IMPROVING STUDENT ENJOYMENT, TIME MANAGEMENT, AND PERFORMANCE USING A SECURE, IMMERSIVE GAMIFIED LEARNING PLATFORM

Mac Malone

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2023

Approved by:

Fabian Monrose

Diane Pozefsky

Tessa Joseph-Nicholas

Jay Aikat

Thomas Ball

ABSTRACT

Mac Malone: Improving Student Enjoyment, Time Management, and Performance
Using a Secure, Immersive Gamified Learning Platform
(Under the direction of Fabian Monrose)

We analyze the thesis that a fully immersive gamified learning platform benefits students, improving their enjoyment, time management, and performance by engaging them with the subject matter. We assert that an immersive experience centered around a true game produces better results than exercises with just a few gamified elements (e.g., points, badges, and leaderboards), which are the most common forms of gamification. To this end, we present Riposte, an online gamified learning platform with a 2D action game at its core. To analyze student behavior and support less gamified assignments (to serve as contrasting examples), Riposte is designed to be a single, unified experience for students, with an integrated development environment (Jupyter Notebook) built in. In outlining this system, we give particular emphasis to its security, as it was developed for use in the "Introduction to Computer Security" course at our university. The topic of cybersecurity presents a number of interesting challenges for an online gamified learning platform – aspects of the framework must be insecure enough to be hackable, but secure enough not to be abused. We evaluate the system over three semesters, highlighting many pitfalls and lessons learned but, in the end, demonstrating that it successfully improved student enjoyment, time management, and performance. Furthermore, based on our experiences with the platform, we offer a number of suggestions and recommendations that we hope will help those considering gamification as a winning educational strategy in computer science, cybersecurity, and beyond.

ACKNOWLEDGEMENTS

I would like to thank Roman Rogowski, Deven Desai, Ryan Court, Forest Li, and Kedrian James for their assistance and guidance during the early stages of the Riposte framework. I would also like to thank Murray Anderegg, Daniel Cowhig, Mark Snyder, and all that folks who work at our department's IT help desk. They made Riposte's large infrastructure a relatively painless endeavour, relieving me of the burden of managing hardware, networking, backups, or crashes, while giving me great freedom in configuring our machines.

A big thanks goes to Jan Werner for helping with the many aspects of Riposte research (teaching, administration, testing, etc.) and just generally being a great mentor during his time at Chapel Hill. I similarly appreciate the help of Yicheng Wang, a fellow student and developer. His vast knowledge of statistics helped cover my deficiencies when I was just starting out and continue to be a vital resource whenever I get stuck.

Most of all, I would like to thank my adviser, Fabian Monrose. Without his outreach after I took his course, I would likely not presently be a PhD student in Computer Science. He has guided me steadily through the many twists and turns of my late undergraduate and graduate career and introduced me to the world of research I previously knew very little about. I do not think I could have had a better advisor.

I would also like to thank my other committee members: Diane Pozefsky, Tessa Joseph-Nicholas, Jay Aikat, and Thomas Ball. I have interacted with all of you in various forms, borne witness to your expertise, and developed great respect for each of you. It honors me that you would serve on my committee. Last but not least, I would like to thank all those who took our course, used our platform, and played our game. Your insights and experiences are what make this thesis possible and many of your efforts have made a lasting impression upon me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A-D	Anderson-Darling (statistical test)
API	Application Programming Interface
CBC	Cipher Block Chaining
CBL	Challenge-Based Learning
CD	Continous Development
CI	Continous Integration
CTF	Capture-The-Flag (competition)
HTTPS	HyperText Transfer Protocol Secure
IT	Information Technology
KC	Knowledge about Concepts
KH	Knowledge about How to proceed
KM	Knowledge about Mistakes
KP	Knowledge about Performance
KR	Knowledge about Results
KTC	Knowledge about Task Constraints
PGP	Pretty Good Privacy
PvP	Player versus Player
PvE	Player versus Environment
REST	REpresentational State Transfer
TLS	Transport Layer Security
UI	User Interface
VM	Virtual Machine
WSS	WebSocket Secure

CHAPTER 1: INTRODUCTION

Given the increasing reliance on technology throughout every aspect of modern life, building a secure digital ecosystem is critical. Unfortunately, even as cyberspace continues to transform the way we live, many vulnerabilities in critical infrastructure are left unchecked, exposing us to a myriad of threats. Today, cybersecurity attacks and breaches are all too familiar events, underscoring the need for better defenses.

Sadly, the reality is that we still lack a cyber-savvy work force that can help defend against even known systemic risks. Training the next generation of cybersecurity experts requires teaching today's students *hands-on skills*. Indeed, Mason and Pike [54] may have put it best: "educating a cybersecurity professional is similar to training a pilot, an athlete or a doctor. Time spent on the task for which the person is being prepared is *critical* for success." (Emphasis added.) However, the success of such education depends *heavily on how interesting the exercises are* [17]. Those who are not sufficiently motivated to learn new concepts or apply them on their own will gain less. Furthermore, engagement is not solely a boon for cybersecurity. It has also been shown to improve student attitudes, reduce stress, and disincentivize academic dishonesty [12]. Finally, the pandemic has demonstrated that educational approaches must be able to transition to a remote-learning environment if necessary. **Therefore, we have three goals: (1) provide students with hands-on education that ensures understanding of the techniques needed to tackle everyday problems, (2) make the exercises interesting, promoting full engagement and obtaining the many benefits it provides, and (3) support distance learning.**

For hands-on education, one can leverage *student-centered* and *challenge-based learning* (CBL) techniques [65]. In student-centered approaches, learners are challenged to draw on prior learning, acquire new knowledge, work as a team, and use their creativity to arrive at solutions as part of an

active learning exercise. In CBL, learners participate in some form of competition and typically learn more quickly as they are often motivated by the common goal of potentially winning a prize.

To make exercises interesting, one can transform them into games. Gamification is a natural fit for computer science education, and its use is particularly appealing in cybersecurity [92] – the adversarial nature of cybersecurity meshes well with the competitive nature of games. That said, the introduction of gamified content into educational settings has been hotly debated [2, 21].

Supporters of gamification encourage teachers to integrate these methods into their classrooms because game elements enhance learning by increasing engagement and motivation [13, 35, 60]. Senko and Dawson [76] found that a "wanting to win" mindset improves the performance of participants especially when they are accompanied by strategies that support feelings of mastery. Detractors, on the other hand, argue that games prompt powerful emotional responses (e.g., curiosity, satisfaction, and frustration) and by including game elements in educational settings we may be creating high levels of stereotype threat or detrimental upward social comparisons – both of which have been shown to negatively influence academic performance [22, 24, 59]. Nevertheless, the findings of Hamari et al. [35] and Bai et al. [2] are encouraging and indicate that gamification can be a winning strategy for both learning outcomes and engagement.

In cybersecurity, gamification often takes the form of "capture-the-flag" competitions (CTFs) or wargames [8, 34, 84]. These activities generally take place within a *cyber range*, an isolated virtual environment that simulates a real system infrastructure (vulnerabilities and all) [92]. In a CTF, players are tasked with finding data, called *flags*, hidden within the range and must leverage cybersecurity techniques to discover them [84]. A wargame, in general, is a model or simulation of a real-world conflict [34]. One popular type of wargame in cyberscurity is a red-blue exercise [8, 38]. Teams of players either attempt to compromise the range (a *red team*) or protect it from intrusion (a *blue team*). Whichever team performs the best wins. While such activities can be engaging, they lack many features of a real computer game and are largely just cybersecurity exercises with added competitive aspects (namely points, badges, and leaderboards). Furthermore, such minimal gamification is all-too-common within the educational context, even outside cybersecurity [2, 21].

Such interventions lack many of the facets that make games so apt for computer education in general and cybersecurity education in particular [34, 92]. For one, video games are very popular among children and young adults and many enter the field of computer science due to their influence [23, 26, 77]. Thus, leveraging a real game can tap into this connection and increase motivation. Furthermore, as games are software, facets of game design can often serve as a great practical example of many important computer science concepts (e.g., graphics, distributed systems, AI, etc.). In the context of cybersecurity, cheats and anti-cheat mechanisms provide a fun and engaging way to analyze and apply security principles.

Thesis Statement

> *A fully immersive gamified learning platform benefits students, improving enjoyment, time management, and performance by engaging them with the subject matter.*

Our Innovations

Seeking to improve upon existing methods to achieve our goals, we created Riposte – an online platform that assists in the learning process and contains an embedded game at its core [51, 52]. It combines a number of industry and educational best practices into a novel package, and offers several important features, including: (*i)* a distributed infrastructure that supports large classes and rapid updates, (*ii)* a built-in integrated development environment (Jupyter Notebook) that allows students to begin developing right away with no machine setup or configuration, (*iii)* an advanced auto-grader that provides automated feedback about each student's performance and gives personalized hints for common mistakes, (*iv)* a standard gamified UI with points, badges, and a leaderboard (*v)* a 2D action game with which students can interact both via play and via programming for a deeper gamified experience, and (*vi)* a modular API that allows instructors to easily create and customize assignments and game challenges to adapt to students' needs and promote their growth. **While our platform was initially designed with cybersecurity in mind, the result is generic and the lessons learned can be applied to many educational contexts.**

We evaluated our system across three iterations of the "Introduction to Computer Security" course at our university. We found that gamification successfully improved student engagement and learning outcomes, with a positive correlation between students's self-reported interest in gamification and their final grade in the course [50]. The integrated development environment also proved popular, with students stating that they "liked that Jupyter notebook was used for the assignment" and statistical analysis showing a significant improvement in student scores. The grader's feedback received a more lukewarm reception, but still showed a positive improvement in performance [52]. Furthermore, during the evaluation, we discovered that the design of our preparatory and gamified assignments organically improved student time management (as measured by earliness of work and time-on-task). Our investigation into the matter led us to conclude that student time management is heavily impacted by academic stress and that gamified (and/or low risk) assessments reduce it. Students heavily engaged by gamification dread assignments less, work earlier, and perform better.

We have reported on aspects of each of these innovations in previous work published and/or accepted to various venues:

- We reported on the success of gamification and leaderboards at SIGCSE '21 [50].
- An outline of Riposte's infrastructure and components appeared at SIGITE '21 [51].
- The design and evaluation of the custom autograder was presented at SIGCSE '23 [52].
- An analysis of the effects of gamification and practice tasks on student's time management will appear at ITiCSE '23.

We draw on this research throughout this work.

CHAPTER 2: BACKGROUND AND RELATED WORK

2.1 Educational Tools

For conventional cybersecurity education, the "Principles of Computer Security Lab Manual" [88] provides exercises with instructions for educators. Unfortunately, while these exercises offer good introductory material, they only teach students how to use existing tools, without providing a good understanding of *what* the techniques employed by these tools are and *how* these techniques can be adapted for new scenarios. More widely adopted exercises are provided by the SEED labs [25] and the EDURange [89] projects. These standalone labs can be helpful, but we found that they fall short in their ability to engage students as they offer limited solution spaces for demonstrating levels of mastery.

On the gamified front, there are the growing number of courses that incorporate some form of "capture-the-flag" (CTF) activities [8, 16, 42, 84] or otherwise introduce gamified content [75] for enhancing cybersecurity skills. These classes can be widely popular, but the learning outcomes typically center on penetration testing on a particular threat or vulnerability. Švábenský et al. [86] take a twist on the prototypically CTF-based approach, providing an offering that focuses on assessing students' ability to use gamification techniques (e.g., storyboarding, level design) to promote engagement on a specific topic in cybersecurity.

Outside cybersecurity, the STEAMiE Education Engine [63] provides instructors with the tools to develop 3D educational games for STEM courses. Its primary focus is on teaching engineering and physics through designing and interacting with simulations. On the other hand, the Simple Academic Game Engine (SAGE) [66] is designed to teach computer science to students by having them design and implement games within SAGE. As such, like Riposte, the developed game itself is not educational per-se, but serves as a vector for teaching programming concepts. However,

unlike SAGE, Riposte is not a game engine that allows students to develop their own games, but rather a game that allows students to exploit and automate it. This approach was a better fit for our cybersecurity use case as it avoids the need to teach unnecessary programming concepts, while still utilizing a game properly focused on engaging gameplay rather than education (as STEAMiE would create).

2.2 Autograding Techniques

Wilcox outlines, discusses, and compares a wide variety of different technical strategies for automatically grading computer science assignments [91]. These can be collected into three broad categories: black-box analysis, source code analysis, and invasive analysis.

Black-box analysis treats student code as a black box, interacting with the program only through standard process I/O. This can be done in a number of ways: from simply piping in template input and comparing the output to some fixed reference (e.g., via `diff`) to fully custom test scripts that construct input and analyze output in complex ways. On the other hand, *source code analysis* does not run the student programs at all, but instead statically analyzes student code for style and functionality. *Invasive analysis* serves as something of a midpoint between the two approaches: test code interacts with student code directly, allowing the grader to use built-in features of the underlying language to assist in grading.

For example, one can use metaprogramming techniques such as reflection to analyze student code and extract important definitions. Alternatively, one can require students to write abstractions and procedures in well-defined manner that enables them to be invoked directly in test code. Lastly, if students are provided libraries to use, instructors can augment these to record behavior relevant for grading (i.e., leverage instrumentation).

Each type of analysis has its strengths and weaknesses, often balancing power with security concerns. For instance, invasive analysis is the most powerful but also poses the greatest security concerns. The techniques it utilizes can also be used by students to find grader code and reverse

engineer assignment solutions. Thus, extreme care must be taken not to leak too much information when using invasive analysis, which limits what can be done.

Conversely, source code analysis is easily secured and great for assessing style, but makes analyzing runtime behavior infeasible. Lastly, black-box analysis lies in the middle, trading some of the power of invasive analysis for security and some of the security of source code analysis for power. It supports runtime analysis and separates student code from grader code, inhibiting the use of invasive analysis techniques by both the student and the instructor. It still presents all the security concerns common to running any black box unknown program, but such concerns can be mitigated if the program is run within a secure environment. Thus, a sophisticated approach will often mix all three strategies – leveraging source code analysis to assess style, invasive analysis to give rich feedback where possible, and black-box analysis to provide security where not.

2.3 Secure Grading Systems

One way to secure a student program is to run it from within a virtual machine (VM). This almost completely isolates the program environment from the host machine, providing strong security guarantees. However, provisioning VMs is time and resource intensive and renders communication between the grader and the student program extremely difficult.

A simpler solution is to use containers. Containers are a lightweight alternative to virtual machines, providing a similar level of isolation of the internal environment from that of the host machine but with a much lower setup cost. Virtualization is also performed at the system level, enabling test code and the isolated black-box program to communicate through standard process I/O. However, spinning up a container still takes time and the isolation provided by a container, while great for security, also limits the student program's interoperability with other elements of the host machine, which can be undesirable.

Thus, another alternative solution is to use a jailed sandbox. With a jailed sandbox, the student program is run in the same environment as grader but with fewer privileges. For example, the grader run as root, but runs the student program through a unprivileged student account. The grader

can also use tools like `chroot`, `rlimit`, and `seccomp` to limit student processes' access to the file system, memory and CPU, and system calls, respectively. However, unlike containers and virtual machines, constructing a sufficiently secure jailed sandbox is difficult, as it is easy to overlook some aspect of the environment a student program can improperly leverage to gain an advantage.

Peveler et al. [69] compare using jailed sandboxes versus Docker containers for autograding. They demonstrate the feasibility of both approaches, but note that jailed sandboxes do provide some performance advantages when one needs to quickly spin up and tear down many such environments.

2.4 Autograding Jupyter Notebooks

Most germane is the approach of Manzoor et al. [55] for autograding Jupyter notebooks within the Web-CAT autograding framework [28] and the Canvas learning management system. Like us, they wish to incorporate the user-friendly environment of Jupyter within a larger learning platform but discover a number of pitfalls with the stock solution for notebook grading, `nbgrader` [36].

Specifically, while `nbgrader` does provide a convenient UI for building assignments and denoting tests, the auto-grading it does must be in the form of unit tests and must be applied manually to the code. Thus, it does not address how notebooks should be submitted for grading, how its results should be reported to students, or how to secure the system from misuse, and it still restricts the types of grading analysis that can be performe on student code. Manzoor et al. address these issues by using `nbgrader` to create assignments, Web-CAT to autograde them, and Canvas to collect the results. Their end product was quite popular, with 75% of students stating they would recommend others to use a similar approach. They also observed a statistically significant increase in mean student grades, indicating automation had a positive effect on student performance.

We were encouraged by these results, but could not directly apply them. Like many other instructors who use home-grown systems [90], we already had a custom gamified learning platform we needed to use (for other assignments in the course, which focus on the embedded game). Thus, we could not switch to the Web-CAT and Canvas system used by Manzoor et al. Instead, we took cues from their design and related research to create our own autograding framework, which we

tailored to our use case. This also allowed us to focus more on the *security* of the system, something Manzor et al. did not address, but was of particular concern to us given our educational topic of cybersecurity.

2.5 Grading Feedback Taxonomy

Previous work by Narciss [61] and Keuning et al. [43] outlines the many kinds of feedback a grading system can provide. There are six common categories: knowledge about performance, knowledge about results, knowledge about mistakes, knowledge about task constraints, knowledge about how to proceed, and knowledge about concepts.

Knowledge about performance (KP) and *knowledge about results* (KR) cover simple measures of performance and task completion (e.g., grades and rubrics). *Knowledge about mistakes* (KM) includes test failures, compile errors, runtime errors, style issues, and performance issues. Such information can be limited (e.g., pass/fail of a meaningfully named test) or detailed (e.g., full stack traces and error messages). *Knowledge about task constraints* (KTC) and *knowledge about how to proceed* (KH) provide hints to the learner. KTC hints are reminders that list the steps needed to complete a task and highlight missing task requirements. KH hints are suggestions that help with error correction and figuring out next steps. Finally, *knowledge about concepts* (KC) covers what a tutor might provide: explanations of the relevant subject matter and examples illustrating certain concepts. Such help can either be provided when the student requests it or automatically offered when the system detects the student is struggling.

An ideal, fully automated feedback system would be able to provide each kind of feedback, but most systems focus on KP/KR/KM. One reason for this is that the other kinds of feedback usually require custom scripting on the part of the instructor to identify relevant mistakes and provide intelligent hints. KM feedback, however, can be templated via example I/O, error reporting, and specific resource limits and is already common in computer science outside education in the form of test suites and linters. Furthermore, in the case of KC, many autograding systems are used within

the context of a conventional class with an instructor that provides such content, creating little demand for its automation.

2.6 Student Time Management

There is a plethora of research on student time management both in computer science and outside it. This includes detailed reports on observed behaviors and their effects [29, 48, 70, 82] along with interventions designed to improve areas of concern [6, 20, 40, 44, 56]. In our system, we pay particular attention to the metrics of earliness and time-on-task, so works with a special focus on those are the most germane to our interests.

Work by Leinonen et al. [45] best addresses the metric of earliness. It summarizes the relevant research and highlights the general consensus that students who start work earlier perform better. The authors then test this theory in a course of their own. While they observe a positive correlation between earliness and grade, they note that even among those who started a day before the deadline, the most common grade was 5 (the highest). Thus, they argue that a simple correlation is likely an inadequate explanation. We build upon this work by demonstrating more anomalous results and providing a possible explanation.

For time-on-task, the current state of the art is also by Leinonen et al. [46]. In it, they compare coarse- and fine-grained metrics of time-on-task and their relationship with student performance. They measure coarse-grained time-ont-task using the time elapsed from first keystroke to first submission and then remove all breaks between keystrokes of 10 minutes or more to obtain their fine-grained time-on-task metric. Fine-grained time-on-task is shown to be universally better than coarse-grained at predicting performance, and they use it to predict final course grades with 93% accuracy (via a random forest classifier). Like them, we also account for breaks in our time-on-task metric, though ours are coarser due to a difference in activity metric (we did not track activity to the level of keystrokes). In contrast, we find time-on-task to be a frail metric that does not always correspond well to performance, which highlights potential caveats with the state-of-the-art understanding.

Work(s)	Games	Security	Assessment	Performance	Time Mgmt	Tools
Bai et al. [2, 3]	✓			✓		
Baniassad et al. [4]			✓	✓		
Benotti et al. [5]			✓	✓		✓
Brown [6]					✓	
Carlisle et al. [8]	✓	✓		✓		
Cook et al. [14]			✓	✓		
da Rocha Seixas et al. [15]	✓			✓		
Dabrowski et al. [16]	✓	✓				
Denny et al. [19, 20]			✓	✓	✓	
Dichev and Dicheva [21]	✓	✓		✓		
Du and Wang [25]		✓		✓		✓
Edwards et al. [28, 29, 40, 56]	✓		✓	✓	✓	✓
Hamari et al. [35]	✓	✓				
Hao et al. [37]			✓	✓		
Jordan et al. [42]	✓	✓		✓		✓
Keuning et al. [43]			✓			
Landers and Landers [44]	✓			✓	✓	
Leinonen et al. [39, 45, 46]				✓	✓	✓
Macan et al. [48]				✓	✓	
Manzoor et al. [55]			✓	✓		✓
Misra and McKean [57]				✓	✓	
Narciss [61]			✓			
Nicol and Macfarlane-Dick [62]			✓	✓		
Nykl et al. [63]	✓					✓
Parberry et al. [66]	✓					✓
Peveler et al. [69]		✓	✓			
Rao [70]			✓	✓	✓	
Schreuders et al. [75]	✓	✓				✓
Trueman and Hartley [82]				✓	✓	
Švábenský et al. [85, 86]	✓	✓	✓	✓		
Vigna et al. [84]	✓	✓				✓
Weiss et al. [88, 89]		✓				✓
Wilcox [90, 91]			✓	✓	✓	
Wolfenden [92]	✓	✓		✓		

Table 2.1: A list of the major topics covered by notable related works. Works with a common first or second author are grouped together to provide insight into that author's breadth of expertise. The topic of "Assessment" covers grading, feedback, and skill assessment. "Tools" includes works whose primary focus is on the presentation or evaluation of novel technology.

CHAPTER 3: PLATFORM DESCRIPTION

The system we developed, Riposte, is a novel learning platform designed initially for cybersecurity education, but which supports hands-on computer science education more generally. The name, Riposte, comes from the fencing term meaning "a quick thrust given after parrying an opponent's lunge" because we find that the dynamics of fencing accurately reflect the interplay between blackhats and security professionals. For example, a hacker exploits a vulnerability (lunges) then experts close the security hole (parry) and track down the blackhat (riposte).

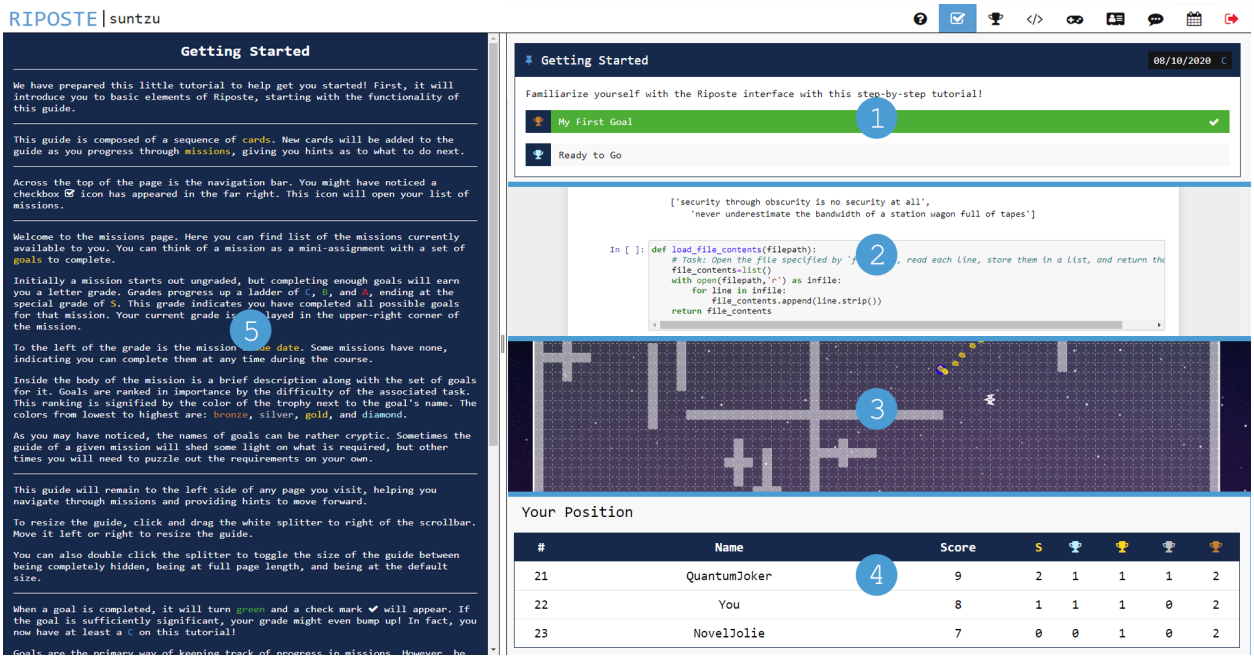


Figure 3.1: A collage of the modules of Riposte: (1) Missions, (2) Code, (3) Game, (4) Leaderboard, (5) Guide.

Riposte leverages a modular design that allows for the easy selection and presentation of disjoint services (henceforth referred to as *modules*). These modules include a ① **Missions** module that allows students to track their progress throughout assignments; a ② **Code** module that provides students with a Jupyter Notebooks to write Python code online to complete assignments; a ③ **Game**

module with a 2D action game that contains a variety of challenges that help teach various cybersecurity concepts; a ④ **Leaderboard** module that ranks students according to their achievements within the platform; and a ⑤ **Guide** module that assists students in navigating the interface and progressing through the assignment. See Figure 3.1 for a collage of these modules.

Students can navigate between these different modules using buttons in the web UI, and new modules can be added relatively painlessly to the user interface. For example, in our course, we add a calendar module for students to schedule office hours and a toy chat module that functions as an SQL injection exercise.

3.1 Design Principles

The technical design of Riposte has three over-arching principles: usability (for both the student and the instructor), security, and accountability. Achieving these goals simultaneously is non-trivial, especially given the nature of a cybersecurity course – **aspects of the platform must be insecure enough to be hackable, but secure enough not to be abused.**

Usability

To make the platform usable for students, we take special care to ensure that the interface is user-friendly and interacts with technologies (e.g., JavaScript, Python) that students are already familiar with. Furthermore, to support distance learning and cross-platform portability, Riposte is a web application with a development environment (Jupyter Notebook) and reverse engineering tools (Chrome Developer Tools) builtin. Students access Riposte through a single web portal that supports both custom login credentials and federated authentication (e.g., with Google Account Services). Once logged in, the web client connects with the service endpoint(s) assigned to the student. Thus, all students need to get started with Riposte is a web browser (Chrome) and a login. This eliminates many of the pain points such as virtual machine configuration and obscure toolchain setups that, in our experience, made onboarding cybersecuirty students difficult.

On the other hand, to be useful to instructors, the platform needs to be flexible, extensible, and easy to manage. It needs to be simple to create exercises for a wide variety of topics and quickly customize them to adapt to different students and their changing needs. As an additional challenge, cybersecurity exercises often test different levels of skill – some assignments highlight simple security vulnerabilities, while others push students to develop complex exploits. Such sophisticated solutions are often only relevant if the simpler solution does not exist. To this end, Riposte has a convenient exercise creation API with a modular design that allows vulnerabilities to be toggled on a per-exercise basis. Also, to enable rapid change and ease management, Riposte has a continuous integration / continuous deployment (CI/CD) pipeline to support agile development and a RESTful API for database management.

Security

Our goal is to support hands-on exercises in both offensive and defensive security. Students are expected to be able to modify the client, bombard the server, and, inevitably, break things – while not impeding the progress of other learners. Thus, security is paramount. To achieve this, we focus on the CIA triad of confidentiality, integrity, and availability and the seminal work of Saltzer on secure design principles [72–74, 79].

In our setup, each student is provisioned a cloud VM that contains their instances of relevant modules, including the Jupyter notebook environment and the game server (see Figure 3.2). These modules and Riposte as a whole are deployed through Docker containers, enabling fast setup and easy scalability. To add a new student machine, we provision a new VM, register the endpoint with the main Riposte server (through its RESTful API), and boot up the Docker containers.

The entire system is locked behind access control. The infrastructure is on the university network behind a firewall. Access to the web portal is password protected and the students' passwords are randomly generated dice words to ensure a sufficient level of complexity. The RESTful interface uses a complex API key that is required on each request, fulfilling the security principle of complete mediation [73, 79].

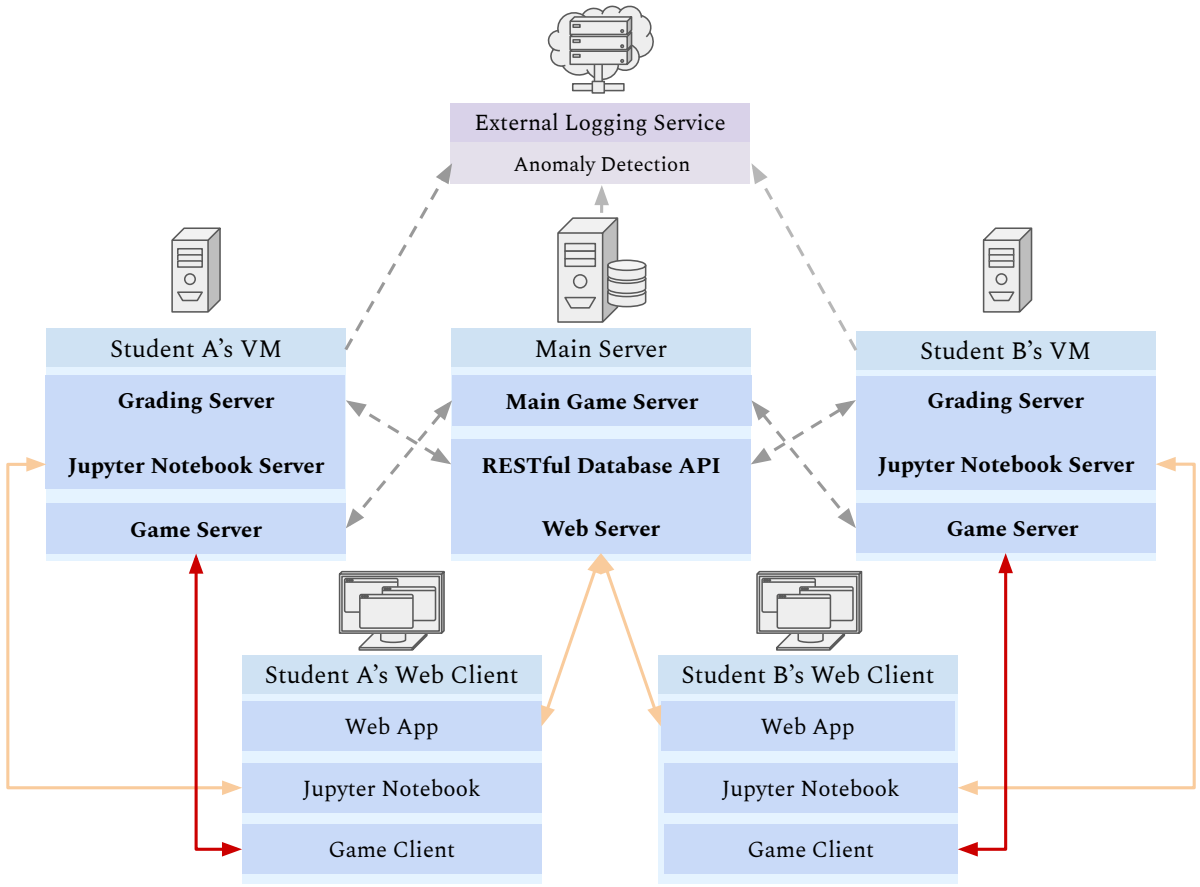


Figure 3.2: *Overview of the server architecture (best viewed in color).* Boxes with bold services are run together within a single isolated Docker container. Arrows represent communications between services. Red solid ones are those that the students can monitor and are encouraged to explore and exploit. Yellow solid ones can technically be seen but are not the focus, and gray dashed ones are invisible to students.

To protect confidentiality and follow the principles of least privilege and least common mechanism, each student user can only connect to their own set of services. Furthermore, users cannot directly access their VM – they can only interact with it through the containerized services. Only the instructors and department IT can log in and administer the machines. Finally, to prevent snooping, connections to the web servers are secured via TLS (webpages through HTTPS and game communication through WSS).

This isolation and strict access control also helps protect the integrity of data stored on students' VMs and the main server. However, the potential for students to break their own machines or for us instructors to accidentally break the whole thing remains. To guard against this, the entire infrastructure is backed up on a regular basis. Futhermore, to ensure availability, we set up services

that run both inside and outside of the Docker containers to immediately restart any server or container should they crash.

Accountability

Good security practice dictates that one keeps logs for accountability in the event of a compromise [73, 79]. Monitoring student behavior can also give an instructor insight into students thought processes and help resolve pain points. As such, Riposte tracks student progress and keeps detailed logs of platform activity. To ensure the availability and integrity of logs even in the event of a machine crash or system compromise, logs are forwarded to an external service.

We use this data to diagnosis complicated system failures, analyze student behavioral patterns, and detect suspicious activities (e.g., potential malicious cyber attacks and/or academic dishonesty). Some of the data recorded is public knowledge – displayed in the leaderboard or through achievements in the Missions module, while other data is kept private. In an effort to respect student's potential privacy concerns, we sought and received approval from our university IRB and consent from the students to use the data collected from by the platform for research purposes [49].

3.2 Hackable Game Engine

Unlike many other gamified learning platforms, Riposte does not merely have gamified elements like points, achievements, and leaderboards, it also has an embedded game. This ties into our thesis that the full benefits of gamification can not be experienced without incorporating a real game.

In Riposte's game, the core gameplay loop is controlling a ship-like avatar on a 2D game board that can move and fire projectiles to complete specific objectives. That is, it is a basic 2D action game. Hacking the game allows student to augment their avatar with more advanced capabilities such as teleporting, moving through walls, and laying mines. The game supports both player versus player (PvP) and player versus environment (PvE) game modes and supports play in both modes. In PvP, the goal is to reduce the opposing players' (or the opposing teams') life total to zero. In PvE, players face off against a variety of AI challenges with differing goals: solving

puzzles, navigating mazes, defeating bosses, surviving waves of enemies, *etc.* In both modes, there can be an arbitrary number of players and enemies – the only limit factors are game board size, resource constraints, and whether teaming is enabled for a challenge. The challenges themselves can be locked behind various authentication mechanisms such as codes found in other exercises or vulnerable cryptographic ciphers players need to exploit.

Therefore, despite its presentation as an ordinary game, Riposte serves as a vector to teach a wide range of computer programming and computer security concepts. For example, learners log into the game using a separate set of credentials from the main Riposte portal, which forms the basis for an online password cracking exercise where students have to crack the instructors' passwords and defeat them at PvP. We also taught various cryptanalysis techniques (e.g., CBC byte flipping) by having students forge challenge unlock codes. In addition, the communication protocol of the game is similar to that of many common web applications (JSON encoded messages over a WebSocket connection to perform remote procedure calls). Therefore, by teaching students how to hack the game, they get hands-on experience with real-world technologies while learning the basics of threat analysis and reverse engineering.

The game is modular, with its challenges and game modes heavily encapsulated, making customization straightforward. Anticipating the flexibility needed for making changes to the platform to adjust our lessons each semester, we took special care to make it easy to design new challenges and enemy AIs. In developing the challenges, we also looked for ways to measure varying levels of skill – i.e., different challenges need to enable and disable different kinds of exploits to accurately assess the specific learning objective. For example, a common exploit used in many challenges allows a player to teleport freely around the board. However, we disable this exploit in assignments where the learning objective demands that the students develop step-by-step maze-solving algorithms, keeping the completion of the assignment tied with the achievement of the learning objective. Riposte supports this flexibility via per-challenge configurations. An example of how the API is used can be seen in Figure 3.3.

```
// Phase 1: Infiltrate and Steal the Gold
const deadends = this.layMaze(this.spawnTile);
const goldTile = this.prng.choice(deadends);
const gold = this.layTile(Gold, goldTile);

// Phase 2: Defeat the Security Ambush (with Hacks)
gold.on.collect.add(() => {
  this.shard.clearTiles();
  const deadends = this.layMaze(goldTile);
  const destructions = [];
  this.nemeses = new Set();
  for (let i = 0; i < 5; i++) {
    const tile = this.prng.choice(deadends);
    const bot = this.spawnBot(Sniper,
      this.shard.map.tileIndexToCenterPixel(tile),
      {fireSpread: false});
    this.nemeses.add(bot);
    destructions.push(
      new Promise((resolve, _reject) =>
        bot.on.destruction.add(() => {
        this.nemeses.delete(bot);
        resolve();
      }))
    );
  }
  this.retarget();
  this.disallowPlayerWeaponry = false;
  this.disallowPlayerTeleport = false;

  // Phase 3: Escape by Reaching the Flag
  Promise.all(destructions).then(() => {
    this.disallowPlayerTeleport = true;
    this.disallowPlayerWeaponry = true;
    this.layFlag(this.prng.choice(deadends));
  });
});
```

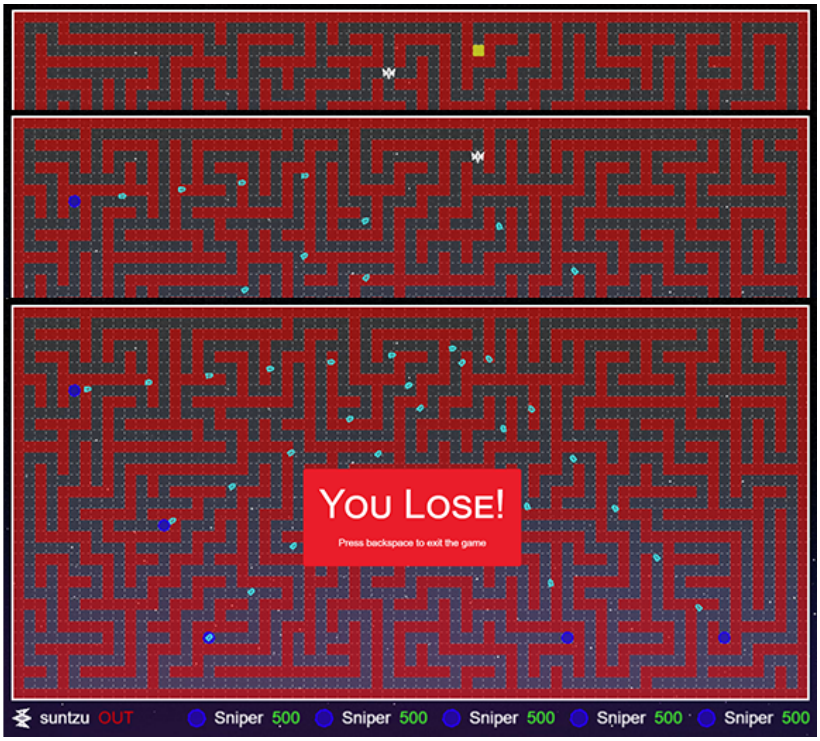


Figure 3.3: Code and gameplay of an example challenge, Heist, that demonstrates the challenge creation API. Heist has three phases: (1) solve a maze to find the gold, (2) defeat the adversaries that appear when collecting it, and (3) solve another maze to escape. Players do not initially know about these different phases and are tasked to complete the challenge with minimal retries.

In it, we design a challenge called Heist with three phases. The challenge's name and its goals are meant to invoke the (rudimentary) image of a fictional bank heist. First, we use the API to lay out a random maze with a gold tile for the player to acquire. Second, once the player steals the gold (by touching it), we spawn sniper enemies (the bank's security force) in random dead-ends of the maze to ambush the player. Finally, should the player defeat these snipers, we re-randomize the maze (to represent a security lockdown) and lay down a goal flag the player must reach to escape.

For this final leg of the challenge, we disable a potential hack that allows players to teleport and also disable their ability to use weaponry (e.g., preventing them from firing projectiles that can destroy maze walls). This forces students to actually solve the maze and not just beeline to the end. The challenge as a whole is designed to train them to expect the unexpected (an important part of cybersecurity) as they are not informed about the different phases but are merely tasked to complete the challenge with minimal retries (ideally zero). It also requires them to leverage previously learned

cybersecurity techniques to hack the client and augment it with additional capabilities to allow them to defeat difficult enemies (i.e., the ambush of otherwise impossible to defeat snipers).

The distributed design of Riposte also factors into the game. In our setup, each student connects to their own instance of the game server hosted in a Docker container on their VM (see Figure 3.2). We termed the per-student game servers, *satellites*. Satellites prevent students from interfering with one another's games and their containerization allow us to design assignments that require exploitation of the game server binary. To also support teamwork and a communal leaderboard, a main server exists that manages the shared state of the satellite servers. Results from games played on each satellite are communicated securely to the main server, and each satellite can query the main server for information about other satellites, allowing for team play. In a team play session, one student hosts the game on their server and every other player joins that game through an in-game lobby interface. The other players' clients learn the host by querying the main server and verifies those players' logins by querying the main server as well.

3.3 Web-Based Development Environment

Hands-on learning in computer science demands coding. While students could code on their own machines, doing so has disadvantages. For one, it assumes that learners already know how to properly setup and manage their development environments, which is not always the case. This is especially true in computer security, where topics like binary reverse engineering and traffic analysis can require students to setup and run VMs and other complex software. Much of the common tooling is written in low-level languages and is frequently complicated enough that even experienced computer scientists outside the domain have a hard time managing it. As such, although setting up and managing a development environment is an arguably essential skill for cybersecurity practitioners to learn, it is often beyond the scope of an introductory course.

To minimize frustration and lower the barriers to entry, we integrated the necessary development environment tools into Riposte in a manner that required little to no setup on the part of the student. We were also encouraged by the results of Benotti et al. [5] and Valez et al. [83], which highlighted

the positive effects a web IDE could have on students. This integration provided another key advantage – it enabled us to track student interactions with the tools and use this information to advance the learning process. It also allowed us to more easily combine the development environment with other elements of Riposte (e.g., the Missions, Game, and Guide modules).

Specifically, we used the Chrome Developer Tools to teach students how to perform traffic analysis, reverse engineer websites, and alter the game client. While the use of browser developer tools might not be appropriate for an advanced security course in which hands-on experience with lower-level tools is part of the learning objective, it worked well for our introductory course, which focused on the concepts being taught, rather than the tools used. Additionally, we provided each student with their own Jupyter notebook *hosted on the Riposte infrastructure*. Each student's Jupyter notebook server is simply a container running on their assigned VM and linked up to the main Riposte site. Thus a learner only needs to log into the site to access their notebook.

Jupyter itself is a web client for writing Python code. Code in Jupyter is organized into *notebooks* which are made up of many *cells* (see Figure 3.4). A cell can either be a snippet of Python code or some prose written in a markup language, allowing users to freely interweave formatted text and code. Python cells can be run individually and print their output below them, allowing for quick feedback on how a piece of code functions. A cell can also support other output formats (e.g., graphs). These features make Jupyter notebooks a very popular way of sharing code online, especially in the data science community. Thus, it is a great IDE to integrate into Riposte, as it is both easy-to-use and practical. Jupyter has also been tested in academia before by Cardoso et al. [7] showing that it can be an excellent tool to support the learning process.

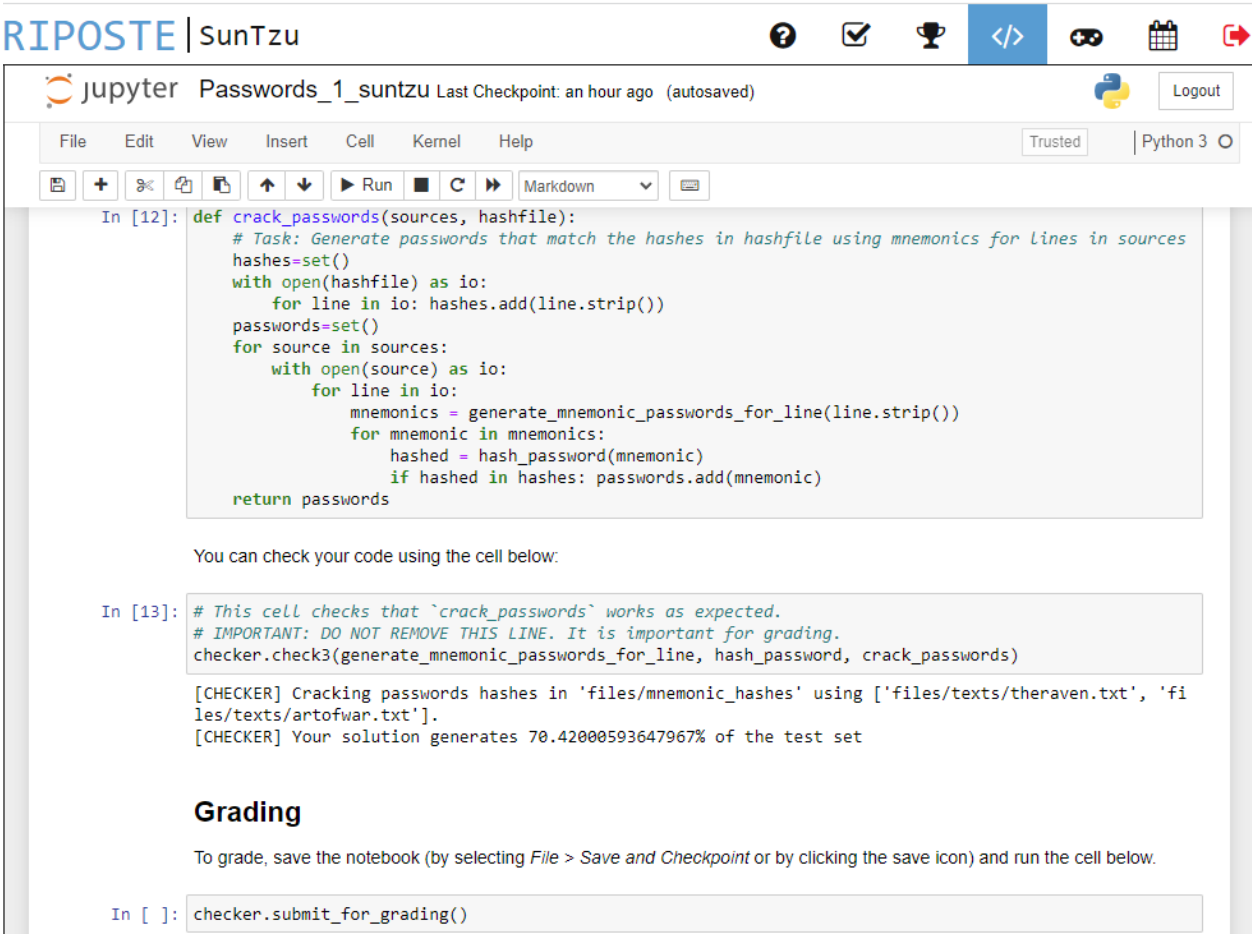


Figure 3.4: A Jupyter notebook in the learning platform.

3.4 Automatic Grading System

Riposte's grading system has three parts: a per-assignment inline aide, a general grading server, and a per-assignment grader. The aide submits a student's notebook to the grading server, which runs the assignment grader on it and reports the results to the learning platform. This pipeline is diagrammed in Figure 3.5.

The aide is an insecure Python module that is accessible to students while they are coding. It provides an assortment of quick checks to help students verify that their code satisfies basic task requirements before they formally submit their notebook. Its invocations also serve as instrumentation points for the grading server that students are informed not to delete. Students can submit multiple times, as previous research shows this to be beneficial [14].

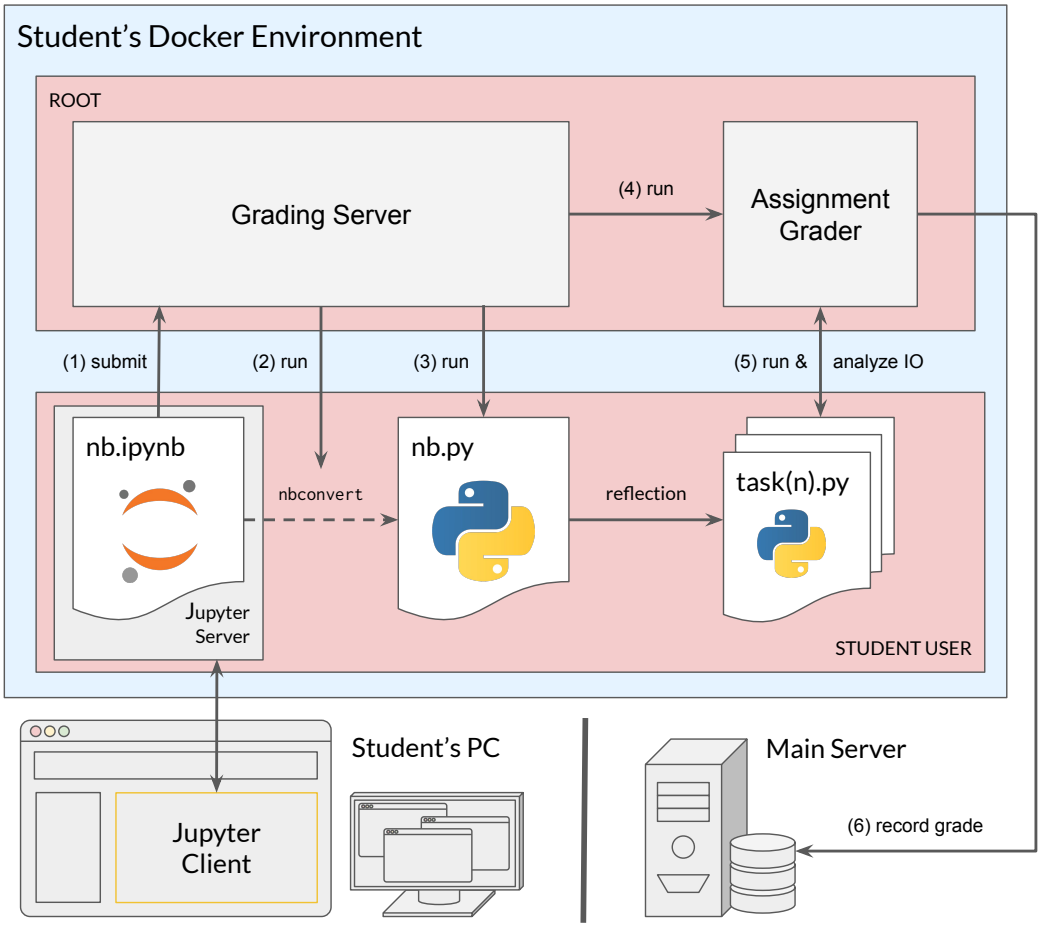


Figure 3.5: *Overview of the grading architecture.* Students edit their Jupyter notebook using the web client embedded in the learning platform. When ready for grading, they (1) submit the notebook to the grading server, which (2) runs `nbconvert` to transform it into a single Python file, which is (3) run by the grading server to produce small tasklet Python files via reflection in the aide. The server then (4) runs the assignment grader, which (5) runs and analyzes the tasklets for correctness and (6) records the grade in the main database.

The Docker container with the Jupyter notebook environment is split in two: the Jupyter notebook server runs under an unprivileged user (a jailed sandbox) while the grading server runs separately under `root`. The environment also has relevant tools, libraries, and Python packages for the tasks we give students pre-installed. The grading server also runs student solutions under the same unprivileged user they used for development. This shared system environment was designed to prevent common grading failures caused by unanticipated incongruities between the student and the grading environment. However, it also means that students have more control over the grading environment and can install packages and tools the instructor may not want them to use. For us,

this is not a problem, as the semi-structured, challenge-based, gamified nature of our assignments allows us to give the students significant leeway to solve problems their way and minimizes the chances of pre-packaged solutions being readily available.

On submission, the grading server converts the notebook into a Python script using the standard `nbconvert` tool provided by Jupyter. This script is then run in the jailed sandbox with a different aide module. Instead of performing checks, this module uses reflection to extract the student's code into separate distinct scripts that are augmented to pass command-line inputs to specific functions and output their results. Afterward, the assignment's individual grader executes these scripts in the jailed sandbox with different inputs and checks the outputs for correctness.

The assignment grader is a mix of black-box and invasive analysis. Invasive analysis (e.g., reflection and instrumentation via the aide library) is used to split the notebook into multiple tiny scripts that can be black-box analyzed (in the jailed sandbox) for correctness. The black-box analysis ensures user code cannot discover and exploit the grader code to achieve improper assessments while the invasive analysis allows students to write relatively free-form code that is smartly transformed into something more amenable to black-box analysis. What makes the invasive analysis secure here is that it is run separately from the grader in the jailed sandbox and it has no knowledge of the grader's strategies.

Results from the grader are sent to the learning platform to record and display in its various interfaces – e.g., awarding trophies on the missions page and ranking up on the leaderboard. Once a student has completed a goal and received the corresponding trophy, the goal remains completed even if student breaks their solution in future submissions. A student may thus be considered to have completed all tasks even if their final submission does not.

3.5 Grades and Feedback

The feedback provided by the aide is substantially more detailed than that provided by the grader. The grader provides only basic KP/KR/KM. It notes which tasks were completed and whether there were runtime errors or incorrect outputs then assigns an overall grade. This limited

level of detail is designed to prevent students from using the grader as an oracle to incrementally fix their code and/or figure out test cases the grader is using and hard code solutions to them without actually solving the problem and completing the learning objective of the assignment.

In contrast, the aide provides detailed KM (e.g., what output was expected versus what was actually produced), general KTC, and some KH (e.g., hints about edge cases they may have missed or strategies for improving their solutions to address task constraints). An experienced student can even use Python's reflection utilities to print out the contents of the aide library for themselves and see what test cases it is using. Solutions, however, are pre-computed to prevent students from easily discovering the algorithms they need to write. As the aide's test cases are just a subset of what the grader checks, hard coding these solutions will not help the students pass the grader and thus mitigates the dangers of such leakage.

Other forms of feedback – such as knowledge about concepts (KC) – are provided statically through the assignment instructions embedded within the notebook or dynamically through lectures and office hours. In the future, we are considering better integrating such feedback into the platform itself so that it could function as an independent learning tool separate from the classroom, but that is not the current focus of our system. Instead, our approach simply augments existing instruction; it does not fully replace it.

CHAPTER 4: PEDAGOGY

In his seminal work on principles of instructional design [32, 33], Gagne et al. outlined nine steps of the learning process that all learning systems should incorporate [18]. These steps are diagramed in Figure 4.1. Our goal is to support each step and facilitate learning.

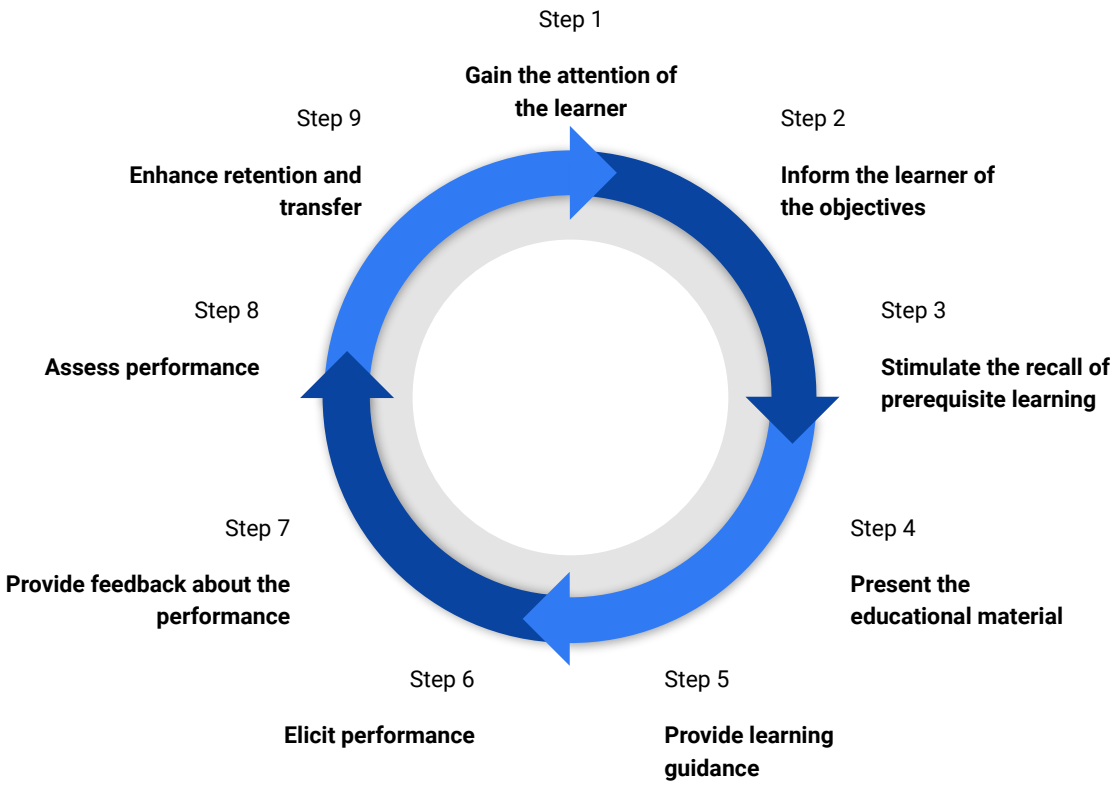


Figure 4.1: Gagne's nine steps of the learning process.

Step 1 is about motivating students to engage with the learning process. This is where gamification comes in. Step 2 is the domain of the Missions module: assignments and learning objectives are disseminated in a gamified form to students, leveraging the gaming notion of trophies (achievements) to denote concrete tasks. Step 3 is accomplished through preparatory assignments we call

Field Exercises, which are special exercises designed to assess students' knowledge of prerequisites. These low stakes pretests determine whether learners have the necessary background knowledge for upcoming assignments, and, in the case of the first exercise, whether the course is right for them.

As this study utilized Riposte within an existing academic course, Steps 4-5 were mainly provided through the course itself (e.g., via lectures, office hours, etc.) rather than Riposte. Conceivably, the course could be made fully asynchronous as well. Regardless, we augment it with the Guide module, which progresses through step-by-step instructions as students complete tasks and can provide hints if the system detects that a learner is getting lost. We created a fully integrated tutorial on the functionalities of the Riposte platform within the guide that can be completed by the students on their own. Thus, Riposte can support being used as a completely standalone learning platform without an academic course backing it. Steps 6-8 are handled through a combination of the Game, Code, and Missions modules. Step 8 is also the domain of our custom automatic grading system.

A common challenge that arises with Step 8 of the learning process is striking a balance between assessing a student's growth versus their proficiency at a set of tasks. Ideally, education grows a student to a requisite knowledge of proficiency in a field, but this is hardly universal. Assignments are generally graded based on whether students have completed certain learning objectives. That is, they measure whether students are *proficient* in a task. However, different students may need very different amounts of growth to achieve such a level of proficiency. The same task can require only a small improvement from one student but a vast jump for another. If the latter does improve (and even potentially improve more, in relative terms, than the other student), but not enough to fully reach proficiency, they will still be ranked low and graded harshly under a pure proficiency model. This can discourage them and turn them away, despite their promising learning potential, because the fruits of their labor were not rewarded. We have found that individualizing assignments (e.g., randomizing parameters, providing a variety of different tasks, and only requiring students to complete a subset of them) can help with this while also mitigating academic dishonesty.

Finally, Step 9 concerns retaining what is learned and transferring it to real world situations. We do this by designing our assignments in a cumulative manner, making sure relevant concepts are

continually repeated and built upon to ensure retention. We also try to make real world connections in each of our assignments, and we developed the Riposte learning platform to closely mirror aspects of real world tools and techniques accordingly.

4.1 Challenge-Based Skill Assessment

In designing a gamified cybersecurity curriculum for our learning platform, we found that the semi-structured, challenge-based learning (CBL) exercises common to gamification present unique challenges for skill assessment. Standard measures like the CIISec skill framework [64], have a sliding scale between knowledge and application, but our learning platform is focused on CBL exercises that contain a practical test of knowledge in which the student must apply the knowledge learned to solve a problem, so all but the lowest assessable skill level require some form of application. Furthermore, our goal is to measure students' mastery of concepts as potential future practitioners, so a lower emphasis is placed on their ability to be up-to-date with recent developments (unlike the CIISec skill framework).

As a result, we devised our own seven-tiered scale to measure skill and assess the difficulty and learning outcomes of assignments. Similar to Parrish et al. [67], we associate a list of "action verbs" to help readers get a better sense of expectations when we say that a learner demonstrates a specific skill level. Our categorization of skill levels is as follows:

❶ *Knowledge*: Acquired the knowledge taught and can manifest that by answering related questions. Most exams test at this level. **Verbs**: define, recognize, memorize, categorize.

❷ *Demonstration*: Can apply learned knowledge in a previously demonstrated way. Show-your-work exams and small assignments generally fall here. **Verbs**: replicate, reproduce, demonstrate, validate.

❸ *Adaption*: Can apply learned knowledge in contexts not seen before. Most traditional assignments tend to fall under this category. **Verbs**: adapt, paraphrase, expand, modify.

❹ *Familiarity*: Demonstrates an understanding of the problem domain and can independently close any knowledge gaps they may have when completing a task within it. Large assignments and small projects often fall here. **Verbs**: investigate, research, explore, analyze.

❺ *Cross-Domain Synthesis*: Demonstrates familiarity in several groups of concepts across different domains, and can draw on and synthesize their skills from these domains to complete a challenging task. Large projects fall in this category. **Verbs**: relate, compare, contrast, combine, survey.

❻ *Innovation*: Demonstrates deep understanding in the given domain to the point of inventing something new. Thesis-level work and particularly novel projects can demonstrate this level of skill in an area. **Verbs**: create, design, develop, hypothesize, theorize, invent.

❼ *Mastery*: Exhibits a deep understanding for a subject to the point of being able to direct, advise, and teach others. It is often hard to observe skill at this level in a conventional assignment, but successful group projects can sometimes reach this level. **Verbs**: teach, supervise, assess, advise, lead.

For each exercise in our curriculum we assess at least two metrics of difficulty using these skill levels: the *skill floor* (i.e., the minimum skill level required for success), and the *skill ceiling* (i.e., the maximum observable skill level). For grading purposes, demonstrating skill floor level proficiency is sufficient to earn a grade of C for an assignment. We also strive, when possible, to have multiple objectives for each exercise that target different skill levels within the range of the exercise's skill floor and skill ceiling to better assess the student's performance. These different objectives map to the trophies presented in the Missions module, and a trophy's color (bronze, silver, gold, or diamond) indicates the increasing level of skill needed to earn it. A student's grade is derived from the number and difficulty of the trophies they earn and the formula used is provided to students, letting them know the challenges they will need to overcome and what areas in their skill set they will need to improve to achieve the grade they want.

4.2 Course Curriculum

To help illustrate the nature of our course, demonstrate the use of our skill assessment framework, and provide context for our evaluation of the Riposte platform, we provide a brief summary of the major exercises we have taught leveraging it. We note that this curriculum is not a recommendation for specific topics to be covered in a cybersecurity class, but rather an explanation of the gamified learning experience from which we observed what worked and what did not.

Exemplary Assignments

We have three distinct styles of assignments: preparatory, conventional, and (fully) gamified. Preparatory assignments are low-risk assessments called *Field Exercises* that test prerequisite skills of later assignments. Conventional assignments are standard programming assignments that use our automatic grading system. Gamified assignments leveraged Riposte's 2D action game to provide a fully immersive gamified learning experience. From each of these categories, one assignment stands out as the best example of its style. We will discuss those now and use them later in the evaluation when comparing and contrasting styles.

Preparatory Assignment: Programming Prerequisites

Before delving deep into a specific topic in cybersecurity, we often give a preparatory assignment that serve as low-risk assessment of the necessary prerequisites. Missing prerequisites can cause a student to perform badly on an assignment even if they completed the learning objective. For example, a student may have trouble communicating over a WebSocket and this may inhibit them from completing an online password guessing assignment even if they learned how to generate good password candidates. Preparatory assignments help mitigate this by highlighting what areas students need to review to prepare for subsequent normal assignments. In structure, our preparatory assignments are much like our other assignments (be they gamified or not) and resemble more the practice task model of Denny et al. [19] rather than the simplified assessments of Edwards et al.'s

syntax exercises [27] or Parsons Problems [68]. The only differences are the topics covered and the way they are graded. Grades on these assignments do not impact the student's overall course grade. Instead, receiving an A (or higher) on the assignment will guarantee the student a minium of a C- on the subsequent normally graded assignment.

The best exemplar of this style is the first preparatory assignment of the semester, which tests students ability to perform important programming tasks (e.g., file I/O, string and set manipulation, asynchronous WebSocket communication) and program standard algorithms (e.g., Levenshtein distance). We rate the skill floor and skill ceiling of the assignment as ❹ (Familiarity) as we do not teach any of these skills to students, rather we request students use their problem-solving skills and previously learned knowledge of programming to complete the assessment.

Conventional Assignment: Offline Password Guessing

Conventional assignments are standard programming assignments that use our automatic grading system. However, they are also semi-structured and exploratory, reflecting our field of cybersecurity. The best exemplar of this style is our offline password guessing assignment. In it, students are given a suite of hashed passwords and tasked with cracking them by generating mnemonic password candidates [93] from a set of files containing quotations of famous literary works (e.g., Sun Tzu's *The Art of War* and Edgar Allen Poe's *The Raven*).

To do this, they write a generator function that produces mnemonic passwords using common word-to-character and character-to-character transformations. While some ideas for basic transformations were provided, more complex transformations in the test set required independent research and out-of-the-box thinking to produce. Students must also consider the efficiency of their password generation algorithm as inefficient solutions can easily run out of memory or run for hours. Furthermore, the grader places limits on system resources and how long their program can run, so even if they are willing to wait, the grader will not. The skill floor for that assignment is level ❹ (Familiarity) for understanding password generation habits and general programming, while the skill ceiling is level ❻ (Innovation) as breaking some of the targets' passwords required ingenuity.

Gamified Assignment: Web Security & Client-Side Modification

Our fully gamified assignments are built around the 2D action game embedded in Riposte. Students complete in-game challenges to earn trophies and improve their standing on the learning platform's leaderboard. Students are given a conventional grade based on the number of trophies they earn. Most challenges cannot be completed by playing normally. Instead, students must use cybersecurity techniques to bypass otherwise impossible obstacles (e.g., undefeatable bots, encrypted unlock codes, etc.). In this way, fully gamified assignments are completely unlike preparatory and conventional assignments – they are centered around completing challenges through cheat-based gameplay rather than finishing normal programming exercises.

The best exemplar of this style is our assignment on client-side modification and web security. In it, students hack the game by modifying its client code using Chrome Developer Tools. For example, one challenge, Blindspot, has bots positioned at 30° off of the cardinal directions from the player's avatar, and the player can only shoot at 45° offsets. To win, students need to inspect the network traffic to learn about the Riposte game API and uncover how their avatar fires. They then modify the client to alter their firing angle so they can beat the bots. Students also use their hacks in a post-assignment PvP tournament that we host between students to see who developed the most game-breaking cheats. Prizes are given to the winners of the tournament.

The skill floor for this exercise is level ❸ (Adaption) as it requires the student to apply the knowledge of client-side modification to different challenges, while the skill ceiling is level ❻ (Innovation) as the students can (and do) devise very creative client side hacks to win the challenges. While the students are shown the basic protocols used by the Riposte game, they could also demonstrate innovation regarding traffic analysis by searching through the protocol space and utilizing hidden commands.

Other Assignments

In our multiple years of using Riposte, we have taught a number of other exercises. However, unlike the ones mentioned above, they do not best represent one of the three distinct styles (e.g.,

because are too short, insufficiently gamified, or given as exams). We mention here some of the longer exercises that will be relevant again when discussing the evaluation of our platform.

Online Password Guessing

As a followup to the offline password guessing assignment, we have students perform online password guessing. In this assignment, students attempt to crack the passwords of instructors' game accounts. To mimic information available on social networks, students are given a short bio for each member of the teaching staff to inform their efforts. Once logged in as the instructor, students are tasked with beating them in PvP by joining a battle with their own account and then defeating the instructor. Successfully cracking all these accounts requirsd applying the mnemonic password generator developed in the offline assignment along with new strategies, such as dictionary attacks and social engineering. Students are evaluated based on how many instructors they were able to defeat in PvP. The skill floor for this assignment is level ❸ (Adaption) for understanding password generation habits and level ❹ (Cross-Domain Synthesis) for general programming, while the skill ceiling is level ❻ (Innovation) as some of the instructors' passwords require ingenuity beyond what was taught in class.

Malleability of Cipher Block Chaining Mode

In this assignment, students learn how challenges are locked using game codes. The 'correct' code used to unlock a challenge is a AES-CBC encrypted ciphertext derived from the challenge description. Given access to a decoding oracle and a known ciphertext, the students are asked to write a program that would perform a CBC byte-flipping attack to create the correct unlock code for any challenge. They are also given a custom challenge within the game to visualize the attack as well as utility functions for interacting with the decoding oracle from their Jupyter notebook. In class, students are shown the basics of the technique and use that knowledge to perform the attack on a 1-block message. Students must then extend the idea to generate the several-block-long code of a locked challenge. They were also asked to research and find real world parallels to the

attack on Riposte and explain how encryption was misused in each case. The skill floor is level ❹ (Familiarity). Students can perform level ❺ (Cross-Domain Synthesis) if they do their research and make connections to known vulnerabilities in existing software.

Leveraging Web of Trust

The focus of this assignment is asymmetric cryptography. Students exchange PGP keys with the Riposte server and their teammate(s). They then must encrypt and sign all network traffic when communicating with the server. The challenge is to produce and send a sequence of "move" messages that solve a randomly generated maze. The layout is sent by the server in an encrypted and signed message (encrypted using the student's own public key).

Students are tasked with creating a custom client capable of extracting the maze layout, solving the maze, and sending the encrypted and signed move messages required to complete the challenge. Furthermore, students must upload a valid certificate to Riposte to certify their win. The initial certificate we provide to them is expired, but contains details on how to roll over to the new certificate. As with the midterm, students are graded based on how many times they die, encouraging them to verify the correctness of their solution before playing for real. We team students to encourage them to use one account to test and debug their solution and then perform perfectly on the other.

This assignment is complicated programmatically and requires level ❺ (Cross-Domain Synthesis) between knowledge of maze solving algorithms, distributed systems, and encryption. The difficulty is further exacerbated by the strictness of secure communications. However, the students are told almost exactly what they need to do from a security perspective (i.e., how to sign, encrypt, and verify messages). The skill floor for that part is thus only level ❷ (Demonstration).

CHAPTER 5: PRELIMINARY EVALUATION

We evaluated Riposte over three fall semesters of the "Introduction to Computer Security" course at our university from 2019 to 2021. In Fall 2019, only the game and leaderboard modules were fully implemented. By Fall 2020, the platform was nearly finished, but we had not yet designed our custom grader. Fall 2021 thus served as a full, complete test of the platform.

The class averages 50-60 students, approximately 85% of whom are male. The 2019 semester was taught in-person, the 2020 semester was online, and the 2021 semester was a hybrid (with in-person lectures and online labs). The curriculum consists of 6-9 exercises, including a midterm and a final exam. Most exercises last for a week or two, and some of the exercises are teamed (students work in groups of two or four).

All exercises are done within the Riposte platform, providing a fully immersive gamified experience. Students are graded on 5-point letter scale of F, C, B, A, S where S means they completed every task for an assignment (including bonus tasks) and rewards them extra points on the leaderboard. After each exercise, students are asked to complete a 5-point Likert scale questionnaire about various aspects of the platform and their engagement with the course. We received approval from our university IRB and consent from the students to use the data collected from the surveys and by the platform for research purposes [49].

5.1 Early Challenges

The results from Riposte's debut in Fall 2019 semester were encouraging. We received lots of positive qualitative feedback on the use of the game and observed a number of promising trends [50]. That said, there was much room for improvement. In what follows we will discuss some early missteps we made and how we rectified them.

Leaderboard Design

Leaderboards are a popular gamification technique for enhancing engagement through social comparisons. During Fall 2019, we had a traditional leaderboard in which the rankings of everyone (including the bottom-most performers) was public. Leaderboards of this type motivate some, but can be a demotivating factor for others [3, 11].

There are no established best practices for the deployment of leaderboards, but several interesting findings have emerged from recent studies [3, 10, 31, 41]. In particular, research from Jia et al. [41] argues against showing the lowest-ranking participants, as this can produce negative social comparisons and thus emotional distress for such students. As such, in subsequent iterations, we implemented a hybrid approach that shows the top N (e.g., 8) players plus the viewing student's immediate neighbors (the players ranking just above and just below them).

Leaderboard Standing

In the Fall 2019 iteration of our course, assignments used each team's leaderboard position as one component of their grade. For the gamified exercise, part of this ranking was also based on their relative order of completion compared to the rest of the class (i.e., the students who completed the assignment first got the highest marks and the ones who completed it last got the lowest). This metric was implemented to discourage procrastination, which we observed to be a major problem in the previous exercise.

While this approach succeeded in its goal, it caused numerous other problems. Students who were able to begin working shortly after the assignment was released got a boost, which frustrated students with busier schedules. It also meant that once a student has successfully completed an assignment, they were in no danger of being knocked down on the leaderboard by the success of other students. This contrasts with other assignments in which students who produced a better solution later could surpass earlier players. One side-effect of this difference is that it removes a major disincentive to sharing solutions (besides the moral one), as students cannot be beaten

by a later solution of the same quality. Due to these negative consequences, we abandoned the completion order metric after this assignment.

Off-topic Difficulties

Students frequently struggled with parts of exercises that were irrelevant to the security concepts we were trying to assess. 11 of 49 students wrote that they struggled with writing asynchronous code in JavaScript inhibiting their ability to complete assignments. 6 of 49 students wrote that they had issues with the web of trust assignment because they were unaware that their local editor was surreptitiously inserting newlines (thereby leading to incorrect hashes). Additionally, students found the lack of clarity in our manual grading process intimidating and frustrating. Namely, there was no rubric, so if students had not fully completed assignments, it was hard for them to know what grade they would get. To address those limitations, after the Fall 2019 semester, we created the notion of Field Exercises to assess prerequisite knowledge, integrated Jupyter Notebook into Riposte to standardize student's development environment, and created an automated grading system to give students real-time feedback on their progress

5.2 Impact of New Changes

We added Field Exercises and the integrated Jupyter Notebook in Fall 2020 and supplemented it with a custom autograding system in the Fall 2021 semester. Regarding these new changes, we had two major research questions:

- **RQ1**: Do students like our changes and did they significantly impact performance?
- **RQ2**: Do the grader and aide provide sufficient feedback on student work?

For RQ1, we theorize that students will like Jupyter Notebook and that the Field Exercises will help them complete subsequent assignments. Moreover, the unified environment and speedy feedback will mitigate the off-topic difficulties discussed above and thereby improve performance. Previous work has also shown Jupyter Notebooks to be popular and improve performance [55].

For RQ2, the aide is designed to provide detailed feedback while the grader is not, thus we expect that the aide will be more warmly received than the grader. Furthermore, a student's impression of grader is likely to be tainted by their final grade. If a struggling student receives a poor grade from the grader and they cannot figure out how to improve their solution, they are likely to blame the grader's feedback. This gives birth to the following hypotheses:

- **Hypothesis 1a**. *Students will like that Jupyter Notebook was used for the assignments.*
- **Hypothesis 1b**. *Students will find the Field Exercises help them complete subsequent assignments.*
- **Hypothesis 1c**. *The new system will positively impact student performance.*
- **Hypothesis 2a**. *Students will find the aide to provide sufficient feedback and the grader less so but still reasonable.*
- **Hypothesis 2b**. *Students with a higher grade will perceive the grader's feedback as more sufficient.*

Data Analysis

At the conclusion of each assignment, students were asked the following Likert-scale questions:

- **Notebook interest**. I liked that Jupyter notebook was used for this assignment.
- **Field Exercise interest**. The field exercise helped me to tackle the tasks in this assignment.
- **Aide/grader feedback**. 'The [aide/grader] gave me sufficient feedback on my work.

For RQ1, we found that students consistently agreed with the *notebook interest* question across every exercise and the final exam. The median was 5 (strongly agree) for each instance, the mean ranged from 4.19 to 4.44, and the standard deviation ranged from 0.77 to 1.01. Similarly, students generally agreed with the *Field Exercise interest* question. The median was ranged between 4 (agree) and 5 (strongly agree), the mean ranged from 3.73 to 4.34, and the standard deviation ranged

from 0.80 to 1.05 for each instance. Figure 5.1 shows the distribution of the median response for each question across all assignments in which it was asked. **The data provides strong evidence to support Hypothesis 1a and Hypothesis 1b – students really liked the use of Jupyter notebooks in the course and generally agreed that the Field Exercises helped them tackle subsequent assignments.**

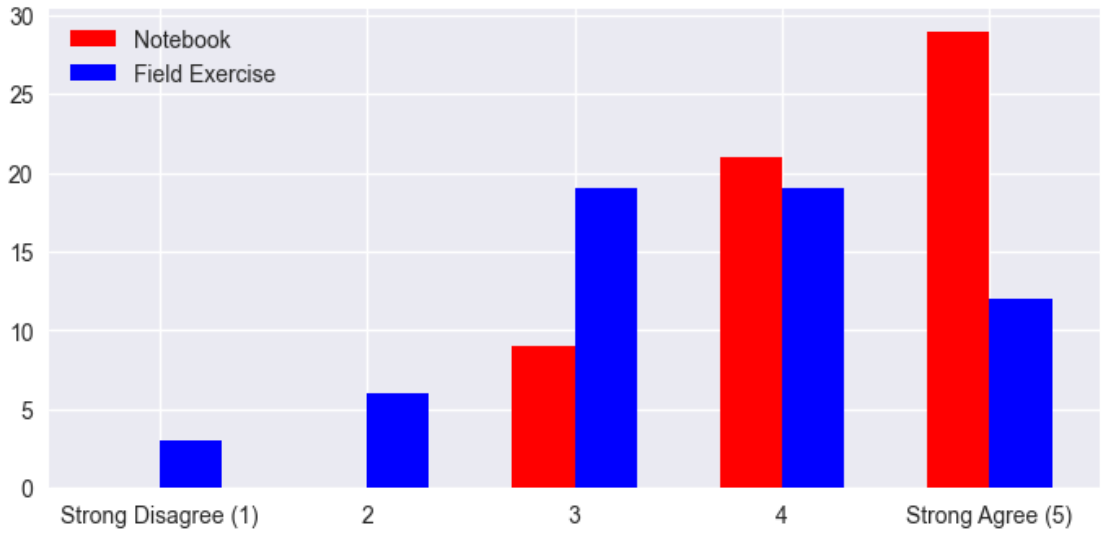


Figure 5.1: Students' median responses on their interest in the Field Exercises and Jupyter Notebook.

To analyze the impact of the new changes on student performance (**Hypothesis 1c**), we compared the results of the offline password cracking assignment across each iteration of the course through the metric of percentage of passwords cracked (a consistent goal across the different versions of the assignment). For the Fall 2021 semester with the new grading system, the mean was 81.36% and the standard deviation was 12.76%. For Fall 2020, where we used an older grading system, they were 71.86%/21.41%, and for Fall 2019, where we did not use Jupyter or automated grading at all, they were 75.61%/9.49%. Using a two-sample, unpaired, unequal variances t-test, we found a statistically significant ($p < 0.05$) positive difference in the mean between Fall 2021 (new system) and Fall 2019 (no system), a statistically weak positive difference ($p < 0.15$) between Fall 2021 (new system) and Fall 2020 (old system), and no statistically significant difference ($p > 0.15$) between Fall 2020 (old system) and Fall 2019 (no system). This provides good evidence that **the Jupyter notebook combined with the new grading system had a positive impact on student**

performance, supporting Hypothesis 1c and producing a similar result to that of Manzoor et al. [55].

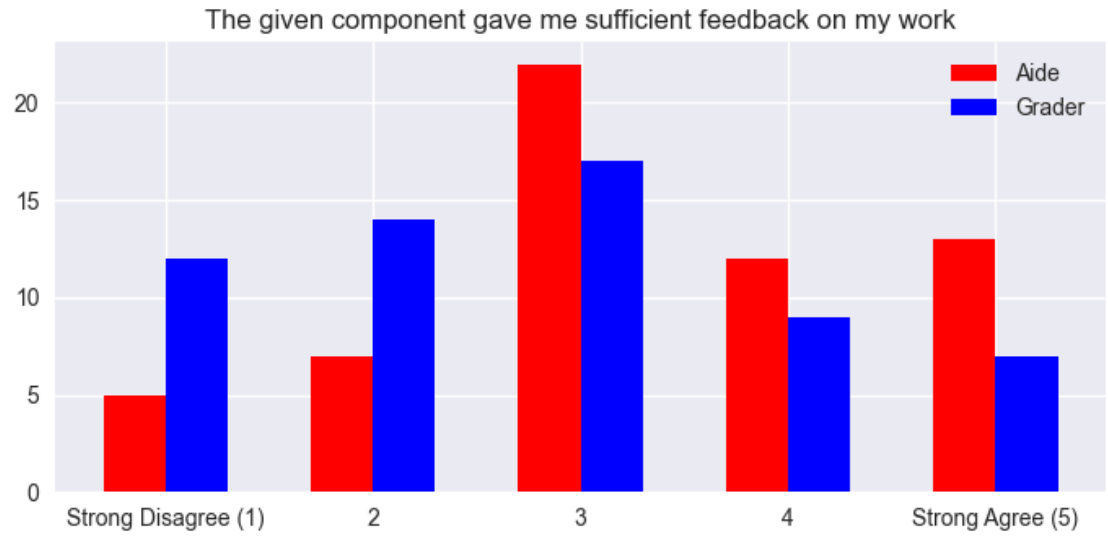


Figure 5.2: Students' views on whether the grader/aide gave them sufficient feedback.

For RQ2, we analyzed the survey responses for *aide/grader feedback* on the offline password cracking assignment. A histogram of the responses can be seen in Figure 5.2. As the figure shows, the response to the aide leaned positive while the response to the grader leaned negative. The responses to the aide had a mean of 3.35 and a standard deviation of 1.2 while the responses to the grader had a mean of 2.75 and and a standard deviation of 1.28. Both had a median of 3.

We performed both a parametric Student's t-test (two tailed, paired) and a nonparametric Wilcoxon signed-rank test (matched) to compare the aide and grader responses for the assignment. We applied both because there is debate about whether a parametric or nonparametric test is more appropriate for Likert-scale questions like our own [80]. Both tests showed a statistically significant difference ($p < 0.01$), indicating that, as expected, the aide's feedback was deemed more sufficient then that of the grader (**Hypothesis 2a**).

To determine if student grades influence perception (**Hypothesis 2b**), we computed the correlation between students grade (1/F-5/S) and their response to the grader (where grade is the independent variable) using both Pearson's r (parametric) and Spearman's ρ (non-parametric). We found a statistically insignificant ($p > 0.15$) and slightly *negative* correlation (r, ρ = -0.069), which

was rather surprising, and provides no evidence for Hypothesis 2b, suggesting that **students' grades were not a factor into their opinion of the grader.**

5.3 Pitfalls and Lessons Learned

While we expected students to find the grader's feedback less sufficient than that of the aide, the response to both were less stellar than we hoped. However, listening to students' qualitative reactions provided us insight into their reasoning, which we discuss here.

Feedback Expectations

Many students expressed confusion about the the grader/aide dichotomy – *"The difference between the [aide] and the grader was frustrating," "the [aide] implements [grading] differently than the autograder."* When the grader did not mirror the aide, they viewed the aide as providing insufficient feedback. Furthermore, even students who understood the dichotomy still felt that grader could provide more feedback. For example, *"I'd simply say a little more specificity on output specifications, errors/specification violations, and reports would be helpful, but I also understand if that compromises the integrity of the assignment."*

Students provided a number of suggested feedback improvements. For example, they wanted the grader to show the exact number of passing test cases, rather than the thresholds (e.g., low, medium, and high) we had decided on. It made them feel better to see the number slowly climb up as they improved their code rather than jump up at unknown intervals. As one student stated: *"it was a lot more encouraging to see the percent [in] the [aide] slowly getting [higher] and mildly discouraging to see 'Low' in the grader."* While such a suggestion is reasonable, others wanted more details. For example, one student asked for the *"[s]pecification of inputs [for] the grader."* Reading between the lines, some students seemed to be trying to use the grader as an oracle to improve their code and were unhappy that it was too difficult to do so.

Another indication that the grading system was serving as a crutch was students' tendency to never test their code themselves, but to rely entirely on the feedback (i.e., KM/KTC/KH) of the

grader and aide – a behavior commonly found in previous research [9, 70]. Students often attempted to divine from the grader whether their code had a specific behavior (e.g., worked on particular edge case). We, as instructors, then had to highlight (e.g., in office hours) that they could write a simple test themselves to see if their code was behaving. Part of problem here may stem from the fact that few courses at our university teach students how to properly test their code and that students may have already been habituated in previous courses to use automated graders as their test suite.

Previous research is split on whether the lack of testing is problematic, if the potential for abusing feedback is significant, and how much feedback would be too much. Some authors note that greater feedback improves performance and achievement – a consistent finding in the research [62] – and that this is indicative of the approach's success [37, 58]. Others argue that this performance is merely a by-product of the student relying on the feedback itself to derive solutions and that they will thus have a hard time accomplishing real-world tasks where such feedback is inaccessible [4, 9]. For example, in Rao's intervention, 60% of students reported they had come to rely on the autograder to verify solutions [70]. This view is further supported by the common finding that students make significant progress in their learning even when external feedback is quite impoverished [62].

As our educational topic was cybersecurity, a field in which feedback on the success or failure of an approach is often minimal, we leaned towards providing too little rather than too much feedback. However, this can, as observed, frustrate students, so in areas in which the real-world applications are more prone to well-structured problems with rich feedback, it may be wise to provide more. Regardless, how much feedback is appropriate should be a question an instructor carefully considers when designing a grading system. The instructor should also inform students exactly how much feedback they should expect from the system to avoid the confusion we observed.

Resource Limitations

Another major area of concern was resource management. The assignment in which the grader was analyzed (offline password cracking) is very sensitive to memory and CPU overuse. Students found optimizing their programs difficult and the errors provided by the grader opaque. Some

specifically cited *"clearer expectations on the parameters of the memory cutoff"* as a pain point. This was not helped by the grader initially producing just a generic "grader crashed unexpectedly" message due to its restricted error reporting.

Furthermore, in our setup, the Jupyter notebook is accessed indirectly through the web and students do not have access to the machine it is running on. Therefore, resource overuse by student's code in the grader or in the notebook can cause the entire system to hang. This results in the student losing access to their Jupyter notebook and requires them to wait for the resource monitor to restart the system before they can resume coding. As solutions can be both time and memory intensive, it can also take a long while (tens of minutes) before the problem emerges and resolves itself.

One potential solution is to give the Docker environment a fixed resource limit that is low enough to allow the VM to continue handling connections. A UI action could also be added to the learning platform to reset the environment as a student's last resort. However, this is still very disruptive to a student's workflow, so resource limits within the environment itself can be instituted to try and avoid this situation. Such limits should ideally be the same between the grader and the notebook itself to make sure code that works in the notebook works in the grader. However, generous limits means the grader can not grade much code in parallel (e.g., run multiple test cases simultaneously), reducing turnaround time (another common complaint of students).

Therefore, when using an autograding system for exercises that may be resource intensive, it is important to carefully consider how the resources for the grader and the student's coding environment are managed. If reasonable solutions are expected to be resource intensive, one must carefully consider the trade-offs between strict limits that can prohibit solutions and generous limits that slow grading.

5.4 Final Remarks

Our initial debut of Riposte encountered an number of challenges that we aimed to fix in subsequent iterations. There were missteps in the leaderboard's design and implementation, and students struggled with a number of difficulties unrelated to the cybersecurity learning objectives.

As a result, we adjusted the leaderboard, introduced Field Exercises, integrated Jupyter Notebook, and created our custom autograding system. While these elements did improve student performance and the Field Exercises and Jupyter Notebook proved popular, the feedback of the grader and aide received a more lukewarm reception.

Qualitative responses revealed that the students wanted to rely more on the grader to help fix their code, the merits of which are debatable. They also had a hard time optimizing their code to avoid resource limits, which was exacerbated by the minimal detail initially provided on them. Thus, while we have likely not yet struck the right balance between too much and too little feedback, there is something to be said for keeping it simple,especially since we have already observed performance improvements. We hope the lessons we learned and the recommendations we made can help others avoid the pitfalls we encountered and inspire future work – both inside and outside the specific topic of cybersecurity.

CHAPTER 6: IMPACT OF GAMIFICATION

Given the debate over gamification in the classroom, our biggest questions concerned whether students enjoyed gamification and whether it had a positive effect on their learning outcomes. This issue has two parts: the general effect gamification had on the classroom and the specific impact it had on students at different levels of the leaderboard. Research has shown that individuals tend to make upward social comparisons, so it is likely that using leaderboards in academic environments will promote such comparisons. In small settings like ours (e.g., less than 100 person classes), students are likely to know each other's identities (beyond a pseudonym on a screen) because they usually interact with each other, and when leaderboard position is connected to one's grade, participants have incentives to stay on top of the leaderboard besides their interest in learning and playing the game. That is, rank in itself may be a motivating (or demotivating) force [47, 81]. We split this evaluation into four distinct research questions.

RQ1: How does gamification impact student enjoyment?

One of the main benefits of gamification attested in the literature is an improvement in student engagement and enjoyment [15]. Furthermore, we theorize that assignments centered around a real game (Riposte's 2D action game) will increase this benefit. On the other hand, the leaderboard may promote detrimental upward social comparisons, which may serve as a demotivating force. Thus, we explore three related hypothesis:

- **Hypothesis 1**. *Increased gamification will generally increase enjoyment.*
- **Hypothesis 1La**. *Students who ranked lower on the leaderboard are more likely to feel less engaged by gamification.*

- **Hypothesis 1Lb**. *Negative impacts of relative social comparisons on enjoyment will more heavily affect those that are doing worse.*

RQ2: How does gamification affect students' learning outcomes?

Guided by the positive results of previous work [2, 35], we theorize that gamification will have a significant impact on student learning outcomes. However, we also theorize that the social comparisons resulting from the leaderboard may influence this effect. Specifically, feelings of inferiority can lead to students thinking they somehow learned less than their peers, both in terms of cybersecurity and general computer science. This gives birth to three hypotheses:

- **Hypothesis 2**. *Increased gamification will generally increase student learning outcomes, affecting both grade and perception, in general and specific to the concepts tested.*
- **Hypothesis 2La**. *Students who ranked lower on the leaderboard are more likely to have a lower perceived learning outcome, both in general and specific to the concepts tested.*
- **Hypothesis 2Lb**. *The negative impacts of relative social comparisons on perceived learning outcomes will more heavily affects those who are doing worse.*

RQ3: How does students' enjoyment of gamification affect their perception?

A hallmark of gamification is its open-ended, multi-solution challenges. Students can feel positively challenged by an exercise if they take advantage of its open-ended nature and explore possible, yet difficult, solutions. Mapping out a solution with minimal guidance to a difficult problem provides students with a sense of accomplishment that is hard to obtain in non-gamified settings. Thus, we hypothesize that students who enjoy gamification would be more likely to explore on their own and feel more learned and intellectually challenged and as a result:

- **Hypothesis 3a**. *Students who showed higher interest in gamification are more likely to have a higher perceived learning outcome, both in general and specific to the concepts tested.*

- **Hypothesis 3b**. *Students interested in gamified learning are more likely to find the assignments intellectually challenging in a positive way.*

RQ4: Did the assignments actively engage students?

Early on, we observed that certain students kept playing the game well after they attained near-perfect scores on the challenges. We also received feedback from students that said their only goal was to stay on top of the leaderboard. As such, we suspected that many students spend a lot of time on the challenges because they were particularly engaged by the gamification aspect. Playing for such extended periods may allow for more complete mastery of the taught material and better learning outcomes [44]. To assess that, we evaluate the following hypotheses:

- **Hypothesis 4a**. *Students work on the assignment past what is needed to get full marks.*
- **Hypothesis 4b**. *Students who spend more time on the assignment end up higher on the leaderboard.*

6.1 Data Analysis

At the conclusion of each assignment, student were asked the following Likert-scale questions (the varying wording indicated the different ways the question was asked across the three semesters):

- **General enjoyment**. I enjoyed the assignment.
- **Gamification impact on enjoyment**. The gamification of the topic made me enjoy the subject matter more || improved my interest in the assignment.
- **General perceived learning outcome**. I learned a lot [during the completion of the assignment].
- **Specific perceived learning outcome**. The challenge helped me better understand ⟨topic⟩.
- **Gamification impact on perceived learning outcome**. The gamification of the topic helped me better understand the subject matter.

- **Intellectual challenge**. The assignment challenged me to think strategically || was intellectually challenging || challenged me to think outside of my comfort zone.

We used this data (along with student grades) to determine whether *correlations* exist between variables. The findings can not be used to indicate *causal* relationships as all data was collected after students finished the assignment and is thus uninformative about such relationships [53].

RQ1: Enjoyment

To test whether increased gamification increased enjoyment, we compared the minimally gamified conventional assignment (i.e., offline password cracking) with the maximally gamified assignment (i.e., client-side modification). We applied both a parametric Student's t-test (two tailed, paired) and a nonparametric Wilcoxon signed rank test (matched), because there is debate about which is more appropriate for Likert-scale questions like our own [80]. For both, we found a statistically significant improvement on the *general enjoyment* ($p < 0.0001$) and *gamification impact on enjoyment* ($p < 0.001$) questions for the maximally gamified assignment. The median response improved from a 4 (Agree) to a 5 (Strong Agree) from the conventional assignment to the gamified assignment. Thus, **we find strong evidence for Hypothesis 1**.

To analyze the impact of leaderboard position on this result, we performed a regression analysis where the independent variable was the student's leaderboard position (lower is better) and the dependent variable was their answer to the *gamification impact on enjoyment* question. We were only able to find a weak negative correlation ($\beta = -0.009, p < 0.15$) in one of the assignments. We also performed two regression analyses using the same independent and dependent variables as those from Hypothesis 1a, but this time, looking at the correlation within students who scored above/below the median ("AM/BM") separately. In the BM segments for half of the assignments, we also found a weak negative correlation ($p < 0.15$). While these correlations do spark some concern, given their rarity and weakness, **we can find no consistent evidence to support or reject Hypothesis 1La or 1Lb**.

RQ2: Learning Outcomes

Regarding learning outcomes, we performed the same tests for Hypothesis 2 as for Hypothesis 1. We found statistically significant improvement between the conventional assignment and the gamified assignment for perceived learning outcome ($p < 0.01$), gamification's impact on it ($p < 0.0001$), and grade ($p < 0.001$). The average grade increased from a B on the conventional assignment to an A on the gamified assignment. **This provides strong evidence for Hypothesis 2**.

To analyze the impact of leaderboard position on this result, we also performed a linear regression with leaderboard position as independent variables and *specific/general learning outcomes* as the dependent variables. **We were unable to find statistically significant correlations ($p < 0.15$) in any of the assignments to support Hypothesis 2La**. Performing a similar analysis on the AM and BM segments separately, we were only able to find a weak ($\beta = -0.02$) albeit statistically significant ($p < 0.05$) negative correlation between leaderboard position and perceived general learning outcomes for one of the assignments in each segment. **This provides some evidence to support Hypothesis 2Lb, but not enough to firmly accept or reject it.**

RQ3: Interest & Perception

We performed a linear regression analysis with *gamification impact on enjoyment* as the independent variable and *specific/general learning outcomes* as the dependent variables. We found consistent statistically significant positive correlations ($p < 0.05$) between gamification interest and both types of perceived learning outcome across all but one assignment. For the remaining one, we found a weak positive correlation at ($p < 0.10$). And for *challenging & strategic thinking*, we found statistically significant ($p < 0.05$) positive correlations in two-thirds of the applicable assignments. **These tests provide strong evidence for Hypothesis 3a and 3b.**

RQ4: Engagement

As a proxy to how much students enjoyed playing the game, we evaluated how much more learners played the game after achieving their first win. We looked at the timestamps of when the

games were played to deduce how many hours students were actively playing the game. Also, for each challenge, we provide a metric of student's improvement in subsequent games after achieving their first win. For shooter-based challenges, we use mean shot accuracy – the percentage of shots they fired which hit the enemies, averaged over all pays of that challenge. For maze-based challenges, we instead use mean death count – the number of deaths it takes them to complete the maze, also averaged over all plays of that challenge.

These metrics, in Fall 2019, were used to determine leaderboard position which was, in turn, directly tied to students' grades. This made playing the game continuously a risk-reward strategy. Playing more could potentially improve or reduce your grade if your mean shot accuracy / death count increased or decreased, respectively. This changed in subsequent semesters, so we only use the data from Fall 2019 in this analysis. A summary of our findings is found in Table 6.1.

	Games Played		**Games After 1 Win**		**Hrs After 1 Win**		**Improvement After 1 Win**		
Challenge	Median	Max	Median	Max	Median	Max	Min	Median	Max
River †	35.5	219	35.5	219	7	16	-16%	0	+88%
Calhoun †	23	87	22.5	87	5	10	-8%	+3%	+84%
Blindspot †	53	224	37	214	5	10	-2%	+34%	+91%
Darlene †	42.5	317	39.5	317	4.5	11	0%	+26%	+79%
Amazo †	52.5	345	43	330	4	13	-38%	+7%	+50%
Sniper †	15.5	250	12.5	229	2	4	0%	0%	+87%
Maze-'o-Mine ‡	107	10991	17	905	4.5	23	+6 D	-0.5 D	-4374 D

Table 6.1: Student engagement metrics for the game challenges in the Fall 2019 semester. For improvement, we display the change in mean shot accuracy (+/- % hit) for the shooter challenges (marked by †) and change in mean death count (+/- D) for maze-based challenges (marked by ‡).

As can be seen in the data, some students made large improvements after their first win. There were two elements that facilitated this. First, students were graded in teams of two and we grade based on the student who performed the best. Thus, by repeatedly playing on only one account, a team can find a better strategy and then drastically improve their performance by playing on the other account. Second, in the maze-based challenge, students often wrote their own automated clients which assisted repeated playing. This allowed them to improve a bad initial showing by playing the maze perfectly enough times that their mean death count is reduced to virtually zero.

The 10991 games played, 4374 death reduction outlier is one such case. That team initially used a brute force strategy of random movement and repeated deaths to discover the maze layout (on both team members accounts). After learning in office hours that they could instead just use a proper maze-solving algorithm and that their many deaths would hurt their grade, they automated thousands of wins to improve their mean death count and their grade.

Overall, there are two important takeaways from this data. First, some students seemed very engaged, playing more than 100 rounds over 4-5 hours for almost all the challenges despite getting a win fairly early on. Second, students often kept playing the game even though doing so did not significantly improve the quality of their solution (three challenges had near zero medians). In fact, some ended up in a worse position than where they started (lower accuracy or more deaths). **These factors indicate that students were heavily driven by gamification instead of a rational approach to maximize performance**, supporting Hypothesis 4a and corroborating previous research on gamification's impressive ability to engage students [15].

We also performed a regression analysis with *leaderboard position* (lower is better) as the independent variable and *number of games played* as the dependent variable. **We were able to find a weak negative correlation ($p < 0.15$) in half of the assignments, providing some evidence in support of Hypothesis 4b.**

Summary of Findings

Overall, we believe the significant increase in enjoyment and perceived learning outcomes combined with the minimal impact of leaderboard position and the significant impact of gamification interest on learning outcomes strongly indicate the success of our intervention. It shows that some students are heavily motivated by gamification and that motivation manifested in improved learning outcomes while those for whom gamification could be a demotivating force (those lower on the leaderboard) are not heavily discouraged. Moreover, it supports our main thesis – **assignments with a focus on a real game are able to draw more value from gamification than conventional**

assignments simply situated within a gamified learning platform (with points, badges, and leaderboards).

Furthermore, on a qualitative front, the course continues to receive some of the highest overall student evaluations in the department with some students specifically highlighting the gamified format – *"I think the game format really helped me understand the concepts better, and it was a lot more engaging than typical written homeworks; This class is the most fun CS class I had at UNC, and I have learned so much by working through all the challenges; The module-based, gamified approach to learning was really great. I really enjoyed this class, and it was very different when compared to other CS courses."*

6.2 Challenges With Semi-Structured Gamified Exercises

Unlike other areas of computer science (e.g., software development), in which practitioners can leverage simplifying assumptions to quickly complete a task derived from an external need, cybersecurity practitioners need to be aware of, and repeatedly question, the validity of these simplifying assumptions to either prove the system's security or find exploitable weaknesses. As such, in this specific field, being able to find the right problems to solve is perhaps just as important as being able to solve them.

Our semi-structured gamified approach nicely mirrors this property – learners are given a distant goal and have to find out how to reach it. They explore the platform, discover its underlying assumptions, and develop ways to exploit them to achieve their goal. In most of our challenges, there are a host of assumptions the students can exploit, hidden at various depths within the system, and students are rewarded for capitalizing on these weaknesses.

To demonstrate this, consider the two password guessing exercises. In the offline assignment, students were tasked with coming up with various character-to-character and word-to-character transformations to generate a wide range of mnemonic passwords. While they were given some transformations in class and in the assignment text, relying on these alone was not sufficient to achieve a passing grade. The students therefore needed to explore and brainstorm to complete that

part of the assignment. In the online assignment, the students had to guess the passwords of the targets from bios that mirrored information one might find on real social media. We provided a few examples of common password creation mistakes, but none of these could be applied as-is. Instead, good approaches required the student to get into the headspace of a victim and figure out how they would go about creating passwords [95]. Students needed to identify and properly understand the hints *and* adapt and develop strategies to generate the passwords within the family of passwords hinted at.

For some of the learners, this approached worked very well: they found the exploration aspect exhilarating and derived a high sense of accomplishment from finding the correct path. However, we identified three pedagogical drawbacks: imperfect correlation between learning objective and game goals, lack of a guardrail against depth-first thinking, and insufficient measurement of lower skill levels. We elaborate more on each of these points below. While we will use the two password guessing assignments as a running example to illustrate these drawbacks, they applied to our semi-structured approach in general.

Correlation Between Learning Objective and Game Goals

By nature, there is no unique way of achieving the end goal of a semi-structured assignment. In an ideal semi-structured assignment, however, the various ways of achieving the end goal all require mastery over the intended learning objective. This, however, is tricky to get right, as the students do not necessarily know *a priori* what the learning objectives are. In the password cracking assignment, the learning objective is to understand the techniques people typically use to generate memorable yet relatively secure passwords. In the offline password guessing assignment, this is tested via the student's ability to generate the common transformations needed to replicate a fixed set of mnemonic passwords we provide. In the online password guessing assignment, they are tasked with both identifying and exploiting information subtly hidden within the targets' social media presence, and then expanding upon their work in the offline assignment by developing guessing strategies beyond a single class of passwords (i.e., mnemonics).

However, not all students saw the correlation between the underlying learning objectives and the stated goals of the assignment. One student in particular perceived the first part as a mere password cracking exercise and employed a brute force approach to replicate our mnemonic set – without ever grasping why one would be doing this task. That is, the learner knew that to replicate the answer set, one would sometimes need to transform an 'a' into a '4' or an 'e' to a '3', but never understood *why* someone would apply such a transformation when generating their password. As a result, the student got stuck on the latter part of the assignment because the learning objective of understanding people's password creation habits was never met.

Lack of Guardrails Against Depth-First Thinking

As stated before, in our intervention, students are encouraged and awarded for questioning assumptions and exploring seemingly far-fetched ideas. For the most part, we believe that this is laudable as it models reality. However, being efficient during exploratory stages is a skill that is rarely explicitly covered in education, and as a result, our students were ill-prepared for our semi-structured assignment. We did attempt to teach students what one should do to avoid going down unnecessary rabbit holes (e.g., planning out an attack tree, trying low hanging fruits first, not being afraid to go back to brainstorming), but this was often insufficient. Many students were still thinking in a very *depth-first* fashion, insisting that they are on the right track despite the often extreme complexity of their supposed solutions. This led to a lot of frustration and often impeded their ability to master the intended learning objectives.

Office hours were our goto solution for handling this problem, but students would sometimes wait too late in the assignment to be able to visit us, reorient their thinking, and try a new approach. In future work, to provide students with real-time feedback, one could train a machine learning model to analyze and predict when and where students are struggling and either notify the instructors or provide feedback itself to help reorient students. However, this would require identifying what data such a model needs to make such predictions. Namely, determining what should be the inputs of the model and what qualifies as a proper training set of struggling and successful students. One

thing to note is that a student who struggles with depth-first thinking on one assignment does not necessarily struggle this way on all of them, so the simple metric of the highest and lowest performers is insufficient. In fact, some of the highest performing students in the course originally struggled with this on early assignments.

Insufficient Measurement of Lower Skill Levels

Another potential drawback of a complex semi-structured learning exercise is its high skill floor – a result of its requirement for innovation and lack of explicit instructions. For example, our password security assignment assesses students' ability to develop password guessing strategies instead of their ability to apply standard ones. This is also why we rated it with a skill floor of *familiarity* (level ❹). However, a side-effect of this is that we are unable to test students who have only mastered the material at lower skill levels. For the purpose of grading, there is no difference between someone who was able to efficiently (and exactly) apply the taught methods of guessing passwords versus someone who has no knowledge in the field. As such, this can be very discouraging for students who are at these lower levels, and can even give the perverse impression to students that their efforts are not valued and that it is, therefore, "not worth it" to try. This also has serious impacts on our ability to measure student growth, as it squashes a wide range of skills into a single point, making the desired type of "start-end" grade equivalency impossible to establish.

Based on these observations, we argue that a good semi-structured assignment must be placed within a structured framework that serves as a "safety-net" for students. This safety-net can take many forms, but at the least, it should (*i*) only assist the student when they have gone far off course – i.e., to prevent circumventing the exploratory nature of the assignment, and (*ii*) be accessible and clear enough for students to feel like their efforts are valued.

For our interventions, in-person office hours served as this safety-net for students: those who were stuck would come to office hours, where we would provide more direct hints, explain the motivation behind parts of the assignments, pull students off the wrong paths, and assess if someone is actively trying but just performing at a lower skill level. This worked – feedback for the course

was overwhelmingly positive, and a lot of people specifically praised how useful office hours were. Alas, its success rests on time-consuming, close one-on-one interactions that are not scalable and ill-suited for teaching in the age of distance learning. While we continued to make do with online office hours, automated solutions to these problems are warranted and would be appreciated in future work.

6.3 Final Remarks

We evaluated the impact gamification had on student enjoyment, engagement, and learning outcomes, paying special attention to potential confounding influence of leaderboard position. We demonstrated a significant increase in learning outcomes and enjoyment and few negative effects, even for those lowest on the leaderboard. Furthermore, we found that many students were so heavily engaged that they continued to play the game after achieving full marks.

However, for gamification to be a winning strategy for cybersecurity education [92], instructors must carefully consider game design elements, paying specific attention to isolating the effects of gamification from other confounding factors (such as related elements of semi-structured learning). One outstanding concern is how best to individualize gamification. It is important to assess who benefits most from gamification, but fully understanding all the dimensions is difficult if for no other reason than there are many different learning styles [30]. Recently, Alshammari et al. [1] showed that matching learning styles to learning material can yield better learning outcomes and student satisfaction. While promising, a more systematic assessment is needed.

CHAPTER 7: ANALYSIS OF STUDENT TIME MANAGEMENT

Our grader logs provide us with the opportunity to analyze student behavior and test previous work on the topic of time management. In particular, we were curious as to whether our three different styles of assignments change anything. We also observed that students who start too late run out of time, spending less time on the task and achieving a lower grade than they otherwise would. We theorized this could lead to a correlation between time-on-task and performance simply as a by-product of a correlation between earliness and performance. That is, students who start late will have to spend less time and therefore perform worse (i.e., decreased earliness, time-on-task, and grade) whereas those who started early can spend more time and therefore perform better (i.e., increased earliness, time-on-task, and grade).

We thus examine three related research questions, comparing and contrasting the results across the exemplars of our assignment styles (i.e., preparatory, conventional, gamified). We use the data from Fall 2021 in our analysis, because that was where the full platform was used and where each of these assignments were done individually (without teams).

RQ1: Do students who work earlier and/or spend more time-on-task perform better?

We wish to assess whether our observations align with previous research on time management. Common wisdom [29, 45, 70] suggests the students tend to delay assignments to the last few days before a deadline and this negatively impacts performance. Furthermore, the longer students spend on an assignment, the better they do (i.e., increased time-on-task improves performance). These ideas give birth to four hypotheses:

- **Hypothesis 1a**. *A significant number of students will delay work on their assignments.*
- **Hypothesis 1b**. *Earliness of work will have a significant impact on student performance.*

- **Hypothesis 1c**. *Students who work in the last few days will perform the worse.*
- **Hypothesis 1d**. *Time-on-task will have a significant impact on student performance.*

RQ2: Does assignment design affect student behavior?

We theorize that academic stress is a leading cause of students delaying work. It has been shown that gamification can increase engagement [15] and thereby reduce stress [12]. Similarly, we expect the preparatory assignment's low-risk nature to also reduce stress. If true, the preparatory and gamified assignments should be less stressful than the conventional assignment and, therefore, induce better time management. Thus, we analyze the related hypothesis:

- **Hypothesis 2**. *Student earliness on the gamified assignment will align more closely with the preparatory assignment than with the conventional assignment*.

RQ3: Are earliness and time-on-task interrelated?

Students who work late often run out of time, reducing their time-on-task. This suggests there is a correlation between earliness and time-on-task. However, if students start sufficiently early, this correlation might vanish. Thus, we test the following hypothesis:

- **Hypothesis 3**. *Earliness will have a significant impact on time-on-task if and only if many students delay their work too late*.

7.1 Assessing Earliness

There are multiple ways to measure the earliness of a student's work. One way is to simply measure the start of student's activity on the assignment (e.g., first log entry or first submission) [29, 40, 45]. This can be problematic. Students can take a quick look at the assignment, maybe even solve some problems they find simple, submit, and not come back until the assignment is almost due. Similarly, another way is to analyze the end of student's activity on the assignment (e.g., final log entry or final submission) [29, 40, 56]. If students can submit multiple times, this can also be

problematic, as students who started early and completed most of the assignment may return late to add a few finishing touches. This is an acute concern in our scenario, since each assignment includes extra tasks a student can complete to earn bragging rights (e.g., a higher place on a leaderboard) and thus students are incentivized to work on an assignment past the time needed to perform well.

A distinctly different way to measure earliness is to count how many days a student worked on the assignment. This connects with the notion of the *spacing effect*, which suggests students distributing work over multiple days improves performance [45, 87, 94]. Students who start very late will necessarily spend less days on an assignment. Nevertheless, research has shown students who start very early and work many days can still perform worse, and students who work only a single day can still do very well [45].

Therefore, to create a good balance between earliness and workload, we measure earliness by recording the offset from the deadline of an assignment's relevant event (code submission or game connection) and compute the mean over all such events. A similar approach is also used by Rao [70]. We term this metric the *mean submission time* or *mean connection time* depending on the event used. We use two different events because students do not submit anything on the gamified assignment. Instead, task completion is automatically monitored in-game and points and grades automatically assigned when the relevant requirements are met. Specifically, in the gamified assignment, students complete tasks by leveraging modifications they make to the game client. To apply these modifications, students need to refresh their browser and establish a new game connection. Therefore, like each new submission, each new connection serves as a test of a student's code, making it a great parallel to submission on the gamified assignment.

Results

If students submit primarily submit close to the deadline, the distribution of submission times will be exponential (i.e., many times close to the deadline of 0 and few out on the tail). Thus, to see if they did so (**Hypothesis 1a**), we analyze whether the mean submission times fit such a distribution. Figure 7.1 shows the distribution of mean submission times for the preparatory and

conventional assignments (the two exemplar programming assignments). It should be noted that the preparatory assignment was due on Thursday and the conventional assignment was due on a Tuesday, so neither distribution appears to be influenced by a "weekend effect" (i.e., students did not primarily work during the weekend before an assignment was due).

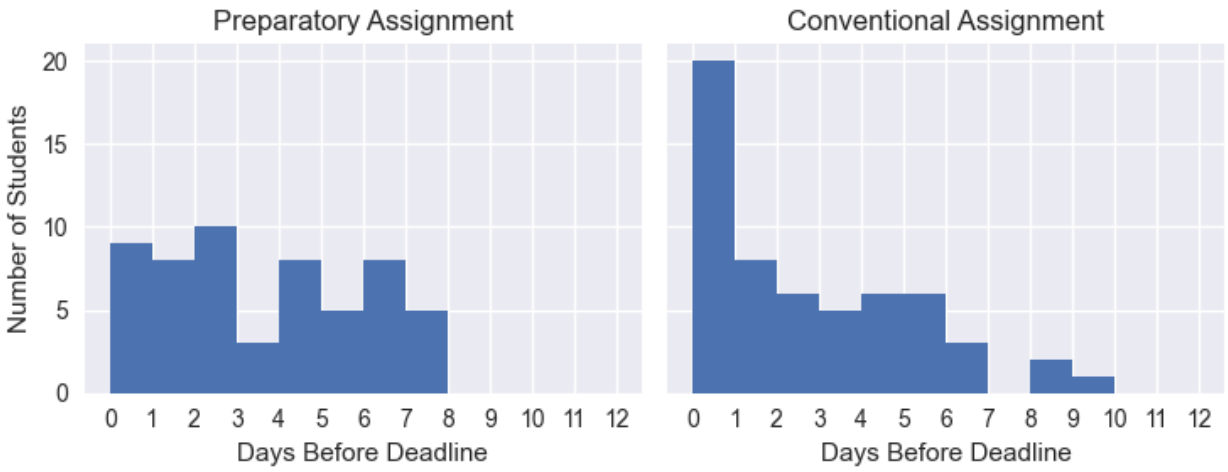


Figure 7.1: Histograms of mean submissions times for the preparatory and conventional assignment. Assignments are distributed about a week (7-12 days) before the deadline. The weekend (Saturday and Sunday) is ranges from days 5-7 on the preparatory and 2-4 on the conventional.

We applied an Anderson-Darling statistical test to determine whether we can reject the null hypothesis that the mean submission times come from an exponential distribution. We use an A-D test for this because, of the three standard goodness of fit tests (Kolmogorov-Smirnov, Anderson-Darling, and Cramer-von-Mises), the A-D test best handles distributions which may have fat tails (e.g., exponential distributions). We found that, for the preparatory assignment, we could reject the A-D hypothesis at a significance level of 1% ($A^2 = 2.56 >$ critical value = 1.94). For the conventional assignment, we could not reject the hypothesis even at a significance level of 15% ($A^2 = 0.71 <$ critical value = 0.91). This indicates that **students submitted the conventional assignment close to the deadline, while they submitted earlier on the preparatory assignment,** providing evidence for Hypothesis 1a on the conventional assignment but not the preparatory assignment.

To see if students' submission behavior affected their assignment grade (**Hypothesis 1b**), we computed the correlation coefficient (two-sided Pearson's r) between their mean submission time

and their final grade on the assignment. We found a statistically significant positive correlation for the conventional assignment ($r = 0.44$, $p = 0.0007$), but a bit weaker and less significant correlation for the preparatory assignment ($r = 0.40$, $p = 0.002$). That is, **students who worked earlier performed better on each assignment, but that effect was more pronounced on conventional assignment,** providing evidence for Hypothesis 1b on both assignments.

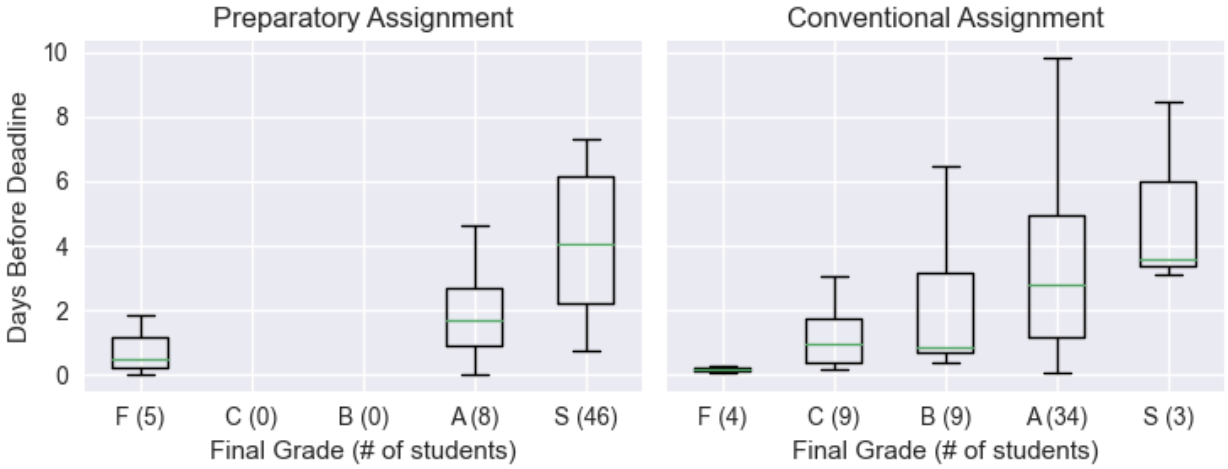


Figure 7.2: A box plot of students' mean submission times categorized by their final grade on the assignment.

Additionally, by Welch's t-test, students performed significantly better on the preparatory assignment than on the conventional assignment ($t = 6.58, p < 0.0001$). The mean grade for the preparatory assignment was an A and the mean grade for the conventional assignment was a B. As students also started earlier on the preparatory assignment compared to the conventional assignment, this provides further support for Hypothesis 1b. Figure 7.2 helps illustrate all this.

To determine whether submitting in the last few days was particularly significant (**Hypothesis 1c**), we split the students in two based on their mean submission time and a threshold (12, 24, 36, 48, 60, and 72 hours before the deadline) then computed the correlation coefficient (two-sided Pearson's r) between mean submission time and grade of the two subsets for each threshold. We also performed a two-sided Welch's t-test of two independent samples (which, unlike Student's t-test has no assumption of equal variance) between the two subsets to test whether the average grade differed significantly.

For the preparatory assignment, there was no significant difference between the subsets at any threshold. This aligns with the observation that students were not frequently submitting during the last few days of the preparatory assignment. In contrast, for the conventional assignment, we found that a split on 48 or 60 hours led to similar sized subsets that differed significantly ($p < 0.05$). 28 students had a mean within the 48 hours and 29 did not (and vice versa for 60) and there was a statistically significant ($t_{48} = 4.20$, $t_{60} = 4.05$, $p < 0.001$) difference in performance between the subsets but an insignificant, weak correlation ($r < 0.15$, $p > 0.50$) between mean submission time and performance within them.

In other words, students who mostly submitted within two days of the conventional assignment's deadline performed worse (had a median grade of B) whereas those who submitted earlier did better (had a median grade of A). Futhermore, within each group, how early or late one submitted was not statistically significant. This provides evidence for **Hypothesis 1c** and supports our theory that **many students started the conventional assignment too late and ran out of time, hurting their grade. Conversely, students worked earlier on the preparatory assignment, giving them more than enough time to do well on the assessment.**

Gamification

To analyze whether gamified assignment was more like the preparatory or the conventional (**Hypothesis 2**), we compared them. Figure 7.3 plots the mean connection time of the gamified assignment both as histogram and as a boxplot categorized by grade, paralleling Figure 7.1 and Figure 7.2 for the programming assignments.

The data, as expected, is observed to more closely align with that of the preparatory assignment than the conventional assignment. Specifically, the mean submission time across students was 3.6 days on the preparatory assignment and 2.9 days on the conventional assignment, whereas the mean connection time across students was 3.8 days on the gamified assignment (i.e., similar to the preparatory). Also, like the preparatory assignment, the distribution across days is not exponential even at a 15% significance level by an A-D test (A^2 = 5.00 $>$ critical value = 1.94).

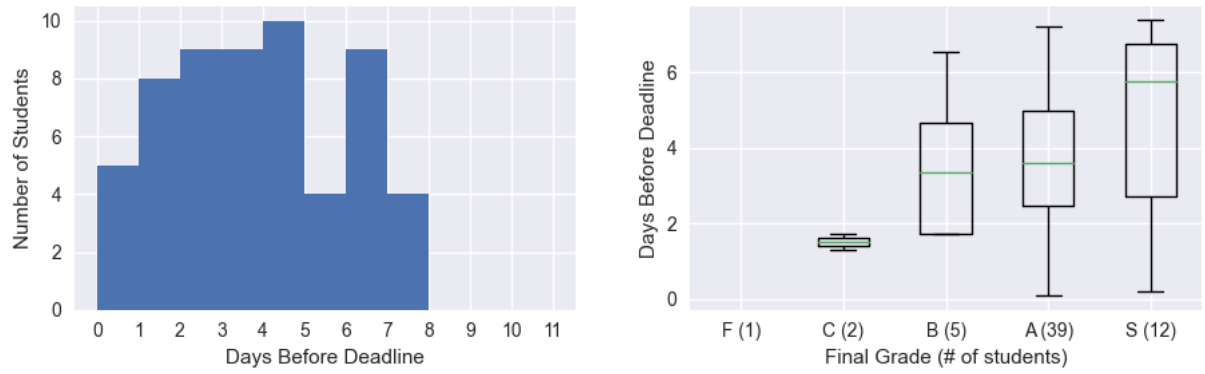


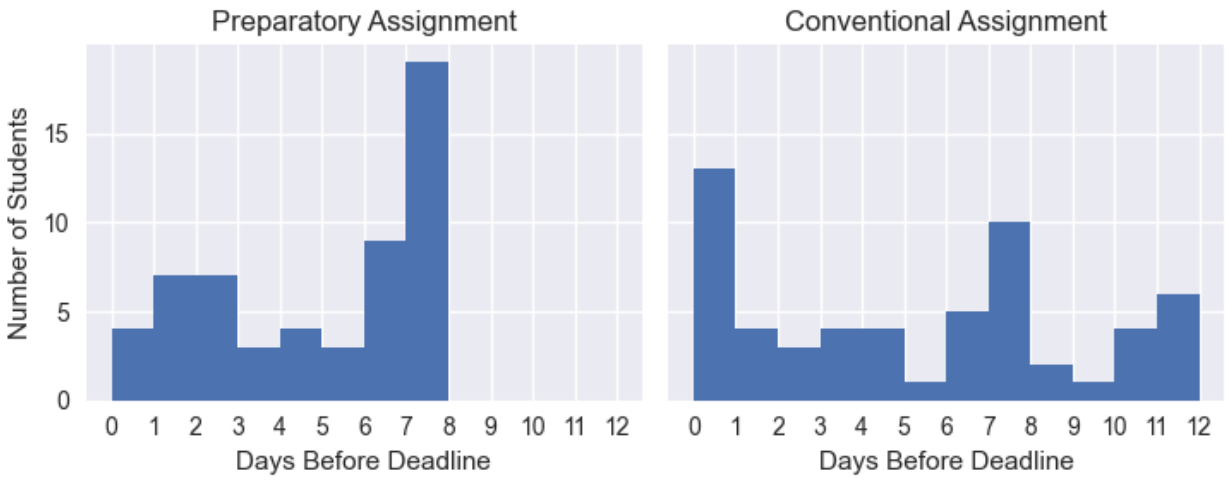
Figure 7.3: The mean time the students connected to the game on the gamified assignment as a histogram (left) and as a boxplot categorized by grade (right).

We also computed the correlation coefficient (two-sided Pearson's r) between the mean connection time and the grade on the gamified assignment. The correlation ($r = 0.27$, $p = 0.039$) was much weaker than the correlation between mean submission time and grade on both the preparatory assignment ($r = 0.40$, $p = 0.002$) and the conventional assignment ($r = 0.44$, $p = 0.0007$).

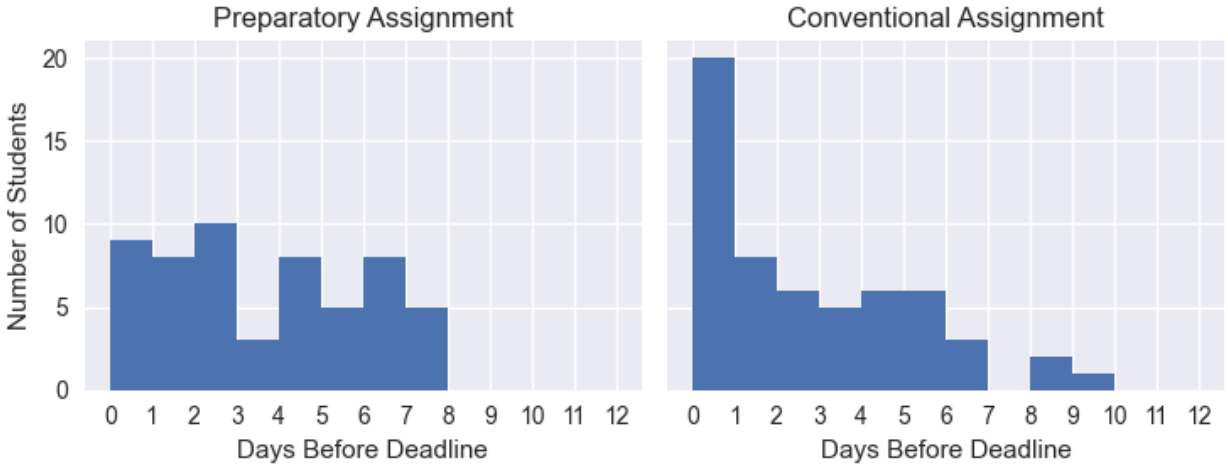
All in all, student time management on the gamified assignment was more like the preparatory assignment than the conventional assignment even though it was normally graded. This supports **Hypothesis 2** and indicates that the low risk nature of the preparatory assignment is not necessary to reproduce the improvement in student time management seen there. In fact, a highly gamified yet normally graded assignment can produce even stronger results. This provides evidence for our theory that **academic stress is the cause of students' poor time management** and that **assignments with high engagement or low risk can mitigate stress and thereby improve time management**.

Does Metric Matter?

With some results in hand, we wish to test whether our choice of metric for submission time was a good one, as there were many other alternatives – first submission time [29, 40, 45], last submission time [29, 40, 56], number of days worked [45, 87, 94], etc. To begin, we compare the distribution of first submission times (the most popular metric of earliness) with that of mean submission times (our metric). See Figure 7.4.



a First Submission Time



b Mean Submission Time

Figure 7.4: A comparison of the histograms of the first and mean submissions times for the preparatory and conventional assignment. Assignments are distributed about a week (7-12 days) before the deadline.

The first submission times suggest most everyone worked early on the preparatory assignment. The mean submission times, however, show that workloads were more uniformly distributed across the period. Similarly, while first submission times suggest that students' earliness was relatively uniformly distributed across the conventional assignment, the mean submission time indicates that most of their work was crammed into the final few days.

Next, we computed the correlation coefficients (two-sided Pearson's r) between the popular metrics of earliness and the student's grade and compared them to results we obtained for mean submission time. On the conventional assignment, where the correlation for mean submission time

was strong and statistically significant ($r = 0.44$, $p = 0.0007$), the correlations for first submission time ($r = 0.40$, $p = 0.002$), last submission time ($r = 0.29$, $p = 0.03$), and days worked ($r = 0.31$, $p = 0.02$) were all weaker but still significant. The preparatory assignment was similar. The correlation for mean submission time was the strongest ($r = 0.40$, $p = 0.003$), and the correlations for first submission time ($r = 0.30$, $p = 0.03$) and last submission time ($r = 0.31$, $p = 0.02$) were weaker. Days worked even had a statistically insignificant *negative* correlation ($r = -0.18$, $p = 0.19$). Finally, on the gamified assignment, where mean submission time had a weak and barely significant correlation ($r = 0.25$, $p = 0.065$), the correlations for first submission time ($r = 0.09$, $p = 0.52$), last submission time ($r = 0.14$, $p = 0.31$), and days worked $r = 0.20$, $p = 0.12$) were all weaker and statistically insignificant. In summary, **the metric of mean submission time was a stronger predictor of student performance than others common in the literature on all assignment styles**. To our knowledge, this is a new result, as few others use this metric and those that did (e.g., Rao [70]), did not compare it with others.

7.2 Measuring Time-on-Task

We measure time-on-task through the proxy metric of total interaction time. Interaction time is the time difference between platform log entries. Total interaction time is the sum of all such time differences, but with a caveat: to account for breaks, we exclude interaction times above a certain threshold (the break threshold) when summing them – an approach also taken by previous work [46]. A longer threshold has a higher chance of false negatives (including shorter breaks as long work sessions) while a shorter threshold has a higher chance of false positives (excluding long work sessions as breaks). From experimenting with different thresholds, we observed that the break threshold setting has a major impact on the resulting analysis.

To determine an appropriate threshold, we compared the distribution of total interaction times for different thresholds with the students' self-reported time-on-task from their responses on a post-assignment survey. Since our objective time-on-task metric only measures the time the students were actively coding and not time they may have spent thinking, researching, or otherwise focused

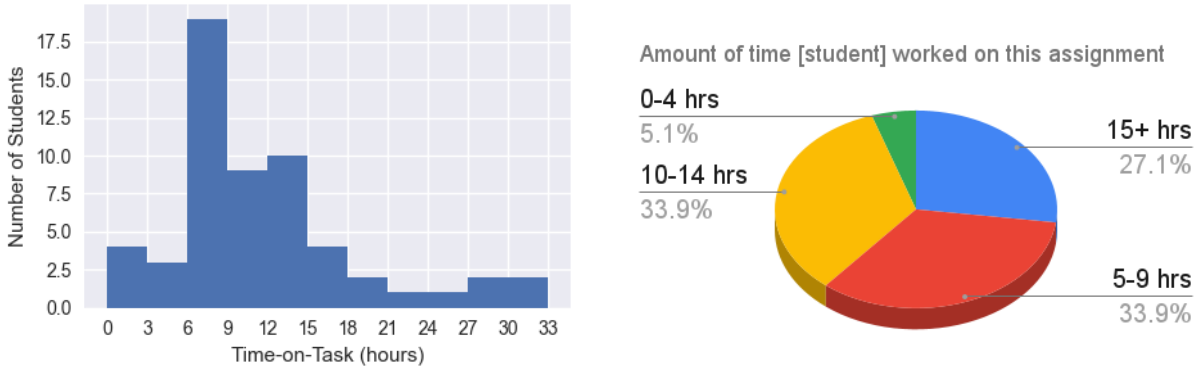


Figure 7.5: A comparison of the observed time-on-task of students for the conventional assignment with a 30 minute break threshold (left) versus the self-reported time-on-task on the survey (right).

on the assignment, we expect the measured time-on-task to be less than the self-reported time-on-task. Considering this, we chose a break threshold of 30 minutes, as this resulted in interaction times that best approximated the self-reported time-on-task without overshooting it (see Figure 7.5). Our break threshold is higher than previous work [46] because we have a coarser metric for activity (log entries versus keystrokes).

Results

We computed the correlation coefficient (two-sided Pearson's r) between a student's total interaction time and grade. A positive correlation for total interaction time (i.e., more time-on-task, higher grade) would support **Hypothesis 1d** and follow previous research (e.g., Leinonen et al. [46]). As expected, we find a statistically significant positive correlation on the conventional assignment ($r = 0.36, p = 0.007$). However, on the preparatory assignment, we find a statistically insignificant and slightly *negative* correlation ($r = -0.12, p = 0.37$), which is quite extraordinary. Figure 7.6 helps illustrate the difference.

While our results are contrary to common wisdom [46], our theory that earliness and time-on-task are interrelated (RQ3) may help explain them. To verify that earliness impacts time-on-task, we computed the correlation coefficient (two-sided Pearson's r) between mean submission time and total interaction time. On the preparatory assignment, where few students started late, we found a statistically significant negative correlation ($r = -0.36, p = 0.008$). Conversely, on the conventional

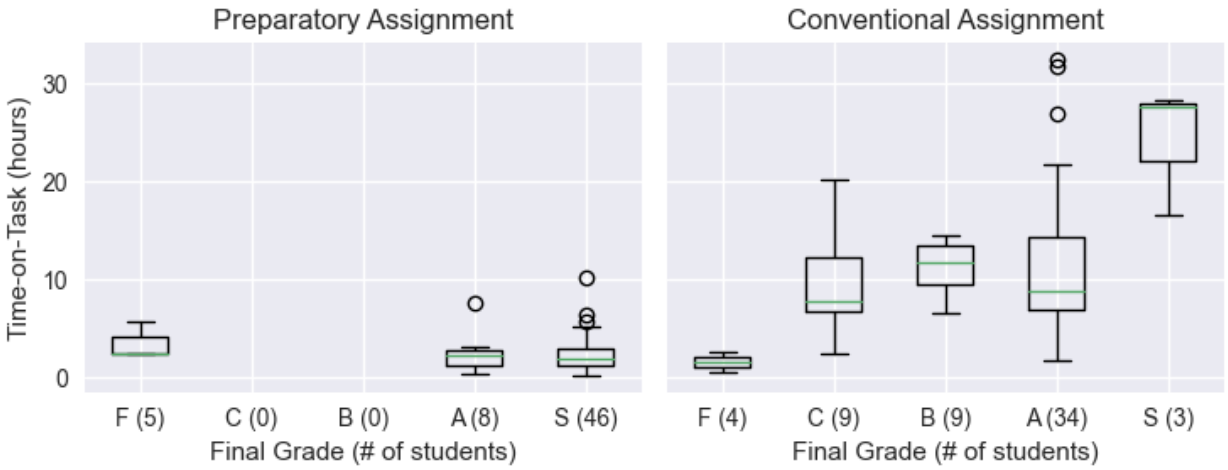


Figure 7.6: A box plot of students' time-on-task categorized by their final grade on the assignment.

assignment, where many students started late, we found a statistically significant positive correlation ($r = 0.31, p = 0.02$). This supports **Hypothesis 3** and provides evidence for our theory that **students running out of time is a significant confounding factor in time-on-task analysis**.

However, it is not the only factor in time-on-task analysis. We also computed the correlation coefficient (two-sided Pearson's r) between mean submission time and total interaction time for the gamified assignment and found a statistically significant positive correlation ($r = 0.28$, $p = 0.031$). Despite few students starting late, how early one started was still had a significant impact on ime-on-task, contradicting Hypothesis 3. Furthermore, we also found a significant positive correlation between total interaction time and grade ($r = 0.29, p = 0.03$). Lower performing student still spent less time on the assignment, despite starting early enough to spend more.

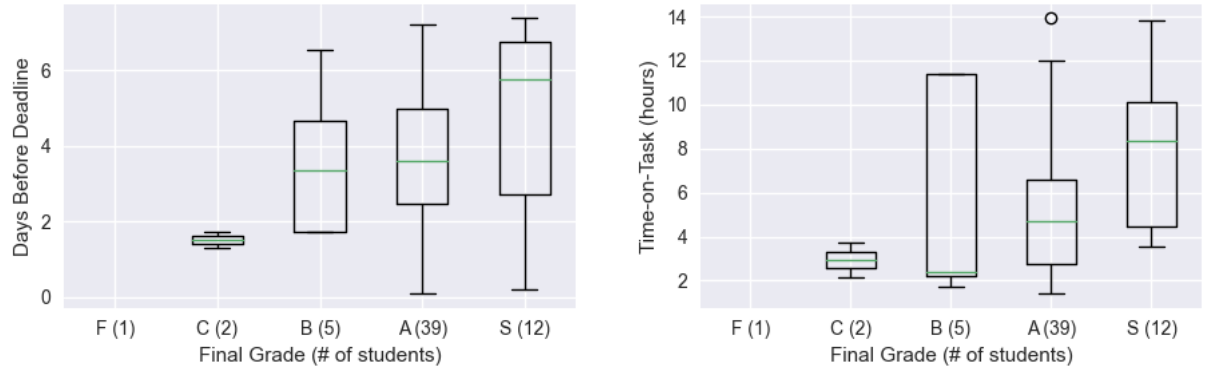


Figure 7.7: Boxplots of the student's mean connection time (left) and time-on-task (right).

We have two explanations for why this is. First, many lower performing students simply choose to spend less time, presumably satisfied with their grade. Despite B students working days before the deadline, some only spent a few hours on the assignment (see Figure 7.7). Second, the gamified assignment was highly engaging and students continued playing the game long after what was required to achieve full marks (see Chapter 6). As students who were heavily engaged were also shown to perform better, increased engagement increases time-on-task and performance, producing a correlation between the latter two. Furthermore, the earlier a heavily engaged students starts, the longer they can play, producing a correlation between earliness and time-on-task. Therefore, **engagement is another significant confounding factor in time-on-task analysis**.

7.3 Insights and Takeaways

Earliness of work and time-on-task are often cited as key factors in determining student performance [45, 46]. Our results indicate that, while earliness and time-on-task do often correlate with student performance, the correlations do not always hold and can even be inverted, and that there are many confounding factors to consider.

Earliness

Students need enough time to properly finish their work, and our results indicate that this is the reason for the usual correlation between earliness and performance. If students start too late, they will run out of time and underperform. However, if they give themselves the time to work they need, how early they start does not matter.

Therefore, like Leinonen et al. [45], **we suggest that using earliness of work as a pure predictor of performance is unwise**. Instead, **we recommend simply measuring whether students gave themselves enough time to finish their work and researching the reason why they did or did not.**

Stress

Students know procrastination is a problem and research has shown that verbal reinforcement of this fact does not significantly change their behavior [70]. Instead, prior work highlights that incentives must be built into the design of assessments to improve student time management [56]. Existing interventions focus on coercing students to start early and work often using strategies like intermittent scheduled feedback [20], limited daily submissions [40], and time management visualizations [39].

We, however, conjecture that most students already know how to manage their time well; they simply do not do so. We base this on our results, where student's time management varied across assignments and was actually quite good on some. Instead, we theorize that students delay work because of stress and dread. This stress can be reduced by providing less stick (lower risk) or more carrot (fun and engagement). Hence, students worked early on the low-risk preparatory assignment and the highly engaging gamified assignment, but delayed the stressful conventional assignment.

To our knowledge, **there is little research investigating the use of assignment or course design to reduce stress and thereby improve time management and avoid procrastination.** In fact, the intervention in Irwin et al. [40] notably *increased* student frustration. In contrast, there is substantial psychology research on academic stress, its causes, and its relation with time management and procrastination [6, 57, 78], but there is little insight on how one should structure a course to reduce it. Therefore, **we recommend further research into this area and suggest designing assignments with more carrot and/or less stick.**

Time-on-Task

On simple preparatory assignments, students can achieve high marks regardless of time spent. In fact, unprepared students can spend more time on the task and still underperform. Therefore, one underlying factor to consider is the average completion time for students of different level of skills. If students who know the material well can finish quickly while those who are still learning will need more time, the usual correlation between time-on-task and performance may even be negative.

Another confounding factor is student time management. Students who start too late can run out of time, spending less time and receiving lower grades on an assignment than they otherwise would have. Conversely, students who could potentially achieve more if they spent more time on assignment may choose not to, even if they have ample time to do so. Both behaviors can generate a correlation between time-on-task and performance, but they are very different in cause.

Finally, there is gamification and engagement to consider. If students enjoy the task at hand and are motivated to continue working even after achieving the highest marks, they will spend more time on the assignment than necessary, artificially strengthening the correlation between time-on-task and performance.

Considering these caveats, we find time-on-task to be an ambiguous metric and we recommend that time-on-task be decomposed into its constituent factors rather than analyzed as a whole. For example, in Leinonen et al. [46], the machine learning model for predicting performance from time-on-task was much more accurate than simple linear correlations. It may be that it learned caveats like these to improve its predictions.

7.4 Final Remarks

Students can struggle with time management, and we have examined the impact that different assignment styles can have on this behavior. On a conventional assignment, students followed common wisdom: most students worked on the assignment close to the deadline and earliness and time-on-task were strongly correlated with performance. However, on a preparatory or gamified assignment, many students started and finished early, and the correlation between with performance was weaker and less significant. Furthermore, on our preparatory assignment, time-on-task was negatively (albeit insignificantly) correlated with performance, clearly defying standard expectations.

Our results suggested that student time management is more complex than previously understood, so we presented the theory that one key element in this equation is academic stress. If an assignment causes too much stress, students delay it; otherwise, they organically manage their time better. Gamified assignments and preparatory assignments reduce stress either by increasing engage-

ment (more carrot) or reducing risk (less stick). In contrast, on a conventional assignment, stress makes them delay work on it, sometimes for so long that they run out of time and underperform.

Regardless of whether our explanation is correct, we hope instructors can use our results to organically improve student time management in their own courses. Expanding on the idea that preparatory assignments help, one might preface a complex task with simpler practice tasks, e.g., following an approach like that of Denny et al. [19]. Alternatively, one may seek to increase student engagement through gamification [15]. Or, like us, one may use a combination of the two [50, 85].

CHAPTER 8: SUMMARY AND CONCLUSION

The demands of cybersecurity education motivated us to find a system that could provide students with hands-on education, make exercises interesting, and support distance learning. To meet these needs, we turned to the world of online gamified learning platforms. However, we found the existing cybersecurity solutions of CTFs and wargames lacking.

These exercises merely use gamification as a vector for competition and progress tracking (in the form of points, trophies, and leaderboards) – an all-too-common feature of educational gamification even outside cybersecurity [2]. Instead, we advanced the thesis that a fully immersive gamified experience where exercises are based around a true game would be a more effective educational tool. To evaluate this thesis, we leveraged Riposte, an online gamified learning platform we designed around this concept.

In the outline of Riposte, we highlighted important design considerations both general to computer science learning platforms and specific to our topic of cybersecurity. Our system is modular and agile, making it easy to develop and customize assignments and toggle various features and security exploits when appropriate. This allows us to quickly adapt to student needs and disable simpler solutions as the course progresses. To reduce setup costs and mitigate student struggles with environment configurations, the platform is entirely online, with a development environment (Jupyter Notebook) built in. Its distributed infrastructure is built with security and scalability in mind, allowing us to quickly onboard new students and ensure they do not break each other's setups.

We also described our pedagogical approach, highlighting how Riposte facilitates each of Gagne's nine steps of the learning process. The unique nature of semi-structured, challenge-based assignments also motivated the creation of a novel 7-tiered skill assessment framework to track student abilities and guide assignment design. We then demonstrated its use by describing a selection

of assignments from our curriculum, emphasizing the three major styles: preparatory, conventional, and gamified.

Riposte debuted in the Fall 2019 semester of the "Introduction to Computer Security" to much acclaim. We received considerable positive qualitative feedback from students at the end of the semester regarding how interesting and engaging the assignments were, with some even raving that this was by far the most fun class they had taken in computer science. However, there was room for improvement. There were missteps in the leaderboard's design and implementation and students struggled with an number of difficulties unrelated to the cybersecurity learning objectives (e.g., writing asynchronous JavaScript).

As a result, in subsequent iterations, we adjusted the leaderboard, introduced Field Exercises, integrated Jupyter Notebook, and created our custom autograding system. While these elements did improve student performance and the Field Exercises and Jupyter Notebook proved popular, the feedback of the grader and aide received a more lukewarm reception. Qualitative responses revealed that the students wanted to rely more on the grader to help fix their code, the merits of which are debatable. They also had a hard time optimizing their code to avoid resource limits, which was acerbated by the minimal detail initially provided on them. Thus, while we have likely not yet struck the right balance between too much and too little feedback, there is something to be said for keeping it simple, especially since we already observed performance improvements.

Turning to our thesis, we evaluated the impact gamification had on student enjoyment, engagement, and learning outcomes, paying special attention to potential confounding influence of leaderboard position. We demonstrated a significant increase in learning outcomes and enjoyment and few negative effects, even for those lowest on the leaderboard. Furthermore, we found that many students were so heavily engaged that they continued to play the game after achieving full marks. Such results corroborate well the commonly stated benefits of gamification [2, 15].

We also analyzed the effect of assignment design on student time management. On a conventional assignment, students followed common wisdom: most students worked on the assignment close to the deadline and earliness and time-on-task were strongly correlated with performance.

However, on a preparatory or gamified assignment, many students started and finished early, and the correlation with performance was weaker and less significant.

Our results suggested that student time management is more complex than previously understood, so we presented the theory that academic stress is a key element in this equation. If an assignment causes too much stress, students delay it; otherwise, they organically manage their time better. Gamified assignments and preparatory assignments reduce stress either by increasing engagement (more carrot) or reducing risk (less stick). In contrast, on a conventional assignment, stress makes them delay work on it, sometimes for so long that they run out of time and underperform.

Overall, we found that our fully immersive gamified learning environment successfully improved student enjoyment, time management, and performance. The fully gamified assignments centered around Riposte's 2D action game did so better than the minimally gamified conventional programming assignments, providing strong support for our thesis. However, our experience with gamified education was not without its pitfalls and missteps.

It took multiple interventions to successfully develop each component and elements like the feedback of the grading system could still use more refinement. Furthermore, we learned that there are a number of aspects beyond these simple metrics to consider, such as navigating the pitfalls of semi-structured exercises (e.g., depth-first thinking). A promising avenue for future research is the possibility of using advanced AI models (e.g., ChatGPT) to provide students real-time guidance and on-demand explanation of concepts to address such struggles [71].

Finally, while our course did receive many great reviews and the overall sentiment was positive, there were still times when a subsection of students was displeased with gamification. Thus, if gamification is to be adopted as a winning educational strategy, it will be important to assess who benefits from gamification and figure out how best to individualize the gamified learning experience to those it serves, while providing off-ramps to those it does not.

REFERENCES

[1] Mohammad Alshammari, Rachid Anane, and Robert J. Hendley. 2015. The Impact of Learning Style Adaptivity in Teaching Computer Security. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (Vilnius, Lithuania) *(ITiCSE '15)*. Association for Computing Machinery, New York, NY, USA, 135–140. https://doi.org/10.1145/2729094.2742614

[2] Shurui Bai, Khe Foon Hew, and Biyun Huang. 2020. Does gamification improve student learning outcome? Evidence from a meta-analysis and synthesis of qualitative data in educational contexts. *Educational Research Review* 30 (2020), 100322. https://doi.org/10.1016/j.edurev.2020.100322

[3] Shurui Bai, Khe Foon Hew, Michael Sailer, and Chengyuan Jia. 2021. From top to bottom: How positions on different types of leaderboard may affect fully online student learning performance, intrinsic motivation, and course engagement. *Computers & Education* 173 (2021), 104–297. https://doi.org/10.1016/j.compedu.2021.104297

[4] Elisa Baniassad, Lucas Zamprogno, Braxton Hall, and Reid Holmes. 2021. STOP THE (AUTOGRADER) INSANITY: Regression Penalties to Deter Autograder Overreliance. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 1062–1068. https://doi.org/10.1145/3408877.3432430

[5] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J. Gomez. 2018. The Effect of a Web-Based Coding Tool with Automatic Feedback on Students' Performance and Perceptions. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, MD, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 2–7. https://doi.org/10.1145/3159450.3159579

[6] Robert T. Brown. 1992. Helping Students Confront and Deal with Stress and Procrastination. *Journal of College Student Psychotherapy* 6, 2 (1992), 87–102. https://doi.org/10.1300/J035v06n02_09

[7] Alberto Cardoso, Joaquim Leitão, and César Teixeira. 2019. Using the Jupyter Notebook as a Tool to Support the Teaching and Learning Processes in Engineering Courses. In *The Challenges of the Digital Transformation in Education*, Michael E. Auer and Thrasyvoulos Tsiatsos (Eds.). Springer International Publishing, Cham, 227–236. https://doi.org/10.1007/978-3-030-11935-5_22

[8] Martin Carlisle, Michael Chiaramonte, and David Caswell. 2015. Using CTFs for an Undergraduate Cyber Education. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. USENIX Association, Washington, D.C., USA. https://www.usenix.org/conference/3gse15/summit-program/presentation/carlisle

[9] P.M. Chen. 2004. An automated feedback system for computer organization projects. *IEEE Transactions on Education* 47, 2 (2004), 232–240. https://doi.org/10.1109/TE.2004.825220

[10] Katheryn R. Christy and Jesse Fox. 2014. Leaderboards in a virtual classroom: A test of stereotype threat and social comparison explanations for women's math performance. *Computers & Education* 78 (2014), 66–77. https://doi.org/10.1016/j.compedu.2014.05.005

[11] David Codish and Gilad Ravid. 2014. Personality based gamification-Educational gamification for extroverts and introverts. In *Proceedings of the 9th CHAIS Conference for the Study of Innovation and Learning Technologies: Learning in the Technological Era*, Vol. 1. The Open University of Israel Ra'anana, 36–44.

[12] Jerusha O. Conner and Denise C. Pope. 2013. Not Just Robo-Students: Why Full Engagement Matters and How Schools Can Promote It. *Journal of Youth and Adolescence* 42, 9 (01 Sep 2013), 1426–1442. https://doi.org/10.1007/s10964-013-9948-y

[13] Thomas M. Connolly, Elizabeth A. Boyle, Ewan MacArthur, Thomas Hainey, and James M. Boyle. 2012. A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education* 59, 2 (2012), 661–686. https://doi.org/10.1016/j.compedu.2012.03.004

[14] Amy Cook, Alina Zaman, Eric Hicks, Kriangsiri Malasri, and Vinhthuy Phan. 2022. Try That Again! How a Second Attempt on In-Class Coding Problems Benefits Students in CS1. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE '22)*. Association for Computing Machinery, New York, NY, USA, 509–515. https://doi.org/10.1145/3478431.3499362

[15] Luma da Rocha Seixas, Alex Sandro Gomes, and Ivanildo José de Melo Filho. 2016. Effectiveness of gamification in the engagement of students. *Computers in Human Behavior* 58 (2016), 48–63. https://doi.org/10.1016/j.chb.2015.11.021

[16] Adrian Dabrowski, Markus Kammerstetter, Eduard Thamm, Edgar Weippl, and Wolfgang Kastner. 2015. Leveraging Competitive Gamification for Sustainable Fun and Profit in Security Education. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. USENIX Association, Washington, D.C., USA. https://www.usenix.org/conference/3gse15/summit-program/presentation/dabrowski

[17] Melissa Dark. 2014. Advancing Cybersecurity Education. *IEEE Security & Privacy* 12, 6 (Nov 2014), 79–83. https://doi.org/10.1109/MSP.2014.108

[18] Fadi P. Deek, Ki-Wang Ho, and Haider Ramadhan. 2000. A critical analysis and evaluation of Web-based environments for program development. *The Internet and Higher Education* 3, 4 (2000), 223–269. https://doi.org/10.1016/S1096-7516(01)00038-0

[19] Paul Denny, Andrew Luxton-Reilly, Michelle Craig, and Andrew Petersen. 2018. Improving Complex Task Performance Using a Sequence of Simple Practice Tasks. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE '18)*. Association for Computing Machinery, New York, NY, USA, 4–9. https://doi.org/10.1145/3197091.3197141

[20] Paul Denny, Jacqueline Whalley, and Juho Leinonen. 2021. Promoting Early Engagement with Programming Assignments Using Scheduled Automated Feedback. In *Australasian Computing Education Conference* (Virtual, SA, Australia) *(ACE '21)*. Association for Computing Machinery, New York, NY, USA, 88–95. https://doi.org/10.1145/3441636.3442309

[21] Christo Dichev and Darina Dicheva. 2017. Gamifying education: what is known, what is believed and what remains uncertain: a critical review. *International Journal of Educational Technology in Higher Education* 14, 1 (20 Feb 2017), 9. https://doi.org/10.1186/s41239-017-0042-5

[22] Pieternel Dijkstra, Hans Kuyper, Greetje van der Werf, Abraham P. Buunk, and Yvonne G. van der Zee. 2008. Social Comparison in the Classroom: A Review. *Review of Educational Research* 78, 4 (2008), 828–879. https://doi.org/10.3102/0034654308321210

[23] Betsy James DiSalvo and Amy Bruckman. 2009. Questioning Video Games' Influence on CS Interest. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (Orlando, FL, USA) *(FDG '09)*. Association for Computing Machinery, New York, NY, USA, 272–278. https://doi.org/10.1145/1536513.1536561

[24] Adrian Dominguez, Joseba Saenz de Navarrete, Luis de Marcos, Luis Fernandez-Sanz, Carmen Pages, and Jose-Javier Martinez-Herraiz. 2013. Gamifying learning experiences: Practical implications and outcomes. *Computers & Education* 63 (2013), 380–392. https://doi.org/10.1016/j.compedu.2012.12.020

[25] Wenliang Du and Ronghua Wang. 2008. SEED: A Suite of Instructional Laboratories for Computer Security Education. *J. Educ. Resour. Comput.* 8, 1, Article 3 (Mar 2008), 24 pages. https://doi.org/10.1145/1348713.1348716

[26] Brianna Dym, Cole Rockwood, and Casey Fiesler. 2023. Gaming Together, Coding Together: Collaborative Pathways to Computational Learning. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1035–1041. https://doi.org/10.1145/3545945.3569833

[27] John Edwards, Joseph Ditton, Dragan Trninic, Hillary Swanson, Shelsey Sullivan, and Chad Mano. 2020. Syntax Exercises in CS1. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 216–226. https://doi.org/10.1145/3372782.3406259

[28] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain) *(ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 328. https://doi.org/10.1145/1384271.1384371

[29] Stephen H. Edwards, Jason Snyder, Manuel A. Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. 2009. Comparing Effective and Ineffective Behaviors of Student

Programmers. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop* (Berkeley, CA, USA) *(ICER '09)*. Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/1584322.1584325

[30] Richard M Felder. 1993. Reaching the second tier. *Journal of college science teaching* 23, 5 (1993), 286–290.

[31] Panagiotis Fotaris, Theodoros Mastoras, Richard Leinfellner, and Yasmine Rosunally. 2016. Climbing up the leaderboard: An empirical study of applying gamification techniques to a computer programming class. *Electronic Journal of e-learning* 14, 2 (2016), 94–110. https://academic-publishing.org/index.php/ejel/article/view/1747

[32] Robert M. Gagne and Leslie J. Briggs. 1974. *Principles of instructional design.* Holt, Rinehart & Winston, Oxford, England. ix, 270–ix, 270 pages.

[33] Robert M. Gagne, Walter W. Wager, Katharine C. Golas, and John M. Keller. 2004. *Principles of instructional design* (5th ed.). Cengage Learning, Samford, CT, USA.

[34] Andreas Haggman. 2019. *Cyber Wargaming: Finding, Designing, and Playing Wargames for Cyber Security Education*. Ph. D. Dissertation. Royal Holloway, University of London.

[35] Juho Hamari, Jonna Koivisto, and Harri Sarsa. 2014. Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. In *2014 47th Hawaii International Conference on System Sciences*. 3025–3034. https://doi.org/10.1109/HICSS.2014.377

[36] Jessica B. Hamrick. 2016. Creating and Grading IPython/Jupyter Notebook Assignments with NbGrader. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, TN, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 242. https://doi.org/10.1145/2839509.2850507

[37] Qiang Hao, David H. Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H. Arakawa, Alistair Turcan, Timothy Poehlman, and Tyler Greer. 2022. Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education* 32, 1 (2022), 105–127. https://doi.org/10.1080/08993408.2020.1860408

[38] Kristin E. Heckman, Michael J. Walsh, Frank J. Stech, Todd A. O'Boyle, Stephen R. DiCato, and Audra F. Herber. 2013. Active cyber defense with denial and deception: A cyber-wargame experiment. *Computers & Security* 37 (2013), 72–77. https://doi.org/10.1016/j.cose.2013.03.015

[39] Kalle Ilves, Juho Leinonen, and Arto Hellas. 2018. Supporting Self-Regulated Learning with Visualizations in Online Learning Environments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 257–262. https://doi.org/10.1145/3159450.3159509

[40] Michael S. Irwin and Stephen H. Edwards. 2019. Can Mobile Gaming Psychology Be Used to Improve Time Management on Programming Assignments?. In *Proceedings of the ACM Conference on Global Computing Education* (Chengdu, Sichuan, China) *(CompEd '19)*. Association for Computing Machinery, New York, NY, USA, 208–214. https://doi.org/10.1145/3300115.3309517

[41] Yuan Jia, Yikun Liu, Xing Yu, and Stephen Voida. 2017. Designing Leaderboards for Gamification: Perceived Differences Based on User Ranking, Application Domain, and Personality Traits. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1949–1960. https://doi.org/10.1145/3025453.3025826

[42] Craig Jordan, Matt Knapp, Dan Mitchell, Mark Claypool, and Kathi Fisler. 2011. CounterMeasures: A game for teaching computer security. In *2011 10th Annual Workshop on Network and Systems Support for Games*. 1–6. https://doi.org/10.1109/NetGames.2011.6080983

[43] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2016. Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 41–46. https://doi.org/10.1145/2899415.2899422

[44] Richard N. Landers and Amy K. Landers. 2014. An Empirical Test of the Theory of Gamified Learning: The Effect of Leaderboards on Time-on-Task and Academic Performance. *Simulation & Gaming* 45, 6 (2014), 769–785. https://doi.org/10.1177/1046878114563662

[45] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2021. Does the Early Bird Catch the Worm? Earliness of Students' Work and Its Relationship with Course Outcomes. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Virtual Event, Germany) *(ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 373–379. https://doi.org/10.1145/3430665.3456383

[46] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2022. Time-on-Task Metrics for Predicting Performance. *ACM Inroads* 13, 2 (May 2022), 42–49. https://doi.org/10.1145/3534564

[47] Weiwen Leung. 2019. How Do One's Peers on a Leaderboard Affect Oneself?. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland, UK) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3290605.3300397

[48] Therese H Macan, Comila Shahani, Robert L Dipboye, and Amanda P Phillips. 1990. College students' time management: Correlations with academic performance and stress. *Journal of Educational Psychology* 82 (1990), 760–768. https://doi.org/10.1037/0022-0663.82.4.760

[49] Mac Malone and Fabian Monrose. 2021. Game & Match: The Effects of Various Gamified Elements in a Computer Security Course. Univeristy of North Carolina at Chapel Hill. IRB #21-1879..

[50] Mac Malone, Yicheng Wang, Kedrian James, Murray Anderegg, Jan Werner, and Fabian Monrose. 2021. To Gamify or Not? On Leaderboard Effects, Student Engagement and Learning Outcomes in a Cybersecurity Intervention. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 1135–1141. https://doi.org/10.1145/3408877.3432544

[51] Mac Malone, Yicheng Wang, and Fabian Monrose. 2021. An Online Gamified Learning Platform for Teaching Cybersecurity and More. In *Proceedings of the 22st Annual Conference on Information Technology Education* (SnowBird, UT, USA) *(SIGITE '21)*. Association for Computing Machinery, New York, NY, USA, 29–34. https://doi.org/10.1145/3450329.3476859

[52] Mac Malone, Yicheng Wang, and Fabian Monrose. 2023. Securely Autograding Cybersecurity Exercises Using Web Accessible Jupyter Notebooks. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE '23)*. Association for Computing Machinery, New York, NY, USA, 165–171. https://doi.org/10.1145/3545945.3569862

[53] Charles F. Manski. 1993. Identification of Endogenous Social Effects: The Reflection Problem. *The Review of Economic Studies* 60, 3 (07 1993), 531–542. https://doi.org/10.2307/2298123

[54] Daniel Manson and Ronald Pike. 2014. The Case for Depth in Cybersecurity Education. *ACM Inroads* 5, 1 (Mar 2014), 47–52. https://doi.org/10.1145/2568195.2568212

[55] Hamza Manzoor, Amit Naik, Clifford A. Shaffer, Chris North, and Stephen H. Edwards. 2020. Auto-Grading Jupyter Notebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, New York, NY, USA, 1139–1144. https://doi.org/10.1145/3328778.3366947

[56] Joshua Martin, Stephen H. Edwards, and Clfford A. Shaffer. 2015. The Effects of Procrastination Interventions on Programming Project Success. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, NE, USA) *(ICER '15)*. Association for Computing Machinery, New York, NY, USA, 3–11. https://doi.org/10.1145/2787622.2787730

[57] Ranjita Misra and Michelle McKean. 2000. College students' academic stress and its relation to their anxiety, time management, and leisure satisfaction. *American journal of Health studies* 16, 1 (2000), 41.

[58] Joydeep Mitra. 2023. Studying the Impact of Auto-Graders Giving Immediate Feedback in Programming Assignments. In *Proceedings of the 54th ACM Technical Symposium on Computer*

Science Education V. 1 (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 388–394. https://doi.org/10.1145/3545945.3569726

[59] Dominique Muller and Marie-Pierre Fayant. 2010. On Being Exposed to Superior Others: Consequences of Self-Threatening Upward Social Comparisons. *Social and Personality Psychology Compass* 4, 8 (2010), 621–634. https://doi.org/10.1111/j.1751-9004.2010.00279.x

[60] Cristina Muntean. 2011. Raising engagement in e-learning through gamification. *International Conference on Virtual Learning* (Jan 2011).

[61] Susanne Narciss. 2008. Feedback strategies for interactive learning tasks. In *Handbook of research on educational communications and technology* (3rd ed.), David Jonassen, Michael J. Spector, Marcy Driscoll, M. David Merrill, Jeroen van Merrienboer, and Marcy P. Driscoll (Eds.). Routledge, New York, NY, USA, 125–143. https://doi.org/10.4324/9780203880869

[62] David J. Nicol and Debra Macfarlane-Dick. 2006. Formative assessment and self-regulated learning: a model and seven principles of good feedback practice. *Studies in Higher Education* 31, 2 (2006), 199–218. https://doi.org/10.1080/03075070600572090

[63] Scott Nykl, Chad Mourning, Mitchell Leitch, David Chelberg, Teresa Franklin, and Chang Liu. 2008. An overview of the STEAMiE Educational game Engine. In *Annual Frontiers in Education Conference*. F3B–21–F3B–25. https://doi.org/10.1109/FIE.2008.4720454

[64] Chartered Institute of Information Security. 2019. Skill Framework v2.4. https://www.ciisec.org/Skills_Framework

[65] Geraldine O'Neill and Tim Mcmahon. 2005. Student-centred learning: What does it mean for students and lecturers? *Emerging Issues in the Practice of University Learning and Teaching* 1 (01 2005). http://eprints.teachingandlearning.ie/id/eprint/3345

[66] Ian Parberry, J. Nunn, Joseph Scheinberg, Erik Carson, and Jason Cole. 2007. SAGE: a simple academic game engine. In *Proceedings of the 2nd Annual Microsoft Academic Days on Game Development in Computer Science Education*. 90–94.

[67] Allen Parrish, John Impagliazzo, Rajendra K. Raj, Henrique Santos, Muhammad Rizwan Asghar, Audun Jøsang, Teresa Pereira, and Eliana Stavrou. 2018. Global Perspectives on Cybersecurity Education for 2030: A Case for a Meta-Discipline. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion)*. Association for Computing Machinery, New York, NY, USA, 36–54. https://doi.org/10.1145/3293881.3295778

[68] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. 157–163.

[69] Matthew Peveler, Evan Maicus, and Barbara Cutler. 2019. Comparing Jailed Sandboxes vs Containers Within an Autograding System. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 139–145. https://doi.org/10.1145/3287324.3287507

[70] Dhananjai M. Rao. 2019. Experiences With Auto-Grading in a Systems Course. In *2019 IEEE Frontiers in Education Conference (FIE)*. 1–8. https://doi.org/10.1109/FIE43999.2019.9028450

[71] Thomas Rid. 2023. Five Days in Class with ChatGPT. https://alperovitch.sais.jhu.edu/five-days-in-class-with-chatgpt/

[72] Jerome Saltzer and M Frans Kaashoek. 2009. *Principles of computer system design: an introduction*. Morgan Kaufmann, Burlington, MA, USA.

[73] J.H. Saltzer and M.D. Schroeder. 1975. The protection of information in computer systems. *Proc. IEEE* 63, 9 (1975), 1278–1308. https://doi.org/10.1109/PROC.1975.9939

[74] Spyridon Samonas and David Coss. 2014. The CIA strikes back: Redefining confidentiality, integrity and availability in security. *Journal of Information System Security* 10, 3 (2014), 21–45. https://jissec.org/Contents/V10/N3/V10N3-Samonas.html

[75] Z. Cliffe Schreuders, Thomas Shaw, Aimée Mac Muireadhaigh, and Paul Staniforth. 2018. Hackerbot: Attacker Chatbots for Randomised and Interactive Security Labs, Using SecGen and oVirt. In *2018 USENIX Workshop on Advances in Security Education (ASE 18)*. USENIX Association, Baltimore, MD, USA. https://www.usenix.org/conference/ase18/presentation/schreuders

[76] Corwin Senko and Blair Bryant Dawson. 2017. Performance-Approach Goal Effects Depend on How They Are Defined: Meta-Analytic Evidence From Multiple Educational Outcomes. *Journal of Educational Psychology* 109 (2017), 574–598. https://doi.org/10.1037/edu0000160

[77] Rebecca Sevin and Whitney Decamp. 2016. From Playing to Programming: The Effect of Video Game Play on Confidence with Computers and an Interest in Computer Science. *Sociological Research Online* 21, 3 (2016), 14–23. https://doi.org/10.5153/sro.4082

[78] Fuschia M. Sirois. 2014. Procrastination and Stress: Exploring the Role of Self-compassion. *Self and Identity* 13, 2 (2014), 128–145. https://doi.org/10.1080/15298868.2013.763404

[79] Richard E. Smith. 2012. A Contemporary Look at Saltzer and Schroeder's 1975 Design Principles. *IEEE Security & Privacy* 10, 6 (2012), 20–25. https://doi.org/10.1109/MSP.2012.85

[80] Gail M. Sullivan and Jr Artino, Anthony R. 2013. Analyzing and Interpreting Data From Likert-Type Scales. *Journal of Graduate Medical Education* 5, 4 (12 2013), 541–542. https://doi.org/10.4300/JGME-5-4-18

[81] Emily Sun, Brooke Jones, Stefano Traca, and Maarten W. Bos. 2015. Leaderboard Position Psychology: Counterfactual Thinking. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI EA '15)*. Association for Computing Machinery, New York, NY, USA, 1217–1222. https://doi.org/10.1145/2702613.2732732

[82] Mark Trueman and James Hartley. 1996. A comparison between the time-management skills and academic performance of mature and traditional-entry university students. *Higher Education* 32, 2 (01 Sep 1996), 199–215. https://doi.org/10.1007/BF00138396

[83] Martin Valez, Michael Yen, Mathew Le, Zhendong Su, and Mohammad Amin Alipour. 2020. Student Adoption and Perceptions of a Web Integrated Development Environment: An Experience Report. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, New York, NY, USA, 1172–1178. https://doi.org/10.1145/3328778.3366949

[84] Giovanni Vigna, Kevin Borgolte, Jacopo Corbetta, Adam Doupé, Yanick Fratantonio, Luca Invernizzi, Dhilung Kirat, and Yan Shoshitaishvili. 2014. Ten Years of iCTF: The Good, The Bad, and The Ugly. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. USENIX Association, San Diego, CA, USA. https://www.usenix.org/conference/3gse14/summit-program/presentation/vigna

[85] Valdemar Švábenský and Jan Vykopal. 2018. Challenges Arising from Prerequisite Testing in Cybersecurity Games. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, MD, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 56–61. https://doi.org/10.1145/3159450.3159454

[86] Valdemar Švábenský, Jan Vykopal, Milan Cermak, and Martin Laštovička. 2018. Enhancing Cybersecurity Skills by Creating Serious Games. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. Association for Computing Machinery, New York, NY, USA, 194–199. https://doi.org/10.1145/3197091.3197123

[87] Matthew M. Walsh, Kevin A. Gluck, Glenn Gunzelmann, Tiffany Jastrzembski, and Michael Krusmark. 2018. Evaluating the Theoretic Adequacy and Applied Potential of Computational Models of the Spacing Effect. *Cognitive Science* 42, S3 (2018), 644–691. https://doi.org/10.1111/cogs.12602

[88] Richard Weiss, Jens Mache, and Erik Nilsen. 2013. Top 10 Hands-on Cybersecurity Exercises. *Journal of Computing Sciences in Colleges* 29, 1 (Oct 2013), 140–147. https://dl.acm.org/doi/abs/10.5555/2527148.2527180

[89] Richard S. Weiss, Stefan Boesen, James F. Sullivan, Michael E. Locasto, Jens Mache, and Erik Nilsen. 2015. Teaching Cybersecurity Analysis Skills in the Cloud. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, MO, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 332–337. https://doi.org/10.1145/2676723.2677290

[90] Chris Wilcox. 2015. The Role of Automation in Undergraduate Computer Science Education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, MO, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 90–95. https://doi.org/10.1145/2676723.2677226

[91] Chris Wilcox. 2016. Testing Strategies for the Automated Grading of Student Programs. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, TN, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 437–442. https://doi.org/10.1145/2839509.2844616

[92] Brad Wolfenden. 2019. Gamification as a winning cyber security strategy. *Computer Fraud & Security* 2019, 5 (2019), 9–12. https://doi.org/10.1016/S1361-3723(19)30052-1

[93] Weining Yang, Ninghui Li, Omar Chowdhury, Aiping Xiong, and Robert W. Proctor. 2016. An Empirical Study of Mnemonic Sentence-Based Password Generation Strategies. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1216–1229. https://doi.org/10.1145/2976749.2978346

[94] Iman YeckehZaare, Elijah Fox, Gail Grot, Sean Chen, Claire Walkosak, Kevin Kwon, Annelise Hofmann, Jessica Steir, Olivia McGeough, and Nealie Silverstein. 2021. Incentivized Spacing and Gender in Computer Science Education. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (Virtual Event, USA) *(ICER '21)*. Association for Computing Machinery, New York, NY, USA, 18–28. https://doi.org/10.1145/3446871.3469760

[95] Leah Zhang-Kennedy, Sonia Chiasson, and Robert Biddle. 2013. Password advice shouldn't be boring: Visualizing password guessing attacks. In *2013 APWG eCrime Researchers Summit*. 1–11. https://doi.org/10.1109/eCRS.2013.6805770