Designing a Computer Game to Teach Computer Science Concepts (2023)

Aidan Chuang-Yu Lim

***Abstract*— .**

This capstone project report focuses on the development of a computer game designed to cultivate the inquisitive mindset of a software tester. The project narrows its scope to emphasize not only testing the expected "happy paths" of software logic but also exploring the "unhappy paths," such as interactions like a player navigating through obstacles in a game environment.

Our quantifiable goals center around providing users with a progressive learning experience. This experience starts with coding logic for a simple game - the classic Snake. We structured the learning journey across four levels. Level 0 begins with the basic task of moving a single box, gradually progressing to Level 4, where users can enjoy a fully functional Snake game.

The motivation behind this project stems from the author's experience as a working Software Test Engineer. As an undergraduate, the author recognized a niche in software testing education that extended beyond the typical coursework. The project draws inspiration from various university courses, showcasing the relevance of testing within the broader context of software engineering and computer science.

The report delves into the challenge of guiding users towards answers without directly revealing them while mitigating issues stemming from syntax unfamiliarity. It explores the efficacy of using games as a pedagogical tool, a method well-documented for its effectiveness in facilitating learning.

The intended audience for this project is first-year students pursuing degrees in software engineering and computer science, possessing a fundamental understanding of coding. This game serves as a gateway for them to explore the multifaceted world of software testing.

I. INTRODUCTION

In a world increasingly shaped by digital innovation, the value of computer science education has transcended the confines of academia. It is now a fundamental life skill, indispensable for navigating the complexities of the modern age [1]. Despite this growing importance, the traditional approaches to teaching computer science often fall short in engaging students [2] and fostering a profound understanding of these essential skills.

This capstone project embarks on a distinctive and innovative path, bridging the gap between the demands of computer science education and the necessity for captivating, practical, and effective teaching methods. Rather than adhering to conventional pedagogy, our project seeks to immerse students in a puzzle-based learning experience through the creation of a computer game. The core objective is to impart fundamental computer science concepts, with a specific emphasis on nurturing an inquisitive mindset, particularly within the context of software testing.

In the realm of software testing, it can be easy for learners to predominantly focus on evaluating the "happy paths" within software logic [3] – the well-trodden routes that lead to expected outcomes. However, this project boldly ventures into the less explored and potentially treacherous terrain of "unhappy paths." Here, we confront the unexpected interactions, edge cases, and glitches that mirror real-world software challenges. By diving into these challenging waters, it is hoped that students gain not only a deeper understanding of testing but also a resilience to tackle complex software issues.

The primary aim of this project is to guide students through a structured journey that commences modestly and gradually expands, mirroring the evolution of a software tester. Users begin with simple coding logic, akin to moving a single box, and progress through multiple levels, culminating in the creation of a fully playable Snake game. This journey provides a starting point for learners with no knowledge of testing to get a glimpse into a potential mindset approach.

The motivation behind this endeavor springs from the unique position of the author, who not only pursues this capstone project but has also gathered valuable experience as a working Software Test Engineer at TechTime Initiative Group Limited (4 months) and ACC New Zealand (12 months). With two distinct positions in both the private and public sectors, the author has learned from the best practices and challenges of two different testing environments. These experiences have illuminated a critical niche within the domain of computer science education, one that extends beyond traditional coursework. This is shown in the Victoria University of Wellington's Engineering and Computer Science Departments not offering testing courses, which is only taught sporadically in various courses. A quote by a first year teacher in this department goes as follows, "COMP102/103 only provides a few examples of how to perform testing well, and it could be exemplified better" – Karsten Lundqvist.

The craft of teaching testing should be more than just offering answers; it should involve guiding learners without explicitly revealing solutions. If this is to be believed, games emerge as a powerful tool for effective learning. Games have long been recognized for their capacity to captivate and educate learners through the amalgamation of challenge, interactivity, and

This project was supervised by Supervisor Name 1 (primary), Supervisor Name 2, and Supervisor Name 3 (add/remove as necessary). Please indicate affiliation of any external (non-ECS) supervisors.

rewards.

This project is aimed squarely at first-year students pursuing degrees in software engineering and computer science, students who are already equipped with basic coding knowledge and who are now poised to explore the intricate world of testing. As this report progresses, it will provide a detailed exploration of the project's design, development, and evaluation, offering insight into the potential of a game-based approach to computer science education, especially in the field of software testing. Subsequent sections will delve into the conceptual framework, implementation, and findings, providing a comprehensive understanding of the problem addressed and the innovative solution developed.

II. RELATED WORK

This section delves into the existing body of research and literature that informs the context of our capstone project. To provide an overview, we will critically analyze and compare previous approaches, highlighting their strengths, weaknesses, and areas where knowledge gaps exist. Additionally, we aim to establish the novelty and significance of our work by elucidating how it extends or addresses the limitations of prior studies.

A. Game-Based Learning in Computer Science Education

Game-based learning has gained considerable recognition in recent years as an effective pedagogical approach. [3] [4] [5] Prior studies have explored the integration of games into various educational contexts, including computer science. Notable works such as Gee's concept of "good learning principles" in games and Shaffer's research on game-based problem-solving have demonstrated the potential of games as educational tools. [4]

While these studies have made substantial progress in emphasizing the value of games for learning, they often lack a specific focus on software testing and the development of an inquisitive mindset among learners. This is where our project distinguishes itself by centering on these specific aspects and aiming to bridge this educational gap.

B. Challenges in Software Testing Education

The realm of software testing carries its own set of unique challenges and intricacies. Numerous studies have addressed the educational needs in software testing [3] [4] [5], acknowledging the gap between academic coursework and the real-world practices of software testing. These works underscore the importance of practical experience and hands-on learning.

However, they frequently fall short in proposing innovative pedagogical approaches. The conventional methods of teaching software testing, centered on theoretical lectures and assignments [6], often struggle to instill the mindset needed to explore "unhappy paths." Our project seeks to go beyond these conventional approaches by introducing a game-based learning platform, which offers hands-on, practical experience to foster the skills and mindset essential for software testing.

C. Existing Game-Based Learning Solutions

Several existing game-based learning solutions in the field of computer science education offer insight into the potential of this approach. Notable examples include "Scratch," a visual programming language that introduces coding concepts through interactive and creative game development, and "CodeCombat," a platform that gamifies coding challenges. However, while trying to find games with software testing online, the results found by google scholar lacked relevancy, and none were found with even the basic concept of testing, being that of edge-case testing.

While these solutions are commendable in their approach, they often lack the specific focus on software testing and exploring the nuances of "unhappy paths." Our project's novelty lies in its dedication to addressing these areas, catering to students with an interest in software testing and introducing them to a practical, interactive, and engaging learning environment.

III. DESIGN

In this section, we delve into the intricate design aspects of the game, exploring the thought processes and considerations that shaped its development.

A. Initial Approach and Challenges

At the project's inception, the aim was to create a game that provided users with a visual representation of a while loop and required them to exit the loop, primarily by falsifying the loop's condition. This approach, while conceptually sound, presented several challenges, primarily stemming from potentially ambiguous rules governing user actions within the game. For example, restrictions on using the "return" keyword introduced uncertainty, making it difficult to clearly define what actions were permitted. This ambiguity was deemed counterproductive to the goal of fostering a sense of exploration and creative problem-solving, fundamental to effective learning.

B. The Shift Towards Software Testing

The pivotal shift in the game's design philosophy marks a significant turning point in the project's evolution. It embodies a departure from the initial concept of teaching computer science concepts through while loop manipulation, pivoting towards a more holistic approach that integrates software testing principles. This transformation was motivated by several key considerations:

1. Real-World Relevance

Drawing from the real-world practices of the software industry, it became apparent that software testing is not solely about identifying and fixing defects in code. Instead, it encapsulates a broader scope, emphasizing a deep understanding of code functionality and the assurance that it

operates as intended. The author's realization prompted the incorporation of testing as a fundamental element in the game's design.

2. Code Quality Assurance

The new approach places a heightened emphasis on code quality and correctness. It challenges players not only to write functional code but to prove its effectiveness through manual testing. By actively involving users in the testing process, the project aims to instill a sense of responsibility for code quality, a crucial aspect of professional software development.

3. Fostering Critical Thinking

The integration of software testing introduces a higher level of critical thinking [5]. Players are no longer limited to code composition but are encouraged to think analytically and strategically, considering the various scenarios that may impact their code's behavior.

4. Learning by Doing

The new approach aligns with the philosophy of "learning by doing." While the initial concept relied on users manipulating while loops to achieve a specific outcome, the software testing approach empowers users to not only comprehend code but also to validate its functionality. This hands-on experience is instrumental in reinforcing comprehension and retention of concepts.

5. Comprehensive Skill Development

The integration of software testing concepts broadens the educational scope. It equips learners with skills that extend beyond coding, encompassing problem-solving, quality assurance, and debugging, which are integral to successful software development.

6. Alignment with Industry Practices

This educational shift mirrors contemporary industry practices. In the professional software development landscape, testing is a fundamental component of the software development lifecycle. Thus, equipping learners with testing skills from the outset fosters a seamless transition from the educational setting to professional software development environments.

7. Practical Experience

By making software testing a central element of the game, users not only grasp theoretical knowledge but also gain practical experience. They learn to design test cases, execute them, and assess code behavior. These are skills that are highly transferable to real-world software testing roles.

C. Game Design

1. Game Concept and Objectives

This project's game is a game for software testing. User's starting a level are met with four different game panels on the screen. There is an input text panel which acts as a Java IDE that allows the user to enter textual code logic, of which the user can then dynamically compile written code with an executable button in a second panel, to then see in action through reflection in the third panel being that of a board panel. The user can then use keyboard inputs to test the written logic of the game and receive console output of what manual tests have passed, and error messages for incorrect behaviour.

2. Players testing experience

Players experience testing by being required to not only correctly implement code logic for the current level, but to show they have done so. An example of this is one of the first puzzles I had designed, a simple player in a one-dimensional array, shown to the user in the game panel. The user was then tasked to implement up movement logic with the keyboard arrow key, where the down arrow key logic was shown. However, while the instructions may have emphasized that a necessary requirement to pass the level was to correctly move up in the array off an up arrow key press, the player would also need to implement logic to check if they player is in the top most section of the array, and not move upwards as it would be out of bounds. With this game emphasizing the necessity to pass the level being that both the happy path scenario (not being at the top of the array on up arrow key press moves the player up) as well as the unhappy path (being at the top of the array on up arrow key press does not move the player up), it is hoped that the user that does not correctly implement the unhappy path logic, would while attempting to pass the level, press the up arrow key while on the top most section of the array, witness the player leave the game array area, and be able to make the self-realization why their current logic does not meet the specifications, and can therefore fix their code and try again. This cycle can help learners understand that while testing may not seem important, it can be integral to correct software development outcomes.

3. Narrative

The levels of this project's game work towards creating a well known classic game, Snake. While the first levels may touch on basic movement handling, the later levels show an apple-eating snake that grows as it eats. This helps create a sense of achievement to the players, as they start out with smaller concepts in the earlier levels, the code that they have created is maintained through out the later levels, where the more difficult concepts of snake can then be added on to what was already done.

D. System Architecture

The system architecture for the game was conceived to accommodate various interactive elements essential to the learning process. These elements are meticulously designed to enhance usability, user engagement, and educational effectiveness:

- Code Editor: At the heart of the system architecture lies the code editor. This is the dynamic environment where players write, modify, and interact with Java

code. The code editor is more than just a text box; it's an essential platform where users experiment, make mistakes, and learn from them.

- Real-Time Visualization Game Board: A core component of the system architecture, the real-time visualization game board, provides players with a visual representation of their code execution. This visualization makes the program's behavior explicit and comprehensible. It enables learners to observe the consequences of their coding decisions in real-time, enhancing their understanding of code execution flow.
- Compile Button and Hint Instructions: To support the learning process, the system incorporates a compile button. This button facilitates the execution of the user-written code, providing feedback on its functionality. Additionally, hint instructions are integrated into this component, offering guidance and suggestions to assist learners in solving challenges effectively. This proactive approach to user support aligns with educational principles and reinforces the learning experience.

The strategic placement of these components within the architecture is intended to ensure that each element is readily accessible to the user. The design choice to quarter these components into each of the four corners of the game window panel offers several advantages:

- Visibility: The layout enables users to have a comprehensive view of all elements simultaneously. This visibility is essential for users to seamlessly refer between different frames during gameplay.
- Ease of Reference: The game window panel layout makes it convenient for users to reference the code they've written, the real-time code execution visualization, and any available hints or guidance, enhancing the user's problem-solving experience.

The architectural design concept was meticulously planned and validated through the use of paper prototypes. These prototypes were presented to fourth-year students, providing invaluable insights into usability, intuitiveness, and the overall effectiveness of the system's layout.

E. User Interface

The user interface design is an integral aspect of the project. It focuses on ensuring that users can interact with the game with ease and clarity. Key features of the user interface design include:

- Intuitive Navigation: The user interface is designed with a clean and intuitive layout that allows users to easily navigate between different game components.
- Visual Clarity: The user interface leverages visual elements to ensure that users can readily understand the feedback, hints, and visualizations provided during gameplay.
- Support for Learning: The user interface includes elements that guide and assist users in their learning journey, such as hint instructions and visual feedback mechanisms.
- Scalability: The design is crafted to support potential scalability, enabling the addition of new features and levels as the game evolves.
- The user interface is a critical aspect of the game's success. Its design promotes an immersive and user-friendly experience, ensuring that players can focus on learning and problem-solving rather than struggling with the game's interface.

The user interface is a critical aspect of the game's success. Its design promotes an immersive and user-friendly experience, ensuring that players can focus on learning and problem-solving rather than struggling with the game's interface.

1. Encouraging Exploration and Problem Solving

The game design motivates learners to explore various coding solutions to complex challenges. It promotes the kind of thinking required to solve practical, real-world problems. By engaging with the code, observing its behavior, and making adjustments based on practical results, users gain a deeper appreciation for the iterative and problem-solving nature of software development.

2. Emphasis on Edge Cases

One of the most critical aspects of practical software development is addressing edge cases – situations that lie outside the norm but can have a significant impact on program behavior. The project actively challenges users to consider and test edge cases. By confronting these situations within the game, learners develop the skills to handle real-world scenarios more effectively.

3. Learning by Doing

The fundamental philosophy of this project is "learning by doing." Instead of passively receiving information, learners actively engage with the material. They write code, test it, debug it, and see the consequences of their actions in real-time. This practical, hands-on experience is invaluable in reinforcing their understanding and retention of programming concepts.

4. Bridging the Transition to Real-World Software Development

The ultimate goal of this project is to facilitate a seamless transition from the educational setting to professional software development environments. By immersing learners in practical problem-solving, it equips them with skills that align with industry expectations. It cultivates the critical thinking, problem-solving, and practical coding skills required to excel in real-world software development scenarios.

In summary, the project's approach to bridging the gap between theory and practice is a transformative educational strategy. It empowers learners to think practically, encourages exploration, emphasizes the importance of edge cases, and fosters an environment of "learning by doing." By providing an interactive and practical learning experience, this project equips learners with the skills they need to excel in real-world software

development scenarios, effectively addressing one of the most significant challenges in computer science education.

F. Key Design Principles

The success and effectiveness of the educational game are underpinned by a set of key design principles that shape its structure and mechanics. These principles guide the development of the game, ensuring that it delivers a meaningful and impactful learning experience.

1. Modular Programming:

The game embraces the principle of modular programming, which involves breaking down code into small, self-contained functions. This modular approach enhances code maintainability and scalability, a practice that mirrors real-world software development. It encourages users to structure their code in a way that makes it easy to understand, test, and extend. This is shown in my game by the break down of classes into different areas of the game, with set level classes interacting with the same level helper classes, but with level specific information kept in solidarity to keep the different levels logic separated.

2. Simulate Real-World Testing Scenarios:

One of the core principles of the game design is to simulate real-world software testing scenarios. Users are not merely asked to write code; they are also required to verify its functionality through testing. This hands-on experience mirrors the responsibilities of professional testers and underscores the importance of thorough testing in software development. Users playing my game will have visual feedback of failure to stop a player incorrectly moving through player blocking walls, and would be able to understand what is necessary changes to the logic need to be implemented to complete the level.

3. Error Detection:

Users are challenged to identify and resolve code issues themselves. The game encourages an active and problem-solving mindset. By requiring users to locate and rectify problems in their code, the design promotes a deeper understanding of how code functions and where potential errors might occur.

4. Incremental Agile-Based Levels:

The game's progression is structured according to the principles of incremental development, following an agile methodology. Each level builds upon the concepts and code introduced in the previous one. This iterative approach mirrors real-world software development, where features are developed incrementally, tested, and refined. It reinforces the idea that software development is an ongoing, iterative process.

5. Reward-Based Motivation:

In contrast to traditional educational methods that may involve long stretches of reading and theory, the game design introduces a reward-based motivation system. Users are motivated and incentivized through the completion of game levels. This gamified approach provides immediate feedback and gratification, enhancing engagement and retention. It stands in contrast to traditional, text-heavy educational courses that can be less engaging.

These key design principles work in tandem to create a dynamic and effective educational tool. They encourage best practices in coding, highlight the importance of software testing, foster problem-solving skills, mirror real-world development, and provide motivation for users to advance through the learning process.

In summary, the game's design principles not only teach computer science and software testing concepts but also prepare learners for the challenges and expectations of professional software development. These principles form the foundation of an engaging and enlightening learning experience that equips users with both theoretical knowledge and practical problem-solving skills.

G. Promoting Software Testing Awareness

The game design project extends beyond its role as an educational tool; it carries the crucial objective of raising awareness about software testing and promoting better software engineering practices within the industry. This section elaborates on how the project serves as a catalyst for these broader goals.

1. Spreading Awareness:

The project acts as a medium for introducing individuals, especially those with an interest in understanding software testing, to the world of software testing. It serves as an entry point that demystifies the domain, making it accessible to those who may not have prior experience. By providing an engaging and interactive learning experience, the project raises awareness of the importance of software testing within the software development process.

2. Educational Outreach:

The game design is particularly well-suited for educational institutions and training programs. Its engaging and interactive nature makes it an appealing alternative to traditional teaching methods. It can be used in academic settings to introduce students to the concepts of software testing, thereby nurturing the next generation of software engineers and testers.

3. Practical Application of Knowledge:

Beyond theoretical knowledge, the project emphasizes the practical application of software testing concepts. By challenging users to apply testing methodologies and strategies in the game, it underscores that software testing is not merely theoretical but an active, integral practice in software development. This is very much shown in the game with users needing to realize that in game with a movable player on a game grid, a player should maintain with in the game grounds and comply with this boundary edge-case.

4. Fostering Industry Best Practices:

The project's emphasis on software testing principles and practices aligns with industry best practices. By familiarizing learners with testing methodologies, error identification, and quality assurance, it contributes to the development of skills that are directly applicable in professional software development and testing roles.

5. Understanding the Importance of Testing:

The project goes beyond the mechanics of coding to instill in learners a deep appreciation for the role of software testing in ensuring the reliability and quality of software products. It communicates the idea that testing is not an afterthought but an essential and continuous aspect of the software development process.

6. Real-World Relevance:

By introducing learners to the practical aspects of software testing and quality assurance, the project ensures that education aligns with the skills demanded by the software industry. This practical relevance is vital for learners seeking to enter the workforce with a deep understanding of the industry's expectations.

7. Spreading Industry-Accepted Practices:

As the project reinforces industry-standard practices related to coding, testing, and problem-solving, it serves as a means of spreading and normalizing these practices among learners. This, in turn, contributes to the development of better software engineering practices within the industry.

8. Enhancing Software Quality:

By educating learners about the importance of software testing and quality assurance, the project indirectly contributes to the enhancement of software quality. Learners who understand the significance of testing are more likely to incorporate testing practices into their development work, which ultimately leads to better software products.

In summary, the project's contribution to raising awareness about software testing and promoting better software engineering practices within the industry is a testament to its broader impact. By providing an interactive and practical learning experience that emphasizes the significance of software testing, it cultivates a new generation of software professionals who are well-equipped to meet industry expectations, contribute to software quality, and advocate for the importance of testing in software development.

IV. IMPLEMENTATION

The "Implementation" section takes a closer look at how the technical solution outlined in the "Design" section was transformed into a tangible artifact. It provides a detailed account of the components used, technical drawings or diagrams, and the rationale behind implementation choices.

A. Components Used

In this section, we delve into the specific components that were integral to the realization of the educational game project. These components form the technical foundation of the game, each playing a unique role in delivering the intended learning experience. Here, we explore these components in greater detail:

1. Programming Language: Java

Choice Rationale: The project's core functionality is powered by the Java programming language. Java was selected for several compelling reasons. Its platform independence ensures that the game can be accessed on various operating systems, making it highly accessible to a broad audience. Additionally, Java is renowned for its extensive libraries and robust support for graphical user interfaces (GUIs), aligning perfectly with the project's requirements for interactive and visually engaging learning.

2. User Interface (UI) Framework: Java Swing

Choice Rationale: The graphical user interface of the educational game is built using Java's Swing framework. Swing provides a rich set of tools and components for designing GUIs, making it a natural choice for creating an interactive and intuitive user experience. Its extensive features facilitate the development of user-friendly interfaces, ensuring that learners can easily navigate and interact with the game.

3. Code Compilation and Execution: Java Reflection

Choice Rationale: To dynamically compile and execute user-written Java code within the game, Java's reflection capabilities were harnessed. This feature allows users to observe the immediate results of their code, providing invaluable feedback and enhancing their comprehension of programming concepts. By incorporating reflection, the project supports a "learn by doing" philosophy, which is essential for effective education.

4. Real-Time Visualization

Choice Rationale: Real-time visualization of code execution is a critical component of the game's educational strategy. This visualization is achieved by leveraging Java's graphical capabilities, complemented by custom graphical rendering. Through real-time visualization, learners can witness the practical outcomes of their code, gaining a deeper understanding of code execution flow and behavior.

5. Code Editor: Java Swing Text Components

Choice Rationale: The code editor is the central element of the educational game, and it is implemented using Java Swing text components. These components provide a platform for users to write, edit, and interact with Java code directly within the game environment. This choice ensures that learners have a familiar and convenient interface for coding and experimentation.

6. Hint and Guidance System

Choice Rationale: A custom hint and guidance system was

developed within the Java application to provide users with helpful instructions during gameplay. This system enhances the learning experience by offering context-specific support when users encounter challenges. The custom hint system was designed to align with the project's educational objectives, ensuring that users receive timely and relevant guidance.

Each of these components plays a distinct role in the project's realization. They are carefully chosen to ensure that the game is not only educational but also interactive, accessible, and responsive to user needs.

In summary, the components used in the project's implementation are the building blocks that transform the theoretical design into a functional and engaging educational tool. They are selected with a focus on accessibility, interactivity, and alignment with the project's educational objectives.

B. Rationale for Implementation Choices

The choices made in implementing the educational game project were not arbitrary but carefully considered to align with key project objectives. Below, we explore the rationale behind these implementation choices in greater detail:

1. Programming Language: Java

Choice Rationale: Java was selected as the primary programming language for the project due to its platform independence. Java applications can run on various operating systems without modification, making the game accessible to a broad audience. This aligns with the project's goal of reaching students and learners from diverse backgrounds and technical environments. Additionally, Java's extensive libraries, strong community support, and robust graphical capabilities made it a suitable choice for developing the game's interactive and visually engaging elements.

2. User Interface (UI) Framework: Java Swing

Choice Rationale: Java Swing was chosen for designing the graphical user interface (UI) of the game. Swing is known for its comprehensive set of GUI components and tools, making it an ideal choice for creating an interactive and user-friendly interface. Its ability to handle complex layouts and its support for custom components, along with its compatibility with Java, ensured that the UI design aligned with the project's vision of an engaging and intuitive learning experience.

3. Code Compilation and Execution: Java Reflection

Choice Rationale: Java's reflection capabilities were harnessed to dynamically compile and execute user-written Java code within the game. This choice was driven by the project's commitment to providing immediate feedback to users. Reflection enables learners to observe the real-time consequences of their code, enhancing their understanding of programming concepts. It aligns with the project's educational approach of "learning by doing," a methodology proven to be effective for knowledge retention.

4. Real-Time Visualization

Choice Rationale: Real-time visualization of code execution was deemed essential for the project's educational strategy. Java's graphical capabilities, complemented by custom graphical rendering, provided a means to create this visual representation. By allowing users to witness the practical outcomes of their code in real-time, the project reinforces the learning experience by providing a tangible connection between code and execution behavior.

5. Code Editor: Java Swing Text Components

Choice Rationale: Java Swing text components were utilized to implement the code editor, enabling users to write and interact with Java code within the game environment. This choice ensures that learners have a familiar and user-friendly interface for coding and experimentation. By using Swing, the project enhances usability and accessibility, allowing learners of varying technical backgrounds to engage with the code effectively.

6. Hint and Guidance System

Choice Rationale: A custom hint and guidance system was developed within the Java application to provide users with relevant instructions during gameplay. This choice was guided by the project's educational objectives, which seek to offer support and assistance to users when needed. The custom hint system ensures that learners receive timely and context-specific guidance, helping more effective learning experience.

In summary, each implementation choice in the project was driven by considerations of usability, accessibility, and alignment with educational goals. Java was selected for its platform independence and extensive libraries, Swing for its rich GUI capabilities, reflection for immediate feedback, and custom graphics for real-time visualization. The code editor choice ensures user-friendliness, and the hint system supports learning objectives. These choices collectively form the technical foundation of the project, ensuring that the educational game is not only educational but also interactive, user-friendly, and responsive to learners' needs.

V. EVALUATION

The "Evaluation" section aims to demonstrate the project's performance in alignment with established goals and specifications, with a specific focus on the user testing and its findings, along with the manual and automated testing.

A. User Testing

1. User Testing Participants

User testing played a vital role in validating the project's functionality, educational content, and overall user experience. The testing phase involved three students who willingly participated in the evaluation process. These students were carefully selected to represent the target audience, primarily those with very little experience regarding automated testing. Their diverse levels of familiarity with programming made them ideal candidates for comprehensive feedback.

2. Feedback and Iterations

All three students who participated in the user testing displayed a positive inclination toward the project's core idea of using gamification for computer science education. They resonated with the concept of actively engaging with code to understand programming concepts, especially the integration of real-time code visualization.

However, the user testing process revealed some consistent issues that warranted immediate attention. Students expressed concerns regarding the visual elements of the game, specifically related to clarity and aesthetics. Additionally, they pointed out that certain instructions required further refinement to enhance the learning experience.

Notably, the project's iterative approach allowed for swift and responsive adjustments based on the feedback received during user testing. The feedback served as valuable guidance for enhancing the game's visuals and refining instructions. These iterative improvements were essential in ensuring a more user-friendly and informative experience.

3. Ethical Approval

It's worth highlighting that the process of conducting user testing with the involved students was carried out under the umbrella of ethical considerations. The user testing procedure was approved and covered under the ENGR489 Human Ethics Application, ensuring that all participants' rights and interests were protected throughout the evaluation process. Ethical considerations included obtaining informed consent from participants, maintaining the confidentiality of user data, and adhering to all ethical guidelines.

The combination of user testing insights and ethical practices not only shaped the project's refinement but also underscored our commitment to conducting research with a strong sense of responsibility and integrity. The valuable input received from the user testing phase, along with the subsequent iterative improvements, contributed significantly to the project's overall quality and alignment with the needs and expectations of its target audience.

B. Manual Testing

Thorough manual testing, conducted diligently by the project's creator, was integral to evaluating the functionality, usability, and educational content of the game-based learning platform. This hands-on approach encompassed various aspects, each contributing to a detailed assessment of the project's performance.

1. Functional Testing:

Objective: The primary goal of functional testing was to ensure that all aspects of the game, including the code editor, real-time visualization, and user interface elements, worked seamlessly and as intended.

Scope: Functional testing covered the entire user journey within the game. Test scenarios were thoughtfully designed to evaluate core functionality, such as code input, execution, and visualization of code behavior. Specific emphasis was placed on the accurate representation of coding concepts, including loops, conditions, and software testing practices.

Results: Thorough functional testing affirmed that the project delivered a robust and reliable user experience. It was carefully ensured that users could interact with the code editor, execute code, and observe the expected behavior. The game effectively conveyed computer science and software testing concepts through its interactive elements.

2. Usability Assessment:

Objective: Usability testing aimed to evaluate the intuitiveness, user-friendliness, and overall user experience provided by the project.

Scope: Rigorous usability assessments involved traversing the game as a user with varying levels of familiarity with computer science concepts. This meticulous exploration of the user experience involved attempting coding challenges, interacting with the user interface, and scrutinizing the educational content.

Results: Usability testing, conducted with meticulous attention, highlighted the project's strengths in providing an intuitive and user-friendly experience. It was thoughtfully ensured that users found it easy to navigate the game, access educational content, and perform tasks. This careful exploration also identified valuable insights for refining the user interface and further enhancing usability.

3. Content Quality:

Objective: Thorough manual testing also extended to an evaluation of the quality and effectiveness of the educational content within the game. This detailed evaluation focused on the relevance and accuracy of the content in conveying computer science and software testing concepts.

Scope: Through meticulous review and testing, the content within each level was thoroughly examined. This scrutiny involved assessing the content's alignment with the intended learning objectives, accuracy of code explanations, relevance to real-world software testing practices, and the educational value of the content.

Results: Content quality testing, conducted with thorough consideration, confirmed that the project diligently aimed to deliver on its educational promises. The content was carefully reviewed to ensure it effectively conveyed computer science and software testing concepts. Thorough testing of the content reinforced its educational value and accuracy.

The results of this thorough manual testing, conducted by the project's creator, provided a detailed assessment of the project's functionality, user experience, and educational content. It affirmed that the project engaged users effectively, provided a seamless learning experience, and accurately conveyed complex coding and testing concepts. This meticulous evaluation served as a solid foundation for the project's development and refinement.

C. Automated Testing

Automated testing played a pivotal role in ensuring the project's functionality and the quality of user manual testing,

with the goal of hopefully validating manual testing efforts and performance checks.

1. User Manual Testing Validation:

Objective: The central objective of automated testing was to potentially validate that users had rigorously conducted manual testing on their code logic, adhering to the project's educational objectives.

Implementation: Automated tests were strategically integrated into the game's levels, aiming to evaluate users' manual testing efforts. These tests were designed with the hope of assessing whether users had thoroughly tested their code to meet specific criteria defined for each level.

Results: Automated tests aimed to potentially provide immediate feedback to users, with the hope of affirming that they had conducted sufficient manual testing on their code logic. Successful completion of these tests was expected to demonstrate the user's commitment to exploring and testing their code comprehensively, reinforcing the educational value of the project.

2. Performance and Reliability Checks:

Objective: Automated testing was also employed with the aim of ensuring the performance and reliability of the game's core functionalities. This aimed to potentially identify potential issues, regressions, and performance bottlenecks.

Implementation: A suite of automated tests was implemented to potentially assess the performance of the game, including response times, resource utilization, and scalability. These tests aimed to confirm that the game performed reliably and met its intended performance benchmarks.

Results: Automated performance testing was intended to potentially validate that the game operated within the specified performance parameters. It was hoped that users would experience a smooth and responsive learning environment, potentially enhancing their overall experience.

3. Educational Feedback:

Objective: Automated testing was not limited to technical assessments. It was also designed with the hope of providing educational feedback to users based on their code's behavior.

Implementation: Automated tests aimed to assess whether the user's code accurately implemented the targeted educational concepts, such as loops, conditions, and software testing practices. Feedback was provided with the hope of guiding users in their learning journey.

Results: The integration of educational feedback through automated tests was intended to empower users to not only pass technical assessments but also reinforce their understanding of coding and software testing principles.

The results of automated testing, including the validation of user manual testing efforts and performance checks, collectively aimed to contribute to the project's ability to provide users with a robust, educational, and engaging learning experience. It was hoped that users would recognize their efforts in conducting manual testing on their code and reinforce their commitment to exploring and understanding coding and software testing concepts.

D. Future Enhancements and Recommendations

While the project has achieved its primary goals and garnered positive feedback from users, there is always room for further improvement and expansion. Here, we outline potential areas for future enhancements and provide recommendations to elevate the project's educational impact.

1. Expanding Content:

One avenue for future enhancement is the expansion of educational content within the project. This could involve the creation of additional levels that delve into more advanced computer science and software testing concepts. As learners progress through the game, they may benefit from encountering challenges that address more complex scenarios and scenarios.

Recommendation: Collaborating with educators and domain experts to identify advanced topics and challenges that align with curricular requirements. This expansion would enable the project to cater to a broader range of learners, from beginners to those seeking more advanced challenges.

2. Enhancing Interactivity:

To further engage users and foster collaboration among learners, enhancing the project's interactivity is a promising direction for development [7]. Introducing more interactive elements, such as collaborative challenges or multiplayer capabilities, could provide users with the opportunity to work together on coding challenges or problem-solving tasks.

Recommendation: Exploring the integration of real-time collaborative features, enabling users to collaborate on coding tasks, test solutions together, or engage in competitive challenges. This social aspect could enhance the project's educational impact by promoting teamwork and peer learning.

3. Advanced Analytics and Reporting:

Implementing advanced analytics and reporting features could provide educators and administrators with valuable insights into users' progress and performance. This data-driven approach could help tailor educational content to individual needs and measure the project's impact more effectively.

Recommendation: Incorporating features that track and analyze user performance, identifying areas where users may be struggling or excelling. Customized reports and analytics could guide educators in tailoring their teaching strategies and content.

4. Integration with Learning Management Systems (LMS):

To seamlessly fit into educational institutions and facilitate the integration of the project into formal curricula, considering compatibility with popular Learning Management Systems (LMS) is crucial. LMS integration could streamline the adoption of the project in educational settings.

Recommendation: Exploring partnerships or development efforts to ensure compatibility with widely used LMS platforms. This integration could simplify user management,

content distribution, and assessment.

5. Accessibility Features:

Ensuring accessibility for all users is paramount. To reach a wider audience, the inclusion of accessibility features, such as support for screen readers and adherence to accessibility standards, can make the project more inclusive and user-friendly.

Recommendation: Collaborating with accessibility experts to conduct audits and incorporate enhancements that ensure the project is usable by individuals with diverse needs, including those with disabilities.

These recommendations for future enhancements are intended to advance the project's educational impact, making it an even more valuable tool for learners, educators, and institutions. By expanding content, enhancing interactivity, utilizing advanced analytics, integrating with LMS, and prioritizing accessibility, the project can continue to evolve and serve as a dynamic and effective learning resource.

To wrap up this entire evaluation section, the incorporation of these performance metrics provided a well-rounded evaluation of the project's functionality and its alignment with its educational goals. This comprehensive assessment ensured that the project met its objectives while considering scalability and the delivery of educational content effectively.

VI. CONCLUSIONS AND FUTURE WORK

A. Conclusions

As we draw the curtains on this capstone project, it's fitting to reflect on the journey we undertook to create a game-based learning platform focused on teaching computer science and software testing concepts. The project has not only met key goals and specifications but has also laid the foundation for envisioning future work and opportunities that extend beyond its current scope.

1. Engagement and Practical Learning:

The project's central accomplishment lies in its ability to actively engage learners and provide a hands-on experience in software testing. The progressive learning approach guides users from fundamental coding concepts to the intricacies of testing, fostering an inquisitive mindset.

2. Effective Pedagogy:

The project seamlessly aligns with principles of educational psychology and game design, establishing itself as an effective pedagogical tool for teaching computer science concepts, particularly software testing. Its success in conveying complex concepts in an engaging manner is noteworthy.

3. User-Centered Design:

A key driving force behind the project's success has been feedback from fourth-year engineering students. Their input has been instrumental in shaping the game's difficulty progression, concept understandability, and overall intuitiveness.

B. Future Work

As we gaze into the future, the completion of this capstone project unveils a realm of opportunities for further innovation and growth in the field of educational technology and computer science instruction. The groundwork laid by this project offers a strong foundation upon which future work can flourish. Consider the following potential avenues for future projects and initiatives, each contributing to the enrichment of the educational landscape:

1. Expansion of Game Modules:

The project's success in imparting fundamental software testing concepts sets the stage for a compelling prospect – expanding the game's content with additional modules. Future projects could introduce modules that delve into a broader spectrum of software testing scenarios. This expansion might encompass specialized areas such as performance testing, security testing, and more. By broadening the scope of topics, the project can cater to a diverse audience with varying interests and career aspirations.

2. Integration with Real-World Tools:

To bridge the divide between theory and practice, future work could focus on the integration of the game with real-world software testing tools and platforms. This strategic integration would empower users to interact with industry-standard software testing frameworks and tools, offering a practical and authentic learning experience. It's an endeavor that aligns the project with the demands of the software testing industry and equips learners with relevant skills.

3. Adaptation for Diverse Audiences:

Ensuring inclusivity in education is paramount, and the project can further this goal by adapting the game to suit diverse audiences. This adaptation could involve tailoring the game to the needs of learners from different age groups and educational backgrounds. The objective is to make the game a versatile educational tool that serves a broad demographic. This versatility could lead to its adoption in a wide range of educational settings.

4. Research on Learning Outcomes:

Beyond the immediate success of the game, future work should include in-depth research on the learning outcomes of users. This research goes beyond assessing the game's effectiveness; it delves into how learners assimilate and apply the knowledge acquired through the game. The data derived from this research would be invaluable for educators and developers, providing insights to refine and optimize the game continually.

5. Assessment of Long-Term Impact:

Understanding the long-term impact of the game on learners' ability to apply software testing concepts in professional settings is a significant endeavor. This longitudinal assessment involves tracking how learners apply their knowledge in real-

world scenarios. It provides insights into the sustained impact of the educational experience, enabling developers to fine-tune the game to better align with the practical demands of the software testing industry.

6. Collaborative Projects:

Collaboration is the cornerstone of innovation. Future work may involve collaborative projects with other capstone initiatives or educational courses. Such collaborations can result in the development of complementary tools and resources that enhance the broader educational ecosystem. The project, when part of a larger network of educational resources, strengthens the collective impact on computer science education.

In summary, the culmination of this capstone project marks not an end but a new beginning. It invites exploration into a realm of exciting opportunities for future work, each with the potential to advance the field of educational technology and computer science instruction. By pursuing these avenues, a '489 student can embark on projects that expand horizons, drive innovation, and contribute to the ever-evolving world of computer science education.

REFERENCES

[1] Rasulov Inom Muyidinovich, . (2020). Advantage And Methodological Problems Of Teaching Computer Science In Modern Schools. The American Journal of Interdisciplinary Innovations and Research, 2(10), 13–16. https://doi.org/10.37547/tajiir/Volume02Issue10-03

[2] McInerney, C. (2010). Having Fun with Computer Programming and Games: Teacher and Student Experiences. In: Hromkovič, J., Královič, R., Vahrenhold, J. (eds) Teaching Fundamentals Concepts of Informatics. ISSEP 2010. Lecture Notes in Computer Science, vol 5941. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-11376-5_13

[3] S. Tobias, J. D. Fletcher, and A. P. Wind, “Game-Based Learning,” Handbook of Research on Educational Communications and Technology, pp. 485–503, May 2013, doi: https://doi.org/10.1007/978-1-4614-3185-5_38.

[4] M. E. Auer and Thrasyvoulos Tsiatsos, The Challenges of the Digital Transformation in Education : Proceedings of the 21st International Conference on Interactive Collaborative Learning (ICL2018) - Volume 1. Cham: Springer International Publishing, 2020. https://peer.asee.org/23791

[5] R. D. Craig and S. P. Jaskiel, Systematic Software Testing. Artech House, 2002. Accessed: Oct. 20, 2023. [Online]. Available: https://books.google.co.nz/books?hl=en&lr=&id=2_gbZYZcZXgC&oi=fnd&pg=PR19&dq=software+%22testing%22+critical+thinking&ots=sW7sIOaIH2&sig=aNy7zJg_l28euwgqRaV75nd4nfo&redir_esc=y#v=onepage&q&f=false

[6] D. Carrington, “Teaching software testing,” Jan. 1996, doi: https://doi.org/10.1145/299359.299369.

[7] Mäntylä, M.V., Smolander, K. (2016). Gamification of Software Testing - An MLR. In: Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S., Mikkonen, T. (eds) Product-Focused Software Process Improvement. PROFES 2016. Lecture Notes in Computer Science(), vol 10027. Springer, Cham. https://doi.org/10.1007/978-3-319-49094-6_46