DPGame: Game-Based Learning for Dynamic Programming Algorithms

Ying Zhu[0000−0002−9155−2315]

Georgia State University
Atlanta, USA
yzhu@gsu.edu

Abstract. Dynamic programming is one of the most challenging topics in data structures and algorithms courses. Students often struggle to grasp dynamic programming techniques and apply them to solve problems. The common advice to students is to study lots of dynamic programming examples. However, analyzing source code is not an engaging experience for most students. This issue motivated us to design a game-based learning environment for dynamic programming. Although game-based methods have been used in computer science education, there have been no examples of games specifically designed to teach dynamic programming. In this paper, we introduce DPGame, a multi-level puzzle game that facilitates the learning process by guiding students through the construction of dynamic programming algorithms via increasingly challenging examples. In the game, the player's goal is to construct a logical dependency diagram by answering a series of questions, similar to assembling a visual puzzle. This game can help students identify recurring logical patterns in dynamic programming and fosters a high-level understanding of the algorithm prior to tackling the low-level details of code implementation.

Keywords: Game-based Learning · Computer Science Education · Dynamic Programming · Algorithm.

1 Introduction

Data structures and algorithms are the cornerstone of computer science education. In addition to basic data structures and algorithms, students also need to learn advanced algorithms such as dynamic programming, divide and conquer, greedy algorithms, graph algorithms, etc. [3, 14, 7] Dynamic programming is one of the most elegant algorithms and also one of the most difficult algorithms to master [12, 14, 7]. For example, Skiena [14] wrote, "..., until you understand dynamic programming, it seems like magic. You must figure out the trick before you can use it." Rawat and Meenakshi [12] wrote, "The most difficult questions asked in competitions and interviews, are from dynamic programming."

Indeed, students in Data Structures and Algorithms classes often struggle to understand dynamic programming. The difficulty of dynamic programming lies

in its recursive logic. In a dynamic programming algorithm, a problem is solved from the results of several overlapping subproblems, which are solved based on the results of several smaller-scale subsubproblems, and so on. It can be tricky to identify the logical dependencies between a subproblem and the relevant, smaller-scale subsubproblems.

The common advice for learning dynamic programming is to study a lot of dynamic programming examples. Kleinberg and Tardos [7] wrote that "it typically takes a reasonable amount of practice before one is fully comfortable with it." Skiena [14] wrote, "Dynamic programming is best learned by carefully studying examples until things start to click." Rawat and Meenakshi wrote a book [12] analyzing many dynamic programming examples. However, analyzing tricky source code and reading pages of dense technical description, though necessary, are not engaging experiences for many students. How can we make this learning process more interesting and engaging? Motivated by this question, we explore the possibility of designing a game-based learning environment for dynamic programming.

Although game-based learning has been applied to computer science education [9, 15, 8, 4, 10, 6, 13, 5, 11, 17], we have not seen any examples of game-based learning environments specifically designed for dynamic programming.

We have designed a multi-level puzzle game called DPGame to help students understand the logical patterns of dynamic programming algorithms. At each level, students are presented with a coding problem, and their goal is to construct the dependency diagram of a dynamic programming algorithm by answering a series of questions. These questions guide students to identify subproblems and establish the logical dependencies between a subproblem and its related subsubproblems. Each time a student answers a question correctly, a component is added to the diagram. The diagram is completed when all questions are answered correctly. As students progress through the game levels, the problems become increasingly complex. Through this process, students gain a deeper understanding of dynamic programming. With this high-level knowledge, they can then tackle the low-level details of coding.

2 Related Work

Game-based learning has long been used in computer science education [9, 15, 8, 4, 10, 6, 13, 5, 11, 17]. Various studies have shown that game-based learning can increase student interest and engagement [8, 6, 15, 17]. Videnovik, et al. [17] conducted a comprehensive review on game-based learning in computer science education. Based on this review, pedagogical strategies in game-based learning can be classified into two categories: learning by playing games or learning by designing games. The majority of the reviewed cases fall into the category of "learning by playing games." Videnovik, et al. [17] also analyzed the topics covered in game-based learning and found the three most frequently covered topics are object-oriented programming, computational thinking, and block-based programming.

Most existing game-based learning methods tend to focus on foundational programming topics. For example, Leutenegger and Edgington [8] used a "learning by game development" approach to teach object-oriented programming topics such as class, object, inheritance, pointers, etc. Some games focused on basic data structures and algorithms topics, such as array, linked-list, queue, stack, and recursion [19, 18, 16, 15]. However, we have not seen any examples of game-based learning for dynamic programming.

Our method is a "learning by playing game" environment in which students learn the computational thinking of dynamic programming, an advanced topic that has not been addressed in previous game-based learning research.

3 Dynamic Programming Overview

Dynamic programming is an advanced algorithm that solves a problem by dividing it into a hierarchy of dependent, overlapping subproblems [3, 14, 7]. The solution to a subproblem is based on the solutions to several smaller-scale subsubproblems [3]. To design a dynamic programming algorithm, we need to properly define the logical dependencies between each subproblem and its subsubproblems.

Dynamic programming is often used to find the optimal solution to a problem. For these optimization problems, recognizing the optimal substructure is critical. This principle asserts that an optimal solution to a problem depends on the optimal solutions to its subproblems [3, 14, 7]. Each step of dynamic programming might involve selecting the best option to optimize specific parameters, such as distance, cost, or profit. The decision for each subproblem may depend on the outcomes of prior subproblems.

As discussed earlier, dynamic programming is one of the most challenging algorithms to master. Our game, described in the following section, is designed to help students overcome this difficulty. It focuses on helping students identify subproblems and construct the intricate dependencies between them.

4 Game Design

4.1 Overview

DPGame is a text-based game played in Google Colab, Jupyter, or other Python Development Environments. The game is played in the following steps.

1. *Challenge*: A coding problem is presented to the student (player).
2. *Gameplay*: A question is presented with multiple choices. The student selects an answer.
3. If the answer is correct, go to the next step. Otherwise, go to Step 2.
4. *Visual Feedback*: Add a new component to the dynamic programming dependency diagram.
5. If the diagram is complete, go to the next step. Otherwise, go to Step 2.

6. *Game Over*.

At each level, the player is given a coding question, such as those typically seen on the premier coding practice website LeetCode [1]. In our game, the player's goal is to correctly answer the questions to gradually complete a logical dependency diagram, like putting together a visual puzzle. Our purpose in designing this game is not to teach students how to write code but to understand the key logic of dynamic programming.

4.2 Structure of the Game

Figure 1 shows the overall structure of DPGame. The Puzzle Manager (described in section 4.4) is responsible for selecting the questions to be presented to the player. The questions are selected from a Questions and Answers data file (described in section 4.3). The data file is separated from the Puzzle Manager so that an instructor can modify the data file without changing the game code.

The Diagram Manager draws the diagram using PlantUML [2], an open-source tool for creating diagrams based on a simple scripting language. The specific diagramming scripts are stored in the Questions and Answers data file along with the questions. If the player answers a question correctly, the Puzzle Manager will send the corresponding diagramming script to the Diagram Manager to add a new component to the dependency diagram.

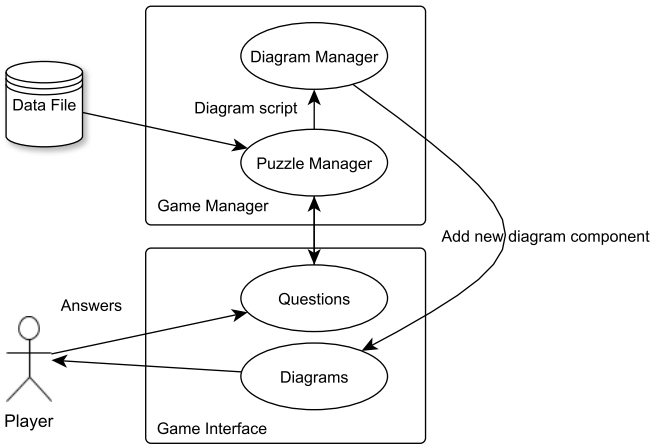

Fig. 1: The overall structure of DPGame. If the player's answer is correct, the Puzzle Manager will notify the Diagram Manager to update the dependency diagram. If the player chooses the wrong answer, the Puzzle Manager will select another question based on the internal game mechanics.

One of the benefits of this approach is its convenience. The game is played within the Python development environment with no extra installation or setup. Instructors can easily share the games with the students via Google Colab. After playing the game, students can quickly move on to writing the Python program while referring back to the dynamic programming dependency diagram without leaving Google Colab.

4.3 Data File

The data file is a text file divided into multiple levels. Each level starts with a problem description and PlantUML script for the visual component, followed by questions and answers. Figure 3 is a screenshot of a questions-and-answers file.

The questions and answers are organized in a tree structure (Figure 2). Each problem has multiple Primary Questions (PQ). Each primary question is associated with a PlantUML script (Figure 3). Each time a primary question is answered correctly, the PlantUML script is sent to the Diagram Manager to add a new component to the dynamic programming dependency diagram. If all the primary questions are answered correctly, the diagram is completed.

Each Primary Question is associated with multiple Alternative Questions (AQ). If the student selects the wrong answer to a primary question, an Alternative Question is selected and displayed.

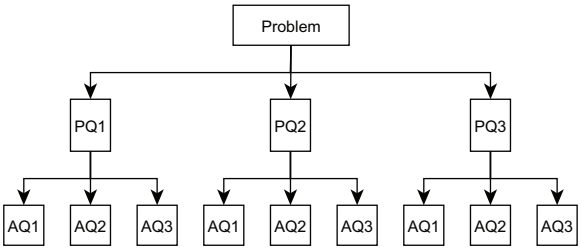

Fig. 2: The questions for each problem are organized in a tree structure. Here a PQ represents a primary question, and an AQ represents an alternative question.

4.4 Puzzle Manager

The Puzzle Manager manages the game mechanics (rules). It retrieves the questions and answers from the data file and presents them in a sequence based on a modified depth-first traversal algorithm, as shown below.

1. Select and display the next primary question. When all the primary questions are answered correctly, go to Step 6.

```
@start

// Problem Description
> You are given an integer array cost where cost[i] is the cost of ith step on a staircase. Once you
pay the cost, you can either climb one or two steps. You can either start from the step with index 0,
or the step with index 1. Return the minimum cost to reach the top of the floor.

# Primary Question. ? means the question. + means the correct answer. - means the wrong answer. *
means this is a primary question.

?* To calculate the minimal cost for the ith step, the minimal costs of which steps need to be
considered?
+* (i-1)th step
+* (i-2)th step
-* (i-3)th step
-* All the previous steps

# PlantUML script for the diagram
$* {
state "(i-1)th step " as S_1
S_1: Min Cost
state "(i-2)th step" as S_2
S_2: Min Cost
state "(i)th step" as S
S: Min Cost
}

# Alternative question. ** means this is an alternative question.
?** From which step can you get to the ith step in one step?
+** (i-1)th step
-** (i-2)th step
-** (i-3)th step
-** (i-4)th step
```

Fig. 3: This is the screenshot of the beginning part of a question-and-answer data file. The questions, answers, and diagram scripts are marked by special symbols so the game program can parse them.

2. If the answer is correct, send the PlantUML script to the Diagram Manager and go to Step 1.
3. If the answer is incorrect, go to Step 4.
4. Select and display the alternative questions associated with the current primary question.
5. After all the alternative questions are answered, display the primary question again. If the answer is correct, go to Step 2. Otherwise, go to Step 7.
6. Game is won.
7. Game is lost.

The alternative questions are designed to guide the students to find the correct answer to the primary question.

4.5 Visual Feedback: Dependency Diagram

The dynamic programming dependency diagram (see Figures 4 and 5) is a novel visualization technique that we have developed to illustrate the logical dependencies between subproblems. It includes three major components: the state of a subproblem, sub-states within a state, and the logical dependencies between them.

The state of a subproblem (shown as a box) is defined by parameters that need to be calculated. For instance, in the Fibonacci sequence, each subproblem's output is a Fibonacci number. In optimization problems, these parameters are what we aim to optimize, such as minimizing costs or maximizing profits.

Each subproblem can have multiple sub-states, representing different choices or scenarios. For example, in a stock buying optimization problem, the choice for each day is "hold" or "sell".

Logical dependencies vary by problem, from simple sums to selecting max (or min) values or complex operations such as searching. These are visualized by connecting states and sub-states with linked operators.

We use PlantUML to create these diagrams. By breaking down the diagram into components and associating each with a question, we create challenges for the player to assemble the visual puzzle.

Further details on the dynamic programming dependency diagram are discussed in another paper [20].

5 Example

Figures 4 and 5 are screenshots of a DPGame played in Google Colab.

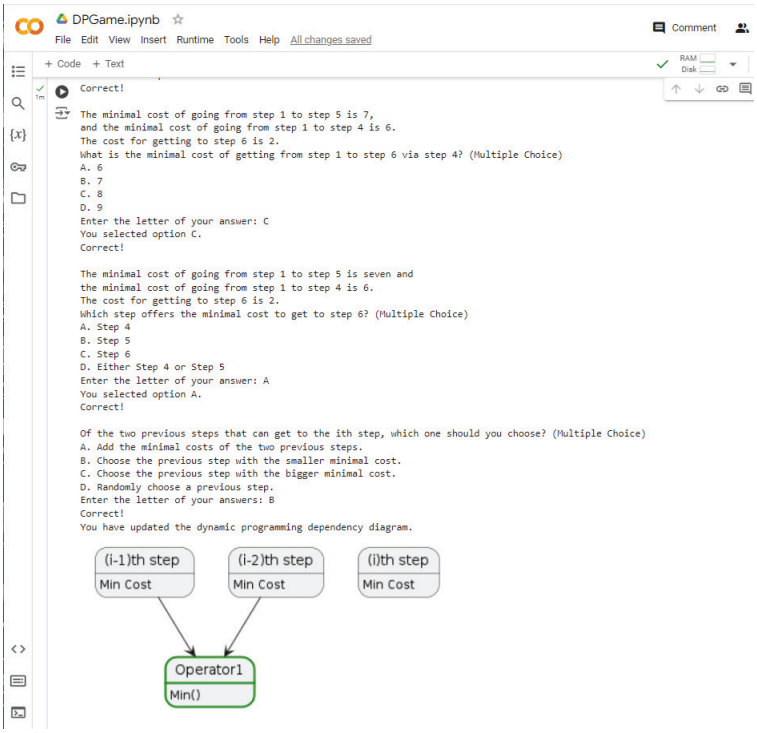

Fig. 4: This is the screenshot of a DPGame session. The dependency diagram is partially constructed.

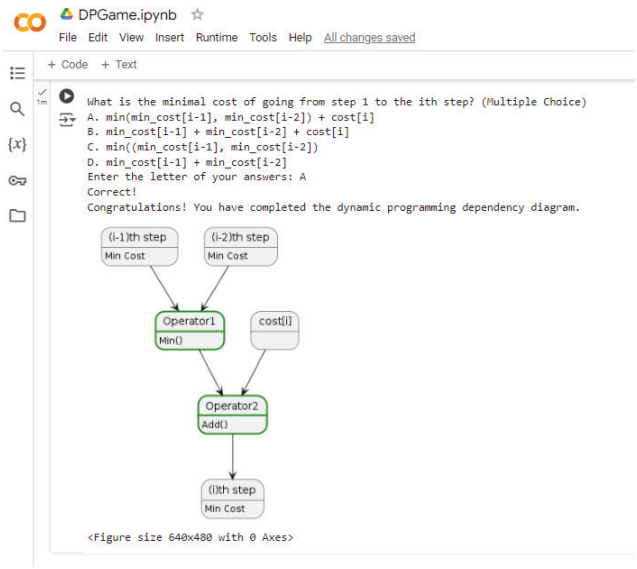

Fig. 5: This is the screenshot of a DPGame session. The dependency diagram is completed.

6 Conclusion and Future Work

Dynamic programming is one of the most difficult techniques in data structure and algorithm design. We have developed a multi-level puzzle game to help students learn the key logic of dynamic programming. In this game, students construct a dynamic programming dependency diagram by answering a sequence of questions, like putting together a visual puzzle. Our goal is to increase student engagement and help them understand the intricate dependency logic between subproblems in dynamic programming. The game is played inside a Python coding environment such as Google Colab, so the game feels like an integral part of programming.

We plan to add more dynamic programming problems to this game and continue incorporating it into our data structures and algorithms courses in the coming semesters. We will conduct qualitative user studies by collecting and analyzing student feedback, as well as quantitative analysis by measuring students' performance in solving dynamic programming problems before and after playing the game.

References

1. LeetCode. `https://leetcode.com/problemset/`, Last accessed on 2024-05-15
2. PlantUML. `https://plantuml.com/`, Last accessed on 2024-05-15

3. Cormen, T., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, 4th edn. (2022)
4. Corral, J.M.R., Balcells, A.C., Estévez, A.M., Moreno, G.J., Ramos, M.J.F.: A game-based approach to the teaching of object-oriented programming languages. Computers & Education **73**, 83–92 (2014). https://doi.org/10.1016/J.COMPEDU.2013.12.013
5. Garcia, I., Pacheco, C., Méndez, F., Calvo-Manzano, J.A.: The effects of game-based learning in the acquisition of "soft skills" on undergraduate software engineering courses: A systematic literature review. Computer Applications in Engineering Education **28**, 1327–1354 (2020). https://doi.org/10.1002/cae.22304
6. Kanika, Chakraverty, S., Chakraborty, P.: Tools and techniques for teaching computer programming: A review. Journal of Educational Technology Systems **49**, 170–198 (2020). https://doi.org/10.1177/0047239520926971
7. Kleinberg, J., Tardos, E.: Algorithm Design. Pearson (2005)
8. Leutenegger, S., Edgington, J.: A games first approach to teaching introductory programming. In: Proceedings of 38th SIGCSE Technical Symposium on Computer Science Education. pp. 115–118 (2007). https://doi.org/10.1145/1227310.1227352
9. Luxton-Reilly, A., Simon, Albluwi, I., Becker, B.A., Giannakos, M., Kumar, A.N., Ott, L., Paterson, J., Scott, M.J., Sheard, J., Szabo, C.: Introductory programming: A systematic literature review. In: Proceedings of Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE). pp. 55–106. ACM (2018). https://doi.org/10.1145/3293881.3295779
10. Popat, S., Starkey, L.: Learning to code or coding to learn? A systematic review. Computers & Education **128**, 365–376 (2019). https://doi.org/10.1016/J.COMPEDU.2018.10.005
11. Priyaadharshini, M., NathaMayil, N., Dakshina, R., Sandhya, S., Shirley, R.B.: Learning analytics: Game-based learning for programming course in higher education. Procedia Computer Science **172**, 468–472 (2020). https://doi.org/10.1016/J.PROCS.2020.05.143
12. Rawat, K., Meenakshi: Dynamic Programming for Coding Interviews: A Bottom-Up Approach to Problem Solving. Notion Press (2017)
13. Scherer, R., Siddiq, F., Viveros, B.S.: A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. Computers in Human Behavior **109** (2020). https://doi.org/10.1016/j.chb.2020.106349
14. Skiena, S.S.: The Algorithm Design Manual. Springer, 3rd edn. (2020)
15. Su, S., Zhang, E., Denny, P., Giacaman, N.: A game-based approach for teaching algorithms and data structures using visualizations. In: Proceedings of SIGCSE Technical Symposium on Computer Science Education. ACM (2021). https://doi.org/10.1145/3408877
16. Tabuti, L.M., de Azevedo da Rocha, R.L., Nakamura, R.: Proposal of method for converting a physical card game to digital for logical reasoning competencies on the data structure subject. In: Proceedings of the IEEE Frontiers in Education Conference. pp. 1–9 (2020). https://doi.org/10.1109/FIE44824.2020.9274001
17. Videnovik, M., Vold, T., Kiønig, L., Bogdanova, A.M., Trajkovik, V.: Game-based learning in computer science education: a scoping literature review. International Journal of STEM Education **10** (2023). https://doi.org/10.1186/s40594-023-00447-2
18. Zhang, J., Atay, M., Caldwell, E.R., Jones, E.J.: Reinforcing student understanding of linked list operations in a game. In: Proceedings of the IEEE Frontiers in Education Conference. pp. 1–7 (2015). https://doi.org/10.1109/FIE.2015.7344132

19. Zhang, J., Atay, M., Smith, E., Caldwell, E.R., Jones, E.J.: Using a game-like module to reinforce student understanding of recursion. In: Proceedings of the IEEE Frontiers in Education Conference. pp. 1–7 (2014). https://doi.org/10.1109/FIE.2014.7044093
20. Zhu, Y.: Visualization techniques for the design and analysis of dynamic programming algorithms. In: Proceedings of the 28 International Conference Information Visualisation. IEEE (2024)