Engineering Courses on Computational Thinking Through Solving Problems in Artificial Intelligence

https://doi.org/10.3991/ijep.v7i3.6951

Piyanuch Silapachote(✉) and Ananta Srisuphab
Mahidol University, Thailand
piyanuch.sil@mahidol.edu

Abstract—Computational thinking sits at the core of every engineering and computing related discipline. It has increasingly emerged as its own subject in all levels of education. It is a powerful cornerstone for cognitive development, creative problem solving, algorithmic thinking and designs, and programming. How to effectively teach computational thinking skills poses real challenges and creates opportunities. Targeting entering computer science and engineering undergraduates, we resourcefully integrate elements from artificial intelligence (AI) into introductory computing courses. In addition to comprehension of the essence of computational thinking, practical exercises in AI enable inspirations of collaborative problem solving beyond abstraction, logical reasoning, critical and analytical thinking. Problems in machine intelligence systems intrinsically connect students to algorithmic oriented computing and essential mathematical foundations. Beyond knowledge representation, AI fosters a gentle introduction to data structures and algorithms. Focused on engaging mental tool, a computer is never a necessity. Neither coding nor programming is ever required. Instead, students enjoy constructivist classrooms designed to always be active, flexible, and highly dynamic. Learning to learn and reflecting on cognitive experiences, they rigorously construct knowledge from collectively solving exciting puzzles, competing in strategic games, and participating in intellectual discussions.

Keywords—computational thinking; artificial intelligence; collaborative problem solving; active learning; constructivism in college classrooms

1 Introduction and related works

Teaching and learning computational thinking or CT, were originally realized over three and a half decades ago [1]. It is marked by Seymour Papert's revolutionary book *Mindstorms: Children, Computers, and Powerful Ideas* [2] that was first published in 1980. It brought computer literacy into primary education. The aim was to draw out constructionism in every child and to leverage cognitive development, intellectual and mental abilities to acquire, transform, and consolidate knowledge. Papert discussed his decade-long experience with the Logo programming language he helped design at Massachusetts Institute of Technology in the 70s. Logo is linked back to Jean Piaget's psychological theory of constructivism [3] and a predecessor of today's popular Lego

Mindstorms series [4]. Logo, a Greek name for thought, is a language for learning that is graphic and logic oriented. Learning to program hands-on, Logo enables preschool-aged children to create visual graphics by commanding and controlling of a pixelated turtle robot. A turtle moves forward, turns left or right, relative to its own position. As a turtle is instructed to move, a pen that is attached to it draws onscreen line graphics and complex geometric shapes. Thence, understanding of the underlying mathematics is genuinely embodied. Moreover, Logo features a body-syntonic reasoning. Children determine the outcomes of their turtle graphics by imagining themselves as a turtle being programmed and imitating what happens.

Computational thinking resurfaced in a 2006 communication of ACM (Association for Computing Machinery) by Wing [5] and has since been gaining strong momentum worldwide. CT has much influence on several aspects of curriculum designs from primary and secondary educations to undergraduate courses. Wing states:

"Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science."

Wing highlighted the richness of abstract thinking in computing and computer science or CS. She placed an emphasis on the complexity of information when it is encoded in multiple layers, including both intra- and inter-relations between and across layers [6]. These mental abstractions are automated by computing machinery.

Designing a course on computational thinking often reflects on computer science and programming. The three terms, though strikingly related, bear different messages and interpretations. CT emerged in mathematics and CS but CS is certainly not the only discipline CT is practiced in, is applied to, or has impact on [1]. Computational thinking is a skill whilst computer science is an academic subject. Programming for computer scientists and engineers is analogous to constructing proofs in mathematics. Programming is neither necessary nor required to develop computational thinking, but mastery of CT skills is vitally crucial for effective learning of programming [7].

Computational thinking has actively been installed into a large number of schools and universities worldwide. Despite variation in details, the fundamental principles of the concepts of CT curriculum remain the same. Wing [5] kicked it off with solving a complex problem by reducing it to simple ones where we already know a solution or an approach. She underscored abstractions, recursive procedures, heuristic methods, parallel processing, and conceptions of scales. Lu [7] focused on vocabularies and notations comprising CT. Barr [8], on bringing CT to K12, asserted the capabilities of data collection, analysis, and representation. The International Society for Technology in Education and the Computer Science Teachers Association put more emphasis on operational definitions [9]. The College Board for Advanced Placement (AP) in CS underlined communications and collaborations [10]. From the United States to the United Kingdom, Computing at School, the forefront of the new Computing National Curriculum, described not only the conceptual framework of CT but also pedagogic techniques that demonstrate CT skills: reflecting, coding, designing, analyzing, and applying. These constitute the scientific methods for computer scientists [11].

Not only have schools incorporated computational thinking into their curriculums, but universities have also added CT as an introductory CS and engineering course or as a general education for other disciplines. Genuine implementations of CT classes, nevertheless, are still largely varied. There is not yet a universally adopted syllabus or a comprehensive textbook. Instead, the community relies on shared resources as the field rapidly evolves. There is an excellent mix of old-time mathematical games, logic puzzles, and a decent number of creative activities to learn basic sorting techniques. The CS Unplugged Project of New Zealand is infamous for learning computer science without a computer [12]. It provides free indoor and outdoor learning activities to engage learners of all ages to computational thinking. Those who prefer undertaking it with block programming environment are familiar with Scratch [13], which allows creators to create interactive arts, animations, stories, and games. Providing a stepping stone, embarking on an intensive track, computational thinking can be rigidly coupled with conventional high-level programming languages such as Python, Java, or C++.

Approaching CT from a different perspective, we present pedagogical benefits and designs of an integration of classical and contemporary elements from AI – artificial intelligence into an introductory course for CS and engineering undergraduates. We propose a comprehensive module on CT comprising a series of carefully selected AI puzzles and games that have been deliberately organized by not only the levels of CT skills required to solve the problems but also coherence, continuity, and cohesion of the underlying fundamentals in data structures and algorithms. Student performance is compared with the same course offered a year earlier before AI was blended in.

In what follows, rationale behind our choice of AI for CT is provided in Section 2. Sections 3-6 describe the structure of our classrooms. We primarily focus on an active learning environment, involving collaborative problem solving (CPS) strategies. How characteristics of AI problems uniquely foster students' motivations and engagements are discussed. AI effectively captures student attention and encourage their cognitive reasoning. Following in Sections 7-10 are the different constituent components of CT. We demonstrate how they are complemented by a mix of problems from intelligent systems and computational intelligence. Drawing a path toward machine automation with inspiration from ordinary applications, our course goes from the art of problem solving to data structures and algorithms while a computer or a source code is never a requirement. We discuss results and give a concluding remark in Section 11 and 12.

2 Rationale for integrating AI into CT for introductory engineering and computing courses

A major challenge in designing an active and collaborative problem solving class is an exhaustive selection of intellectual activities and exercises, that not only promotes computational thinking but is also practically interesting and pleasantly enjoyable. We conjecture that problem solving alone is too broad when targeting computer science and engineering entering undergraduate students. Algorithmic-oriented constructivist activities efficaciously offer much greater benefits, involving the fundamental control flow structures in programming: sequential, selection, and repetition.

We embrace artificial intelligence because AI continues on a rise, dominating the field of computer science and influencing interdisciplinary research. Not only has AI become a core requirement for most third and fourth year undergraduates in CS, but students can also easily relate to intelligent systems, from everyday board games to embedded features in familiar online and mobile applications. Being connected to a domain at hand naturally gains attention. Consequently, maintaining the momentum becomes painless. Problems in AI are hard but they can be neatly broken down into multiple levels of difficulties and complexities for diverse groups of students. Settings can be standalone sessions or a small series. More importantly, solving problems in artificial intelligence is comprehensively completed with various components that are perfectly aligned with computational thinking skills. Elementary ingredients comprise logical reasoning, critical and analytical thinking, foundations of mathematics, and decision-making. Lateral thinking and creativity are indispensable in designing state representations. An intermediate level of systematic planning process is a necessity to analyze flows of control. Abstraction and abstract thinking are chiefly imperative to a visualization of both data structures and algorithms. Details of these computational thinking aspects that are embedded in AI problems are presented in Sections 7-10.

3 Structuring an introductory course on computational thinking for entering CS and engineering undergraduates

Regardless of its specific name, the underlying principles of a CT course remain unchanged. A course on computational thinking is designed to connect learners with computing, to build logical and analytical skills necessary to efficaciously formulate a problem from its specification, and to develop algorithmic thinking. Computational thinking is crucial to programming. Without a practice on computational thinking, for many even an introduction to programming is awfully difficult. A computer can be programed to turn a complex task into a simple one. Struggling to work out a problem computationally in the first place, how could one devise and solve it using syntax and semantics of a never seen before programming language? We design and structure our computational thinking course as a prerequisite to programming. Primarily focused on evolving mental tools and developing metacognitive skills, no computing machine is ever required. Key elements of our lesson plan are summarized in Table 1. Our active constructivist college classrooms are described in Section 4, student motivations and engagements in Section 5, and the realization of computational thinking in Section 6.

Table 1. Keys to facilitate collaborative computational thinking in active learning classrooms

Learning Environment	**Encourage Engagements**	**Unlock Unforeseen Usages of CT**
Teach less, learn more	Tangible and relatable	Reinforced exercises
Structured activities	Challenge with a twist	Interactive simulations
Student centered	Competitive games	Focus on a specific goal
Highly dynamic	Enjoyable and fun	Demo of advanced courses

4 The classroom environment uses active learning approaches with a focus on collaborative problem solving strategies

Designing any skill-based subject always poses a challenge. Neither lecture-based teaching nor rote learning approaches are appropriate or beneficial. Developing a skill requires significant amount of hands-on experiences on several practical exercises. As Bonwell and Eison [14] summarized, exclusively and passively receiving information from lecture constrains student learning. Students learn more being active. They must be engaged and devoted physical and psychological energy into academic experience. Students should be doing things besides listening. College classroom discussions are preferable to lectures when the learning objectives involve developing thinking skills and applying knowledge to new circumstances [15]. Discussion, reflection, as well as attitude play a major role in active teaching and learning models, especially to invoke higher-order thinking, including analysis, synthesis, and evaluation [16].

We have setup active classrooms where students spend most of their class-time on learning-by-doing. We provide large open space for everyone to learn, to express their thoughts and ideas, and to argue or to criticize contrasting viewpoints. Students can gradually construct their own intellectual knowledge; instructors are there to facilitate, to coach, to guide, and to provide a network of support system. Learning environment is generally friendly and largely dynamic. Student performances are constantly being observed and evaluated on an individual basis, considering academic achievements as well as efforts. Being learner-centric, while the objectives and the outcomes of every class are rigidly maintained, detailed implementations are adjusted in accordance with the needs and the desires of the entire class.

No classes are passive. Chalk-and-talk lectures are used only when instructions are given. They are kept short, delivered concisely in no more than ten-to-fifteen minutes block of time. This is in alignment with a short attention span of today's digital native students. A new challenge is presented in every session. Problems are meticulously selected and carefully sequenced by their relative difficulties and in accordance with Bloom's taxonomy of cognitive learning domain that is remembering, understanding, applying, analyzing, evaluating, and creating [16]. Though some are individual tasks, most assignments require collective teamwork and good team effort. In the following, we highlight three peer-elements that flawlessly complement our active constructivist classroom experience. They are peer discussion, peer teaching, and peer reflection.

4.1 Peer discussion

Students work in small groups, usually groups of three, four, or five. Everyone is strongly encouraged to get involved and to fully engage in every step of solving every problem. A group works collaboratively to figure out a solution to a problem at hand. Intellectual discussions are key to success. Group members are constantly discussing with one another. As facilitators and coaches, instructors have to monitor and listen to their conversations. We join them not only when there are questions, but also when they accomplish a milestone or when they unfortunately take a turn off the right track.

Encouragement is of primary vitality. When a group is stuck, bogging down, feeling regrettably discouraged, instructors must be there to guide them through. We provide hints, give clues, pointers, and suggestions, but never give out a complete solution. Time and again we keep reminding students that their experience of solving problems and their ability to think computationally is our ultimate goal. Catching their feeling at the very second they arrive at a hard-earned solution is beyond rewarding; allowing students to reflect on their moments of success effectively build self-motivation and encouragement to overcome any obstacle and solve another problem with confidence.

4.2 Peer teaching

Learning by teaching others is placed at the top of the pyramid [17], exhibiting the maximum retention rate above every other learning approaches. Peer-teaching can be effectively and efficiently incorporated into any active classrooms. This is especially well suited for first-year undergraduates who may have entirely different high-school backgrounds. Some may be able to solve a particular problem easily, but have a hard time on another. Some may learn quickly, while others need a bit more time. Instead of reckoning on these as an issue in a classroom, we turn it around and have students assist one another. Teaching others allows them to reconsider details that they may have overlooked either accidentally or intentionally. Those who teach typically gain deeper understanding of the problem, the solution, and, more importantly, the process. The ones who learn gets to express their viewpoint and to find out how a friend solves it. A peer-teaching exercise remains a win-win situation so long as it does not turn to be an unconstructive competition (i.e. to aim at overtaking each other for better scores and grades). This is where a coach must always encourage and maintain constructive collaborative efforts, ensuring every student to learn together and to succeed together.

4.3 Peer reflection

Thinking about one’s own thinking process and learning experience is fundamental to metacognition and transformative learning. Engaged higher-order thinking process, students reflect on their cognitive knowledge and regulation. Along with a reflection, they were asked to comment about the class: anything good or bad, anything they like or dislike; related to the contents, teaching or learning approaches, or even friends or instructors. Social factors or cultural traits may hinder some to express comments and criticisms. We relaxed it and made it easier by announcing a reflection as optional and anonymous. We intentionally and openly paid no attention to who wrote each note and allowed a friend to drop it off in a shared box. Student reflections are invaluable, mirroring and recognizing their unique perspectives. As it turned out, the quicker we respond to the class, either verbally or in action, the more they speak up the next time. This echoes how their every voice is listened to and respected. Before long, students adjust. They become comfortable enough to reflect face-to-face on the spot. Particular attention is paid to students’ spontaneous reactions and negative feedback, so changes toward improvements could be made. Students truly appreciate it when their opinions matter; their participations subsequently increase; they start learning with happiness.

5 AI problems are tangible and genuinely practical; naturally and effectively fostering student motivation and engagement

George Polya brilliantly summarized a process of finding a solution to any problem in four simple but powerful steps, none of which should be overlooked. He described it in his exceptionally wonderful book, "How to Solve It," published in 1945 [18]:

"First, we have to understand the problem; we have to see clearly what is required. Second, we have to see how the various items are connected, how the unknown is linked to the data, in order to obtain the idea of the solution, to make a plan. Third, we carry out our plan. Fourth, we look back at the completed solution, we review and discuss it."

Problems in artificial intelligence possess unparalleled characteristics that tailor to these four phrases of Polya. A number of AI problems are naturally tangible, which significantly aids in understanding of the various constituent components, conditions, data, and the unknowns. Having background knowledge or prior involvements makes things easier. It provides an insight and creates a connection; enabling individuals to imagine it deeper and clearer. These can be things we have seen before with an added twist, considering an old task from an unexpectedly different angle. With little or no confusion on what is required, the focus is fully attended to the challenges of creative thinking. Examples of AI-related problems that incorporate machine intelligence into age-old board games are tic-tac-toe, checker, chess, or the game of Go; most of which involve search strategies and analysis of an enormously large and difficult to evaluate search space. Observing how a computer plays and wins the game automatically draw out individual interests, engagements, and motivations to seek solutions.

Another preeminent feature of problems in AI domain is that they typically appear complicated at first glance but the underlying implementation is surprisingly simple. For example, it sounds rather difficult and complicated, especially to novice engineers and programmers, to write an application for a computer to play a perfect game of tic-tac-toe against a human. A 2D game-board, nonetheless, can simply be modeled using either one- or two-dimensional arrays and a gameplay can easily be simulated through a series of selection procedures. Being assigned a seemingly difficult task, students frequently display a frantic look, calling out that it is unfair. It is not only challenging but simply beyond their capabilities and the scope of the class. We patiently wait until their experience would turn them around after they have arrived at a solution in a fair amount of time. Instead of being discouraged or intimidated, this learning experience genuinely enhances the joy of accomplishment. They proudly and successfully solve the problem that they originally imagined to be rather impossible. This further triggers inspiration and motivation. Students become excited about their next challenges in the upcoming classes. They look forward to getting their hands dirty on another problem.

Looking back at the process of figuring out such an AI problem, although it is not extremely demanding, good planning is without a doubt required (i.e. Polya's second principle). Recollecting from their past experience, students discover how a computer must move its game pieces in order to win against a human who may play optimally. They guess and check, try trial and error, eliminate possibilities, look for patterns or special cases, or devise a rule or a heuristic method. In carrying out a plan, instead of working on a real computer application (this is a computational thinking course with no programming required), students undertake roleplaying. One team acts as a robot, the computer they have designed, and the others are just being themselves, humans. This energetically creates a lively, cheerfully entertaining, and stimulating classroom, in which students are unlikely to be asleep. The ability to transparently incorporate a variety of competitions into problem solving and computational thinking activities is another favorable peculiarity intrinsic to puzzles and games in artificial intelligence. Structurally constructive tournaments are practically powerful in college educations; they effectively gain and maintain students' attention and engagement [19].

During the fourth and last phase, different solutions are compared, commented, and analyzed. Together with the class, we reflect on pros and cons of varying approaches, flaws or drawbacks, and plausible improvements. Student performances and efforts are assessed and evaluated. The next activities are then selected accordingly, filling-in student curiosities and those skills that appear lacking. To this end, AI offers not only a wide range of diverse problems, but also variations of similar problems and distinct problems that employ similar procedures. As an example in connection with tic-tac-toe, a two-dimensional lattice mesh can be customized, creating contrasting AI search problems. Constraints can be mixed and matched, forcing different search strategies. These practicable exercises not only reinforce the previously learnt concepts but also allow knowledge building from something students already know.

6 Simulations and live-demonstrations unlock the unforeseen necessities and uses of computational thinking skills

Especially early on in their undergraduate studies, students often question lessons being delivered: what they are for and why they have to do seemingly useless works. Courses with abstract contents, computational thinking clearly included, fall into the high-risk, if not dangerous, zone in this particular aspect. Importance of these skills is often not realized until they are inevitably needed; worse yet it is generally difficult to convince students of things they cannot foresee. Realistic realizations of applications in artificial intelligence persuasively relax these constrains. Students are able to focus on an unambiguous objective such as reaching a goal state or winning the game. A live-demonstration of a complete and interactive AI program after every CPS session effectively boosts the tangibility of a problem (in some occasions, a demonstration is advantageous in the beginning of the activity when a problem is introduced). Students are thrilled to see a working system, being convinced that it is indeed plausible. We let students test the program first-handed. Just providing them space to learn, explore, experiment, and discover, many have come up with several ingenious questions and innovative ideas. An intellectual discussion then continues. Furthermore, they enjoy peeking at its constituent parts that will be covered in advanced courses later in their curriculum. It firmly reassures them that what they are studying today will be useful in a near future and provides better realization of what they are preparing for.

7 Solving AI problems is a comprehensive exercise for crafting computational thinking toward machine automation

We designed our course by aligning several problems in AI with fundamental CT skills. Creative CPS classroom activities were carefully planned in ascending order of their complexity (following Bloom's taxonomy of cognitive learning domain [16]). In Table 2, we organize the core essences of CT into conceptual understanding, practical skills, and operational approaches, each divided into a problem solving phase, a path toward automation (data structures and algorithms without coding nor programming), and everyday innovative applications in social functions. Implementations of these computational thinking aspects using artificial intelligence problems and exercises are described below: elementary problem solving in Section 8, intermediate algorithmic thinking in Sections 9, and 10. We discuss results and conclude in Section 11 and 12.

Table 2. Essences of CT are organized into conceptual foundations, compulsory skill sets, and how to approach them hands-on. Learning of these CT skills are divided into three successive phases: elementary problem solving, intermediate algorithmic thinking toward automation, and their everyday functions in innovative applications.

Conceptual Understanding	**Skills and Practical Implementations**	**Hands-on Approaches**
Problem solving	Recognizing patterns	Tinkering
- Problem formulation	Mathematical modeling	Exploration
- Problem decomposition	Logical and lateral thinking	Creative designs
- Representation of states	Multiple layers of abstraction	Trials and errors
Toward automation	Data virtualization	Tracing
- Data structures	Systematic thinking	Debugging
- Flows of controls	Algorithmic insights	Simulation
- Recursive procedures	Step-by-step instructions	Perseverance
Innovative applications	Computer programming	Teamwork
- CT is for everyone	Evaluations and analysis	Collaboration
- CT is everywhere	Scales and parallel computing	Communication

8 AI problems instinctively promote problem solving skills

Solving problems in artificial intelligence requires computational and systematic thinking. Each involves several problem solving paradigms: guess and check, drawing diagrams, creating tables, looking for patterns, using variables, or working backward [18]. AI problems typically have not one, but multiple correct solutions. This allows students to actively brainstorm in groups, to explore various designs, to defend and to debate their ideas, to compare and contrast pros and cons of different methodologies. Through discussions, not only is their critical and analytical reasoning reinforced but their communication skill is also challenged. They need to figure out a way to clearly and completely express their thoughts either verbally or in writing to let their friends understand. Describing logical reasoning is not as simple as it may appear; practicing it is key. Individually or collaboratively, students search for a solution by playing and tinkering. Through trials and errors, they succeed in one step and fail in another. Once one solution is found, attempting to find another may trigger them to think outside the box, to approach the same problem from a different angle. Ultimately, arriving at the best possible answer is second to developing their computational thinking skills.

A number of steps are expected in figuring out most AI problems. Spontaneously and automatically, this force one to write down his or her solution in smaller steps. It is, however, rarely a straightforward task to write down a state, i.e. a snapshot in time of the environment and the agents in it. Students instinctively draw out their creative side, applying lateral thinking skill. Without formal lessons, they are literally learning knowledge representation and reasoning in both computational thinking and artificial intelligence. They formulate a problem, defining and parsing their own computational language. Solving it, they break a problem down into smaller, more manageable sub-problems that are easier and simpler to solve. In effect, the essence of abstraction is implicitly being employed; indispensable specifics are highlighted whereas irrelevant and minor details are discarded. All these aforementioned skills comprise an integral component in both problem solving and computational thinking.

As a concrete example, we successfully conducted a lively class utilizing a classic AI missionaries and cannibals puzzle [20], often referred to as river-crossing problem. Using only one two-person boat, the goal is to transport three missionaries and three cannibals from one side of the river to the other without any missionaries being eaten. Missionaries remain safe on either river bank so long as they are not outnumbered by cannibals. We provided six stone pieces, three white and three black. These supplies not only got them started, but also made it much more enjoyable. Some groups folded a boat origami; others used their pencil case as a boat. After a few trials and errors, most were quick to find some solution. Then, we revealed the real challenge: asking students to present their solution step-by-step on a piece of A4 paper. Showing off their beautiful artistic skills, most immediately opted for drawing a pictorial diagram.

At this stage, students had completed a problem solving step. We now guided them to the next phase of thinking computationally – how to formulate and represent this river crossing puzzle in such a way that it can later be programmed on a computer. The representation of the problem is always crucial. A bad representation could make it significantly more difficult to solve. Taking into account conditional constraints, we gently blended in the use of numerical tuples and ordered pairs for representation of states, leading to a concise solution proposed by Amarel in 1971 [21]:

$$331 \rightarrow 310 \rightarrow 321 \rightarrow 300 \rightarrow 311 \rightarrow 110 \rightarrow 221 \rightarrow 020 \rightarrow 031 \rightarrow 010 \rightarrow 021 \rightarrow 000$$

where each state consists of three numbers. It describes the number of missionaries, followed by the number of cannibals, both on the initial river bank. The last number indicates the side of the river the boat is on: 1 for the initial bank and 0 for the other.

It is worth reminding ourselves that students are naively solving problems assigned to them. Their approaches are derived solely from authentically practical viewpoints. Though relevant, their strategies are not based on any theoretical basis in AI. After all, this is a computational thinking module, not an artificial intelligence course. Formal conceptions of intelligent systems are neither taught nor expected. A large number of student solutions may not be well-structured but they do accomplish the objectives of the course that is to first-hand experience CT and to involve their metacognition.

9 AI problems neatly feature control flow structures

Control flows: sequential, selection, and repetition, are next to tackle. Once agreed on a representation (and a solution) of the river crossing puzzle, we questioned how students would program a computer so it performs the same or similar logical steps to search for solutions. Instead of a solution path, they sketch a search space. Lines were drawn, linking successive states. Arrows were used to signal a valid transition from one state to another. These basically reciprocate sequential flows. Constraints validate possible moves resulting in a number of states not drawn. Omitted states are either unreachable or invalid (containing more cannibals than missionaries or having a boat crossing the river without anyone rowing it). This process: *sequentially* enumerating all possible next states and *selecting* only states that are valid based on the constraints, iterates. It *repeats* until the goal state is reached, i.e. everyone safely crosses the river.

This exercise is the first installment of the fundamentals of control flow structures. Control flow stands at the core of computing concepts and programming routines. It is generally embedded in an assorted set of interesting problems and puzzles involving computational thinking and analysis. Its extensively broad range of applications does provide an excellent opportunity to revisit the ideas many times later in the course, reinforcing the cores and preparing students for their advanced computing courses.

Besides, at a completion of this activity, every student has a more or less complete but unstructured flowchart, i.e. a workflow or a diagram that represents an algorithmic process. This lays out the groundwork for their following lessons on flowchart and unified modeling language commonly covered in programming courses. By then, the topic will not be surprisingly new to students, making it easier for them to understand. They have seen it and have done it. Essentially, they got an entire blueprint. An added component will only be how to write it formally using proper standardized symbols.

10 AI problems allow practical explorations of fundamental data structures, algorithmic thinking and design, and recursion

Contents in many intelligence systems have strong linkage with data structures and algorithms. Designing and implementing a computer program to simulate an AI agent requires a solid comprehension of data representation, data processing, data stores, data flows, and algorithmic procedures. Although this is also undeniably true for most if not all programming projects in other disciplines, the field of AI uniquely offers superior variations in computational complexity both space and time. It covers a wide range: from basic to intermediate and advanced levels. As a result, an excellent mix of class activities can be designed to match with student interests and their abilities.

10.1 AI and games: revisiting state representations, followed by a practice in data and memory virtualization of one-dimensional structures

Arrays are the first data structures usually introduced after primitive data type, e.g. a character, integer, floating-point number, and boolean. Accomplished programmers have mastery over array indexing and manipulation, for instance, slicing or extracting subarrays, addition and subtraction of elements in arrays, comparison and non-linear operations on arrays. A familiar puzzle such as tic-tac-toe is a fine starting point for a one-dimensional array structure. Even though the physical board is a two-dimensional planar structure, implementing it with a nine-cell one-dimensional array is common in practice (illustrated in Fig.1a and Fig.1b). This exercise handily incorporates not only array indexing but also modular arithmetic. Mapping from one- to two- dimensional indexes, the column number in 2D is equal to its corresponding index in 1D modulo by 3, which is the number of elements in each row. Going further, one may formulate a binary representation of the state of the game and involve bitwise operations, all of which are mandatory in CT for computer science and engineering studies.

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

(a) 2D array indexing

0	1	2
3	4	5
6	7	8

(b) 1D array indexing

$(-1,-1)$	$(-1,0)$	$(-1,1)$
$(0,-1)$	$(0,0)$	$(0,1)$
$(1,-1)$	$(1,0)$	$(1,1)$

(c) changes of indexing for traversal

Fig. 1. Array indexing in one- and two-dimension

Aho and Ullman defined the field of computer science as *"the mechanization of abstraction… computer science is a science of abstraction – creating the right model for thinking about a problem and devising the appropriate mechanizable techniques to solve it"* [22]. By the same token, computational thinking involves automation of abstraction through computing [6]. Thenceforth, the significance of abstractions and data abstractions cannot be overstated. Proficiency in CT requires virtual visualization of data structures and memory allocations. We have found that it is never too early to introduce it to every computer science and engineering student; as a matter of fact, the earlier the better. Without further delay, abstraction of data structures can be promptly integrated into this part of our computational thinking sessions.

After students have informally learned the concepts of arrays (again, this is CT; it is not Data Structures), the class is divided into two teams to compete against each other. Any game that utilizes a simple one- or two-dimensional array structure can be installed. Familiar board games are good choices; tic-tac-toe is one example. Gaining a momentum by reckoning on natural competitiveness in college students, the game starts with a normal play. Challenges are gradually added as a competition continues. Our ultimate goal is to facilitate a gameplay that implicitly drives or even forces every student to construct his or her own virtual game pieces and board and virtually carry out moves on it. Notes, papers, pens, pencils, and all stationaries are disallowed. Each team instantaneously realizes that they need to agree on some representation on which their communications are based. Knowledge and state representation is reinforced and put to use. Effectiveness of their representation model is then right away tested.

Picking up the pace, talking and discussion among team members are not permitted and every member of the team must in turn call out a move. An instinctive desire to win, coupled with peer pressure not to make a false move, forces everyone to focus on data virtualization as the game progresses. Essentially, students are playing as a team while practicing virtualization skills as an individual. Our outcome of this activity was nothing less than impressive. Students energetically dove in; they had fun. Thinking it was easy they quickly sensed the barriers. Playing a familiar game in an unusual way was confusing at times, however, once they get the knack of it, the game advanced quite rapidly. Most importantly, it highlights the power of their mental tools. Tools of the mind is a concept attributed to Vygotsky [23]. He affirmed that cognitive learning and behavior in small children remain largely limited until they learn to use their own mental tools. Instructional strategies should aggressively foster cognitive functions, extending and transforming not only students' physical but also their mental abilities. Utilizing the tools of the mind to help us think logically and computationally, to solve problems, and to create solutions is a noticeably challenging task that requires a lot of experience and practice. It is an essential skill mastered by experienced programmers.

10.2 AI and search: traversing two-dimensional grid structures, implementing breath first search and depth first search strategies

The field of AI contains a large collection of logical problems on 2D grids that are fundamentally associated with graph or tree data structures. Intellectually entertaining puzzles and games keep students engaged while they are inevitably learning. We look at Wumpus World, an infamous toy example originally named Hunt the Wumpus and published in the Best of Creative Computing [24]. On a 4-by-4 square grid of a pitch-dark cave, an agent plays hide-and-seek to search for a pile of gold while avoid being killed by the Wumpus, a mysterious dangerous monster, and falling into a bottomless pit trap. Unable to directly see what lies ahead, the agent must strategically analyze and thoroughly plan its path based on clues it can perceived. A stench is present in the four cells neighboring the Wumpus and there is a breeze surrounding a pit.

Another intensive assignment, this fun puzzle excellently combines knowledge or state representation, critical thinking and logical reasoning, planning, plus axioms and situation calculus. To avoid dangers, a player, acting as an AI agent, needs to examine multiple sequential and conditional control flows all at once. Algorithmically, hunting the Wumpus entails exploration and search on two-dimensional arrays. An attempt not to overlook any cell during a search and not to visit any cell more than once leads to a systematic traversal on a two-dimensional grid. Ordinarily by intuition, students wind up performing either a breadth first search (BFS) or a depth first search (DFS). Both are common floodfill path finding routines on a 2D mesh or lattice structure. A live-demonstration of these search techniques on a computerized Wumpus World not only enhances their understanding, but also boosts their motivation and engagement. Connecting it to greater, more realistic applications that literally employ the same or similar search methods is an inspiration, providing insight into how far these simple strategies can go. Further extensions to queue and stack abstract data types for BFS and DFS, respectively, are optional, depending upon the dynamic of the entire class.

Attending to details on data structures and algorithms, programming how an agent makes a move on a 2D grid pushes the context of CT towards another aspect of array indexing. Illustrated in Fig.1c, visiting a neighboring cell involves a shift-by-one of an index in either the horizontal or the vertical direction. This is practically very useful in coding of an iterative or repetitive loop. An introduction of such concept for students is of particular interest. It makes the picture more complete and it is not at all difficult.

10.3 AI and recursion: recursive programming concepts, no coding required

Touching on DFS, recursion is practically around the corner. There are no reasons to pass up an opportunity to introduce it. One of the most challenging lessons thus far, tackling recursion for the first time is rather daunting and quite confusing. Beginners can get lost quickly after only a few steps. In spite of the dilemma, a careful tracing of a recursive routine is unavoidably the key to understanding it. An old-schooled pencil and paper practice goes a long way. Convincing a digital native to write this step-by-step, nonetheless, is a tougher concern. The work requires a willingly strong ambition, perseverance, and persistence to iteratively trace it through deeper steps and back. It appears repetitively tedious and painstaking. Any small mistake can potentially break the entire sequence and backtracking to find errors is even harder; this sounds scary. We pulled this off by making it a high stake but highly flexible and enjoyable project. Each student picks his own problem. Overlaying a photo or a drawing of interest onto a 2D grid, they recursively run DFS to flood-fill the area covered by their image. Each is free to decide how big or small of the image, with which they are willing to work. They make their own choice on the amount of details around the edges, the borders of the image, and to which level to carry out DFS. With a lot of encouragement, students gradually responded. The reward of this rigorous exercise was worthwhile. Students were not only extremely proud of their hard work but also indeed mastered recursion.

11 Results and discussions

Evaluating our design of CT with AI module, we tracked student performance in a required introductory computing course, *Problem Solving Techniques*, during two consecutive years at the Faculty of ICT, Mahidol Univ. The original course with 241 students focused on the traditional problem solving style. In the second year with 269 students, elements from AI were blended in. Table 3 details grade distributions of the entire class. With CT and an integration of AI in place of problem solving alone, the percentage of students achieving the highest grade was significantly greater and that of the lowest grades was significantly lower. Changing the course design necessitated us to reshape a performance evaluation rubric. Students has made every effort to think computationally to overcome unseen challenges in solving AI problems in every class rather than passively listen to chalk-and-talk lectures or follow recipe-style exercises. Emphasizing on the process, efforts contribute to a score; a grade is not solely based on answers in exams and physical attendance. Keeping it well balanced, however, no free lunch was handed out – the grade still reflects the student's grasp of the material.

Table 3. Grade distribution of a problem solving techniques course. Listed is the percentage of students getting grades: A, B, C, D or below. We compared two consecutive years: one followed a traditional problem solving style, the other is our approach with CT and AI.

	A	**B**	**C**	**D or below**
Our CT with AI course design	37.918	37.175	19.331	5.576
The traditional problem solving	19.087	55.187	16.598	9.129

12 Concluding remarks

We discussed and demonstrated how solving problems in artificial intelligence is an effective pedagogical approach and instructional design of computational thinking for entering undergraduates in engineering and computing fields of studies. Elements from AI align excellently with the foundations of CT skills and strongly engage every student in active exploration and discovery, constructing their own understanding and knowledge from hands-on experiences. Besides developing technical competence, AI activities spontaneously invite learners to communicate efficiently and effectively and to work collaboratively as a team. Every problem in artificial intelligence is unique; making it an exciting challenge. Each carries a decent amount of complexities that is moderately demanding and difficult for an average freshman to solve alone, whilst it is particularly suited for a small group. Moreover, AI computing can be methodically decomposed into multiple parts, enhancing the understanding of scales and extending to the realization of hierarchical and parallel processing. Beyond college classroom, a number of intelligent agents are repeatedly linked with common everyday application innovations, from frontend functions to behind the scene operations. This familiarity neatly provides continuously insight and creates endless inspiration.

13 References

[1] J. Voogt et al., “Computational thinking in compulsory education: Towards an agenda for research and practice,” *Education and Information Technologies*, vol. 20, no. 4, 2015. https://doi.org/10.1007/s10639-015-9412-6
[2] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic Books, Inc., 1980.
[3] E. Ackermann, “Piaget’s constructivism, Papert’s constructionism: what’s the difference?” *Future of Learning Group Publication*, 2001.
[4] J. Bumgardner, “The origins of Mindstorms,” Wired, 2007.
[5] J. M. Wing, “Computational thinking,” *Communications of the ACM*, vol. 49, no. 3, 2006. https://doi.org/10.1145/1118178.1118215
[6] J. M. Wing, “Computational thinking and thinking about computing,” *Philosophical trans. of the Royal Society Series A*, vol. 366, no. 1881, pp. 3717–3725, July 2008.
[7] J. Lu and G. Fletcher, “Thinking about computational thinking,” in *Proc. ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 260–264, 2009. https://doi.org/10.1145/1508865.1508959
[8] V. Barr and C. Stephenson, “Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community?” *ACM Inroads*, 2011. https://doi.org/10.1145/1929887.1929905

[9] ISTE and CSTA, "Operational definition of computational thinking for K-12 educ.," 2011.
[10] College Board, *AP Computer Science Principles Course and Exam Description*, Fall 2016.
[11] A. Csizmadia et al., *Computational Thinking: A Guide for Teachers*. UK: Computing At School, Part of BCS–the Chartered Institute for IT, 2015.
[12] T. Bell et al., "CS Unplugged: school students doing real computing without computers," *The NZ Journal of Applied Computing and Information Technology*, vol. 13, 2009.
[13] M. Resnick et al., "Scratch: Programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, November 2009. https://doi.org/10.1145/1592761.1592779
[14] C. C. Bonwell and J. A. Eison, *Active Learning: Creating Excitement in the Classroom (ASHE-ERIC Higher Education Report)*. Jossey-Bass, 1991.
[15] W. J. McKeachie et al., *Teaching and Learning in the College Classroom. A Review of the Research Literature*. Ann Arbor: NCRIPTAL, The University of Michigan, 1987.
[16] L. Anderson et al., *A taxonomy for learning, teaching, and assessing: a revision of Bloom's taxonomy of educational objectives*. Longman, 2001.
[17] J. P. Lalley and R. H. Miller, "The Learning Pyramid: Does it point teachers in the right direction?" *Education*, vol. 128, pp. 64–79, 2007.
[18] G. Polya, *How to solve it, a new aspect of mathematical method*, 2nd ed. Princeton, 1973.
[19] P. Silapachote and A. Srisuphab, "Gaining and maintaining student attention through competitive activities in cooperative learning," in *Proc. IEEE EDUCON*, 2014, pp. 295–8.
[20] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, Upper Saddle River, New Jersey, 2009.
[21] S. Amarel, "On representation of problems of reasoning about action," in *Machine Intelligence 3*, D. Michie, Ed. Edinburgh University Press, 1971, pp. 131–171.
[22] A. V. Aho and J. D. Ullman, *Foundations of Computer Science*. New York, NY, USA: Computer Science Press, Inc., 1992.
[23] E. Bodrova and D. Leong, *Tools of the Mind: the Vygotskian Approach to Early Childhood Education*. Englewood Cliffs, NJ: Merrill, 1996.
[24] G. Yob, "Hunt the Wumpus," in *The Best of Creative Computing*, D. Ahl, Ed. Morristown, NJ: Creative Computing Press, 1976, pp. 247–250.

14 Authors

Piyanuch Silapachote received the BS degree with Honors in Computer Science from Cornell University College of Engineering, the MS and PhD degrees in CS from the University of Massachusetts Amherst. She is an instructor in the Faculty of ICT at Mahidol University. Her primary research interests are in computer vision, AI, pattern analysis and recognition, image understanding and processing, machine learning, and biologically-inspired computing. Her interests extend to interdisciplinary research and innovative applications. She is active in computer science and engineering education.

Ananta Srisuphab received the BSc, MSc, and PhD degrees in Computer Science from Mahidol University. He had decade of experience working in the IT industry. He was an information services consultant at Unisys and a manager at Infosoft. Presently, he is an instructor in the Faculty of ICT at Mahidol University. His research includes computational intelligence, connectionist models, machine learning, image and signal processing, embedded systems, CS and engineering education, and ecoinformatics.

This article is an extended version of a paper presented at the 2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), held in December 2016, Bangkok, Thailand. Article submitted 30 March 2017. Published as resubmitted by the authors 29 April 2017.