Tuk Tuk: A Block-Based Programming Game

Chonnuttida Koracharkornradt
Stanford University
520 Galvez Mall, Room 102
CERAS Building
Stanford, California, 94305, USA
kchonnut@stanford.edu

IDC '17, June 27-30, 2017, Stanford, CA, USA

ACM ISBN 978-1-4503-4921-5/17/06.
http://dx.doi.org/10.1145/3078072.3091990

Abstract
Studying computer programming helps children develop computational thinking, a problem-solving methodology that can be transferred to solve everyday problems. Additionally, exposing children to an advanced computational concept of search algorithm allows them to see how different problem-solving techniques are designed to tackle more challenging tasks, and improve their ability to solve problems. We present a block-based programming game called Tuk Tuk for children in kindergarten level (junior version), and elementary and middle school level (standard version). With Tuk Tuk, learners create a computer program in a block-based language to control a car to complete a given task, earn money, reach the next level, and unlock new coding blocks. By completing each task, learners will learn important computational concepts and algorithms, a basis of computational thinking, such as conditionals, iterations, depth-first search (DFS) and breadth-first search (BFS).

Author Keywords
Block-Based Programming; Game

ACM Classification Keywords
D.2.6 Programming Environments

Figure 1: Learners use coding blocks to control a car in the standard version.

Figure 2: Learners move the car to control the car itself (programming-by-example) in the junior version.

Introduction

Studying computer programming helps children develop computational thinking [1]. We present a block-based programming game Tuk Tuk for beginners, specifically kindergarten students (junior version), and elementary and middle school students (standard version), which introduces basic computational concepts, a basis of computational thinking [1]. With Tuk Tuk, learners use coding blocks to control a car in the standard version (Figure 1), or simply move the car (programming-by-example) in the junior version (Figure 2). A chain of coding blocks, such as "pick up", "forward", and "drop off", forms a program to complete a given task after which the player earns money, reaches the next level, and unlocks new coding blocks.

While there are a number of programming platforms with a goal to encourage an understanding of basic computational concepts, our focus is on a concept that does not normally made available to children—the search algorithms i.e. depth-first search (DFS) and breadth-first search (BFS).

Related Work

In order to have children engage in a meaningful way with the learning process, the tools have to provide a low floor (easy to get started), and the designers have to selectively expose the foreground with respect to the learning goal and black box others [2]. Block-based programming language follows the pattern by blackboxing the syntax of programming, which results in an easy-to-use interface, and allows learners to jump into the central ideas of programming right away [3].

While several block-based programming platforms have been developed (e.g. Scratch, Snap), relatively few of these platforms present programming concepts in the game-based learning. Games have been used for educational purposes for a long time as they are not only fun, but also provide a competitive environment where students can compare their abilities and learn from others, which encourages engagement and motivates students in the learning process [4].

In order to teach young children programming concepts that are considered "too advanced" for their age, Topobo [5] uses programming-by-example concept where a child can program a robot by physically pushing and pulling the moving parts. Inspired by this approach, we develop a junior version of Tuk Tuk.

A more advanced programming concept of algorithms is incorporated into the game in order to allows children to see how different problem-solving techniques are designed to tackle the more challenging tasks, and improve children's ability to solve problems [6]. The car metaphor lends itself well to the path finding algorithms using in Google maps, in which DFS and BFS are the simplest graph search algorithms to learn.

Design

1) Design Overview

The main game screen consists of three sections. The "Game Status" section at the top left area of the screen shows player's name, level, money, and current task to be completed. The "Map" section at the bottom left area of the screen runs and displays the result of the program. The "Script" section at the right area of the screen shows the program being developed (Figure 3).

Tuk Tuk allows learners to program by using a block-based language or programming-by-example method,

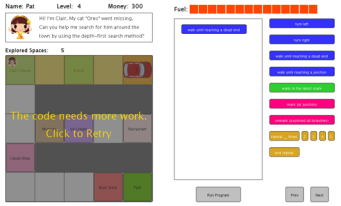

Figure 4: Sample of visual feedback and error message.

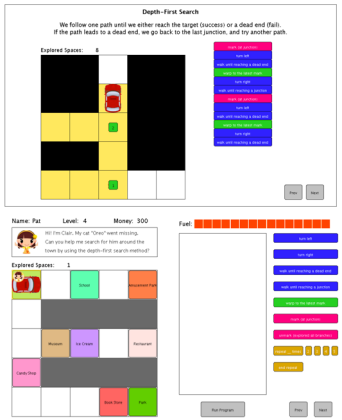

Figure 5: Introducing DFS.

Figure 6: The cat will always appear at the last explored grid.

as well as implement DFS and BFS algorithms. There are multiple solutions to complete each task, and learners have to decide on how to solve the problem.

2) Core Mechanics ("What is possible?")

2.1) Coding Blocks

Tuk Tuk's coding blocks are represented as simple phrases so that learners can intuitively understand what each block does. The blocks are color-coded in order to help learners find blocks in the same category. The coding blocks are also easy to manipulate. Learners can click on a coding block in the palette to add it to the script, and click on a coding block in the script area to delete it from the script (Figure 3). Moreover, the coding blocks are designed to have embedded syntactical error correction. For example, learners can only insert the "else" block after the "if" block is used.

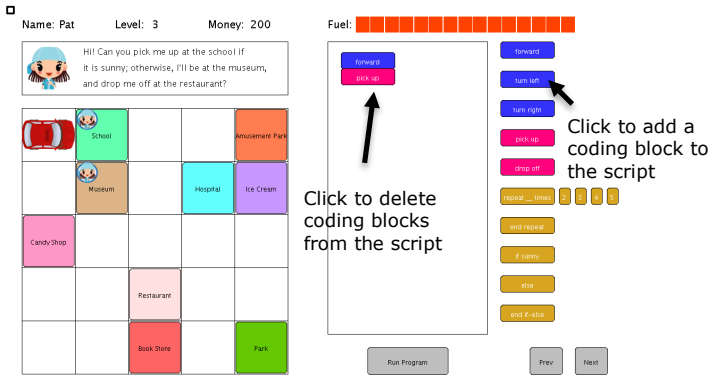

Figure 3: Main game screen layout.

2.2) Fuel Blocks

One fuel block is used for each coding block that learners add to the script. For each task, a limited numbers of fuel blocks are given in order to avoid too long script and encourage learners to think about how to solve the problems efficiently.

2.3) "Run Program" Button and Map Simulation Area

Tuk Tuk allows rapid iteration, the ability to add or delete blocks to a script and see the result quickly. Learners can run a program fragment at any time to see what it does. When the "Run Program" button is clicked, the "Map" section will simulate script execution. Learners can check whether the program do what it is intended to do, and if the program has an error, an error message will appear (Figure 4). These features help learners stay engaged in testing, debugging, and improving their program.

3) Built-in Pedagogy ("How what is possible is made available?")

Tuk Tuk introduces programming concepts as a game where learners obtain additional features when they progress to the next level.

At the beginning, learners only have "forward", "turn left", and "turn right" coding blocks (Figure 1). As they progress to level 2 and 3, "repeat" blocks and "if-else" blocks will be enabled, respectively (Figure 3). At level 4, there will be an instructional simulation of what DFS is, before the DFS coding blocks are enabled (Figure 5). Level 5 is similar to level 4, but focuses on BFS. The task for level 4 and 5 is to search for a cat lost in a town. We also design the solution in a way that the cat will always appear at the last explored grid (Figure 6) to make sure that the learners has utilized the algorithms properly. Next, level 6 and 7 allow learners to explore the differences and advantages of each search algorithm by giving them access to both DFS and BFS coding blocks. However, one algorithm is more

Figure 7: Explore the differences and advantages of each search algorithm.

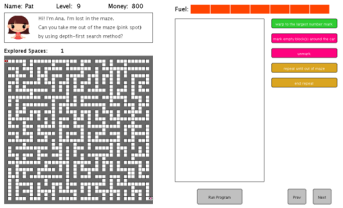

Figure 8: Complex Map.

Figure 9: Preliminary user testing.

preferable than another for a particular map. DFS is more suitable for level 6, and BFS is more suitable for level 7 (Figure 7). Level 8 is similar to level 4 and 5, but focuses on a compact version of DFS. At level 9 and 10, learners get to apply search algorithms to complex maps (Figure 8). The learners will discover the advantage of search algorithms as the last two levels are too large to direct the car to explore all spaces without the usage of search algorithms.

Preliminary User Testing

A 1-hour session interview was conducted with six- and eight-year-old girls (Figure 9). The interface interested the kids and the kids were eager to complete the task. They loved the designs, and immediately understood how to use coding blocks to control the car. In addition, we learned that the game mechanism works very well, i.e., the kids enjoyed the game, wanted to play it again, and had fun trying to beat each other. Finally, the learning goals were met. By completing level-2 task, children understood the concept of iterations as they managed to put the coding blocks inside the "repeat" block in order to make the car repeatedly perform sequence of actions. We hope that further user testing will confirm these initial findings and capture new ways to improve our design.

Conclusion and Future Work

Tuk Tuk is a block-based programming game that helps children develop computational thinking by learning computational concepts, especially search algorithms, through self-directed interfaces. Tuk Tuk can be easily downloaded online and is ready to use.

There are improvements that can be made on Tuk Tuk such as:

- Adding incentive to play the game e.g., learners can use money to decorate the car, or share their score online and see rankings.
- Introducing more advanced computational concepts such as artificial intelligence search techniques in games (e.g. chess, go, etc.) and allow learners to compete their created game agent, etc.

Acknowledgements

We would like to thank Paulo Blikstein, Assistant Professor at Stanford University, and our TAs for Beyond Bits and Atoms course, especially, Richard Davis, Engin Bumbacher, and Chris Proctor.

References

1. Kafai, Y., Burke, Q., & Resnick, M. (2014). *Connected Code: Why Children Need to Learn Programming*. MIT Press.
2. Papert, S. (1980). Mindstorms: Children, computers and powerful ideas. Basic Books, NY.
3. Resnick, Mitchel, John Maloney, Andres Monroy-Hernandez, Natalie Rusk, Evelyn Eastmond, and Karen Brennan. (2009). *Scratch: Programming for all*. Communications of the ACM 52 (11): 60-67.
4. Combefis, S., Beresnevicius G. & Dagiene, V. (2016). *Learning Programming through Games and Contests: Overview, Characterization, and Discussion.* Olympiads in Informatics Vol.10, 39-60.
5. Hayes Raffle and Cristobal Garcia. (2003). *Topobo for Tangible Learning.* MAS 712 Technological Tools for Learning.
6. Miller, B. and Ranum, D. (2011). *Problem Solving with Algorithms and Data Structures using Python: Second Edition.*