Computer programming has become an essential part of K12 education, promoted as a way for students to engage in computational thinking that helps develop students’ ability to analyze and solve problems and prepare them for future careers. Tabletop board games are seen as an effective means to help students learn computer programming. Several board games have been developed for teaching computer science to novice students. Still, many are dominated by simple pathfinding movements lacking comprehensive use of various computer programming concepts or have a considerable gap between the game dynamics and the actual coding that takes place on the computer. This paper presents a design case in which we used Kalmpourtzis’ (2018) elements of educational game design (game elements, learning, and players) to develop a board game that engages players who are learning block-based computer programming. We present the four major prototypes and the challenges for each step. Then, we highlight three main areas in which our design process offers implications for the design of educational board games.

Dayae Yang is a doctoral student in the Learning, Design, and Technology program at the University of Georgia. Her research interest is in designing educational games, with particular interest in games that support computational thinking and coding education.

Theodore J. Kopcha is an associate professor of Learning, Design, and Technology at the University of Georgia. His research focuses on developing and testing unique approaches to STEM education in K-12 classrooms through design-based research.

INTRODUCTION

Computer programming has become an important part of K12 education, promoted as a way for students to engage in computational thinking that helps develop students’ ability to analyze and solve problems effectively and efficiently (Shute et al., 2017; Wing, 2006) and to prepare them for future careers (Gresse Von Wangenheim et al., 2019). Research suggests that learning computer programming skills early at a young age can improve both their skill development and interest in computer science in the future (Bers et al., 2014; Sharma et al., 2019; Tsan et al., 2018). However, learning computer science and programming can be a daunting process for novice learners; it requires multiple higher-level thinking skills and an understanding of abstract concepts (Mcgettrick et al., 2005; Piteira & Costa, 2012; Robins et al., 2003). Students must learn the language of the computer, conceptualizing how different commands and codes will manifest when a program is running.

Adding to the challenge of introducing computer science to younger students is an overall lack of materials and teacher training. The available instructional materials for teaching computer programming do not always take into account the developmental needs and capabilities of the audience (Walton-Hadlock, 2008), resulting in resources and activities that are too difficult for students to accomplish without intervention. In addition, teachers often lack the background knowledge and skills needed to teach children to program a computer (Bell et al., 2009). This is due, in part, to a lack of teacher-preparation programs that help novice teachers learn and teach computer programming (Yadav et al., 2016). Although several block-based programming languages for kids have been developed (e.g., Scratch or Blockly), many teachers continue to express a lack of confidence and concerns with having to teach such an abstract language on the computer to young students (Ericson et al., 2016; Thompson et al., 2013).

Tabletop board games are seen as an effective way to help young children learn computer programming. The use of games in learning, in general, has been shown to enhance students’ interest, engagement, and motivation (Apostolellis
et al., 2014; Barab et al., 2010; Barata et al., 2013; Nah et al., 2014). Games also provide learners with an opportunity to focus on a small set of essential skills that can be mastered in a friendly, engaging manner (Barab et al., 2010). In the context of computer science, board games not only make learning accessible for anyone with or without the digital infrastructure (Bell & Vahrenhold, 2018; Gibson & Bell, 2013) but also are especially effective in helping younger players develop concrete representations of abstract concepts (Hinebaugh, 2009). This is particularly important when dealing with complex concepts such as algorithms, conditionals, and loops that tend to be overwhelming not only for young children to understand but also for adults to teach children (Fotaris et al., 2016; Olsson et al., 2015).

Several board games have been developed for teaching CT and computer science to children - commercial games such as Robot Turtles and Codemaster, and a few that were developed for research purposes (e.g., Apostolellis et al., 2014; Gresse von Wangenheim et al., 2019; Tsarava et al., 2018). These games support computer programming in that they require players to use a combination of codes (e.g., move forward, left, right) or patterns to manipulate objects on the game board and achieve game goals. However, current challenges with such games are that they are generally often dominated by path movement as the primary game dynamic – that is, through simple sequencing of directions rather than a comprehensive use of various computer programming concepts. This dynamic presents a considerable gap between the game and the actual coding that takes place on the computer (Wu, 2018). Despite the popular demand for such games, little attention has been spent in the literature on the process of designing and developing educational games (Gresse von Wangenheim et al., 2019), still leaving many questions about how a board game that teaches computer programming skills should be designed.

This paper presents a design case in which we used Kalmpourtzis’ (2018) elements of educational game design (game elements, learning, and players) to develop a board game to teach young children basic computer programming concepts and skills. The impetus for the project came from a practical issue repeatedly noted by the authors. The online programming environments, Scratch and Scratch Jr., are promoted as essential tools for teaching computer programming skills. However, in our work with local schools, we noticed that both teachers and students often needed an introduction to block-based programming before they could be successful with the available online tools. This gave us the idea for the board game – our thinking was that we could address this practical issue by creating a safe environment in which children and their teachers could learn to use a limited set of block-based codes before moving to an online programming environment. We hoped that the game would help children transition to more sophisticated programming in the future.

With that context in mind, the main goal of the board game was to introduce beginning programmers to the block-based language and basic programming concepts used in Scratch. We had two target audiences in mind for the game. The first was children from grades three to five; the game was designed to help players practice the concepts found in the Computer Science Teachers Association (2017) standards for grades three to five, such as sequencing movements, looping or repeating subroutines, and if-then conditionals. We also thought that the game and game elements would be useful for adults who were just learning about block-based programming. Thus, our second audience was adults interested in learning block-based programming basics so they could support young children in learning CS skills (e.g., novice teachers). Table 1 describes how various programming concepts were incorporated into the game.

The design process involved four iterations of playtesting. The first three playtests were conducted with graduate students in the field of Learning, Design, and Technology; for the fourth playtest, both graduate students and a 5th-grade student at a local school played the game. We intentionally began our playtesting with graduate students because we

<table>
<thead>
<tr>
<th>CS CONCEPTS</th>
<th>IMPLEMENTATION IN THE GAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequences</td>
<td>Players will create and follow codes in a sequence to make their character move in favor of the player's strategy (e.g., deciding to turn right first, then move three steps forward to avoid an obstacle and move towards a reward).</td>
</tr>
<tr>
<td>Algorithms</td>
<td>With limited access to code tiles, players will consider and compare different ways to solve a task and decide which would be the best solution for them.</td>
</tr>
<tr>
<td>Loops</td>
<td>With a limited number of code tiles in their hands, players will use “repeat __ times” to repeat a sequence of codes multiple times rather than listing the same commands repeatedly.</td>
</tr>
<tr>
<td>Conditionals</td>
<td>Players will use “if __ then” statements to move across obstacles or collect more rewards when the condition is met.</td>
</tr>
</tbody>
</table>

**TABLE 1.** Basic programming concepts that were incorporated in the game.
were concerned that children might not tolerate the flaws in the gameplay and learning dynamics inherent in our initial design. The graduate students had little to no prior experience with Scratch programming. Our goal was to work first with adults with an underlying background in learning design to refine the game and learning dynamics. This would, in turn, help us make better use of our time when prototyping with young children.

In the first playtest, the players identified critical issues with the initial game core, gameplay, and learner dynamics. The second playtest occurred after making a major change in the game mechanics. Once the game mechanics were working, the third playtest focused on improving the aesthetics of the game and the learning dynamics. The fourth playtest, which took place most recently, focused on making the game enjoyable while also engaging players in higher-level computer programming concepts.

In the sections that follow, we present our initial design and detail the development process across the four prototypes, highlighting the important challenges that were faced during an educational game design process. We then suggest implications for future designers.

THE GAME CORE

The “game core” is a term used by George Kalmpourtzis in his book *Educational Game Design Fundamentals* (2018). The game core represents the essential intent and purpose for the game being designed. According to Kalmpourtzis, establishing the game core before the creation process is fundamental:

By figuring out their game core, game designers establish a concrete starting point that will help them avoid future issues like technical restrictions, communication problems, finding materials that will affect deadlines, budget, member motivation, and the final impact of the designed game (p. 117).

We chose Kalmpourtzis’ (2018) perspectives to guide our design process. Kalmpourtzis suggested that the game core be established by focusing on three essential aspects of educational game design: (1) the players, (2) the game elements and mechanics, and (3) the learning that will take place. He stated that these three aspects are important because a successful educational game designer must consider who the players will be as well as their preferences and learning goal. These factors would ultimately influence what game elements will be included and how the game mechanics will address the desired learning objectives. Based on this idea, our goal as designers was to balance these essential aspects so that the game is fun while ensuring that the players learn from the game.

FIGURE 1. The goals for our board game in the context of Kalmpourtzis’ (2018) Game Core.

With Kalmpourtzis perspectives in mind, our game design process began by identifying the general objectives within each of the three core aspects (see Figure 1). First, we wanted to create a board game that would be appealing to both adults and children (players) who were learning basic and advanced programming concepts, including conditionals and loops (learning), in a way that was fun and resembled the Scratch programming environment (game). Note that the three elements are interrelated, so it is not always easy to make a clear distinction between each perspective during game design (Kalmpourtzis, 2018). It is important that the designers keep the framework in mind during the initial stages of educational game design; they serve as guidelines that help designers attend to each of the critical aspects of game design, considering how each major design decision affected each aspect, and consciously finding the optimal balance between game, player, and the content to be learned.

In the following sections, we present a design case associated with creating an educational board game that achieved our goal of supporting beginning programmers in learning computer science. The case focuses on how we, as designers, engaged in the process of prototyping that attempted to balance our goals for the game, players, and learning. Below, we present each of the four main prototyping sessions, including the problems that arose from each prototype and our design decisions that helped resolve those problems in each subsequent version. (See Appendix for the summary of challenges in each prototype.)

PROTOTYPE 1: BRAINSTORMING

The overall focus of the first prototype was on developing and testing game mechanics. Playtesting our initial design revealed issues with the complexity of the game mechanics and a need to make gameplay more engaging for players. These issues are described in detail below in terms of the
focal point of our design and the results of the prototyping session.

**Design Focus**

Development of the first prototype began by brainstorming the gameplay mechanics. The prototype was constructed simply using markers, crayons, sticky notes, and readily available objects. Because our target audience was players who were new to programming, the concepts used in the game were intended to be intuitive and easy to understand without any prior knowledge.

The earliest idea for gameplay was to move robots on a grid (see Figure 2). Although we wanted the game to incorporate more advanced programming concepts, we began with movement as it allows novice players to mimic how characters are programmed to move in digital settings and sequence a computer program in a way that makes intuitive sense. Since one of our objectives was to help such players become familiar with the Scratch programming environment, we used words and functions typically associated with Scratch, such as moving a specific quantity and turning to face the desired direction.

The game board was constructed on a grid so that players could assemble code cards (i.e., the sticky notes in Figure 2) to program their robots (round toys) to move and avoid obstacles (crayons) while collecting rewards (blocks) and points (colored spaces). We also included placeholders for constructing blocks of commands. These include spaces for: taking direct action, constructing variables, and constructing repeatable functions. Our thinking was that these would support the learning aspect of the game core. The spaces on the gameboard invited players to apply basic concepts (i.e., movement) while offering space for more advanced programming concepts (i.e., loops, conditionals).

**Results**

Two major issues were found. The first dealt with the balance between the players and the learning: it was difficult for the players to understand what each placeholder meant. We anticipated that this difficulty would create a high entry barrier for beginning players to fully understand the game and find the game fun to play. The second issue dealt with the game: there was not enough space for the robots to move around and the players to assemble block codes on a single game board. Both of these issues became the focus of our improvements in our second prototype.

**PROTOTYPE 2: FROM ONE TO TWO GAMEBOARDS**

Our work on the second prototype (Figure 3) began with a focus on the game and the players. Our goal was to improve the game board design so that game mechanics were more intuitive for players and that there was more room for gameplay. A major design decision in the second prototype was to distribute the game mechanics across two separate game boards - one for computer programming and one for the movement of the robot. Playtesting revealed that the gameplay significantly improved for the players but did not address the learning of computer programming in a substantial way. Detailed issues are described below in terms of the focal point of our design and the results of the prototyping session.

**Design Focus**

Our first prototyping session revealed that players felt constrained by the physical aspects of the game board -- there was not enough room on the board for players to construct blocks of code and simulate the movement of the robot. To address this issue, we ultimately separated the construction of code from the actions of the robot. As shown in Figure 3, gameplay would now take place across two game boards: (1) an "action board" where the players would move the robot, and (2) a "coding board" where players would construct blocks of code. We were inspired by Scrabble and Rummikub in that we used tiles on a grid that could be modified by all participating players. Each tile would become part of a
code block, and the players could combine the tiles to build statements to make their characters move toward a goal.

We anticipated that our design decision about the game (i.e., use two game boards) would address the two other components of the game core: the player’s experience and the learning components of the game. Having two game boards would help reduce the confusion created in the first prototype from having multiple areas on a single board for different types of programming skills (e.g., variables, functions). In turn, this would make it easier for players to develop effective blocks of code from their first move in the game (i.e., learning). It also added more space for characters to move the robot around while more closely resembling the Scratch programming environment, which provides users a separate space for building codes.

Results

The second prototype was playtested with three graduate students who had no or little experience with Scratch. The participants agreed that using tiles to make codes was a unique and fun experience. However, the players’ use of tiles was limited to basic moving and turning movements. Rather than engage in higher-level programming concepts such as conditionals, they were primarily focused on collecting immediate rewards using simple movements. The players also were unclear of the game rule during the play, which significantly slowed down the game. The challenge of finding the balance between the game elements (finding a good game mechanic) that can include the necessary learning objectives (advanced programming concepts) while making the game still appealing for the targeted players (fun and easy to understand) remained.

However, we were not sure at this point whether such a challenge was due to the game design itself or its loose structure and lack of an appealing story or visual design. Such missing parts seemed to limit the motivation of players to play the game long enough to move on from using simple movements to making more complex strategies using advanced statements. Hence, it was necessary to add in the aesthetic elements for our next step to find out if the game did have the potential to prompt players to use advanced statements or if there was a fundamental flaw with the overall game design.

PROTOTYPE 3: MAKING THE GAME APPEALING

The main focus of the third prototype, shown in Figure 4, was twofold. First, we sought to make the game more playable and appealing through improved structure and aesthetics. In addition, we sought to improve players’ use of higher-level computer programming concepts; this was a recurring issue from previous prototypes. Playtesting of this prototype eventually revealed important information about two aspects of the game core: the players and the learning. Specifically, the players were much more engaged with the game as we introduced a gender-neutral aesthetic, but the learning was still limited to sequencing simple movements rather than more advanced programming concepts (e.g., conditionals). Detailed issues are described below in terms of the focal point of our design and the results of the prototyping session.

Design Focus

Several research studies have asserted the need for gender equity in designing instructional materials for computer science education (Barab et al., 2005; Grover & Pea, 2013; Justice & Markus, 2010). With this need in mind, we considered several gender-neutral themes to improve the appeal and aesthetics of the game. Our final selection was “bears on an adventure.” In line with the theme, colored areas in the previous prototypes were replaced with water, grass, and
pit holes. Rewards (honeycombs, berries, fish) and obstacles (rocks and trees) were put on the board with physical objects made of playdough and small toys to provide a more interactive and tangible experience for the players.

With regard to the game, the primary goal was for players to randomly collect tiles of codes and assemble them on the coding board to move their characters on the action board. On the action board, players could earn points by collecting rewards, where each reward type had a different point value. The player with the most points in total would win the round.

On the coding board, the tiles were categorized and color-coded by words and non-words. To encourage learning, the block of code at the center was usable to both players; each round, a player could add to it or modify it to their liking. For example, in Figure 4, the first player constructed a block of code, “Turn [diagonal]” then “Move 2 [squares].” Building on existing blocks of code encouraged players to actively make strategies that could build into more advanced programming. Also, the player that used up their tiles first would end the round and receive bonus points. The goal of this reward was to encourage players to build more advanced codes so that they could use up their tiles more quickly.

**Results**

The prototype was playtested with five graduate students, including three novice players and two returning players. It was clear that the added aesthetics and story attracted the players’ attention and increased their motivation to be more engaged in the game. The participants reported that the game was fun to play and that having physical objects on the board to collect or avoid made the game much more interactive.

However, now that the game had more solid rules and content, several significant limitations were more evident:

The players were confused over the different tile types. For example, the tile “move” was to be used with numbers to mean ‘move [number] steps forward.' Similarly, the “turn” was to be used with an arrow to indicate ‘turn to face [direction of the arrow].’ The variation made gameplay confusing for the players, who needed repeated reminders about how to use the tiles on the coding board to move their character on the action board.

There was still an issue with encouraging players to engage in higher-level computer programming concepts such as conditionals and loops. One primary reason for this issue was that the gameplay required players to wait until they collected all the tiles needed to complete a complex command. This was frustrating and tedious. For example, it might take multiple rounds of gameplay before an if-then statement could be constructed to completion. Often, the need for the statement passed before the tiles could be played. As a result, players found it easier to rely on simple movements rather than attempt more advanced gameplay.

**PROTOTYPE 4: ADDING SCAFFOLDS**

**Design Focus**

Prototype 4 (see Figure 5) addressed the two issues that were revealed in the previous prototype. First, we added structure that reduced player confusion over the meaning of each tile. Second, we continued improving the gameplay to encourage players to engage in higher-level computer programming concepts. Playtesting revealed that both the graduate and the 5th-grade students found the game intuitive and were more motivated to use advanced codes than in previous versions of the game. One challenge arose regarding the retention of players’ engagement and motivation. Detailed issues are described below in terms of the focal point of our design and the results of the prototyping session.

**Results**

The prototype was played with two groups of playtesters: (1) a group of three adult players, including two novices and one expert in computer programming, and (2) one with a fifth-grade student who was familiar with working with Scratch. After one round of gameplay (20 minutes) for each
group, all players agreed that they were motivated to use the advanced programming concepts. When the novice players played the game, they began by using simple movement codes. As the game progressed, they soon began creating more complex codes that involved conditionals and loops. Although returning players began using advanced codes sooner than novices, both groups created a similar level of statements by the end of each round. All playtesters explained that they used thinking processes similar to those needed to program on the computer. It was clear that the added scaffoldings and the revised reward system helped make the game more intuitive while achieving the learning goal.

One common issue experienced by both the graduate and 5th-grade learners was that interest in the game diminished over time. For both audiences, the first 10 minutes of gameplay were exciting and intriguing. They enjoyed thinking through different possible moves to earn rewards and points. The 5th-grade learner intentionally used an if-then conditional early in the game that allowed him to collect points more easily as the game progressed. However, once those points were earned after several rounds, he slowly began to lose interest in the game. The adult players similarly lost interest in gameplay once they began accumulating points, mainly because they were repeating the same gameplay pattern, and it was unclear what it would take to end the game and win it.

Both sets of playtesters recommended placing a time limit on the game so that there was pressure to earn the most points in a short period of time. This would make the win-state of the game more achievable for players and encourage them to continue finding ways to earn the most points possible, potentially sustaining interest as the game progressed. Another suggestion for improving and sustaining interest in the game was to add cards that would introduce fun but unpredictable challenges or rewards. Examples included allowing players to lose a turn, move to a random space on the game board, steal points from one another, or interfere with an opponent’s current strategy.

**REFLECTION**

The purpose of this paper was to present our process of prototyping an educational game that sought to teach novice programmers the fundamental components of block-based programming in a Scratch environment. After four rounds of prototyping with both adults and a young learner, we were able to improve the various elements of the game core (Kalmpourtzis, 2018), including the gameplay, learning, and the appeal of the game to the players. Each prototype resulted in new challenges that not only required design decisions but also helped balance the core elements while meeting our goals for the game. In the following sections, we reflect on our design process, highlighting three main areas in which our design process offers implications for the design of educational board games. The first focuses on the importance of having clear learning objectives when making decisions about the game. The second focuses on balancing learning with the goal of making the game fun for the players. The third focuses on developing in-game support that encourages players to engage in higher levels of computer programming concepts. Detailed explanations are presented below.

**GAME: Making Design Decisions: Having clearly stated goals and objectives to help make critical decisions about the design of the game**

Our game design originated with a clear learning objective: we wanted our players to construct and understand both basic and advanced computer programming skills through gameplay. This clarity proved to be an essential component of our decision-making throughout the game development process. To begin, it helped us decide which language and concepts to include for our target audience. We had several coding languages and tools to choose from, such as Java, Python, Scratch, and C++, as well as an option to make up our own language that taught coding concepts. Although those languages included similar programming concepts, the terms used in each language were distinct from one another. We ultimately chose to mirror Scratch’s block-based programming environment because Scratch is widely used...
by K-5 teachers as an introductory platform for computer programming. By focusing on block-based programming as our primary learning objective, we were able to refine gameplay and mechanics with confidence. It helped us determine which programming terms to include on game tiles and which programming concepts to focus on in our game.

Having a clear objective also helped us make decisions about how to improve the game from one prototype to the next. In some cases, this led to improvements in the game design. Our use of two game boards closely mirrors the Scratch environment in which the coding space is kept separate from the animation area where those codes are executed. In other cases, our clear objective helped us to determine what not to include in our design. For example, playtesters repeatedly suggested that we make the game more exciting by allowing players to attack their opponents. In response, we prototyped a game dynamic that would allow players to attack one another. While we were able to make the attack function work to some extent, it ultimately resulted in gameplay that de-emphasized our learning goal. Rather than attending to the coding component of the game, players instead focused on finding tiles for attacking. We ultimately had to limit the attack function and create other mechanisms that improved gameplay while also achieving our learning goal.

Goal and Design Reflection: Having a clear learning objective helps designers make critical design decisions about what elements and mechanisms to include in the game and to which extent they should accept the various feedback from players.

PLAYERS: What Do Players Want?: Using low-fidelity prototypes to receive honest feedback

Educational game designers are challenged to find the balance between making the game effective for learning while also appealing to the learners who play the game (Acosta & Denham, 2018; Greeff et al., 2017; Hopkins & Roberts, 2015; Kalmpourtzis, 2018). In fact, one of the most critical internal motivations for students to play educational games is to have fun (Long, 2007). This implies the importance of knowing the players’ gameplay experience and what they want in a game to have fun, rather than solely focusing on the educators’ goals. Early and frequent prototyping is essential in getting honest feedback from the players (Kelley & Kelley, 2013; Moggridge & Atkinson, 2007).

Our low-fidelity approach during initial prototyping helped us learn more about what players wanted without expending valuable resources. Our earliest prototypes were made using scraps of paper, small toys, and post-its that we could find in the room. This approach made it clear to playtesters that our prototype was in very initial draft form and need of tremendous improvement. As a result, they could easily understand the concept and intent of the game while being less hesitant to provide straightforward comments when asked to help us improve the prototype. Such feedback immensely helped in learning the players’ preferences without expending our own resources needlessly. It also helped us, the designers, remain emotionally removed from the feedback we were receiving. We intentionally limited the time and effort we invested in creating the prototype. As a result, it was much easier to hear critical yet necessary feedback that helped us quickly improve and test new game components in the next prototype.

Goal and Design Reflection: Using low fidelity prototypes during earlier stages of game development can help designers more easily and quickly determine what the players want to do in the game environment.

LEARNING: Engaging learners in higher levels of concepts: Use of immediate feedback and exclusive rewards

One persistent design challenge had to do with creating opportunities for players to engage in higher levels of programming such as conditionals and loops, rather than being limited to just simple movements. We wanted to encourage players to use such advanced functions but at the same time make the game accessible to both novice and returning players. As revealed in our earlier prototypes, this was difficult to achieve. Most players focused on simpler movements because these were easier to employ and often guaranteed small but immediate rewards. Even when players could earn the largest points at the end of the game for using advanced concepts, the effect was minimal.

One of the major reasons for this challenge was a lack of immediate rewards for using advanced programming concepts. The timing of feedback is a design focus in video game settings. Delayed feedback can be motivating for players who are winning but at the same time discouraging for players who are losing (Turkay et al., 2014). The same idea can be applied to the board game setting. Even though the players in our game could receive the largest points at the end of each round for using advanced concepts, many players were more motivated to use simpler codes instead. This was especially true for players who were not in the lead; simpler codes allowed them to collect immediate rewards more quickly than wait until the end. Thus, the solution we integrated into Prototype 4 was to provide instant rewards exclusively earned by using conditionals. This additional game mechanic was highly effective in encouraging players to be more actively engaged with using higher-level programming concepts as part of their gameplay.

Goal and Design Reflection: Providing immediate and exclusive rewards in a game can motivate players to engage with higher-level thinking.
CONCLUSION
Games are an increasingly popular mechanism for learning. In the context of computer science, they can help beginning programmers engage in computational thinking and acquire computer programming skills prior to working on a computer. This paper, then, provides greater insight into game development and the mechanics that can support learning computer science through gameplay. It is our hope that this design case serves as a foundation for educators and scholars to build upon as we move closer to the goal of improving computer science in K-12 settings.

REFERENCES


Bell, T., & Vahrenhold, J. (2018). CS unplugged—how is it used, and does it work?. In Adventures between lower bounds and higher altitudes (pp. 497-521). Springer, Cham. https://doi.org/10.1007/978-3-319-98355-4_29


## APPENDIX A

### Summary of Challenges by Prototype

<table>
<thead>
<tr>
<th>PHASE</th>
<th>GAME</th>
<th>PLAYERS</th>
<th>LEARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIAL GOAL</td>
<td>Make the game fun to play</td>
<td>Create the game core that appeals to players</td>
<td>Engage in concepts beyond movement-based gameplay</td>
</tr>
<tr>
<td>1</td>
<td>Game mechanics (tiles) were too complex; also, more space was needed to assemble codes on the game board.</td>
<td>The placeholders in the game were difficult to understand for novice players. The game was playable but not exciting or appealing.</td>
<td>The coding mechanics used in the game were significantly different from Scratch.</td>
</tr>
<tr>
<td>2</td>
<td>Added a second game board and simplified tiles/codes, but the game was not very exciting.</td>
<td>The placeholders were removed and the gameplay mechanic simplified.</td>
<td>The simplified mechanic made the game more fun but did not encourage more than simple movements.</td>
</tr>
<tr>
<td>3</td>
<td>Game mechanics supported the assembling of codes in blocks. Aesthetics made the game more interesting.</td>
<td>The game became more appealing, but the language and use of tiles with advanced concepts was still confusing.</td>
<td>Gameplay improved but focused on simple movements due to the mechanics around using tiles with advanced concepts.</td>
</tr>
<tr>
<td>4</td>
<td>Rewards were added to promote engagement, but the game still needed more fun elements.</td>
<td>Scaffolds were added to make the game more intuitive and easier to understand, but the players' interest diminished over time.</td>
<td>Bigger and exclusive rewards were added that encouraged conditionals; this motivated the players to use more complex codes.</td>
</tr>
<tr>
<td>NEXT STEPS</td>
<td>Limit the duration of a single round; increase random elements to make gameplay more exciting and challenging.</td>
<td>Improve ways to engage players in advanced concepts such as loops and variables.</td>
<td></td>
</tr>
</tbody>
</table>