Journal of Information Systems Education

Volume 36
Issue 3
Summer 2025

Teaching Tip *Adventure RPG:* A Text Adventure Game for an Introductory Java Programming Course

Seth J. Kinnett, Tatum Shinedling, and Ben Sunset

Recommended Citation: Kinnett, S. J., Shinedling, T., & Sunset, B. (2025). Teaching Tip: Adventure RPG: A Text Adventure Game for an Introductory Java Programming Course. *Journal of Information Systems Education*, 36(3), 209-223. https://doi.org/10.62273/ASHE6341

Article Link: https://jise.org/Volume36/n3/JISE2025v36n3pp209-223.html

Received:	September 9, 2024
First Decision:	December 11, 2024
Accepted:	March 14, 2025
Published:	September 15, 2025

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Journal of Information Systems Education, 36(3), 209-223, Summer 2025
https://doi.org/10.62273/ASHE6341

Teaching Tip
Adventure RPG: A Text Adventure Game for an Introductory Java Programming Course

Seth J. Kinnett
Tatum Shinedling
Ben Sunset
College of Business
Colorado State University
Fort Collins, CO 80523, USA
seth.kinnett@colostate.edu, tatum.shinedling@colostate.edu,
ben.sunset24@alumni.colostate.edu

ABSTRACT

Engaging students in rudimentary programming concepts is challenging when code examples do not yield practical payoff or are otherwise uninteresting. The purpose of *Adventure RPG* is to enable students to utilize first-semester object-oriented programming concepts to build a text adventure game. In this paper, we describe the incremental development and modular deployment that characterize the game's introduction into the course curriculum. In its earliest stages, the game welcomes players and asks them to select a lineage for their heroes. In its final stage, it is a fully functioning text adventure game utilizing selection statements, loops, methods, classes, objects, arrays, and file input/output. A survey of 60 students revealed that a majority of students scored the activity as highly valuable and self-reported high scores for positivity and participation in the *Adventure RPG* live-coding activities, while also reporting low levels of perceived distraction. The project provides ample opportunities for co-creation and incorporation of student-sourced enhancement ideas. Given the importance of live coding in delivering content in programming courses, this teaching tip provides student-supported content to refresh instructors' live coding exercises and enhance curriculum in introductory Java programming courses.

Keywords: Computer programming, Teaching tip, Code demonstrations, Computing curriculum

1. INTRODUCTION

An introductory programming course is one of the fundamentals of an information systems education. Despite its importance, these courses are often regarded as overly complex and disengaging. In fact, many students in information technology (IT) related majors find introductory programming courses difficult (Ali & Smith, 2014). Furthermore, many students find traditional programming courses to be dull (Lippert & Granger, 1997). Successfully learning to code requires students to fully grasp and apply the concepts to the correct contexts. Traditional education methods, such as slide decks, are generally discouraged in programming pedagogy as they promote a passive approach to learning (Ferreira et al., 2018). Live coding has emerged among the leading strategies for content delivery in programming courses (Selvaraj et al., 2021). In particular, incremental exercises spread over longer time horizons allow students to easily retain information while also being able to go back and refine the work they have previously done (González-Pérez & Ramírez-Montoya, 2022). When students are directly involved in the development of digital assets that represent a complete unit of learning, or learning object, their education improves significantly (Williams et al., 2020). Accordingly, educators must find innovative ways to make learning interactive, engaging, and applicable to real-life scenarios by using live coding techniques over an extended period.

The purpose of this paper is to outline the incremental development of a Java-based text adventure game, *Adventure RPG*, which was implemented via live coding lectures throughout a 16-week introductory Java programming course. *Adventure RPG* addresses the issues outlined by providing a fun, active-learning experience for students in an introductory programming course. *Adventure RPG* incorporates core programming concepts into its development to provide a real-life use of code written in class. According to *Journal of Information Systems Education* guidelines, this Teaching Tip qualifies as a recommended teaching practice, given that it introduces a new teaching method or technique (Lending & Vician, 2012).

The remainder of this Teaching Tip is structured as follows: First, we provide practical justification using the theories of constructivism and active learning. Next, we explain the pedagogical importance of active learning, incorporating fun into course design and games. We then ground the innovation in the context of Education 4.0, the World Economic Forum's framework for higher education in the emergent Fourth Industrial Revolution (4IR). After that, we briefly review the existing literature surrounding the pedagogy of computer programming. To articulate our approach, we then present a narrative surrounding the incremental development of *Adventure RPG* throughout a 16-week semester.

For brevity, the entire source code is not included in this paper, but we invite instructors to request the full source code so they can better understand the comprehensive structure and gameplay mechanics. Throughout this paper, we describe some of the most prominent lecture content, referencing specific code examples that we have included as an appendix. We next present results from a student survey, a collection of student feedback on the exercises, and recommendations for instructors interested in implementing *Adventure RPG* in their programming courses. We conclude with a plan to survey future students and a forecast for future enhancements to the application.

2. BACKGROUND

2.1 Theoretical Foundation: Constructivism

Adventure RPG is a live coding exercise: a technique that leverages the tenets of active learning. Active learning is an instructional strategy that is theoretically grounded in the constructivist learning paradigm. Constructivism is generally attributed to Dewey, who believed that we learn by acquainting ourselves with objects through frequent use (Dewey, 2018). This principle of learning by doing (Kivinen & Ristelä, 2003) depicts the learner as an active participant in the creation of knowledge: a paradigm that accounts for part of its pedagogical popularity (Jones & Brader-Araje, 2002). Piaget, whose conceptualizations of constructivism are seminal, noted that "… all knowledge is tied to action, and knowing an object or an event is to use it by assimilating it to an action scheme…" (Piaget, 1966, pp. 14-15). Key principles of constructivism emphasize that learning is an individual, active, and evolving process (Anthony, 1996).

2.2 Active Learning

While constructivism provides a valuable theoretical framework for how students successfully gain knowledge, active learning serves as its classic operationalization. Active learning is defined by Bonwell and Eison (1991) as any teaching strategy that "involves students in doing things and thinking about the things they are doing" (Bonwell & Eison, 1991, p. 19). Active learning allows students to be involved in their education, reducing the risk of passive content consumption. Examples of active learning strategies in an introductory programming course include project-based learning, class discussion, and field activities (Močinić, 2012). Previous research has shown that employing active learning in the classroom improved student performance, reduced failure rates, and had a greater impact on mastery of cognitive skills (Freeman et al., 2014). Furthermore, active learning aligns with the emergent concept of Education 4.0, the World Economic Forum's framework for curriculum and student experiences to thrive in the emergent 4IR (Elhussein et al., 2023). We developed the *Adventure RPG* application with the goals of Education 4.0 in mind. Digital skills and programming, discipline-specific knowledge, communication, and critical thinking are all elements of *Adventure RPG* that align with the Education 4.0 taxonomy.

2.3 Live Coding

A powerful strategy to integrate active learning in introductory programming education is live coding. Live coding is when the instructor designs and implements a program for the class to follow along and has been found to be significantly more effective than slide decks used during lectures (Rubin, 2013). This strategy fosters engagement within the curriculum and facilitates a hands-on learning experience during class, greatly improving students' chances of success (Menon, 2023; Rubin, 2013). Additionally, Menon (2023) found that students can learn adequate problem-solving skills following an instructor's example.

Live coding enables students to be engaged, actively participate, discuss questions, and gain hands-on experience. It also accommodates students' various levels of coding knowledge; students with more advanced coding skills can assist their peers, resulting in more student engagement. The effectiveness of live coding is shown by student feedback expressing enthusiasm for the live coding exercises (Rubin, 2013). Given the importance of live-coding in the teaching of programming (Raj et al., 2018), the necessity of high-quality live-coding exercises cannot be overstated. Many curricular enhancements are important, but in programming, the literature is especially clear on the importance of live coding to student success (Raj et al., 2018; Rubin, 2013; Selvaraj et al., 2021). However, live coding is only useful if students are engaged in the exercise. Even the best live coding practices can be unsuccessful if students become distracted. It is thus implicitly clear that instructors should endeavor to make their live coding exercises as fun as possible.

2.4 Pedagogical Value of Fun and Games

Using fun in education has been a strategy to engage students and strengthen learning for many years. Research suggests that fun is not a superficial aspect of education; rather, it is essential to incorporate to fight distraction and enhance learning. Fun methods of instruction make learning interesting and engaging for students, resulting in improved concentration and knowledge retention (Mokhtar et al., 2023). Furthermore, fun in the classroom has a significant indirect effect on student attitudes about the subject matter and significant total effect on learning (Tisza, 2021). These findings show how fun can increase student learning while fighting distraction. An important aspect of fun is the positive emotions that are expressed as a result. These positive emotions have a constructive effect on learning, while sadness, anger, and stress negatively affect learning (Tisza et al., 2022). Overall, fun in the classroom makes learning more enjoyable and fights student distraction (Purinton & Burke, 2019), making it an invaluable strategy to incorporate in traditionally difficult subject matters.

One notable way to incorporate fun in an introductory programming course is with games. Games can be considered a classic pedagogy and have consistently been used to teach words, shapes, colors, and basic information throughout childhood (Barber, 2021). Yet, using games in learning has academic merits for all ages. Seethamraju (2011) found the implementation of a business simulation game effective in engaging students. Feedback revealed students enjoy the interactive nature of the simulation and its academic value, which they perceived as more enriching than traditional teaching methods (Seethamraju, 2011). Similarly, game-based learning exercises foster student engagement and in-depth learning (Hartt et al., 2020). It also allows for students to have a deeper understanding of topics that they find challenging (Farkas et al., 2022). Games can be great digital learning tools that support meaningful leaning (Shute & Ke, 2012). Additionally, the development of *Adventure RPG* aims to replicate the high levels of motivation and engagement typically found in gaming for educational purposes (Cheong et al., 2024).

Games appear to be particularly timely interventions today, as instructors seek to improve their classroom experiences for Generation Z students. This generation is especially familiar with digital games and derive a high level of engagement from the social, emotional, and cognitive motivation they experience (Maulana, 2024). Because of this motivation, Generation Z students are very comfortable with games. Generation Z students crave the fun, kinesthetic aspects of games and desire them in learning (Mendoza, 2019).

2.5 The Pedagogy of Computer Programming

The importance of introductory programming courses does not go unnoticed; most information systems programs require at least one programming course (Babb et al., 2014). Although these courses are both important and still prevalent in Information Systems (IS) education, teaching coding efficiently is seen as a challenge. Prior teaching tips do not underestimate the uphill battle that teaching first-time programmers effectively can be. These courses are the foundation for information systems education, and if concepts are not grasped early, students struggle in the long-run when they are enrolled in more advanced computing courses (Zhang et al., 2020). Java is one of the most popular programming languages to use in the information systems curriculum (Smith & Jones, 2021), showing the need for proper introductory Java education.

The reputation of teaching these courses has enabled instructors to explore different and engaging teaching strategies to find what is most successful. Prior literature has warned against the use of traditional lectures leveraging slide decks in programming courses, which can lead to students zoning out (Zhang et al., 2020). The adverse effects of traditional lecture methods highlight the need for more effective teaching strategies to emerge. Prior *Journal of Information Systems Education* Teaching Tips surrounding programming courses have made a range of valuable contributions to the information systems curriculum. Sengupta (2009) provided an excellent scaffolding for coding instruction using comment first coding. Other Teaching Tips (e.g., Cavaiani, 2006; Menon, 2023; Sharma et al., 2020; Zhang et al., 2020) have likewise explored—along with making other valuable contributions—the many merits of live coding in detail and how to implement such techniques in programming courses. This paper extends this strong foundation by offering a comprehensive and interactive semester-long live coding activity.

In a meta-analysis of teaching and learning computer programming, many factors of successful programming instruction were tested. It was found that instructors can shape their teaching to fit the needs of their classes without sacrificing efficiency. Additionally, trying new and effective teaching practices does not obstruct student learning (Scherer et al., 2020). This both recognizes the need for innovative teaching methods and displays the importance of meeting student-specific needs. This approach provides a unique and effective way to teach fundamental introductory programming principles. Overall, programming pedagogy continues to be ripe for innovation and benefits from the use of new and creative teaching practices. Our answer to this call for engaging course content comes in a text adventure game: *Adventure RPG.*

3. THE TEACHING PROCESS

3.1 Introduction to Teaching *Adventure RPG*

Adventure RPG is a Java-based text adventure game that is developed throughout the semester in an introductory Java programming class for information systems students. The topics taught in class are incrementally added to *Adventure RPG* such that students will implement an object-oriented text adventure game by the end of the semester, utilizing solutions native to the modules taught in class (see Table 1). Further, *Adventure RPG* uses incremental development, which aligns with Agile methodologies. Our legacy lecture content largely consisted of comparatively bite-sized, stand-alone exercises. *Adventure RPG*, on the other hand, is built throughout the entire semester, which we suggest provides a sense of continuity and anchors various course concepts in a way that independent exercises do not.

Module	Topic Summary
1	Course Overview, History of Programming Languages
2	Java Fundamentals (data typing, variables, constants)
3	Selection Statements (if/else/else if/switch)
4	Loops (while, do-while, for)
5	Methods & Method Overloading
6	Arrays (one and two-dimensional)
7	Classes & Objects (data encapsulation, constructors, accessors & mutators)
8	Advanced String Manipulation & File I/O

Table 1. Curriculum Summary

The decision to build the game as lecture material as a live coding exercise—rather than as a series of graded assignments—emerged from a blunt assessment of the course's needs. The lecture content was ripe for expansion and enhancement, and we sensed that this was the greatest immediate priority. Instructors should feel empowered to use some or all of *Adventure RPG* as assignments, quizzes, or exams, or otherwise modify our content and approach as their course needs dictate. We now summarize the implementation of the game as taught incrementally through the eight course modules.

Student participation is a key feature of *Adventure RPG*. During live coding exercises, students actively contribute by typing along with the instructor, participating in discussions the instructor facilitates during the activity, and debugging their code if they have an error. In addition to choosing their own output messages, many students collaborated with one another if they missed something. Students are also encouraged to create their own code with *Adventure RPG*. Common student-driven customizations we observed include customization of messaging, additional or custom lineages, new hero roles, and extensions to either Hero or Monster class definitions.

3.2 Programming Basics, Variables, Constants, Primitive Data Types, and Input/Output (I/O)

In the module following course introductions, *Adventure RPG* is briefly introduced along with code surrounding basic variable and constant declarations and program I/O statements, providing students an opportunity to think about unique applications of coding basics. The introduction of input/output commands, including declaration and instantiation of the Scanner class and the suite of output statements related to *System.out.print()*, provides early opportunities to develop messaging and elicit basic inputs to begin the game.

We implement the concept of hero *lineage — 1: ELF, 2: ORC, 3: HUMAN —* as one collection of game start-up prompts, followed by the elicitation of the hero's *role — 1: FIGHTER, 2: MAGE, 3: ROGUE*. Soon after the scanner and data types are introduced, *Adventure RPG*'s code begins (see Figure 1; all figures are provided in the Appendix). The instructor explains importing and creating a Scanner. Next, the instructor demonstrates the declaration of variables and constants for the name and lineage of the hero. Furthermore, the instructor discusses *System.out.println()* to display a message prompting the user to give the hero's name. The instructor discusses the process of getting an integer input from the user by utilizing the assignment statement *intLineage = scr.nextInt()*. Subsequently, selection statements can utilize the constants for more intuitive implementation.

3.3 Selection Statements and Basic Error Catching

After the introduction of selection statements, a natural opportunity exists to extend the basic elicitations from the prior module and implement *if/else if* ladders to the code by using the previously declared constants *ELF*, *ORC*, and *HUMAN* as the parameters for the *if/else if* block. The instructor explains that the integer *intLineage* relies on user input, and with *if/else if* statements, the code can be manipulated to select the lineage that coincides with user input. Figure 2 contains an example of the *if/else* statements that coincide with the user input to select the hero's *lineage*. The instructor may also use this as an opportunity to explain tests for equality compared to assignment statements. Demonstrating *(intLineage == ORC)* as the correct notation and showing the adverse result from using *(intLineage = ORC)* are important dimensions of this code presentation. This allows for a deeper understanding of these concepts while also preventing a common error made by students when first learning *if/else* statements: misconstruing the appropriate use cases for = and ==. To encourage class participation, the instructor can survey the class to suggest the text response to players after a lineage is selected. This approach provides students with a personal touch in the code written, which may not always occur in introductory programming assignments.

3.4 Loops

The select lineage process can be enhanced with error handling when loops are introduced to the course. Specifically, students enhance the lineage selection process from only using *if/else* and *System.exit(0)* statements to prompting the user via a loop and continuing to prompt the user until a correct selection is obtained. This is an opportune time to deploy a *do/while* loop. Students see how the loop serves as built-in error checking by preventing the game from advancing until the user makes a proper selection. To demonstrate a *do/while* loop, we declare a new integer, *intRole.* Then, we encapsulate prior selection statements within *a do/while* loop.

As seen in Figure 3, the *do/while* loop is established to continue looping if the input does not equal FIGHTER, MAGE, or ROUGE. Inside the loop is the prompt for the user to select a fighter type, and an i*f/else if* block that outputs different statements based on what fighter type is chosen by the user. The output statements provide an opportunity to gather input from the class, as well as a creative output when a fighter type is chosen. This can boost participation and allow students own creativity to personalize *Adventure RPG*. We also replicate this loop structure to enhance the lineage selection process. This is an opportune time to add error handling with *try/catch* structures.

3.5 Methods

The code is around 50 lines at this point, so the module on Methods is a good time to clean up the *main()* method and "outsource" functionality to other methods in the main RPG class. Creating three new methods: *void startStory()*, *String startHeroName()*, and *int startHeroLineage()* allows for cleaner code and streamlines the process while also providing examples of both void and non-void methods. To illustrate a void method, we implement *void startStory()*. This method provides a customizable introduction to the context of the game. The use of the method illustrates how some functionality can be packaged and modularized even if there are no return values to be obtained. In our implementation, we print a game title, introduction, and story context: *A kingdom in need of a hero*. To emphasize what makes a void method unique, the instructor stresses the absence of a return value. In contrast, the method *String startHeroName()* illustrates a non-void method. The method uses the process previously written in the code to obtain the hero's name based on user input. To emphasize what makes a non-void method unique, the instructor highlights how a *String* variable needs to be returned. The instructor demonstrates the specific return type of the method and how it requires a return statement with the same data type. Replacing the code previously written with this method shows how methods can be used to simplify code to make it neater while accomplishing the same goal.

As shown in Figure 4, we create the non-void method, *int startHeroLineage()*. Similarly to the method *startHeroName()*, this method contains code that is already written but is now stored in its own method, which reduces the quantity of code remaining in the main method. We demonstrate how to replace the existing *main* method code with a call to *startHeroLineage()*. It returns an integer that corresponds to the hero *lineage*. In addition to the lineage selection process, we also encapsulate the hero's *role* selection in a non-void method *startHeroRole()*. This method assigns the hero's *role* based on user input. The user input correlates with a constant (i.e., *1: FIGHTER, 2: MAGE, 3: ROGUE*). Overall, the different methods highlighted in *Adventure RPG* provide several examples of both void and non-void methods.

3.6 Arrays

Our implementation of an array centers upon the game's map, and we utilize a void method that receives a 2D integer array to create a map for the hero to traverse. The method is created using *void printMap(int[][] map)*. Because classes and objects have not yet been formally introduced, and due to the complexity of building arrays of objects, we opted to use the primitive data type of integer as the foundation for the 2D array. Inside the array, a legend occupies the space. This can include different locations, monsters, and treasure. By assigning the indexes different numbers, the programmer can input different objects to achieve different outcomes.

Integers, while suitable for the base of the map, are less compelling when printed, so we implemented a translation logic to use characters, as outlined in the *void* method *printMap()*, which accepts a 2D integer array as a parameter and prints the map. The main loop of the game requires reprinting the map numerous times. As seen in Figure 5, this method prints a visual representation of the array to the console, providing a map for *Adventure RPG* players. Every constant declared in the array corresponds with a different component of the map, such as MONSTER, HERO, CASTLE, and OBSTACLE. This code also allows for different symbols and colors to represent the different game elements. Setting up the map with constants as the foundation allows it to be customizable in the future and gives students another chance to be creative.

3.7 Classes and Objects

At this point in the code, there are many methods initialized to play *Adventure RPG*. We next add the use of classes and objects to the gameplay through the creation of two new classes. They each accomplish different tasks and provide insight on the different functions of classes. *Hero* (see Figure 6) and *Monster* (see Figure 7) classes are initialized to create hero and monster objects. The *Hero* class begins by defining a public class named "Hero" as well as static constants to represent the three different lineages. This code enables reiteration of the importance of the contrast between static and instance variables. Following the initialization of variables, the hero constructor needs to be created. This constructor establishes the initial values for selected class variables—*name*, *lineage*, and *role*—along with assigning varying combinations of health and damage of the hero based on the lineage selected by the player. Each lineage has different advantages and disadvantages; for example, the *Elf* lineage has the most health but does the least amount of damage, while the *Orc* lineage has the least amount of health but does the most damage.

The *Monster* class is created to introduce enemies to the game for more exciting gameplay. Figure 7 demonstrates the creation of the *Monster* class. It starts with the definition of a public class called "*Monster*" and declares static constants for monster health and type. Instance variables are then created for the characteristics of a monster. The constructor then sets the Boolean variable *isAlive* to true and initializes *monsterType.* It also sets the damage and health of the monster based on the type. For example, a *Goblin* has a health of 10 and a damage of 15. This constructor allows for different monsters to be created that have varying health and damage capabilities.

3.8 File Input/Output

The final module in the course introduces students to basic file input and output using console and graphical user interface solutions. To incorporate this functionality to *Adventure RPG*, we introduce the ability to load different maps from .txt files. This functionality replaces the original hard-coded map and introduces the *loadMap()* method and related functionality. The .txt file can also be altered to showcase important game design elements. For instance, if a student were to make the .txt file the only integer that corresponds with obstacles, the game would not be functional. Additionally, altering the .txt file to include more *Monsters* than the original map would make the game more difficult for players looking for a challenge. Using .txt files provides exciting opportunities to enhance the gameplay of *Adventure RPG*.

To perform file input, a method must be implemented into the code. The method *buildMap()* accepts zero parameters and returns a 2D integer array. We utilize the *JFileChooser* class to read the files. The *buildMap()* method creates a 10 x 10 map from the input file chosen (see Figure 8).

First, the method initializes the key components of the method, such as the 2D integer array, a *File* object, *JFileChooser*, a scanner, and primitive data types. The method prompts the user for the file using *fileChooser.showOpenDialog(null)* and incorporates an *if/else if* to approve or cancel the selection. The method then reads the file and prints an error message if it is not found. Additionally, we create a Scanner object to read and parse the data from the file. The next section of the method reads data from the file and converts it into a 2D integer array. To keep track of the current row and column numbers, the integers *col* and *row* are initialized. The delimiter is set, and values are read as the data is parsed. A catch is thrown for *NumberFormatException* to ensure the file input is always numeric. Finally, the method returns the *map* array to create the map to be traversed by the player.

3.9 Additional Methods

The most significant portion of functionality needed to "stitch the game together" is possible only after the introduction of classes and arrays. Accordingly, this collection of coding exercises could become larger than feasible for live coding within the time allotted; this was the case with our implementation of the game. We propose instructors may address this by posting certain supplemental code snippets on a learning management system and advising students how to add them to their existing code. Methods including *battle()*, *handleMove()*, and the main method's core game-driving logic (see Figure 9) all fall into this category. We would like to allocate more class time to incorporating these methods as live-coding exercises in future iterations of this course. Because game maps can be created at random or read from files, eliminating the file loading portion would be the natural choice if an instructor wanted to devote more time to the advanced methods that are available after Module 7.

The driver of the gameplay consists of a loop that prompts users to select a direction to move, a decision which causes some variation of a call to the *handleMove()* method which tests the player's intended location against the map, determining whether the space is empty, occupied by a monster, or contains the castle the hero seeks. If a player enters a monster-occupied space, the *battle()* method uses random number generation, along with the character's lineage and role, to determine outcomes against monsters. Ultimately, the player wins when all monsters are defeated, and the player reaches the castle square.

4. EVIDENCE

Following the guidance of Lending and Vician (2012), we sought to measure the impact of the pedagogical innovation.

We chose to evaluate student perceptions of the *Adventure RPG* live-coding activities and overall course quality. The Student Response to Instructional Practices (StRIP) instrument, developed by DeMonbrun et al. (2017), is grounded in constructivist theory and measures constructs surrounding student responses to instructional practices. While constructivism provides the theoretical foundation, the StRIP provides practical justification to gauge student reactions to course interventions like *Adventure RPG*. In particular, the StRIP instrument measures the relationship between student perceptions about active instruction interventions and perceptions of the overall course in which they occurred. DeMonbrun et al. (2017) define active instruction "as occurring when students are engaged with the course content in any individual activity" (p. 276). To measure student-perceived engagement with the instruction, the instrument evaluates five dimensions: value, positivity, participation, distraction, and evaluation of the overall course.

DeMonbrun et al. (2017) describe value as the extent to how students see an activity as worthwhile. It determines whether students found benefit in the activity, saw value in the activity, and if they felt their effort was worth it. A high value score indicates that students saw value in the activity, the time spent, and the effort involved. The low end of this factor means students disagree that they saw value in the activity, the time spent, and the effort they put in. Positivity includes the positive or negative feelings students have about the activity and classroom environment. A high positivity score shows students feel positively about the activity, instructor, and classroom environment, while a lower positivity score displays the opposite.

Participation measures the degree of student participation or resistance. A high participation score exhibits students showed the positive parts of behavioral engagement, participated actively, and put effort into the activity. A low participation score displays students showing open resistance, passive nonverbal resistance, or partial compliance. Open resistance includes student objection in ways that are not constructive. Passive nonverbal resistance are any activities where students are appearing to engage but are doing something else. Partial compliance occurs when students complete a task with minimal effort or while being preoccupied with the structural details. Distraction occurs when students distract themselves or others during learning. A high distraction signifies students are focused and avoid other activities. Yet, a low distraction score indicates students are engaging in distracting behaviors. The last term, evaluation, reflects how students rate the course or instructor at the end of the semester. If the evaluation score is higher, students rate the course and instructor highly while a low evaluation score means students rate the course and instructor poorly (DeMonbrun et al., 2017).

We modified DeMonbrun et al.'s (2017) questionnaire to fit the context and goals of our study. Namely, it was impractical to include the full bank of survey questions originally used by DeMonbrun et al. which would necessitate a very high sample size for our survey threshold. Although the literature varies on recommendations for respondent size versus number of survey items, Pett et al. (2003) recommend 10-15 respondents per survey item. Therefore, utilizing the original survey would, at minimum, require 170 respondents; considering that our classes rarely exceed 20-30 students, we selected a single question from each construct.

Throughout this reduction, we were mindful to preserve the original construct of measure even when only a single question is used. Namely, we asked: Is there a risk that the question might actually measure a different construct? In the scale at hand, these are not concerns. For example, for the *Value* dimension, we used the survey item: "I saw the value in the *Adventure RPG exercises*." We suggest that it is reasonable to conclude that the omission of multiple questions surrounding this construct could hardly lead to the conclusion that this item is not a good faith attempt to measure the construct *Value.* Such considerations are standard when adapting a scale to a unique scenario. The resultant five-question survey asks students to rate how much they agree with each statement by selecting from a Likert scale of 1 (Strongly Disagree) to 5 (Strongly Agree). Table 2 outlines the five factors measured, associated survey questions, and the mean & standard deviation of the responses. Each question was leveraged directly from the general scale proposed by DeMonbrun et al. (2017).

Factor	Survey Question	Results
Value	I saw the value in the Adventure RPG exercises.	*M* = 4.37, *SD* = 0.69
Positivity	I enjoyed developing the Adventure RPG game.	*M* = 4.17, *SD* = 0.81
Participation	I participated actively (or attempted to) in the Adventure RPG game development.	*M* = 4.38, *SD* = 0.72
Distraction	I surfed the internet, checked social media, or did something else instead of building out Adventure RPG.	*M* = 1.68, *SD* = 0.91
Evaluation	Overall, this was an excellent course.	*M* = 4.17, *SD* = 0.85

Table 2. Survey Findings

A total of 60 students completed the survey (*N = 60*). As shown in Table 2, students generally reported strong agreement with the value of the exercises, their perceptions of positivity, self-assessment of participation of the exercises, and low levels of distraction during the exercise. Students generally agreed the course overall was excellent. 92% of respondents agreed or strongly agreed they saw value in the *Adventure RPG* exercises. 78% agreed or strongly agreed that they enjoyed developing the game. 90% agreed or strongly agreed in their active participation in the exercises. Only 5% reported any agreement with our questions about distraction. Most students reported low levels of distraction and did not agree with statements about multi-tasking or simply ignoring the *Adventure RPG* live-coding exercises. Finally, 80% of students agreed or strongly agreed with our question about overall course excellence.

All of the positive factors exceeded the mean value for student perception of overall course excellence. It is likely that the game improved overall course perception scores, and students' perceived value, positivity, and participation exceeded the degree to which they favor the course overall. More work remains to be done to ensure the course overall is perceived highly, but the relative popularity of the *Adventure RPG* live coding exercises suggests it is a worthy compendium to programming course curriculum. Student support of the activities was generally strong and was found in both course evaluations, verbal ad-hoc communications, and even a nomination of the instructor for a university-wide teaching award. Selected feedback from these students is shown in Table 3.

The responses display the effectiveness of the project in teaching an introductory programming course. Notably, many students highlight how they felt *Adventure RPG* was more beneficial than traditional exercises. Students felt *Adventure RPG* deepened their understanding of the topic, increased their participation, had a real-world purpose, and allowed them to retain the material better. The student perception to *Adventure RPG* shows students found the tool both beneficial and fun in the learning environment. Collectively, *Adventure RPG* is an endeavor students view as valuable, perceive positively, actively participate in, and engage with minimal distraction. Anecdotal student feedback is positive.

The RPG example he constantly uses and updates throughout the semester has been a far better way to learn the material for me. I actually understand the difficult content better using his project than I do with the book.

I found myself referencing the RPG all the time to help learn concepts and figure out all the working parts in one place. I felt the RPG example was easier to understand and better than some of the quick Java projects we did during lecture.

Building the Java RPG brought the concepts from class to life in a way that was both engaging and practical. When paired with Zybook assignments, it created a well-rounded learning experience that made the material stick better than traditional homework.

I really liked that it was a universal use case, used lots of different variables and methods, was comprehensive amongst basic Java topics, and that it was group-led but individually driven.

The Adventure RPG deepened my understanding of Java in a fun and engaging way. Utilizing different methods to create different aspects of a game was a good way to show real world application of the concepts we were learning, heightening the enjoyment of Java programming.

Compared to the typical practice exercises, the Adventure RPG kept me captivated and helped to bring all content in the course together. I still use that project whenever I need a reference and reminder of all the needed basics in Java!

[The RPG game] was helpful to see each piece of criteria we learned throughout the semester come into one piece of code. It helped see how all the components work together and is a good piece to look back at to see all the basics of Java we learned.

One of the most rewarding projects that we did in [the course] was the RPG game. At first it seemed daunting, but the way it was formatted (learn new topic in class, apply it to normal code, apply it to RPG) made it more fun to learn!

Adventure RPG was my favorite part of the course. When we switched from the regular lecture to Adventure RPG, I felt like I knew the concepts better. It was so exciting to learn something fun that had a tangible outcome. Learning to code in Java was definitely a hard process, but Adventure RPG made it fun and easier!

Adventure RPG was a great way to learn and made a difficult course more enjoyable!

I really liked that Adventure RPG was introduced at the beginning of the semester and as we learned more the game became more complex. It helped my learning, and I paid more attention to it because it was interesting.

Table 3. Student Feedback

5. DISCUSSION AND TEACHING SUGGESTIONS

Reflecting on the game's implementation in our course, we observed substantial positive feedback from students. We did not observe much non-engagement during the coding sessions, similar to the survey findings. The biggest challenge that occurred during the game's implementation was finding sufficient time to cover every code snippet during class. We tended to favor using *Adventure RPG* as one of several examples during each module. In some cases, more advanced topics (e.g., arrays) needed to be introduced first with simpler examples prior to moving to the *Adventure RPG* exercise. This led us to post some of the *Adventure RPG* code on the course learning management system rather than covering every single line in class. Therefore, our advice to instructors is to consider whether there are opportunities to entirely replace prior lecture content, which we did for certain modules earlier in the game's development, or whether the game content is to be added on top of other examples. Similarly, instructors could consider—as we continue to—making more structural changes in the course to provide even more time to focus on *Adventure RPG*.

So far, *Adventure RPG* has been incorporated primarily as live-coding lecture content and not directly as graded assignments. This allowed the game to address our greatest need: fresh lecture content to engage students in core concepts. At the time, we were not lacking in assignment content, which drove some of our decisions. We believe there are several avenues that instructors could pursue to leverage *Adventure RPG* for their own course needs. Some examples include leveraging the code base to develop graded assignments, including aspects of *Adventure RPG* on quizzes and exams, and crowd-sourcing extensions, revisions, and enhancements. The latter activity was the driver behind our first extension to the lecture content, specifically eliciting student enhancements and extensions to the game as end-of-semester extra credit. Students have submitted a variety of collateral material as extra credit ranging: from minor tweaks to class definitions to new suites of functionality (e.g., an item inventory management scheme) to map enhancements incorporating terrain (e.g., forest, desert) to comprehensive rebuilds using dialog boxes (i.e., leveraging the *JOptionPane* class methods).

Developing graded components of the game or requiring students to submit their version of the code from lecture, may be advantageous in further reducing the already small amount of student-reported distraction during the exercises. Ultimately, however, we believe that students should be supplied any missed source code following assignment due dates, so every student has the opportunity to construct a fully functional game by the end of the semester. If critical aspects of the game are purely graded assignments, whereby students are 100% responsible for the code, the ultimate effect may not ultimately be advantageous since it will introduce substantial variation in each student's code, especially as the course content becomes more advanced near the end of the term. Providing the code immediately after lecture also allows for students to revise and refine their code if they missed something during live coding.

While *Adventure RPG* is effective in providing an innovative teaching solution that enhances student learning and participation, enhancements can be made. For example, as currently implemented, *Adventure RPG* does not require any skill to play the game. It determines the outcome of battles based on a random number and the user has limited input that can influence the game's outcome. A solution to this could be to implement a way for the user to input a number, and if that number is guessed correctly, or matches the integer generated randomly, they win the battle. On the other hand, the game's relative simplicity, both to develop and to play, can also be viewed as one of its strengths.

Another improvement that could be made based on user skill during the battle sequence is if the user strikes a certain key quickly enough, they deal more attack damage. Leveling up characters and deploying upgrades based on experience points is another avenue that instructors could pursue, as long as the intent is to deploy the program in an introductory course; however, we were apprehensive about increasing complexity and recommend instructors reserve too many enhancements for upper-level programming courses. We also invite instructors to re-create our code in other languages to fit their own needs and modify it to align with their selected curriculum where applicable.

6. CONCLUSIONS

Not all live-coding exercises are created equal. There are individual differences in every live-coding exercise that influence constructs surrounding a student's intention to engage with the activity via dimensions such as perceived value and perceived positivity. An effective activity evokes a positive student response to instructional practices, reflected in higher levels of these variables. *Adventure RPG* appears to be a useful addition to lecture live coding content in an introductory Java programming course. It may also function well as input for student programming assignments.

The results of our sample of 60 students indicate high agreement with perceptions of the *Adventure RPG* game's value, associated feelings of positivity and participation, while also demonstrating low levels of distraction. Students also rated the course overall with high marks, though they had higher perceptions of factors evaluating *Adventure RPG* than of the course overall. Students have offered substantial qualitative feedback, much of which favorably compares the RPG exercises to legacy exercises or underscores the importance of RPG as a supplement to textbook readings and exercises.

We invite other instructors to evolve and enhance the code as they see fit. This paper contributes to IS education by providing a collection of source code and guidance to enhance the live coding portion of programming course lectures. Given the importance of live coding in programming pedagogy, this paper expands available course material to enhance this aspect of curriculum. Moreover, the game has piqued student interest. In the competitive landscape characterizing today's higher education institutions, providing exciting and positive experiences may be ever more necessary to recruit and retain students in IS programs.

7. ACKNOWLEDGEMENTS

The authors would like to thank James Beattie Jr. for his contributions to this project.

8. REFERENCES

Ali, A., & Smith, D. (2014). Teaching an Introductory Programming Language in a General Education Course. *Journal of Information Technology Education: Innovations in Practice*, 13, 57-67. https://doi.org/10.28945/1992

Anthony, G. (1996). Active Learning in a Constructivist Framework. *Educational Studies in Mathematics*, 31(4), 349-369. https://doi.org/10.1007/BF00369153

Babb, J., Longenecker, H. E., Baugh, J., & Feinstein, D. (2014). Confronting the Issues of Programming in Information Systems Curricula: The Goal is Success. *Information Systems Education Journal*, 12(1), 42-72.

Barber, C. S. (2021). When Students Are Players: Toward a Theory of Student-Centric Edu-Gamification Systems. *Journal of Information Systems Education*, 32(1), 53-64. https://doi.org/10.1287/isre.2020.0968

Bonwell, C., & Eison, J. (1991). *Active Learning: Creating Excitement in the Classroom* (Vol. 1). Association for the Study of Higher Education.

Cavaiani, T. P. (2006). Object-Oriented Programming Principles and the Java Class Library. *Journal of Information Systems Education*, 17(4), 365-368.

Cheong, C., Filippou, J., & Cheong, F. (2024). Towards the Gamification of Learning: Investigating Student Perceptions of Game Elements. *Journal of Information Systems Education*, 25(3), 233-244.

DeMonbrun, M., Finelli, C. J., Prince, M., Borrego, M., Shekhar, P., Henderson, C., & Waters, C. (2017). Creating an Instrument to Measure Student Response to Instructional Practices. *Journal of Engineering Education*, 106(2), 273-298. https://doi.org/10.1002/jee.20162

Dewey, J. (2018). *Democracy and Education*. Myers Education Press.

Elhussein, G., Leopold, T., Silva, A., & Zahidi, S. (2023). *Defining Education 4.0: A Taxonomy for the Future of Learning* [White Paper]. World Economic Forum. https://www3.weforum.org/docs/WEF_Defining_Education_4.0_2023.pdf

Farkas, B., Shang, Y., & Alhourani, F. (2022). Teaching Tip: Teaching Business Process Concepts in an Introductory Information Systems Class: A Multi-Level Game-Based Learning Approach. *Journal of Information Systems Education*, 33(4), 306-323.

Ferreira, C. M., Santos, A. I., & Serpa, S. (2018). Electronic Slideshow Presentations in the Higher Education Teaching and Learning Process. *Journal of Education and Training Studies*, 6(2), 120. https://doi.org/10.11114/jets.v6i2.2818

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active Learning Increases Student Performance in Science, Engineering, and Mathematics. *Proceedings of the National Academy of Sciences of the United States of America*, 111(23), 8410-8415. https://doi.org/10.1073/pnas.1319030111

González-Pérez, L. I., & Ramírez-Montoya, M. S. (2022). Components of Education 4.0 in 21st Century Skills Frameworks: Systematic Review. *Sustainability*, 14(3), 1-32. https://doi.org/10.3390/su14031493

Hartt, M., Hosseini, H., & Mostafapour, M. (2020). Game On: Exploring the Effectiveness of Game-Based Learning. Planning Practice and Research, 35(5), 589-604. https://doi.org/10.1080/02697459.2020.1778859

Jones, M. G., & Brader-Araje, L. (2002). The Impact of Constructivism on Education: Language, Discourse, and Meaning. *American Communication Journal*, 5(3), 1-9.

Kivinen, O., & Ristelä, P. (2003). From Constructivism to a Pragmatist Conception of Learning. *Oxford Review of Education*, 29(3), 363-375. https://doi.org/10.1080/03054980307442

Lending, D., & Vician, C. (2012). Writing IS Teaching Tips: Guidelines for *JISE* Submission. *Journal of Information Systems Education*, 23(1), 11-18.

Lippert, S., & Granger, M. (1997). Peer Learning in an Introductory Programming Course. *Proceedings of the 12th Annual Conference of the International Academy for Information Management* (pp. 123-130).

Maulana, M. R. (2024). Exploring Game Playing Motivation in Generation Z: An Ethnographic Approach. *Journal of Informatics and Computer*, 1(2), 15-23.

Mendoza, K. (2019). Engaging Generation Z: A Case Study on Motivating the Post-Millennial Traditional College Student in the Classroom. *US-China Foreign Language*, 17(4), 157-166. https://doi.org/10.17265/1539-8080/2019.04.002

Menon, P. (2023). Teaching Tip: An Example-Based Instructional Method to Develop Students' Problem-Solving Efficacy in an Introductory Programming Course. *Journal of Information Systems Education*, 34(1), 1-15.

Močinić, S. N. (2010). Active Teaching Strategies in Higher Education. *Metodički Obzori,* 7(2), 97-105. https://doi.org/10.32728/mo.07.2.2012.08

Mokhtar, N., Xuan, L. Z., Lokman, H. F., & Mat, N. H. C. (2023). Theory, Literature Review, and Fun Learning Method Effectiveness in Teaching and Learning. *International Journal of Social Science and Education Research Studies*, 3(8), 1738-1744. https://doi.org/10.55677/ijssers/V03I8Y2023-30

Pett, M. A., Lackey, N. R., & Sullivan, J. J. (2003). *Making Sense of Factor Analysis*. SAGE Publications. https://doi.org/10.4135/9781412984898

Piaget, J. (1966). Biologie et connaissance. *Diogène*, 54, 3.

Purinton, E. F., & Burke, M. M. (2019). Student Engagement and Fun: Evidence from the Field. *Business Education Innovation Journal*, 11(2), 133-140.

Raj, A., Richard, H., Patel, J. M., & Erica, H. (2018). Role of Live-Coding in Learning Introductory Programming. *Proceedings of the 18th Koli Calling International Conference on Computing Education Research* (pp. 1-8). https://doi.org/10.1145/3279720.3279725

Rubin, M. J. (2013). The Effectiveness of Live-Coding to Teach Introductory Programming. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 651-656). https://doi.org/10.1145/2445196.2445388

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2020). A Meta-Analysis of Teaching and Learning Computer Programming: Effective Instructional Approaches and Conditions. *Computers in Human Behavior*, 109, 1-18. https://doi.org/10.1016/j.chb.2020.106349

Seethamraju, R. (2011). Enhancing Student Learning of Enterprise Integration and Business Process Orientation Through an ERP Business Simulation Game. *Journal of Information Systems Education*, 22(1), 19-29.

Selvaraj, A., Zhang, E., Porter, L., & Soosai Raj, A. G. (2021). Live Coding: A Review of the Literature. *Annual Conference on Innovation and Technology in Computer Science Education* (pp. 164-170). https://doi.org/10.1145/3430665.3456382

Sengupta, A. (2009). Teaching Tip: CFC (Comment-First-Coding): A Simple Yet Effective Method for Teaching Programming to Information Systems Students. *Journal of Information Systems Education*, 20(4), 393-400.

Sharma, M., Biros, D., Ayyalasomayajula, S., & Dalal, N. (2020). Teaching Tip: Teaching Programming to the Post-Millennial Generation: Pedagogic Considerations for an IS Course. *Journal of Information Systems Education*, 31(2), 96-105.

Shute, V. J., & Ke, F. (2012). Games, Learning, and Assessment. In *Assessment in Game-Based Learning: Foundations, Innovations, and Perspectives* (pp. 43-58). Springer New York. https://doi.org/10.1007/978-1-4614-3546-4_4

Smith, T. C., & Jones, L. (2021). First Course Programming Languages Within US Business College MIS Curricula. *Journal of Information Systems Education*, 32(4), 283-293.

Tisza, G. (2021). The Role of Fun in Learning. *Extended Abstracts of the 2021 Annual Symposium on Computer-Human Interaction in Play* (pp. 391-393). https://doi.org/doi.org/10.1145/3450337.3483513

Tisza, G., Sharma, K., Papavlasopoulou, S., Markopoulos, P., & Giannakos, M. (2022). Understanding Fun in Learning to Code: A Multi-Modal Data Approach. *Proceedings of Interaction Design and Children* (pp. 274-287). https://doi.org/10.1145/3501712.3529716

Williams, A. R., Windle, R., & Wharrad, H. (2020). How Will Education 4.0 Influence Learning in Higher Education? *Journal of Learning Development in Higher Education*, 17, 1-18. https://doi.org/10.47408/jldhe.vi17.572

Zhang, X., Crabtree, J. D., Terwilliger, M. G., & Jenkins, J. T. (2020). Teaching Tip: Teaching Introductory Programming From A to Z: Twenty-Six Tips From the Trenches. *Journal of Information Systems Education*, 31(2), 106-118.

AUTHOR BIOGRAPHIES

Seth J. Kinnett is a clinical professor of computer information systems in the College of Business at Colorado State University. He earned a Ph.D. in Computer & Information Sciences from the Eugene P. Jarvis College of Computing and Digital Media at DePaul University in Chicago.

Tatum Shinedling is a junior undergraduate honors student pursuing a Bachelor of Science in Business Administration with dual concentrations of Computer Information Systems and Accounting in the College of Business at Colorado State University.

Ben Sunset earned a Bachelor of Science in Business Administration with a concentration in Computer Information Systems from Colorado State University's College of Business. He is currently a Product Cybersecurity Engineer at Woodward, Inc., Fort Collins, Colorado.

APPENDIX

Code Samples

```
Scanner scanner = new Scanner(System.in);

String strHeroName;

int lineage, role; //see constants for translation

final int ELF = 1;

final int ORC = 2;

final int HUMAN = 3;
```

Figure 1. Preliminary Variable & Constant Declarations

```
if (intLineage == ELF) {
  System.out.println("The Elf, a wise choice. The elves have been cunning
      magicians for centuries");
  }
  else if (intLineage == ORC) {
    System.out.println("The Orc. Often misunderstood, the Orcs are formidable
     warriors");
  }
  else if (intLineage == HUMAN) {
    System.out.println("The Human. A balanced hero with modest attack and magic.");
  }
  else { //catch all for anything beyond 1, 2, or 3
      System.out.println("Error, please select 1, 2, or 3.");
      System.exit(0);
  }
```

Figure 2. Selection Statements in *Adventure RPG*

```
do {
  try{
    System.out.println("Enter hero role: 1:FIGHTER, 2:MAGE, 3:ROGUE");
    strRole = scr.next()
    intRole = Integer.parseInt(strRole);
    if (intRole == FIGHTER) {
      System.out.println("I'm shaking in my boots");
    }
    else if (intRole == MAGE) {
      System.out.println("Oh, got ourselves a brainiac I see");
    }
    else if (intRole == ROGUE) {
      System.out.println("Don't pick my pocket :/");
    }
    else {
      System.out.println("Not a quick study, I see. Try again.");
      }
  } catch(NumberFormatException ex) {
  //error handling when converting int to String
  }
} while ((intRole!=FIGHTER) && (intRole!=MAGE) && (intRole!=ROGUE) );
```

Figure 3. do/while Loop for Selecting Hero Role

```
public static int startHeroLineage(Scanner scr, String strHeroName) {
  final int ELF = 1 , ORC = 2, HUMAN = 3;
  int intLineage=0;
  String strLineage;
  do {
    try {
   System.out.println("What is"+strHeroName+"'s lineage?\n1:ELF\n2:ORC\n3:HUMAN");
    strLineage = scr.next();
    intLineage = Integer.parseInt(strLineage);

    //customize user message
    if (intLineage == ELF) {
      System.out.println("The Elf, a wise choice. The elves have been cunning
                magicians for centuries");
    }
    else if (intLineage == ORC) {
      System.out.println("The Orc. Often misunderstood, the Orcs are
                formidable warriors");
      }
      else if (intLineage == HUMAN) {
        System.out.println("The Human. A balanced hero with modest attack and
                  magic.");
      }
      else { //catch all for anything beyond 1, 2, or 3
        System.out.println("Error, please select 1, 2, or 3.");
      }
       } catch (NumberFormatException ex) {
        //error handling when converting int to String
      }
    } while ((intLineage < 1)||(intLineage > 3)); //only loop if we get invalid values

  return intLineage;
}
```

Figure 4. Hero Lineage Selection as a Method

```
public static void printMap(int[][] map) {
// Loop through each square, print depending on what's in the square
  for (int i = 0; i < map.length; i++) {
    for (int j = 0; j < map[i].length; j++) {
      if (map[i][j] == EMPTY) { // Empty - Blank spot
        System.out.print(ANSI_BLACK + "- " + ANSI_RESET);
      } else if (map[i][j] == CASTLE) { // Castle - Win condition!
        System.out.print(ANSI_PURPLE + "E " + ANSI_RESET);
      } else if (map[i][j] == OBSTACLE) { // Obstacle
        System.out.print(ANSI_GREEN + "O " + ANSI_RESET);
      } else if (map[i][j] == HERO) { // That's us - Hero
        System.out.print(ANSI_YELLOW + "H " + ANSI_RESET);
      } else if (map[i][j] == MONSTER) { // Monster Position
        System.out.print(ANSI_RED + "M " + ANSI_RESET);
      }
    } // End Nested For Loop
    System.out.println(); // Spacing
  } // End For Loop
} // End printMap
```

Figure 5. Code Snippet on Map/Array Implementation

```
public class Hero {
    public static final int ELF = 1;
    public static final int ORC = 2;
    public static final int HUMAN = 3;
    String name;
    int health;
    int damage;
    static int wins = 0;

    boolean isAlive = true;
    int lineage;
    int role;

    public Hero(String name, int lineage, int role) {
        this.name = name;
        this.lineage = lineage; //elf, orc, human
        this.role = role; //Rogue,Mage,Fighter

        // Set health and damage based on lineage.
        if (this.lineage == ELF) {
            health = 70;
            damage = 15;
        } else if (this.lineage == ORC) {
            health = 55;
            damage = 20;
        } else if (this.lineage == HUMAN) {
            health = 80;
            damage = 10;
    }
```

Figure 6. Partial Hero Class Definition

```
public class Monster {
    final static int MIN_HEALTH = 0;
    //Base Damage and Base Health
    public static final int GOBLIN=1;
    public static final int SKELETON=2;
    public static final int GUARDIAN=3;
    int monsterType;
    int health;
    int damage;
    boolean isAlive;

    //constructor
    public Monster(int monsterType) {
        isAlive = true;
        this.monsterType = monsterType;
        // Set damage and health depending on type
        if (monsterType == GOBLIN) {
            health = 10;
            damage = 15;
        } else if (monsterType == SKELETON) {
            health = 15;
            damage = 10;
        } else if (monsterType == GUARDIAN) {
            health = 20;
            damage = 5;
        }
}
```

Figure 7. Partial Monster Class Definition

```
public static int[][] buildMap() {
  int[][] map = new int[10][10];
  File file = null;
  JFileChooser fileChooser = new JFileChooser();
  Scanner input = null;
  String strValue; // String verion of place value being read in
  int selection,value;
  // Selection is approve/cancel, value is the integer value being read in

  // Prompt user for file to use
  selection = fileChooser.showOpenDialog(null);
  if (selection == JFileChooser.APPROVE_OPTION) {
    file = fileChooser.getSelectedFile();
    } else if (selection == JFileChooser.CANCEL_OPTION) {
    System.out.println("User hit cancel");
    System.exit(0);
  }

  // Create scanner in file
  try {
    input = new Scanner(file);
    } catch (FileNotFoundException ex) {
      System.out.println("Didn't find file");
      System.exit(0);
    }

    // Read data from file into a 2D int array
    // Declaring variables
    int col = 0, row = 0;
    // Initialize a counter to keep track of the current row and column number

    try {
      input.useDelimiter(",|\n"); // Use delimiters of , AND \n
      while (input.hasNext() && row < 10) {//Keep within bounds, and loop through map.
      strValue = input.next(); // Get next value
      strValue = strValue.trim(); // Trim it
.     value = Integer.parseInt(strValue); // Get the integer value
      map[row][col] = value; // Set value to location in array
      col++; // Increment place value
      if (col == 10) { // Like a typewriter, done with this column? Next row.
        col = 0;
        row++;
        }
      }
    }catch (NumberFormatException e) {
      System.out.println("Invalid file. Non-numeric characters cannot be in the
      map.");
      System.exit(0);
      }
//     */
    return map;
   }
```

Figure 8. buildMap() Method

```
while (hero.isAlive) { // Game loop - While Alive
  String move = scanner.nextLine();// Get input, store as string
  switch (move) { // What are we doing next?
     case "a":   // Left
     handleMove(hero, map, heroX - 1, heroY);
     // Modify position, and pass location, hero, and map down to update.
     break;
     case "d":  // Right
        handleMove(hero, map, heroX + 1, heroY);
        break;
     case "w":  // Up
        handleMove(hero, map, heroX, heroY - 1);
        break;
     case "s":  // Down
        handleMove(hero, map, heroX, heroY + 1);
        break;
     }
  printMap(map);
  System.out.println("Enter move (a for left, d for right, w for up, s for down):");
}
```

Figure 9. Core Logic Prompting User to Enter a Direction and Looping Back

ISSN: 2574-3872 (Online) 1055-3096 (Print)