходящ повече за научна работа. Подходът изисква добре развито абстрактно мислене и затова не е подходящ за ученици.

Недостатък на *programming-first* подходите е, че са съсредоточени в голяма степен върху техниките за програмиране. Често се получава акцентиране върху синтактичните конструкции на конкретния език за програмиране, вместо да се развиват алгоритмичното мислене и умения за правилно моделиране на поставените задачи в термините на използваната парадигма. Когато става въпрос за обучение на деца, *programming-first* курсовете трябва да претърпят сериозни корекции. Затова в учебните програми по компютърно моделиране за III и IV клас [4][3], където има елементи на програмиране, този подход е приложен само частично. Преди да достигнат до същинско програмиране (писане на код), учениците изграждат програмите си в средата Scratch или Kodu блоково – чрез словесно описание на алгоритми, чрез визуални компоненти (обекти) и под формата на игри. По този начин алгоритмичното мислене се развива, без да се налага писането на чист код.

Известни алтернативи на модела *programming-first* са:

– *breadth-first*: паралелно с изучаването на програмиране, алгоритми и структури от данни обучаемите изучават понятия от математиката, компютърните мрежи и архитектури. Така получават по-широка основа;

– *algorithms-first*: основните понятия се въвеждат чрез псевдокод, вместо да се използва конкретен език за програмиране. Освобождаването на обучаемите от синтактичните подробности, свързани с успешното компилиране на програмите, позволява те да се концентрират върху осмислянето и обясняването на алгоритмите, които конструират, проследявайки мислено тяхното изпълнение. Сред сериозните недостатъци на този модел е проблемът с мотивацията. Обучаемите идват с желанието да се научат да управляват силата на компютрите и са разочаровани, когато не виждат директно резултата от труда си;

– *hardware-first*: започва се от машинното ниво и постепенно се преминава към по-абстрактните понятия. Приложим подход за технически училища и университети.

За пълнота ще спомена и няколко съвременни алтернативи:

– *concepts-first:* приложим за изучаване на всички програмни парадигми, включително паралелно програмиране и използва специално проектиран за целта изчистен език. Акцентира върху важните за парадигмата концепции, без да обременява със сложни синтактични конструкции. Така се избягват част от недостатъците на *programming-first* подходите;

– *fundamentals-first*: идеята на този подход е да се започне първо с най-важните концепции, принципи и механизми, след което да се премине към решаването на практически проблеми и езиковите особености. Подходящ подход за обектно ориентираната парадигма;

– *GUI-first:* твърди, че обучението по програмиране трябва да започне директно със създаването на приложения с графичен потребителски интерфейс. Варианти на такъв подход се срещат все по-често, независимо дали става въпрос за обучение на ученици или студенти. Предимствата на този подход са очевидни – приложенията с графичен интерфейс са познати и по-интересни за обучаемите. Главен недостатък е, че много дълго се отлага във времето изграждането на алгоритмично мислене. Късното усвояване на процедурните концепции води до трудности при имплементирането на алгоритмите в програмата;

– *games-first*: естествено продължение на *GUI-first* с акцент върху програмирането на игри;

– *model-first*: вариант на *objects-first*, при който се отделя повишено внимание на концептуалното моделиране. Предимства: по-дълбоко разбиране на процеса на програмиране; фокусиране върху основните програмни конструкции на конкретен език за програмиране;

– *design-first*: този подход е друг вариант на *objects-first*. Той е много близък до *model-first* и залага на идеята, че обучаемите научават най-добре това, което учат първо. Затова те трябва да започнат с придобиване на умения за решаване на проблеми в обектно ориентиран стил, тоест с основите на обектно ориентирания анализ и проектиране. Слабост на този подход е, че поради ограничения в хорариума използваните примери всъщност не представят реален обектно ориентиран дизайн.

Интерес представляват още няколко подхода за въведение в информатиката, които са съобразени с възрастовите особености на учениците и дават положителни резултати:

– *unplugged* подход: преподаване на компютърни науки, но без компютри. Счита се, че помага на учениците да разберат абстрактни понятия и да задълбочат теоретичните си познания;

– *Controlling Physical Systems* – програмиране, управление и игра със специално разработени за обучението физически системи – роботи, сензори, платки, транспортни системи и други, които подпомагат учебния процес. Нашият опит показва, че такъв подход е мотивиращ, особено за по-малките. Освен да програмират, учениците имат възможност да проектират, конструират и всичко това се случва пред очите им. Те участват активно в целия процес. Има налични много дидактически инструменти за реализация на подхода – BeeBots, LEGO WeDo, Sphero, LEGO, Micro:bit.

НАШИЯТ ПОДХОД ЗА УЧИЛИЩЕН КУРС ПО ИНФОРМАТИКА

По същество подходът, който предлагаме в тази статия, е вариант на *objects-first* стратегията. Обектите се въвеждат рано (*objects-early*), но без да се подценяват процедурните конструкции. Практиката ни показва, че в обучението по информатика най-удачно е да се комбинират идеи от няколко подхода. Нашият вариант залага на допълнителна мотивация на учениците чрез:

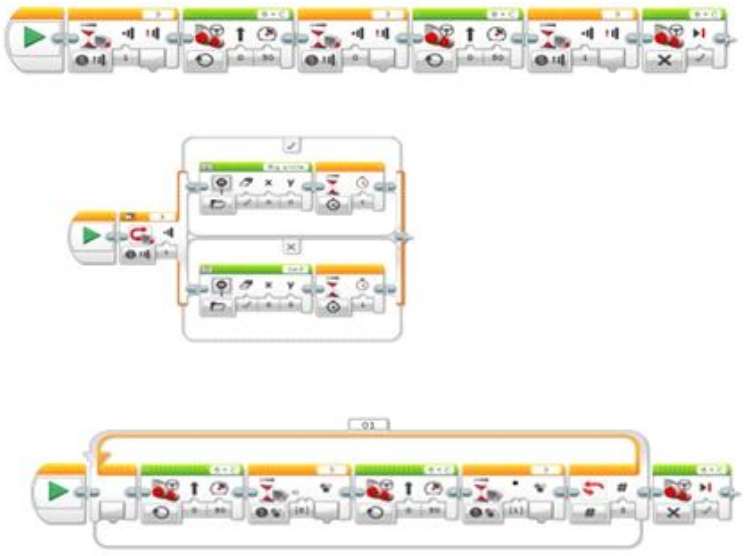
– използване на съвременна среда за разработка (актуална версия на Visual Studio) и изграждане на приложения с графичен интерфейс (WPF и Windows Forms);

– управление на физически системи.

За съвременните ученици е важно да виждат ползата от изучавания материал, затова още в самото начало, преди да се премине към писането на код, под формата на демонстрация, те получават нагледна представа защо е нужно да учат всичко това. Демонстрациите включват софтуер за библиотека, за резервация на стаи в хотел или управление на обекти в средата Scratch чрез прилагане на различни функции върху тях.

След демонстрацията на различен вид готов софтуер и използването на програмируеми, интерактивни ресурси, учебните часове следват предвидените в учебната програма [2] теми. Направената пропедевтика помага на учениците да добият представа защо е нужно да се изучават процедурните елементи (условен оператор, циклични алгоритми, функции). Отново се връщаме към демонстрациите при изучаване на понятието „алгоритъм" и видовете алгоритми, където включваме LEGO робот. На учениците се дават предварително подготвени от преподавателя примери, с които те проследява движението на робота при трите вида алгоритми (Фиг. 1.).

Основно понятие в програмирането е „тип данни". За да мотивираме въвеждането му, обсъждаме характеристиките на софтуер, необходим за касите в хранителен магазин. С насочващи въпроси учениците сами достигат до същността на новото понятие и предлагат правилните типове данни за различните стоки в магазина. Въпросите трябва да подтикват учениците да открият ключовите абстракции и механизми в проблемната област. Те сами трябва да достигнат до основното понятие. Това е и една от идеите на подхода. Така се задържа вниманието на учениците и те остават активни, като участват през целия учебен процес. Разглеждаме примерни случаи на употреба (*use cases*) – На касата сме и искаме да си купим 5 шоколада. Какви са действията



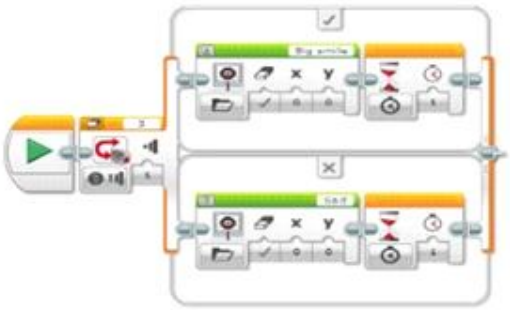
Фиг. 1.
Демонстрация за линеен, разклонен и цикличен алгоритъм

на касиерката? Касиерката къде въвежда тези 5 броя шоколад? Какво е числото, което е въведено? Съществените въпроси са „Какво?“, „Защо?“ и „Как?“.

Подходящи за демонстриране на примери са блокчетата на Scratch и LEGO Mindstorms, показани на Фиг. 2.

Фиг. 2.
Демонстрация на типове данни в Scratch и LEGO Mindstorms

Темите за условен оператор и цикли алгоритми отново започват с демонстрация на LEGO. За визуализиране на условен оператор се използва и сензорът за допир.



Фиг. 3.
Условен оператор в LEGO Mindstorms

Мисията на робота е, след като бъде докоснат, да се усмихне, а ако не е – да покаже тъжно лице (Фиг. 3.).

При демонстрацията на процедурните елементи може да се използват също Micro:bit и Scratch. Добри онлайн дидактически инструменти чрез игри се предлагат и от специализираните сайтове [12] и [13].

Важна част от нашия подход са примерите и задачите, които развиват уменията и затвърждават знанията на учениците. След въвеждането на всяко от заложените в учебната програма понятия е необходимо оперирането с него. Предпочитаните от нас задачи са с приложен характер: създаване на различни видове калкулатори – графичен, геометричен, изчисляване на индекс на телесната маса; приложение за наемане на автомобили; игра на морски шах и други. Опитът ни сочи, че силно положително въздействие оказва създаването на игри.

ЗАКЛЮЧЕНИЕ

Обучението по информатика и компютърни науки винаги е било предизвикателство. Трудностите произлизат както от непрекъснатото развитие на технологиите, така и от променящия се психологически профил на учениците. Експериментирането с нови подходи, средства и инструменти е част от работата на всеки педагог. Отчитайки съвременните изисквания, предложеният от нас подход обединява идеи и добри практики от класическите подходи с прилагането на иновативни инструменти – използва програмирането като средство за изучаване на компютърните концепции; въвежда рано обектите и развива обектното мислене, но без да подценява процедурните конструкции, без които е невъзможно реализирането на алгоритми. Съществен елемент на подхода е мотивирането на учениците чрез разработването на приложения с модерен интерфейс в съвременна среда, решаващи проблеми от тяхното ежедневие, демонстрациите с робот и визуализациите на алгоритми.

Предложените примери и задачи могат да послужат за обогатяване на педагогическата практика на учителите в средното училище.

Поради краткото време, в което прилагаме подхода, все още не можем да дадем точни емпирични данни за ефекта от него. В момента тече дидактически експеримент и изчакваме да тестваме върху по-голям брой ученици. Но въпреки това може убедено да се твърди, че се наблюдават положителни промени по отношение на повишаване интереса към програмирането, изграждането на добри модели на решения на задачите, активно участие в учебните часове, което в крайна сметка води и до по-високи резултати.

ЛИТЕРАТУРА

[1] МОН. 2015. Наредба № 5 от 30 ноември 2015 г. за общообразователната подготовка, стр. 56 // MON. 2015. Naredba #5 ot 30 noemvri 2015 g. za obshtoobrazovatelnata podgotovka, str. 56, https://www.mon.bg/upload/2341/nrdb5_30.11.2015_obshtoobr_podgotovka_1.pdf

[2] МОН. 2016. Учебна програма по информатика за VIII клас (общообразователна подготовка), // MON. 2016. Uchebna programa po informatika za VIII klas, https://www.mon.bg/upload/13463/UP_8kl_Informatika_ZP.pdf

[3] МОН. 2017. Учебна програма по Компютърно моделиране за IV клас. // MON. 2017. Uchebna programa po Kompyutarno modelirane za IV klas, https://www.mon.bg/upload/13767/UP9_KM_ZP_4kl.pdf

[4] МОН. 2017. Учебна програма по Компютърно моделиране за III клас. // MON. 2017. Uchebna programa po kompyutarno modelirane za III klas, https://www.mon.bg/upload/12205/UP_KM_3kl.pdf

[5] **Booch, Gr., Maksimchuk, A., Engle, M., Young, B., Conallen, J., Houston, K. 2007.** Object-Oriented Analysis and Design with Applications, Third Edition, Addison-Wesley Profession, 2007, pp. 44-45

[6] **DeHaan, R. 2009.** Teaching Creativity and Inventive Problem Solving in Science, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2736021/

[7] **Qualls, A., Grant, M., Sherrell, L. 2011.** CS1 Students' Understanding of Computational Thinking Concepts. Journal of Computing Sciences in Colleges

[8] **Shaw, M. (ed.) 1981.** ALPHARD: Form and Content. New York, NY: Springer-Verlag, p. 6

[9] **Wing, J. 2006.** Computational Thinking, Communications of the ACM, 49, (3), 33–35, 2006.

[10] **Манев, Кр., Манева, Н., Христова, В. 2017.** Информатика 8. клас. Учебник по общообразователна подготовка. Изкуства, София, 2017. // **Manev, Kr., Maneva, N., Hristova, V. 2017.** Informatika 8. klas. Uchebnik po obshtoobrazovatelnata podgotovka. Sofia: Izkustva, 2017.

[11] The Joint Task Force on Computing Curricula. 2001. Computing Curricula 2001. Computer Science.

[12] Blockly Games, https://blockly-games.appspot.com/

[13] Desmos Classroom Activities, https://teacher.desmos.com/

ИНФОРМАЦИЯ ЗА АВТОРИТЕ

Габриела Чотова – докторант, Факултет „Математика и информатика", Великотърновски университет „Св. св. Кирил и Методий", e-mail: gabriela.chotova@gmail.com

Ивайло Дончев – доцент, доктор, Факултет „Математика и информатика", Великотърновски университет „Св. св. Кирил и Методий", e-mail: i.donchev@abv.bg

ABOUT THE AUTHORS

Gabriela Chotova – PhD student, Faculty of Mathematics and Informatics,
St. Cyril and St. Methodius University of Veliko Turnovo. E-mail: gabriela.chotova@gmail.com

Ivaylo Donchev – Associate Professor, Ph.D., Faculty of Mathematics and Informatics,
St. Cyril and St. Methodius University of Veliko Turnovo. E-mail: i.donchev@abv.bg