Tabletop Games Designed to Promote Computational Thinking

Frederick J. Poole

Center for Language Teaching Advancement, Michigan State University, Lansing, United States

poolefre@msu.edu

Jody Clarke-Midura

Department of Instructional Technology and Learning Sciences, Utah State University, Logan, United States

jody.clarke@usu.edu

Melissa Rasmussen

Department of Computer Science, Utah State University, Logan, United States

melissa.ann.r@gmail.com

Umar Shehzad

Department of Instructional Technology and Learning Sciences, Utah State University, Logan, United States

agha.umar.s@gmail.com

Victor R. Lee

Graduate School of Education, Stanford University, Stanford, United States

vrlee@stanford.edu

This work was supported by National Science Foundation (NSF) under Grant [number 1837224]

Tabletop Games Designed to Promote Computational Thinking

Background and Context: There is a growing perception that computational thinking can be developed in unplugged environments. A recent trend among these unplugged approaches is the use of tabletop games. While there are many commercial tabletop games on the market that are promoted as teaching computer science learning and/or computational skills, there is little research to support these claims.

Objective: This study investigates the types of tabletop games that are currently being promoted as teaching or requiring computational thinking, who such games are marketed towards, and how game designs could provide opportunities for developing computational thinking.

Method: We conducted a content analysis to explore the type of tabletop games currently being created, their audiences, and the kinds of game mechanics and design features being implemented to teach computational thinking concepts. We present the results of our content analysis and provide design cases of three tabletop games to illustrate how different game designs afford opportunities for learning computational thinking concepts at different age levels.

Findings: In this study, we created a taxonomy of computational thinking tabletop games that identified three primary categories (e.g. code building, code executing, and puzzle games) and one category that includes a combination of the first three categories. Games that fall into our categories share similar learning claims, target audiences, and game mechanics.

Implications: Our taxonomy offers a starting place for instructors who want to explore the use of tabletop games for introducing computational thinking concepts in unplugged settings, suggestions for designers, and areas of investigation for researchers.

Keywords: Tabletop games, computational thinking, content analysis

Introduction

There is a growing perception that computational thinking (CT) can be developed and experienced in unplugged, or non-digital, environments and materials (Bell, et al., 2015). A recent trend among these unplugged approaches is the use of tabletop games (Tang et al., 2020). Tabletop games are analog games that are played on a flat surface and include both board and card games. Researchers have argued for the use of tabletop games to teach CT concepts because they promote computation naturally (Berland & Lee, 2011; Horn et al., 2012) are cheaper and thus scalable (Gresse von Wangenheim et al., 2019), and can promote transfer when learners start programming in digital environments (Lee et al., 2020; Kafai & Vasudevan, 2015). In a seminal study on CT and tabletop games, Berland and Lee (2011) identified several forms of CT that can emerge in game play. In particular, they observed people playing the collaborative tabletop game *Pandemic* and found evidence of CT as the players internalized the rules and created and enacted strategies to optimize gameplay.

While some researchers have designed and evaluated their own tabletop games to promote CT (Apostolellis et al., 2014; Gresse von Wangenheim et al., 2019; Kuo & Hsu, 2020) there are many commercial tabletop games on the market. These tabletop games are marketed as promoting computer science (CS) learning and/or computational skills, although empirical research will need to be done to support these claims. A casual glance at the different games would suggest that there tend to be some similar designs. For instance, many tabletop CT games ask players to plan movement on a two-dimensional grid space that is reminiscent of the turtle in the LOGO programming language. Presumably, the benefits of the turtle interaction metaphor for LOGO would apply to playing these games (Papert, 1980). Others involve players following a linear path with their tokens, common in many other board games. Considering there seems to be some regularity, this paper reports on a content analysis to explore the type of tabletop games currently being created, their audiences, and the kinds of recurring game mechanics and design features being implemented to teach CT concepts. These tabletop games are complex environments involving both gaming and learning mechanics that interact in dynamic ways. Understanding not only the types of games and game designs that

are available, but also how such game designs potentially promote learning of CT concepts can be valuable resource for game designers, researchers, and teachers.

In this paper, we present the results of our content analysis and provide design cases of three tabletop games to illustrate how game designs afford opportunities for learning various CT concepts. By explicitly illustrating how game designs promote learning we provide information for designers to design new games, researchers to identify and evaluate the efficacy of games, and teachers to determine aspects of these games that can be leveraged or enhanced for learning CT in a formal or informal setting. In the following literature review, we first provide our definition of CT and the CT concepts that were used to explore learning potential and objectives within the tabletop games. Next, we review how tabletop games have been used in other educational settings. We then discuss arguments for using tabletop games to promote CT learning and review the studies that have explored this topic to date, specifically investigating the game design approaches employed in the literature.

Literature Review

CT Concepts

Researchers studying CT have pursued a variety of frameworks to identify and categorize the components included in CT (e.g., Brennan & Resnick, 2012; Grover & Pea, 2013; Shute et al., 2017; Weintrop et al., 2016; Wing, 2006). The current study focuses on the frameworks presented by Brennan and Resnick (2012) and Shute et al. (2017). Brennan and Resnick (2012) centered their framework on the block-based coding platform Scratch. They divided ideas into concepts (e.g. loops, parallelism, data), practices (e.g. iteration, debugging, abstracting), and perspectives (expressing, connecting, questioning). Their concepts map onto individual Scratch blocks, whereas their practices deal with designing a program.

While Brennan and Resnick (2012) included coding concepts in CT, others propose that CT is similar to problem solving (Shute et al., 2017). The framework proposed by Shute et al. (2017) is based on a definition of CT as "the conceptual foundation required to solve problems effectively and efficiently, with solutions that are reusable in different contexts" (p. 151) and draws from a number of studies and frameworks, including that of Brennan and Resnick (2012). Shute et al. (2017) identified decomposition, abstraction, algorithms, and debugging as the components of CT that occurred most frequently in the literature, and included them as skills engaged in CT. Abstraction is divided into data collection/analysis, pattern recognition, and modeling; algorithms include subcategories of algorithm design, parallelism, efficiency, and automation.

We developed a framework for categories within CT from a combination of the frameworks from Brennan and Resnick (2012) and from Shute et al. (2017). In this framework, CT includes skills employed in the process of designing a solution to a problem: debugging, abstraction, and algorithm design. In addition, CT includes

the knowledge required to implement a solution in a way that computers understand: control statements, data, language-specific syntax. These six areas of CT are defined in Table 1.

[Table 1 near here]

Research on tabletop games for learning

Tabletop games have been used and investigated for learning in a variety of contexts including medicine (Beylefeld & Struwig, 2007; Kaufman & Flanagan, 2016; Reeve et al., 2008), mathematics (Elofsson et al., 2016; Jimenez et al., 2011; Siegler & Ramani, 2009; Skillen et al., 2018), second language learning (Poole et al., 2019), and climate change (Castronova & Knowles, 2015), among other areas (King & Cazessus, 2014; Thomas et al., 2019). While the topics and content areas for which tabletop games are used is diverse, the rationale for researchers using and investigating such games can be broadly parsed into two categories. On one side are those who argue that games are inherently enjoyable and thus can motivate learners to do seemingly monotonous tasks; on the other side are those who argue that tabletop games involve game mechanics and structures that promote learning.

Researchers who see educational games as a motivating force on their own, regardless of the game mechanics, tend to create and investigate quiz-based or trivia games (Nicholson, 2011). In these games, players typically roll a die or draw a card, and are then prompted with a question related to the topic being studied. Such games have been called chocolate-covered broccoli (Bruckman, 1999) because the game functions as chocolate covering a task the learner finds dull (broccoli). Researchers investigating these types of tabletop games have found that players view them positively (Beylefeld & Struwig, 2007; Ogershok & Cottrell, 2004; Rose, 2011) and have reported learning gains (Rose, 2011; Struwig et al., 2014). Though research has shown that by simply framing an activity as a game, learners tend to rate the activity more positively (Leiberoth, 2015), thus calling into question enjoyment as a measure of validation for games.

Other researchers have investigated how tabletop game mechanics can be leveraged to promote learning. For example, several studies have explored the effect of rolling dice and counting while moving a game piece on number sense and mathematical knowledge (Elofsson et al., 2016, Siegler & Ramani, 2009; Skillen et al., 2018;). These studies generally find that simple game mechanics of rolling dice and then counting and moving a game piece, has a strong impact on learners' number sense. Poole et al. (2019) illustrated how a non-linear board allowed for meaningful language learning opportunities, and further how game design can encourage collaboration by creating tasks (e.g. defeating a 'baddie') that cannot be completed alone. Tabletop games have also been argued to support teaching complex systems because players can manipulate different mechanics and pieces within the game and explore how they affect the system (Castronova & Knowles, 2015).

In recent years, researchers have started exploring the use of tabletop games to promote CT and the learning of CT concepts. Researchers have argued that

tabletop games are ideal for learning these concepts and skills because players often engage in computation even when playing tabletop games not specifically designed to teach such concepts (Berland & Lee, 2011). Further researchers have argued that unlike in digital games, in tabletop games the game mechanics are transparent, and thus more salient to players (Horn et al., 2012).

Research on tabletop games for CT

Research investigating the use of tabletop games to promote CT has focused on researcher-designed games and conceptual arguments. Tabletop games used in these studies, similar to games used in other educational settings, fall into a few categories. Some games promote learning by giving players code to execute and then movement is determined by the result of that execution. Other games attempt to integrate learning via game mechanics that typically involve an action queue and or a puzzle to be solved.

In one of the first studies investigating the use of tabletop games to teach computer science concepts, Singh et al. (2007) invited undergraduate students and five lecturers to play *C-Jump. C-Jump* is a linear tabletop game in which players roll a dice and execute code based on the dice roll. For instance, if a player is on a square with the expression: x+2, then the player adds two to the dice roll and moves the game piece along the board. While the students reported enjoying the game and agreed that it promoted learning, the lecturers were more critical and skeptical of the game's learning value. One lecturer stated that the game focused too much on syntax and not enough on programming skills, such as problem solving.

In game designs that have integrated CT concepts into game mechanics, designers have relied on an action queue. An action queue adds a series of steps (e.g. move forward, turn left) to a queue that will be executed. For instance, Gresse von Wangenheim et al. (2019) designed the game *SplashCode*. In the game, players draw five movement cards (e.g. move forward, turn left) to be placed into an action queue of three cards. These action-queue cards are then executed to move the player's character around an open-grid board. The goal of the game is to get the player's character to a designated location before other players do. In another study, Kuo and Hsu (2020) designed *Robot City,* a game that also has players move around a grid using an action queue of cards. However, players have multiple tasks to complete on the grid. For example, players are asked to pick up material and deliver them to spatial locations. To improve the efficiency and complete tasks quicker they can create more advanced action queues that include conditional statements and procedures.

Other tabletop games that have integrated CT concepts into game mechanics have relied on puzzle features. For example, Apostolellis et al. (2014) designed the game *RabBit EscApe*. In this game, players must manipulate wooden tiles that are connected to each other by magnets to create a path for a rabbit to escape an ape. In another study, Lee, et al, (2020) used the *ThinkFun* game *On the Brink* in a classroom setting to promote CT and transfer to a digital version of the game in *Scratch*. In the game *On the Brink,* players move a robot around a grid towards an endpoint. To do

so, they must identify a pattern of colors hidden in the grid and then explore solutions to the puzzle by adding cards to an action queue that is executed by the players.

Finally, Tsarava et al. (2018) designed three tabletop games and explored student and expert gaming experiences across all three games. Interestingly, the three games fall into the aforementioned categories. The first game they designed is a race-to-the-end game called *The Race*, in which players solve coding problems on a card to advance game pieces across a linear board. They also designed an action-queue game, *Treasure Hunt*, in which two players create an action queue involving movement and control statement cards to move either a crab or a turtle around a grid to collect items. Finally, their last game is a puzzle-based game, *Patterns*, in which players identify patterns to solve puzzles before other players do. They claim that *The Race* teaches control statements and data concepts, *Treasure Hunt* teaches algorithm design and abstraction, and *Patterns* teaches control statements and abstraction.

The present study has three primary goals. First, we identify the landscape of tabletop games that are marketed as teaching CT. Secondly, we offer a taxonomy for categorizing these tabletop games based on design features. Finally, we provide design cases for games that fall into each of our categories to illustrate how game mechanics are intended to promote development of CT skills and concepts within each game type.

Methods

Our analysis includes three steps: search for games, content analysis, and design cases. In this section, we first detail our search strategy for locating commercial tabletop games that make marketing claims around teaching CT. Next, we provide a brief description of content analysis and how this approach was used to summarize the current tabletop games available for learning CT. Finally, we provide a description of design cases.

Search Strategy

This review used a three-phase search. In the initial search the following terms were used:

"computer science" OR "computational thinking" OR "programming" OR "coding" AND "tabletop games" OR "board games" OR card games"

In this initial phase, websites were searched for commercial tabletop games that are believed and/or marketed to promote CT, computer science, and/or general programming skills. Commercial games were our focus because a disproportionate amount of educational games in research has focused on researcher developed games and thus relatively little is known about the design of commercial ones (Lee, 2020). All commercial games that appeared to involve computer science in any form

were included resulting in the identification of 22 games. Publisher websites that specialize in designing computational thinking/computer science games and/or websites that review tabletop games (e.g. https://boardgamegeek.com/) were also identified. In the second phase, both the publisher and tabletop game review websites were further investigated for additional games that made claims to teach CT skills. Nineteen additional games were identified, bringing the total to 41. Finally, in the third phase, we searched for research involving CT tabletop games using the following library databases: *Education Source*, *ERIC*, and *Google Scholar*. The search strings above were used in this search.

This search resulted in 171 articles from *Education Source* and *ERIC*, and 17,000+ articles from Google Scholar. Abstracts for the first 20 pages (200 results) in Google Scholar were examined. After 20 pages the results became irrelevant to games and CT. The abstracts of articles from both searches were then reviewed to confirm that the studies were investigating the CT skills in tabletop games. We then identified tabletop games used in each of these articles that were unique from our already established list and identified 8 additional games to bring the total number of games identified to 49. Next, we examined the inclusion/exclusion criteria used to identify which games were included in this analysis.

Inclusion and Exclusion Criteria

Tabletop games investigated in this analysis were included if they met all the following criteria (See Table 2):

[Table 2 near here]

After applying the inclusion criteria 25 games were dropped, leaving 24 games for analysis. Of the 25 games dropped, 5 were dropped because they were researcher designed games and thus were not readily available to the public, 12 tabletop games were dropped because they did not specifically claim to teach CT concepts, 7 were dropped because they either were not tabletop games or required the use of an iPad or other digital screen, and 1 was dropped because it has not yet been released. It is important to note that the 12 games dropped because they did not specifically claim to teach CT concepts were originally added for review because either a researcher or a reviewer on a tabletop game review website identified them as games that promote CT. While they do not explicitly claim to teach CT, these particular tabletop games may provide valuable insight into the design of future games.

Content Analysis

Content analysis is a research tool used for synthesizing concepts or themes within a selected data. It involves the identification of certain concepts followed by quantification of those concepts with the help of analytical coding and subsequent categorization of those analytical codes. Content analysis can provide insight into trends, features, and relationships between concepts within the target data (Lin et

al., 2019). We applied this technique to the commercial tabletop games identified in our search marketed as teaching CT. The researchers in the present study had access to physical copies of most of the 24 games. For those we did not have copies of, we consulted online instruction manuals and videos to understand how the game was played. Analysis based on manuals exclusively has been fruitful for other studies of tabletop gaming (Garcia, 2017).

To explore the types of game mechanics being used within games, we first created a list of all board game mechanics found on https://boardgamegeek.com/. Then we found examples of each of these board game mechanics within one of the CT games we identified. Game mechanics that were not identified in any of the games were then dropped from the list. Next, we coded all 24 games in terms of whether they contained one of the remaining game mechanics. It was at this point that we recognized that these games could be collapsed into three overarching categories, games that used an action-queue, those that required the execution of code, and those that relied on puzzle-based mechanics. Given that this matched game designs being developed and explored in the literature we felt confident in these categories. Finally, we discussed definitions for these categories as a group and then used these definitions for our coding scheme, see Table 3. Two researchers used this coding scheme to code each of the games. Cohen's Kappa was calculated as a measure of inter-coder reliability. As we see in table 3, Cohen's k for each of the game types was .78 or greater, suggesting good reliability.

[Table 3 near here]

In addition to coding for game type, we also coded each game for the targeted age-group based on suggestions by the publisher, the narrative theme (e.g. robots, space) and complexity (e.g. simple, complex), learning claims made either on the publisher's website or in the instructional manual, and the gameplay configuration (e.g. single-player, collaborative, competitive). Narrative complexity was coded as either simple or complex. A simple narrative provides players with a goal and agents to complete the goal. For instance, in *Cosmic Coding* players are given a spaceship (agent) and a task to collect all the stars (goal). Complex narratives include additional plot information such as information about the setting, agent/character background stories, and/or rationale for completing goals. Finally, the learning claims were identified from the publisher's website or the instructional manual, using the keywords in Table 4 below. Synonymous words or phrases - for example, "programming language" is synonymous with "syntax" - also counted as a learning claim, and all claims were collapsed into the six basic categories described in Table 1.

[Table 4 near here]

Design Cases

Design cases are descriptions of "real artifact[s] or experiences that [have] been intentionally designed" (Boling, 2010, p.2). In the present study, the artifacts are

tabletop games that have been intentionally designed to teach or promote CT skills and/or knowledge. Design cases do not provide validation for designs, rather they are a form of summative discourse about the principles and conjectures that designs are based on. Specifically, we explore the conjectures about learning that are made in tabletop game designs.

To systematically build the design cases, we applied Arnab's et al. (2015) learning mechanic-game mechanic (LM-GM) model to three games that fall into each of the game types identified in our taxonomy. The LM-GM model was designed to help researchers and educators identify serious game mechanics that are argued to be "game components that translate a pedagogical practice/pattern into concrete game mechanics directly perceivable by a player's action" (Arnab et al., 2015, p. 395). To use this model, the authors suggest that users: first identify the learning mechanics (LM) and game mechanics (GM), then describe their relationships and implementation. In other words, how do GM afford opportunities for LM. Next, users should illustrate the dynamic appearance of both LM and GM during gameplay. It's is important to note here that while Arnab et al., (2015) do provide several examples of LM and GM in their paper, they do not explicitly define these examples and neither do they claim that their list of examples to be exhaustive. Rather, it is up to the researcher to user of the framework to define the LM and GM in the analysis. In our design cases, we first provide a brief overview of the game. Then we describe the GM in detail. Finally, we explore the conjectures about learning by applying the LM-GM model. The goal of this model is to "determine at which point gameplay and pedagogy intertwine" (Arnab et al., 2015, p. 398). It is at this intersection that potential learning opportunities afforded by the game can be observed.

Results

Results of the content analysis are organized into our taxonomy that contains four categories of game types: *code building*, *executing code*, *puzzle*, and *combination* (see Table 4). Within each category, we present findings on game configuration, environment, goals, and actions followed by a design case.

Code Building Games

Code Building games are those in which players create code via an action queue and then execute the code. These games are designed for single player (N=4), cooperative (N=2), and competitive (N=2) configurations and involve controlling and moving an agent through a grid-like space (See figure 2). Thus, the coding in these games is based on movement and the syntax is directional arrows. These games draw inspiration from the LOGO Programming language, which also uses directional commands to move a turtle around a grid system (Papert, 1980). Eight games were identified within this category (see Table 5). All games in this category are designed for younger learners who are not reading or are emergent readers. Seven of the games have robot-based themes and one has a space theme. Three of

the robot-themed games use animal robots. The narrative for these games is simple and involves a directive to reach a specific target. For example, in *Coder Bunnyz,* players are told that they must retrieve their carrot and bring it home. Games with a complex narrative provide stories that attempt to provide context for the gameplay. For example, in *Robot Turtles*, an additional story book provides a narrative and rationale for several board configurations. To increase difficulty in these games, obstacles are introduced at higher levels and require players to either create a new path or use control statements to get around them. Finally, learning claims made by these types of games focus on CT employed in designing a solution to a problem, such as algorithm design (N=8), abstraction (N=5), and debugging (N=3). Some games claim to teach control statements (N=4), and some claimed to teach data (N=2) and syntax (N=1) concepts. The following design case provides an in-depth look at one of the games in this category.

[Table 5 near here]

Design Case: Robot Turtles

Overview.

In *Robot Turtles*, two to four players program robot turtles to reach a jewel that is placed somewhere on a grid (see Figure 2). Once a player reaches their jewel, they win the game. The game has many levels and introduces obstacles on the paths to make the play more challenging.

[Figure 1 near here]

Game Mechanics and Rules.

The primary game mechanic in *Robot Turtles* is placing a series of coding cards (Forward, Rotate Left, and Rotate Right) face up that indicate which way a player wants their turtle to move. The players, called Turtle Masters, lay down the cards. Then, the Turtle Mover, a teacher, parent, or more experienced peer, executes the code on the board by moving the player's turtle. A secondary mechanic is the use of function cards. In more advanced levels, players can use function cards to define a series of steps to be called with the use of one card, rather than repeating the same steps. Finally, there is a debugging mechanic. This is a game card that the player can hit if they notice an error in their code.

There are two types of rules that provide parameters around how the game is played. The first rule relates to the obstacles (ice blocks, brick blocks, and crates). Each obstacle requires a unique approach to overcome them. An ice block triggers the use of a laser, the brick triggers a need to go around, and the crate requires the player to push it out of the way. These obstacles allow for more challenging paths and more complex code. The second type of rule is applied to provide scaffolding.

This rule states that in game version one, players only play one card at a time and then the Turtle mover executes it. In the second version of the game, players add three cards at a time and in the last version, players add all the cards needed to the action queue to reach the jewel.

LM-GM Analysis.

Opportunities for learning and developing CT skills come when players create an action queue and when the code is executed or enacted by the Turtle Mover. By creating an action queue (the game mechanic), players demonstrate their ability to plan a sequence of moves (the learning mechanic). Placing a series of cards has been argued to help learners develop an understanding of the computational concept sequencing (Brennan & Resnick, 2012). This illustrates how the LM and GM blend into pedagogy. This game is unique from other action queue games because rules explicitly state that the Turtle Mover (game mechanic) should execute the code and not the player. This may help players understand how code is executed by a computer once a command is given. Further, this places emphasis on writing the code sequence (learning mechanic) rather than the enactment of the code on the game board. Another opportunity for learning is created by using the debug card. When mistakes are made, players are encouraged to slap a debug card (game mechanic), which gives them an opportunity to fix their code (learning mechanic). This may help in developing the players' understanding of programming as an iterative, problem-solving task. Finally, when players use the function card (game mechanic), they can engage in abstraction, by identifying (learning mechanic) a series of codes that can be reused in other parts of their queue. Similar to the tables created in Arnab et al., (2015), Table 6 illustrates how the game mechanics connect to learning mechanics.
[*Table 6 near here*]

Executing Code Games

Executing code games are those in which players are presented with a code or a sequency of code to execute. There are six games in this category. All these games are played with multiple players and have a competitive gameplay configuration. Executing the code requires players to integrate information from either a dice roll (N=2) or from the current state of the board (N=4). These games target older learners with most games suggesting an upper elementary age (10+) as the starting point. The themes in this category are more diverse and include sports (N=1); space (N=2); animals (N=1) and farming (N=1). None of the games we identified in this category have robot themes and all the narratives are simple. Players are presented with a linear board and are given a simple explanation for moving their character to the end goal. For example, in *Code Monkey Island*, the objective is to get three monkeys to the banana grove. The learning claims focus on CT for implementing a solution to a problem, such as teaching syntax (N=5), data concepts (N=4), and control statements (N=5). Only one game in the executing category claimed to teach

algorithm design.

[Table 7 near here]

Design Case: Coding Farmers

Overview.

Coding Farmers is played with two to four players. Each player draws cards from a draw pile and executes those cards to move their tractor across a linear board towards a farm. Along the path to the farm are several obstacles that players must navigate around. Each player can hold a maximum of three action cards in their hand and each action card has instructions written in English as well as in java code (See figure 3).

[Figure 2 near here]

Game Mechanics and Rules.

The primary game mechanic involves drawing an action card and then executing code on the card. For example, one card might read, "if the die roll is greater than 2, then move forward two spaces." This game mechanic acts similar to many quiz-based games in that players are given a problem to solve and then are rewarded with gameplay based on a correct answer. However, it adds in the element of chance by rolling the die. Subsequently, data concepts, and more specifically, variables, can be simulated by rolling the die. The die acts as a variable that is constantly being changed each time a player rolls the die. The value of the die is then used within the pre-given functions to be executed. Most of the problems presented in the cards target control statements. There are two types of action cards: (1) English and Java and (2) Java. In beginner mode, players use action cards with both English and Java. These cards act as a form of scaffolding in that they provide support (e.g. English) for the Java syntax. Once the players become comfortable with the Java syntax, they can play in advanced mode by only using the Java cards.

[Figure 3 near here]

Players can only hold three action cards at a time and can choose to play a card depending on their location on the board. Although the game is played by traversing a linear path, there are obstacles, which makes playing certain cards more strategic at different points in the game. By allowing players to a) only hold a limited number of cards, and b) select which card to play, *Coding Farmers* provide players with a sense of autonomy that may not exist in other linear tabletop games.

LM-GM Analysis.

Coding Farmers intends for players to learn Java through game play. There are three ways that learning is assumed to occur. First, by using the cards (game mechanic) to

execute code containing control statements, it is assumed that players may gain incidental knowledge about the functionality and syntax of control statements. To win the game, players should evaluate code (learning mechanic) on all three cards in their hand and determine which card will allow them to advance farthest on each given hand. Secondly, by transitioning from action code written in simple English to code written in Java, the game provides a form of scaffolding to learners as they move towards scripting. Again, there is an assumption that by observing this code and executing the code repetitively (learning mechanic), learners will make connections between simple English and Java code. Finally, rolling the die (game mechanic) and adding the values (learning mechanic) to the executable code provides a simulation of how variables are applied. Table 8 below illustrates how the game mechanics combine with learning mechanics to provide learning opportunities in the game.
[*Table 8 Near Here*]

Puzzle Games

Puzzle games require players to either identify or match a pattern to complete a task. We identified two puzzle tabletop games in our content analysis. Both games are designed around a robot theme. In one game, the objective is for the player to fix the robot. In the second game, the objective is to direct the robot along a path. Both games are designed around pattern matching. Players are given a pattern or a set of conditions and are tasked with matching the pattern or set of conditions on the game board. The learning claims for these games included algorithm design, control statements, and data.

[Table 9 near here]

Design Case: Rover Control

Overview.

Rover Control is a single player game designed around patterns and paths. The game contains 40 different levels, or puzzles, that increase in difficulty as they progress. The objective of each level is to help the rover move from the starting point to the endpoint. Each level is a graph containing nodes and edges (See figure 5) and the rover can move from one node to another following an edge.

[Figure 4 near here]

Game Mechanics and Rules.

The primary mechanic in rover control is pattern matching. On the board, players have a series of paths connected by number-labeled nodes. Players are given a

starting number and a finishing number to dictate where the rover must go. They are then presented with a color-coded pattern (e.g. red, blue, red) that they must follow to complete the level. The secondary game mechanic comes in the form of decision nodes that have various names (e.g. storage station, data station, charging station) (see figure 6). These decision nodes (game mechanic) prompt the learner to make a decision based on where they are or what they currently hold in the game. For instance, one decision node asks if the player is currently at a charging station, if they are, they should continue on to a green path, if they are not, they should continue on to a blue path. Through these secondary mechanics, the game introduces concepts like loops, conditionals, and variables, which fall into our CT categories of control statements and data.

[Figure 5 near here]

LM-GM Analysis

The pattern matching mechanic (game mechanic) in *Rover Control* promotes the CT skill abstraction by requiring learners to identify (learning mechanic) and then reuse (learning mechanic) a series of steps in different parts of the levels. In addition to teaching abstraction, this mechanic also provides an opportunity to learn algorithmic thinking and sequencing as players simulate (learning mechanic) potential paths through the nodes using the provided patterns. In other words, players must plan a sequence of moves based on the pattern and then test it out. *Rover Control* also promotes deduction through a trial and error approach in which the learner colors a path, simulates the proposed path, and then evaluates the functionality of the path. Based on the results, the player can make adjustments. In Rover Control, players begin with very basic patterns and puzzles and slowly increase in difficulty until the final levels when multiple game mechanics are introduced. Control statements are introduced in this game via special nodes (game mechanic) that ask players to make decisions (learning mechanic) about the next color in their pattern and/or what their game piece currently holds. While these nodes make the pattern matching task more complicated and perhaps more fun, the learning assumption is that players will develop knowledge about control statements incidentally by enacting the puzzles to test their solutions. Table 10 below illustrates how game and learning mechanics provide opportunities for learning.

[*Table 10 Near Here*]

Combination Games

The final set of games includes games that fall into two categories. These were either games with *code building* and *puzzle* mechanics (N=4) or *code building* and *code executing* mechanics (N=4). There were no games with *puzzle* and *code executing* mechanics. The *code building* and *puzzle* mechanics included games that used an action queue while trying to either navigate a puzzle-like path or match a set of conditions. These games claimed to teach algorithm design (N=3), debugging

(N=1), abstraction (N=1), control statements (N=1), and data concepts (N=1). Games that included code building and code executing codes allowed players to create an action queue, but then also prompted learners to execute code or an algorithm on some of the cards in the action queue. These games claimed to teach CT concepts in both the design process (N=7) and solution implementation (N=7). Overall, combination games are more complex and have more diversity in the themes, board types, play styles, and learning claims. Themes in this section include pirates, hackers, space, and robots. Only two games had a complex narrative.

[Table 11 near here]

Due to space limitations, we do not provide a design case for this category. However, the mechanics that are combined are clearly outlined in the previous cases. In the next section, we discuss the implications of these findings.

Discussion

The last five years has seen an increase in the number of commercially produced board games that are marketed as teaching CT skills. Most of these games are aimed at pre-elementary and elementary learners. This paper set out to provide an overview of the games being designed, a taxonomy of the design types, and further to provide insight into how games within that taxonomy promote CT using the LM-GM model. We identified four types of games: *code building*, *executing code*, *puzzle*, and *combination* games.

Our taxonomy parses games based on design features and mechanics. In addition, we found that games within each category share similar targeted audiences and learning claims, further lending credence to our taxonomy. Specifically, we found that *code building* games targeted younger learners and focused more on CT in terms of solution design. *Executing code* games targeted older learners and focused primarily on syntax and control statements. While *puzzle* games and *combination puzzle* games targeted older learners, they included learning claims for CT in both solution design and solution implementation. We further contend that our taxonomy has implications for how players conceptualize CS, design, formal and informal learning contexts, and research, which we discuss below.

Representations of CS

Using board games to introduce CT concepts has potential to broaden participation by targeting a younger and potentially more diverse audience. However, it is also important to consider how the design of the game represents CS and the messages it sends players about what programming is. For instance, *code building* emphasize how computation works from a hardware perspective. In other words, computers must be given commands that will be followed exactly as written. In addition, while these games do provide the players with a goal or objective (e.g. spatial target),

players are allowed autonomy on how they achieve that goal Thus, these games focus on the creativity and open-endedness of programming, but also tend to restrict the idea of programming to simple sequences of navigational instructions. In *code execution games* the focus is on the player's ability to understand and recognize the programming language that is used to implement solutions to a problem. This approach limits the view of programming to a particular context or syntax, but this context-specific knowledge may also transfer the easiest to beginner coding exercises. The third category, *puzzle* games, tends to frame CT in terms of ill-designed problems. In other words, CT or programming involves finding solutions to problems through iterative, trial-and-error like approaches. These approaches, problematically, typically restrict creativity by only accepting one solution. Finally, our fourth category, *combination* games, are game designs that include either *code building* and *puzzle* mechanics or *code building* and *code execution* mechanics. By implementing multiple game designs, these games can provide a broader framing of what CT and programming is.

Understanding the implicit messages that are being delivered through these game designs is particularly important given that research suggests that youth perception of CS can impact whether they choose to opt in (Pantic et al., 2018). Further it is important for educators to recognize not only how CS/CT is being portrayed when using these games, but also what aspects of CS/CT are being left out when they choose to teach with these tools. Likewise, designers should explore other areas of CT that may not be represented by the current landscape of CT tabletop games such as state-based, object-oriented, or functional programming.

Implications for Design

In the three categories of games we identified, we noticed a clear distinction in the audience age groups targeted by designs. Games that target young learners use designs within the *code building games* category. These games have less diversity in themes with nearly all games involving robots in some form. They allow players to create an action queue of movement cards which, when executed, move a game piece towards a spatial goal. Given that these games target younger learners, the use of an action queue with movement cards lowers the threshold for introductory CT as literacy skills are not required (Bers, 2019). Action queues potentially allow for players to attempt to fix problems they find as they enact the queue, which may facilitate decomposition and simulation in the process of debugging. However, many of these games are simple, and after a few turns, players may master the mechanic and thus rarely experience a bug to be fixed.

Most of the *code building games* use a robot theme where the narrative builds on the idea that robots (e.g. computers) need input from the player to complete a task. Although three games distinguish their robots as animal robots, these games could use more variety in thematic choices. Such robot-inspired themes may enforce current stereotypes around computing. Master et al. (2016) found that simply adding stereotypical objects (e.g. computer parts, electronics) to an image of a

computer science classroom influenced female interest in an introductory computer science course. Adding more diversity in the themes applied at this level may attract a larger audience to these types of games (Rusk et al., 2008). In addition, two of the eight games in this category situated the game within a complex narrative. Complex narratives may help young learners make connections between gameplay and the CT skills that the game promotes. Further, such narratives may provide an anchor for transferring skills learned in the game into a future digital programming environment (Bers, 2019). Nevertheless, complex narratives may distract young learners; thus, the use of narratives should be carefully considered. The learning claims within these games focus more on developing skills related to the design of a solution. Some games do add secondary game mechanics in advanced levels of the game that target concepts related to implementing a design, such as control statements and data.

Games that target older learners tend to use game designs found in the *executing code games* category. These games focus on scripting knowledge as learners are presented with code, either written in plain English or in a programming language (e.g. Java), to execute. Executing code in these games determines how the player moves through a linear board. None of the games in this category made use of a complex narrative, and because the code being executed can be placed either on a card or the board, there is no need to apply a logical narrative to explain the task. Thus, in this category, greater variety of themes are used to mask the repetitious execution of codes. A review on the use of games in higher education CS classes found that many games have students execute algorithms in a competitive gaming context to add entertainment to the memorization task (Battistella & Gresse von Wangenheim, 2016). Likewise, the designers of these games appear to view CT and/or computer science as a task of memorizing a syntax or semantics associated with syntax. One way to expand on the executing code design is to allow players to combine cards with executable code into larger functions, as is done in the game *Potato Pirates*. In *Potato Pirates* players can combine up to three cards to build an attack on other players (See Figure 7.). Each card played also contains code like those in many of the *executing code games*, but players can build on that code in a meaningful way.

[Figure 6 near here]

Older learners were also targeted by games in the puzzle category. These embedded CT, both as designing and implementing a solution, into puzzles that required either pattern matching or pattern identification to be solved. Two puzzle games did not include designs from other game categories, and four puzzle games included designs from the *code building games* category. In both the combination and non-combination puzzle games, the difficulty of the puzzles often requires a trial-and-error approach in which the player puts forth an idea, tests it, reflects on the idea and then tries a new approach. These games also have a greater variety of themes. In future designs for puzzle games, designers should explore designs that encourage collaborative play (Zagal et al., 2006). While many of these games allow players to cooperate with a peer, most are designed for a single player. In our own research, we find that when pairs work on puzzles, one player often takes over and

solves the puzzle independently. Further, researchers have noted that a primary benefit of tabletop games is the social interaction that occurs naturally when multiple players engage with the physical environment (Battistella & Gresse von Wangenheim, 2016).

Our taxonomy of tabletop games designed to promote CT learning provides a road map for future designers by illustrating how past designs have been leveraged for specific age groups to elicit CT learning in unique ways. Designing tabletop games for learning is a complex process that not only considers the designer's intent and rationale for a design, but also how those designs are realized and enacted by players (Engelstein, 2017). By exploring how past games have leveraged game and learning mechanics to target CT, designers can either apply similar mechanics to games with novel themes and narratives or they can expand on and improve existing mechanics. Further it is important to identify aspects of CS/CT that are not currently addressed in the board games. Blanco and Engstrom (2020) conducted similar analysis with commercial digital games that made claims for teaching programming. They found that most digital games focused on developing fundamental programming concepts which includes syntax, control structures, among others. However, they noted that few digital games addressed algorithm and design concepts. While algorithm, design, and syntax are well covered in tabletop games, there are still many computing concepts that are not addressed in these games, in particular data structures. Game designers should explore designs that address some of these less covered CS concepts and may consider drawing inspiration from the commercial digital games that are designed to teach programming concepts (Blanco & Engstrom, 2020).

Implications for Formal and Informal Learning Contexts

In the content analysis, we sort games by their primary game mechanics and identify their learning claims. Then within each category, we provide a design case to illustrate how the game mechanics are assumed to promote learning. Game mechanics that promote learning may be ideal starting points for educators to focus on when bringing tabletop games into the classroom or an informal learning space to promote CT. For instance, in *code building games* educators could leverage the action queue; in *executing code games* the focus could be on control statements, and educators looking to develop problem-solving skills associated with CT may consider using the puzzle games. However, many of the game mechanics that are designed for learning control statements and data concepts may not be salient to the learner. These game mechanics may improve the game experience, but at some point, an educator may need to help students see how such concepts relate to computing. Past research has drawn on principles from expansive framing (Engle et al., 2012) to make connections between game mechanics in tabletop games that are similar to their digital instantiations (Lee et al., 2020). For instance, in the game *On the Brink*, players are presented with a color-coded action queue (e.g. blue, yellow, red), in which they can place two movement cards on each color. Then when a player's game piece lands on a specific color they enact the code that is associated

with that color. Focusing on this game mechanic, researchers designed a curriculum that promotes transfer from an analog gaming environment to a digital programming environment by emphasizing the similarities between the action queue and procedures in the digital environment. A similar framework could be applied to the *executing code games* to help students see how the die simulates variables or to point out similarities between quasi-code and syntax used in authentic programming languages. Given the trial-and-error approach that is often required in the *puzzle* games, educators may provide support by demonstrating the debugging process in which players attempt a solution, identify a bug, reflect on their original solution, and then propose an adapted solution.

Our taxonomy provides educators with a starting point in determining which type of game is best for their context. In other words, if educators are working with younger learners, they may start with *Code Building Games*. Once, educators have determined which game they will teach with, they can apply the LM-GM model as we did in this study to identify moments when learning is most probable. Then, by understanding where learning is occurring within the game, educators can design lessons that highlight, enhance, and/or extend learning opportunities found within the game. Finally, taking into account how the game is representing CS the instructor should consider ways of expanding the learners view of CS and/or breaking down stereotypes that may exist within the games.

Implications for Research

For researchers, we believe that our taxonomy not only highlights several potential areas for research, but it also provides a guide for what that research might look like. First, several researchers have chosen to design games for their particular context of study. While some of these games may provide some unique aspects, many of the researcher-designed games could have been replaced by a commercial game. For example, Gresse von Wangenheim's et al. (2019) *Splash Code* is very similar to *Coder Bunnyz.*

Second, in terms of areas rich in research potential, each of the game types within our taxonomy include a set of assumptions about learning based on the game design. For example, within the *code building games,* there is an assumption that by allowing players to create and enact an action queue of movement cards, players will develop skills specifically related to algorithmic design. Researchers have designed assessments (Grover et al., 2014; Román-González et al., 2017; Zhao & Shute, 2019) that could be used to determine if these games do in fact promote CT in terms of algorithmic design. Further, *code building games* often increase in difficulty by adding game mechanics around building functions and obstacles to complicate solution paths. Given that these games use directional cards in action queues to build code, researchers could explore how the visual and spatial aspects of action queues supports learning and understanding of computer science concepts like functions and control statements.

E*xecuting code games* makes two primary assumptions. First, they assume that by playing the game players will learn programming languages and/or structures as a by-product of seeing the code and executing it. Research could explore what learning occurs as a result of playing the game, and how long the players need to play to develop this knowledge. In linear path games, where there is only one direction and one solution, learners may not find the game interesting after a game has been completed a few times. Thus, researchers may want to explore how interest changes over continued play. Secondly, there is an assumption that by simply learning a script, players will learn to program. Thus, there is a question of transfer. Are learners able to take the scripting knowledge learned in these games and apply them to a programming environment? Finally, given that these games tend to focus on control statements, exploring how executing code within these games supports learning and understanding of these concepts is another area for potential research.

In terms of puzzle and combination based games, more so than games in other categories, there is an assumption that by playing games and implicitly engaging with concepts that involve control statements and data, players will notice and learn something about these concepts. However, given that such concepts are embedded within game mechanics and thus not overtly clear to the players, research should investigate if players are aware of and if they notice these concepts that puzzles games claim to promote. Furthermore, given the challenge and problem solving involved in these games, player experience and frustration levels should be considered when exploring the efficacy of these games as learning environments. Research should consider the approaches mentioned in the previous section on implications for teaching and explore ways of leveraging these unplugged learning environments in formal and informal spaces. Further research could explore do these unplugged games work? If so, when and under what conditions? Finally, given the differences in how tabletop games represent the CS field and the objective of introducing programming at a young age, researchers should explore how younger learners perceive programming and the CS field after playing of these tabletop games.

Conclusion

The recent influx of commercial CT tabletop games is undoubtedly a positive trend for young emergent coders. However, having too many options can also make it difficult to choose an appropriate game for one's context. In this study, we created a taxonomy of CT tabletop games that identified three primary categories (e.g. code building, code executing, and puzzle games) and one category that includes a combination of the first three categories. Games that fall into our discrete categories share similar learning claims, target audiences, and game mechanics. In our discussion we illustrate how our taxonomy offers a starting place for instructors who want to explore the use of tabletop games for introducing CT concepts in unplugged settings, suggestions for designers looking to build similar games, and areas for investigation for researchers.

ACKNOWLEDGMENTS

This paper was funded by the National Science Foundation (NSF) grant #1837224.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author, [F.J.P], upon reasonable request.

REFERENCES

Apostolellis, P., Stewart, M., Frisina, C., & Kafura, D. (2014). RaBit EscAPE: A board game for computational thinking. *Proceedings of the 2014 Conference on Interaction Design and Children*, 349–352. https://doi.org/10.1145/2593968.2610489

Battistella, P. & Gresse von Wangenheim, C. (2016). Games for Teaching Computing in Higher Education – A Systematic Review. *IEEE Technology and Engineering Education*, *9*(1), 8-30..

Bell, T. C., Witten, I. H., Fellows, M. R., Adams, R., & McKenzie, J. (2015). CS unplugged: An enrichment and extension programme for primary-aged students. Retrieved from http://csunplugged.org/wp-content/uploads/2015/03/CSUnplugged_OS_2015_v3.1.pdf

Berland, M., & Lee, V. R. (2011). Collaborative Strategic Board Games as a Site for Distributed Computational Thinking. *International Journal of Game-Based Learning (IJGBL)*, *1*(2), 65–81. https://doi.org/10.4018/ijgbl.2011040105

Bers, M. U. (2019). Coding as another language: A pedagogical approach for teaching computer science in early childhood. *Journal of Computers in Education*, *6*(4), 499–528. https://doi.org/10.1007/s40692-019-00147-3

Beylefeld, D. A. A., & Struwig, M. C. (2007). A gaming approach to learning medical microbiology: Students' experiences of flow. *Medical Teacher*, *29*(9–10), 933–940. https://doi.org/10.1080/01421590701601550

Blanco, A. A., & Engström, H. (2020). Patterns in Mainstream Programming Games. *Int. J. Serious Games*, *7*(1), 97-126.

Boling, E. (2010). The need for design cases: Disseminating design knowledge. *International Journal of Designs for Learning*, 1(1).

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada*, *1*, 25.

Bruckman, A. (1999). Can educational be fun. *Game Developers Conference*, *99*, 75–79.

Castronova, E., & Knowles, I. (2015). Modding board games into serious games: The case of Climate Policy. *International Journal of Serious Games*, *2*(3), 41–62.

Elofsson, J., Gustafson, S., Samuelsson, J., & Träff, U. (2016). Playing number board games supports 5-year-old children's early mathematical development. *The Journal of Mathematical Behavior*, *43*, 134–147.

Engle, R. A., Lam, D. P., Meyer, X. S., & Nix, S. E. (2012). How does expansive framing promote transfer? Several proposed explanations and a research agenda for investigating them. *Educational Psychologist*, 47(3), 215–231.

Engelstein, G. (2017). *Gametek: The math and science of gaming*. Ludology.

Garcia, A. (2017). Privilege, Power, and Dungeons & Dragons: How Systems Shape Racial and Gender Identities in Tabletop Role-Playing Games. *Mind, Culture, and Activity*, 24(3), 232-246. doi:10.1080/10749039.2017.1293691

Gresse von Wangenheim, C., Silva de Medeiros, G., Filho, R., Petri, G., Da Cruz Pinheiro, F., Ferreira, N., Hauck, J. (2019). SplashCode–A Board Game for Learning an Understanding of Algorithms in Middle School. *Informatics in Education*, *18*(2), 259–280.

Grover, S. & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, 42(1), 38-43. doi: https://doi.org/10.3102/0013189X12463051

Horn, M. S., Weintrop, D., Beheshti, E., & Olson, I. D. (2012). Spinners, dice, and pawns: Using board games to prepare for agent-based modeling activities. *American Educational Research Association Annual Meeting*.

Jimenez, O., Arena, D., & Acholonu, U. (2011). Tug-of-war: A card game for pulling students to fractions fluency. *Proceedings of the Games, Learning, & Society Conference*, *7*.

Kafai, Y., & Vasudevan, V. (2015). Hi-Lo tech games: Crafting, coding and collaboration of augmented board games by high school youth. *Proceedings of the 14th International Conference on Interaction Design and Children*, 130–139.

Kaufman, G., & Flanagan, M. (2016). High-low split: Divergent cognitive construal levels triggered by digital and non-digital platforms. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2773–2777.

King, C., & Cazessus, M. (2014). Teaching with Audacity: A board game for urban studies. *8th European conference on games based learning: ECGBL2014. Academic Conferences and Publishing International*.

Kuo, W.-C., & Hsu, T.-C. (2020). Learning computational thinking without a computer: How computational participation happens in a computational thinking board game. *The Asia-Pacific Education Researcher*, *29*(1), 67–83.

Lee, V. R. (2020). Let's cut to commercial: Where research, evaluation, and design of learning games should go next. *Educational Technology Research and Development*. https://doi.org/10.1007/s11423-020-09865-3

Lee, V. R., Poole, F., Clarke-Midura, J., Recker, M., & Rasmussen, M. (2020). Introducing Coding through Tabletop Board Games and Their Digital Instantiations across Elementary Classrooms and School Libraries. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 787–793.

Lieberoth, A. (2015). Shallow gamification: Testing psychological effects of framing an activity as a game. *Games and Culture*, 10(3), 229–248.

Lin, T.-J., Lin, T.-C., Potvin, P., & Tsai, C.-C. (2019). Research trends in science education from 2013 to 2017: A systematic content analysis of publications in selected journals. *International Journal of Science Education*, 41(3), 367–387.

Master, A., Cheryan, S., & Meltzoff, A. N. (2016). Computing whether she belongs: Stereotypes undermine girls' interest and sense of belonging in computer science. *Journal of educational psychology*, 108(3), 424.

Nicholson, S. (2011). Making gameplay matter: Designing modern educational tabletop games. *Knowledge Quest*, 40(1), 60.

Ogershok, P. R., & Cottrell, S. (2004). The pediatric board game. *Medical Teacher*, *26*(6), 514–517.

Pantic, K., Clarke-Midura, J., Poole, F., Roller, J., & Allan, V. (2018). Drawing a computer scientist: stereotypical representations or lack of awareness?. *Computer Science Education*, 28(3), 232-254.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Poole, F., Clarke-Midura, J., Sun, C., & Lam, K. (2019). Exploring the pedagogical affordances of a collaborative board game in a dual language immersion classroom. *Foreign Language Annals*, *52*(4), 753–775.

Reeve, K., Rossiter, K., & Risdon, C. (2008). The Last Straw! A board game on the social determinants of health. *Medical Education*, *42*(11), 1125–1126. https://doi.org/10.1111/j.1365-2923.2008.03215.x

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, *72*, 678–691. https://doi.org/10.1016/j.chb.2016.08.047

Rose, T. M. (2011). A Board Game to Assist Pharmacy Students in Learning Metabolic Pathways. *American Journal of Pharmaceutical Education*, *75*(9). https://doi.org/10.5688/ajpe759183

Rusk, N., Resnick, M., Berg, R., & Pezalla-Granlund, M. (2008). New Pathways into Robotics: Strategies for Broadening Participation. *Journal of Science Education and Technology*, *17*(1), 59–69. https://doi.org/10.1007/s10956-007-9082-2

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158.

Siegler, R. S., & Ramani, G. B. (2009). Playing linear number board games—But not circular ones—Improves low-income preschoolers' numerical understanding. *Journal of Educational Psychology*, *101*(3), 545–560. https://doi.org/10.1037/a0014239

Singh, J., Dorairaj, S. K., & Woods, P. (2007). Learning computer programming using a board game–case study on C-Jump. *Proc. of the Int. Symposium on Information and Communications Technologies, Kuala Lumpur, Malaysia*.

Skillen, J., Berner, V.-D., & Seitz-Stein, K. (2018). The rule counts! Acquisition of mathematical competencies with a number board game. *The Journal of Educational Research*, *111*(5), 554–563.

Struwig, M. C., Beylefeld, A. A., & Joubert, G. (2014). Learning medical microbiology and infectious diseases by means of a board game: Can it work? *Innovations in Education and Teaching International*, *51*(4), 389–399.

Tang, K.-Y., Chou, T.-L., & Tsai, C.-C. (2020). A content analysis of computational thinking research: An international publication trends and research typology. *The Asia-Pacific Education Researcher*, *29*(1), 9–19.

Thomas, M. K., Shyjka, A., Kumm, S., & Gjomemo, R. (2019). Educational Design Research for the Development of a Collectible Card Game for Cybersecurity Learning. *Journal of Formative Design in Learning*, *3*(1), 27–38.

Tsarava, K., Moeller, K., & Ninaus, M. (2018). Training computational thinking through board games: The case of Crabs & Turtles. *International Journal of Serious Games*, *5*(2), 25–44.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Zagal, J. P., Rick, J., & Hsi, I. (2006). Collaborative games: Lessons learned from board games. *Simulation & Gaming*, *37*(1), 24–40.

Zhao, W., & Shute, V. J. (2019). Can playing a video game foster computational thinking skills? *Computers & Education*, *141*, 103633.

Table 1. Areas of Computational Thinking

CT Area	**Definition**
Debugging	Detecting, investigating, then fixing errors in a procedure
Abstraction	Finding or constructing patterns within problems and solutions, in order to facilitate understanding.
Algorithm Design	Thoroughly defining steps to solve a problem. These steps may be intended to be executed linearly, in a sequence, or non-linearly, concurrently or in event-driven programming
Control Statements	Choosing between instructions to follow next based on some condition
Data	Structures or methods involved in keeping or modifying values in a memory
Syntax	A system of well-defined rules for communication, i.e., a language

Table 2. Inclusion Criteria

Inclusion Criteria	**Rationale**
The game includes either an instructional manual, instructional video, or in-depth description of gameplay. Further game instructions were in English.	To conduct the analysis, it is important to understand how the game is played. This requires either an instructional video or manual that was presented in English.
The game is either marketed or promoted by 'the company' / 'designer' as a game to promote CT skills, as identified in Table 1 above.	Many board games are argued to contain programming concepts. Thus, this study is only looking at games that are explicit in their intent to teach CT.
It is a tabletop game, which includes board games, card games, and other analog games. Unplugged activities that are not games were not included.	This study is focused on how games and game design is used to target CT skills so only games were examined.
The game can be purchased or acquired freely.	We focus on games that are readily available to the public for use in classrooms or the home.

Table 3. CT Tabletop Games Taxonomy

Game Type	Description	Example	Cohen's Kappa
Code Building (N=8)	Players create code or an algorithm by placing a series of movements or code into an action queue to later be executed by the player, another person, or a robot.	Robot Turtles	0.84
Code Executing (N=6)	Players are given a code or an algorithm to be executed. Movement or progress in the game is dependent on the correct execution of the algorithm. Players do not create code or an algorithm in these games.	Coding Farmers	0.78
Puzzle Games (N=2)	Players are presented with a puzzle that can be solved by identifying or matching a pattern.	Rover Control	1.00
Combination (N=8)	A combination of two game types	Potato Pirates	NA

Table 4: CT Learning Claims

CT Area	Example Claim
Debugging	"Students learn experiential learning, **debugging**, limited syntax, order of operations…" --(Robot Turtles)
Abstraction	"Teaches simple concepts like loops, branches, **functions**, conditionals and advance concepts like **Inheritance**, Parallelism, List, Stack, Queue and Algorithm writing." – (CoderBunnyz)
Algorithm Design	"Playing *Code Master* won't just teach you principles behind programming, you'll also build **planning**, **sequential reasoning** and problem-solving skills." – (Code Master)
Control Statements	"It teaches the child basic commands of a programming language, such as **'if'**, **'else'**, **'switch'**, and introduces variable 'x' concept." – (C-jump)
Data	"It ... exposes kids to fundamental programming concepts like control structures, **data structures**, **Boolean logic and operators**, and **assignment** and **mathematical operations**." – (Code Monkey Island)
Syntax	"Introduces the **basics of java** and programming concepts." – (Coding Farmers)

Note: Keywords are in bold.

Table 5: Code Building games

Game	Year	Age	Theme	Narrative Complexity	Learning Claims
Robot Turtles	2013	4+	Animals/ Robots	Complex	Debugging, abstraction, algorithm design, data, syntax
Coder Bunnyz	2016	4+	Animals	Simple	Debugging, abstraction, algorithm design, control statements, data
LittleCodr	2017	4+	Robots	Simple	Abstraction, algorithm design
Future Coders Robot Races	2018	4+	Robots	Simple	Algorithm design
Code & Go Robot	2016	4+	Animals/ Robots	Simple	Algorithm design
Mojobot	2019	4+	Robots	Complex	Abstraction, algorithm design, control statements
Cody Roby	2014	4+	Robots	Simple	Algorithm design, control statements
Bits & Bytes Card Game	2014	All	Space	Simple	Debugging, abstraction, algorithm design, control statements

Table 6. The LM-GM analysis of Robot Turtles

Game Mechanic	Learning Mechanic	Implementation	Usage
Action Queue	Plan	Adding cards to the action queue	Players add cards to action queue based on a plan that they wish to enact.
Grid/ Capture	Experimentation	Open-ended board with a grid and a jewel placed on the grid as the objective.	The grid constrains the potential movement and provides a space for planning. While the jewel gives the learner a target.
Function Card	Identify patterns	Players are encouraged to use fewer cards via the function card.	By encouraging the use of less cards via the function card mechanic, players are encouraged to identify

			patterns that can be abstracted.
Turtle Master	Observation/ Analyze	Turtle master enacts the code and debug card.	Because the turtle master moves the turtle, the player can observe and analyze their code.

Table 7: Executing Code Games

Game	**Year**	**Age**	**Theme**	**Narrative Complexity**	**Learning Claims**
Coding Farmers	2015	7+	Farming	Simple	Control statements, data, syntax
Code Monkey Island	2014	10+	Animals	Simple	Control statements, data
C-Jump	2005	11+	Sports	Simple	Control statements, data, syntax
Cosmic Coding Game	2019	6+	Space	Simple	Algorithm design, control statements, syntax
Coding is Good	2016	10+	None	None	Syntax
Astro Coders	2018	10+	Space	Simple	Control statements, data, syntax

Table 8. The LM-GM analysis of Coding Farmers

Game Mechanic	**Learning Mechanic**	**Implementation**	**Usage**
Selecting a card to execute	Generalization/ Discrimination	Placing cards to move a tractor	Players must select between three cards on each turn to determine which card will give them the best opportunity to win.
Reading the code on the cards	Observation and repetition	Code to be executed is on cards.	Code is presented in both English and Java script, giving players opportunity to observe both forms multiple times.
Question & Answers	Identify	The code to be executed represents a question to be answered.	Executing the code is how players determine which card is best and how far they can move on the board.

Competition	Competition / Feedback	Players attempt to reach the barn first.	Players race to the end giving them motivation to pick the best card. Players also check that other players are executing code correctly.

Table 9: Puzzle Games

Game	**Year**	**Age**	**Theme**	**Narrative Complexity**	**Learning Claims**
Rover Control	2017	8+	Robot	Simple	Algorithm design, control statements
Robot Repair	2017	8+	Robot	Simple	Data

Table 10. The LM-GM analysis of Rover Control

Game Mechanic	**Learning Mechanic**	**Implementation**	**Usage**
Puzzles	Hypothesis/ Repetition	Players make a plan using game pieces.	Players make hypotheses about how to solve the puzzle, and then test their hypotheses via simulation.
Non-linear paths	Modelling/ Analyze	Players enact their plan by simulating a path.	While simulating a potential solution, players must also analyze why it did or did not work.
Collaboration	Demonstration	Players work together to solve the puzzle.	As players simulate potential solutions, they model paths and provide examples for their peers.

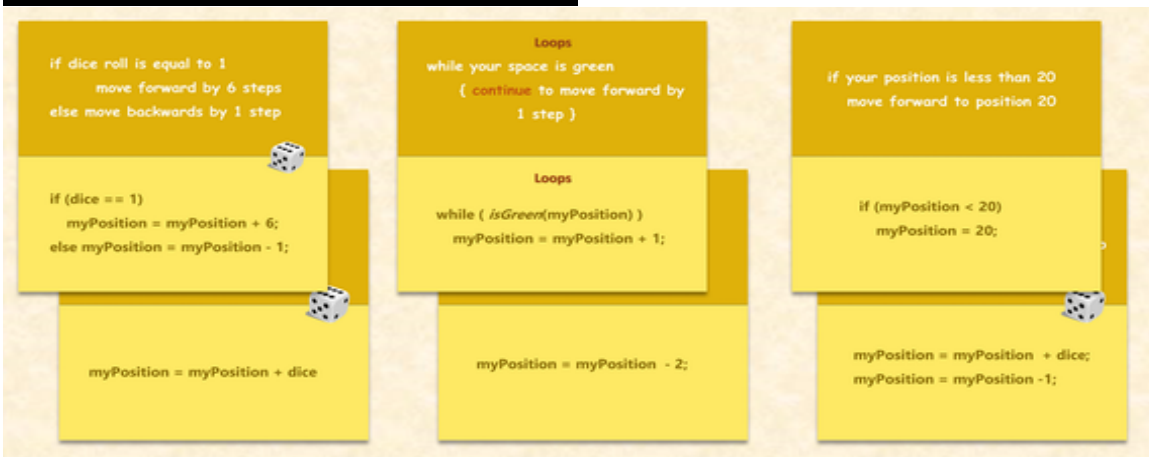
Table 11: Combination Games

Game	**Year**	**Combo Type**	**Age**	**Theme**	**Narrative Complexity**	**Learning Claims**
Code Master	2015	C + P	8+	Space	Simple	Debugging, algorithm design, control statements
On the Brink	2017	C + P	8+	Robots	Simple	Abstraction, algorithm design

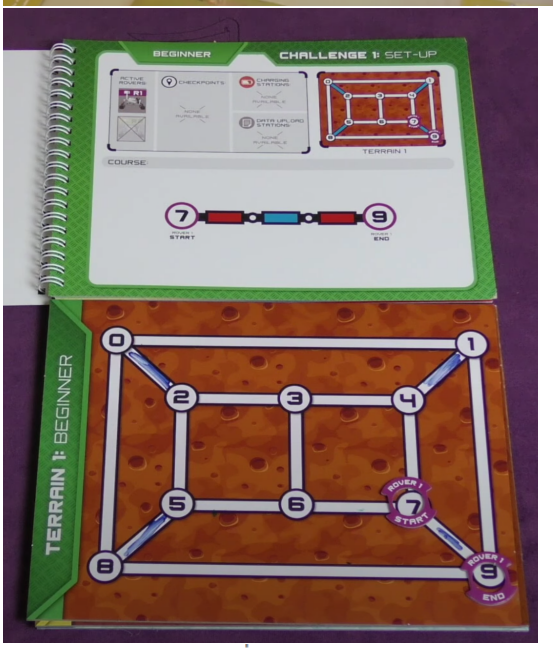
Game	Year	Combo Type	Age	Theme	Narrative Complexity	Learning Claims
Hacker	2018	C + P	10+	Hacker	Complex	Algorithm design
Turing Tumble	2018	C + P	8+	Space	Complex	Data
Race Condition	2011	C + E	NA	None	None	Algorithm design, control statements, data
Potato Pirates	2018	C + E	10+	Pirates	Complex	Debugging, abstraction, algorithm design, control statements, data
CoderMindz	2018	C + E	4+	Robots	None	Abstraction, algorithm design, control statements
Robot Wars	2017	C + E	7+	Robots	Simple	Abstraction, control statements, syntax

*C= Code Building, E=Executing Code, P=Puzzle

Figures




codingFarmers;


BEGINNER
CHALLENGE 1: SET-UP
CHECKPOINTS
CHARGING STATIONS
DATA UPLOAD STATIONS
TERRAIN 1
COURSE
7
START
9
END
TERRAIN 1: BEGINNER
0
1
2
3
4
5
6
ROVER 1
START
7
8
ROVER 1
END
9


Anchor
stack up to 3 action and control cards to prepare for battle
Anchor
stack up to 3 action and control cards to prepare for battle
Roast
Mash
Fry
fry 3 potatoes
Action
While > 4
Mash
mash 2 potatoes
Action