UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

VINICIUS BOSA PETRIS

AVALIAÇÃO DO EFEITO DA INTEGRAÇÃO DE ATIVIDADES DE PROGRAMAÇÃO NO ENSINO DE TESTE DE SOFTWARE EM UM AMBIENTE COM ELEMENTOS DE JOGOS

CAMPO MOURÃO

2021

VINICIUS BOSA PETRIS

AVALIAÇÃO DO EFEITO DA INTEGRAÇÃO DE ATIVIDADES DE PROGRAMAÇÃO NO ENSINO DE TESTE DE SOFTWARE EM UM AMBIENTE COM ELEMENTOS DE JOGOS

Evaluation of the effect of integrating programming activities and software testing in a gamified learning tool

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Marco Aurélio Graciotto Silva

CAMPO MOURÃO

2021

VINICIUS BOSA PETRIS

AVALIAÇÃO DO EFEITO DA INTEGRAÇÃO DE ATIVIDADES DE PROGRAMAÇÃO NO ENSINO DE TESTE DE SOFTWARE EM UM AMBIENTE COM ELEMENTOS DE JOGOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 01/Dezembro/2021

Marco Aurélio Graciotto Silva
Doutorado
Universidade Tecnológica Federal do Paraná

Juliano Henrique Foleiss
Doutorado
Universidade Tecnológica Federal do Paraná

Ivanilton Polato
Doutorado
Universidade Tecnológica Federal do Paraná

CAMPO MOURÃO

2021

AGRADECIMENTOS

Agradeço primeiramente ao meu orientador, Marco Aurélio Graciotto Silva, por esses quase dois anos de orientação e dedicação ao me auxiliar no desenvolvimento desta pesquisa, aos professores Juliano Henrique Foleis e Ivanilton Polato por fazerem parte da banca e da minha jornada acadêmica, ao professor Rogério Aparecido Gonçalves por também ter sido importante nessa jornada e aos outros docentes do Departamento Acadêmico de Computação (DACOM).

Por fim, sou grato aos meus amigos, em especial ao Lucas Gabriel da Silva e ao Wesley Franco Ferreira e a toda minha família: minha mãe Cristina Bosa, ao meu pai Neucir Petris, a minha irmã Carolina Bosa e a minha namorada Laura Helena Figueira Brito, por todo apoio nesse período.

RESUMO

Contexto: O aprendizado de teste de software envolve uma dificuldade por parte dos estudantes devido à falta de material atualizado e uma atitude negativa com relação ao aprendizado. Uma solução plausível para incentivar os estudantes é abordar gamificação no aprendizado. Existem algumas ferramentas que abordam gamificação e ensino, dentre elas o Code Defenders, o qual aborda gamificação entrelaçada a uma jogabilidade intuitiva que permite aos jogadores produzir mutantes e desenvolver casos de testes.
Objetivo: O objetivo deste trabalho foi avaliar o efeito da integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos, através do Code Defenders, avaliando a qualidade da ferramenta e avaliando os códigos de programa e testes de unidades criados pelos estudantes. Duas questões de pesquisa foram tratadas. A primeira questão de pesquisa abordou se a qualidade do ambiente para ensino de teste de software integrado à programação é satisfatória. A segunda questão de pesquisa avaliou se a integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos permite um melhor aprendizado de teste de software.
Métodos: O primeiro passo foi definir a plataforma para responder as questões de pesquisa, escolhendo-se o Code Defenders. Para aprimorar a gamificação foram implementas duas modificações na plataforma. Após a implementação foi proposto aos estudantes do curso de Ciência da Computação que participaram da disciplina de Engenharia de Software a execução de um estudo de caso para implementação de códigos de programa e testes de unidade em Java. Por fim, os participantes responderam um questionário para avaliar o Code Defenders com base no método MEEGA+. Seus respectivos códigos de programa foram avaliados utilizando os testes de unidade de referência, e também seus testes foram avaliados utilizando medidas de cobertura de teste de software.
Resultados: Para a primeira questão de pesquisa, a qual refere-se à qualidade do Code Defender para o ensino de teste de software integrado a programação, ela foi avaliado como boa, sendo frequentemente o jogo considerado relevante para os interesses dos alunos. Na segunda questão de pesquisa, analisando as submissões dos códigos de programa e dos testes de unidade, a maioria dos estudantes conseguiu implementar com êxito os programas especificados e também conseguiu uma boa cobertura com seus testes.
Conclusão: Neste trabalho foi realizado um estudo com estudantes utilizando o Code Defenders e, foi possível observar que a ferramenta apresentou-se como alternativa para aprimorar o aprendizado de testes de unidade e a programação em Java.

Palavras-chaves: Educação em Computação, gamificação, teste de software, análise de mutantes, programação, Code Defenders

ABSTRACT

Context: Software testing learning is difficult for students due to lack of up-to-date material and a negative attitude towards learning. A plausible solution for encouraging students is to employ gamification techniques. There are tools that combine gamification and teaching, among them Code Defenders, which addresses gamification intertwined with intuitive gameplay that allows players to produce mutants and develop test cases.

Objective: The objective was to evaluate the effect of the integration of programming activities in the teaching of software testing in an environment with game elements, as provided by Code Defenders, evaluating the quality of the tool and evaluating the program code and unit tests created by the students. Two research questions were addressed. The first research question investigated if the quality of the environment for teaching software testing integrated into programming is satisfactory. The second research question was about evaluating whether the integration of programming activities when teaching software testing in an environment with game elements allows for better learning of software testing.

Methods: The first step was to choose the platform to answer the research questions, which is how Code Defenders was defined. Two modifications were implemented in the platform to improve gamification. Using the modified platform, we proposed to the Computer Science students who participated in the Software Engineering class to carry out a case study to implement program code and unit tests in Java. Finally, they answered a questionnaire to evaluate Code Defenders based on the MEEGA+ method. Their respective program code was evaluated using reference unit tests, and their test cases were evaluated using software test coverage measures.

Results: For the first research question, which refers to the quality of Code Defender for teaching software testing integrated into programming, it was evaluated as good, with the game being frequently considered relevant to the students' interests. In the second survey question, which considered program code and unit test submissions, most students successfully implemented the specified programs and achieved good coverage with their tests.

Conclusions: In this work, a study was carried out with students using Code Defenders. For this study, it was possible to observe that the tool presented itself as an alternative to improve the learning of unit tests and programming in Java.

Keywords: Computing education, Gamification, Software testing, Mutant analysis, Programming, Code Defenders

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Habilidades de programação e de teste de software são responsáveis pela formação de engenheiros de software que criam produtos de qualidade. No entanto, um dos desafios do ensino de teste de software e programação é ajudar os estudantes a desenvolver essas habilidades. Ambas são importantes e requerem prática e experiência para complementar a aprendizagem (CARRINGTON, 1997).

A programação de computadores é uma habilidade muito útil e pode ser uma carreira muito gratificante. Porém, o ensino e aprendizagem de programação é referido como uma das disciplinas que enfrentam grandes desafios devido ao abandono, desmotivação e retenção de alunos em níveis elevados no primeiro ano são problemas que devem ser resolvidos. Para resolver esse problema, os educadores introduzem no processo de aprendizagem ferramentas para apoiar e facilitar o aprendizado da programação (PITEIRA; COSTA, 2012).

No cenário da indústria de software, os profissionais que atuam no desenvolvimento de software interagem com os testes, tendo ou não um trabalho de 'teste'. Dessa forma, eles precisam adquirir habilidades de teste. Porém, os estudantes da computação formados possuem deficiência nessas habilidades (SCATALON et al., 2019).

Uma maneira para lidar com esse problema é abordá-lo desde o começo do curso, integrando-o amplamente ao currículo. Nesse contexto, destaca-se que o ensino de teste de software é um tópico importante para o ensino da programação: a partir dele os estudantes podem desenvolver suas habilidades de teste e melhorar suas habilidades de programação (SCATALON et al., 2019).

Para atrair os estudantes para o processo de aprendizado de teste de software são utilizados métodos de ensino no qual se ensina programação vinculada a testes. Da mesma forma, mais pesquisas no ensino do processo inicial de aprendizado a programação e especialmente conceitos básicos de teste podem ajudar a melhorar a habilidade de teste de software dos estudantes (SCATALON et al., 2019), promovendo a criação passo a passo da resolução de problemas, a reflexão sobre seu próprio processo de desenvolvimento e a inclusão de práticas de teste de software (SCATALON et al., 2019).

Incentivar os estudantes a estudar programação com testes envolve uma certa dificuldade por parte dos educadores, devido a dificuldade de manter um curso de testes atualizado e apresentar exercícios que são realistas (ANICHE et al., 2019). Estudos relatam a falta de testes adequados para os estudantes. Em particular, a principal dificuldade pode estar relacionada aos estudantes escreverem seus próprios casos de teste, pois para serem devidamente compreendidas e aplicadas, muitas ideias de teste requerem conhecimento prévio e habilidades de programação, que eles ainda estão adquirindo em cursos de programação. Além disso, os estudantes podem apresentar uma atitude negativa em relação ao aprendizado, mesmo sabendo de sua importância (SCATALON et al., 2019). Uma solução potencial é utilizar jogos ou uso de elementos de jogos (gamificação) no aprendizado.

A gamificação é uma técnica para utilizar design de jogos em contextos não-jogos, com o objetivo de reter pessoas e engajá-las a realizarem determinadas tarefas (DETERDING et al., 2011).

Ela permite transformar atividades que são de pouco interesse, por serem desinteressantes ou difíceis, em tarefas divertidas e competitivas, transformando em um desafio de um jogo (FRASER, 2017). A familiarização com o conceito de jogar contra o computador está relacionada com a sensação de conquista obtida quando as metas são atingidas, com o objetivo de satisfazer o jogador. A diversão é vista como uma ferramenta para acelerar a aprendizagem e reter o interesse constante para a realização dos estudos (BISHOP et al., 2015).

No contexto de gamificação quanto ao ensino de programação, o Pex4Fun apresenta como principal elemento de gamificação os duelos de codificação, no qual os estudantes recebem uma implementação do professor referentes a conceitos e tópicos estudados e as implementam. A partir dessa ferramenta, originou-se o Code Hunt, que possui a mesma característica de gamificação, porém o design de sua interface foi aprimorada contando com: animações, sons e frases com analogias a jogo de caça.

Quanto ao ensino de teste de software, o Testing Game alinha conteúdos teóricos e práticos sobre teste de software abordando três técnicas de teste: Funcional, Estrutural e Baseada em Defeitos. Cada uma das técnicas possuem um determinado número de fases, que o jogador deve completar para avançar no jogo.

E por último, a ferramenta no qual será o objeto de estudo do trabalho, o Code Defenders. A plataforma é um sistema gamificado para ensinar teste de software com base nos princípios de teste de mutação. O jogo foi apresentado em sete estudos, onde foram encontrados objetivos comuns de gamificação: aumentar o envolvimento, a motivação e o prazer de aprender o teste de mutação. Outros objetivos foram mencionados em um ou mais desses estudos: produzir testes mais fortes, melhorar as habilidades do aluno, melhorar a compreensão do aluno, promover a adoção entre desenvolvedores e profissionais e melhorar o desempenho do testador. Além disso foram listados os elementos de jogo presente no sistema: pontos, duelos e níveis (JESUS et al., 2018).

Esse trabalho tem como objetivo avaliar o efeito da integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos, explorando as seguintes questões de pesquisa:

- **QP1:** A qualidade do ambiente para ensino de teste de software integrado à programação é satisfatória?
- **QP2:** A integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos permite um melhor aprendizado de teste de software?

Para responder a primeira questão de pesquisa foi utilizado o método MEEGA+. Esse método foi desenvolvido especificamente para a avaliação de jogos educativos para o ensino da computação (PETRI et al., 2019). Ele estabelece um questionário como instrumento para esta avaliação, o qual foi aplicado no final do experimento para os estudantes.

Sendo assim, foram obtidos apenas 3 respostas dos estudantes que participaram do experimento, 4 não responderam, e aplicando a Teoria de Resposta ao Item (TRI) do método MEEGA+, o Code Defenders foi avaliado no nível de qualidade boa referente a qualidade do jogo, levando em

consideração sua gamificação e outros pontos importantes abordados no questionário. Essa nível tem como principal característica apresentar as vezes atividades desafiadoras e proporcionar sentimentos de confiança e satisfação aos jogadores. Frequentemente, o jogo é considerado relevante para os interesses dos estudantes.

Para responder a segunda questão de pesquisa foram avaliados o código do programa e os testes de unidade criados pelos estudantes do curso de Ciência da Computação que estavam matriculado na disciplina de Engenharia de Software. O estudo contemplou duas atividades no Code Defenders, uma síncrona no que foi proposto o desenvolvimento do *bubble sort* e outra assíncrona no qual foi proposto o desenvolvimento da estrutura de dados fila. Os códigos de programa foram analisados com base nos testes de unidade desenvolvidos no desenho experimental que foram utilizados como referência para o estudo. Analisando os testes de unidade através da cobertura de código, de desvios e de mutantes das ferramentas JaCoCo e Pit, o resultado apresentado foi satisfatório no que diz respeito a integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos.

No próximo capítulo (Capítulo 2), é apresentado o referencial teórico, abordando primeiramente as características de gamificação no ensino, seguindo das ferramentas gamificadas para o ensino de programação, através do Pex4Fun e Code Hunt, e para o ensino de teste de software, Testing Game e Code Defenders. Neste caso, o Code Defenders é tratado com mais detalhes, por ser o objeto de estudo do trabalho.

No Capítulo 3, primeiramente são apresentados as duas questões de pesquisas e o objetivo do presente trabalho, em seguida, é descrita a abordagem de implementação de novas funcionalidades para o Code Defenders, habilitando a ferramenta para a atividade de programação pela criação do programa a ser testado. Seguindo pelo capítulo, é especificado o desenho experimental do estudo, no qual apresenta como será realizado o experimento para obtenção e avaliação dos dados dos estudantes, conforme os instrumentos previamente definidos. Para finalizar são apresentados os dois métodos de pesquisa utilizados, o primeiro método tem como objetivo avaliar a qualidade dos elementos de jogos do Code Defenders, e o segundo método apresenta como será realizada a avaliação da qualidade dos teste de unidade.

Os resultados são apresentados no Capítulo 4, que consiste em apresentar como foi realizado o experimento, as primeiras conclusões sobre as experiências de programação e de desenvolvimento de teste dos estudantes antes de utilizar a ferramenta, a análise dos códigos e casos de teste desenvolvidos pelos estudantes considerando a cobertura do código e a evolução de casos de testes de unidades de alguns estudantes, que foram importantes para apresentar o comportamento deles durante o experimento, e a análise do Code Defenders como ferramenta de aprendizado utilizando o método MEEGA+.

Por fim, o Capítulo 5 apresenta a conclusão do trabalho, respondendo as questões de pesquisa através dos resultados que foram obtidos do experimento, limitações e trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Ensino integrado de programação e teste de software

Programar não é uma habilidade única, mas sim uma atividade cognitiva complexa, onde o estudante deve simultaneamente construir e aplicar várias habilidades cognitivas de ordem superior para resolver um problema particular, e ser capaz de criar um modelo mental de como os programas se relacionam com o sistema subjacente, ou criar um modelo claro de fluxo do programa. Muitas teorias foram apresentadas sobre por que aprender a programar é uma tarefa difícil, algumas são atribuídas a própria natureza de programação, outras são atribuídos aos aspectos dos estudantes, e outras estão associadas à metodologia de ensino utilizada (VIHAVAINEN et al., 2014).

A fim de classificar as abordagens do ensino de programação, Vihavainen et al. (2014) realizaram uma revisão sistemática sobre o tema e identificou 60 trabalhos e os classificou conforme suas atividades mais comuns:

- **Colaboração:** atividades que incentivam a colaboração dos alunos em salas de aula ou laboratórios.
- **Mudança de conteúdo:** partes do material de ensino foram alterados ou atualizados.
- **Contextualização:** atividades em que o conteúdo do curso e as atividades foram alinhados a um contexto específico, como jogos ou mídia.
- **CS0:** a criação de um curso preliminar que deveria ser feito antes do curso introdutório de programação; pode ser organizado apenas para, por exemplo, alunos com dificuldades.
- **Tema do jogo:** um componente com o tema do jogo foi introduzido no curso, por exemplo, um projeto com tema de jogo.
- **Esquema de classificação:** uma mudança no processo de avaliação da disciplina; a mudança mais comum foi aumentar a quantidade de pontos recompensados pelas atividades de programação, enquanto reduzia o peso da prova do curso.
- **Trabalho em grupo:** atividades com maior comprometimento com o trabalho em grupo, como aprendizagem em equipe e aprendizagem cooperativa.
- **Computação de mídia:** atividades que declaram explicitamente o uso de computação de mídia (por exemplo, o livro).
- **Apoio de pares:** apoio de pares na forma de pares, grupos, mentores ou tutores contratados.
- **Suporte:** um termo abrangente para todas as atividades de suporte, por exemplo, aumento de horas de professores, canais de suporte adicionais etc.

O teste de software é uma das ferramentas mais importantes para o controle de qualidade de software. Os testes podem ser usados como base para um processo de desenvolvimento ágil e iterativo, eles comunicam requisitos de forma inequívoca e evitam regressões que podem ocorrer

quando novos recursos são introduzidos ou quando um aplicativo é refatorado (KATS et al., 2011). Ao abordar o teste de software mais cedo no currículo de computação, os estudantes têm a oportunidade para desenvolver suas habilidades de teste ao longo do curso, além disso o conhecimento de teste pode ajudar os estudantes a melhorar suas habilidades de programação (SCATALON et al., 2019). Outros benefícios esperados são de promover a programação reflexiva em vez da programação de tentativa e erro, melhorando a compreensão dos conceitos de programação, acostumando-se às práticas e ferramentas da indústria e aumentando a confiança do estudante (LUXTON-REILLY et al., 2018).

Em um mapeamento sistemático, foram encontrados artigos que descrevem a introdução do desenvolvimento orientado a testes (TDD) para cursos introdutórios de programação, mas sem fornecer qualquer avaliação empírica. Segundo Luxton-Reilly et al. (2018), a ferramenta Web-CAT tem sido usada em várias instituições por vários anos para avaliar a qualidade de casos de teste escritos por estudantes em cursos introdutórios de programação e foram encontradas evidências satisfatórias em ensinar novatos a escrever caso de teste de altas qualidade, no uso de TDD para ensinar matrizes e para a utilidade de ensinar testes de mutação em um curso de programação com a linguagem Pascal.

Por outro lado, alguns artigos relatam que ensinar TDD na programação introdutória é um desafio adicional, porque adicionar um requisito de teste pode ser opressor para os novatos e pode ser percebido pelos estudantes como irrelevante ou inútil para o aprendizado de programação (LUXTON-REILLY et al., 2018).

Porém, existem vários métodos que visam ensinar programação com a integração de teste de software aos estudantes. Em um mapeamento sistemático sobre este tópico, Scatalon et al. (2019) identificaram 229 trabalhos, abordando o ensino integrado de teste de software quanto ao currículo (trabalhos sobre a integração de testes no currículo da computação), conteúdos (conteúdos sobre testes para o contexto de cursos introdutórios), tarefas de programação (instruções para realizar tarefas de programação que incluem práticas de teste), processo de programação (método de ensino a programação para iniciantes) e ferramentas (ferramentas de suporte). Os demais tópicos dizem respeito aos resultados da aprendizagem da integração de testes em cursos de programação: qualidade de programas (avaliação do código enviado pelos estudantes), percepções/comportamentos (atitudes dos alunos em relação ao teste de software) e compreensão de conceitos (avaliação do conhecimento dos alunos sobre programação e conceitos de teste).

2.2 Jogos e uso de elementos de jogos (gamificação) em educação

Os jogos são usados de muitas maneiras diferentes no ensino da ciência da computação. Os chamados jogos sérios são jogos que têm outros objetivos além do puro entretenimento e incluem todos os aspectos da educação: ensino, treinamento e informação. Jogos sérios podem ser uma forma motivadora de ensinar tópicos de ciência da computação. Um elemento fundamental nos jogos em

geral é um alto nível de interação entre os jogadores. Um jogador deve estar ativamente envolvido no jogo para ter sucesso nele (HAKULINEN, 2011).

Além dos jogos sérios, a gamificação aparece como outra forma para motivar e aumentar a atividade e retenção do usuário. A gamificação é a ideia de usar elementos de design de jogos em contextos não relacionados a jogos e está gerando um intenso número de aplicações com essa característica - que vão desde produtividade, finanças, saúde, educação, sustentabilidade, bem como mídia de notícias e entretenimento. Vários fornecedores agora oferecem gamificação como uma camada de serviço de software de sistemas de recompensa e reputação com pontos, emblemas, níveis e quadros de classificação. Esta implantação comercial de aplicativos gamificados para grandes públicos potencialmente promete novas e interessantes linhas de investigação e fontes de dados para a interação homem-computador (IHC) e estudos de jogos (DETERDING et al., 2011).

A gamificação tem sido amplamente utilizada em ambientes educacionais com o intuito de aumentar a motivação de aprendizagem dos estudantes, através da utilização de elementos de jogos em um ambiente educacional. Para determinar quais elementos podem ser utilizados não existe uma convenção de nomenclatura ou um processo para desenvolvê-los (TODA et al., 2019).

No que diz respeito às estruturas criadas para auxiliar na implementação de elementos de jogos em um determinado ambiente, existe a estrutura MDA que é dividida em 3 componentes: Mecânica, Dinâmica e Estética (HUNICKE et al., 2004).

A mecânica descreve os componentes particulares do jogo, no nível de representação de dados e algoritmos, a dinâmica descreve o comportamento em tempo de execução da mecânica atuando nas entradas dos jogadores e nas saídas e a estética descreve as respostas emocionais do jogador, quando ele interage com o sistema de jogo (HUNICKE et al., 2004).

Porém, essa estrutura não fornece ao usuário estratégias claras sobre como combinar os elementos de jogos adequadamente, além disso, por ser uma estrutura geral, falta instâncias aplicadas em ambientes educacionais (TODA et al., 2019). Isso dificulta a adoção de gamificação por desenvolvedores ou pesquisadores, apesar do interesse em utilizá-la, eles não têm tempo ou recursos para entender as diferenças e semelhanças a fim de decidir quais elementos usar, bem como quais são mais apropriados no contexto educacional.

A fim de apresentar soluções, Toda et al. (2019) desenvolveram uma taxonomia baseado na estrutura MDA, fornecendo detalhes sobre a seleção, descrição e uso de elementos de jogos para avaliar e projetar sistemas gamificados em ambientes de aprendizagem, além de propor recomendações sobre como organizar hierarquicamente esses elementos semanticamente, para serem usados por designers, professores e outras partes interessadas na educação. A taxonomia é dividida em cinco dimensões relacionadas aos estudantes e ao ambiente: desempenho, ecologia, social, pessoal e ficção. Para cada dimensão, definem-se alguns sinônimos, elementos de gamificação, exemplos de como cada elementos podem ser aplicado em um ambiente educacional e algumas vantagens e desvantagens quanto ao seu uso (TODA et al., 2019).

2.2.1 Desempenho

Estes são elementos relacionados à resposta do ambiente, que podem ser usados para fornecer um *feedback* para o estudante, tais como: reconhecimento, nível, progressão, ponto e estatística. Se o estudante não recebe nenhuma resposta da sua progressão no ambiente, ele pode se sentir desorientado, pois suas ações não têm nenhum tipo de *feedback* (TODA et al., 2019). Os elementos de gamificação desta dimensão são:

- **Reconhecimento:** representado por emblemas, medalhas, troféus e conquistas. É um tipo de *feedback* que elogia o conjunto específico de ações dos jogadores, por exemplo, concluir uma tarefa em um prazo pode levá-los a ganhar um troféu. O reconhecimento é um dos elementos mais utilizados em aplicações gamificadas.
- **Nível:** também conhecido como nível de habilidade. Fornece ao jogador novas vantagens à medida que avançam no ambiente, por exemplo, os estudantes ganham um nível após concluir um determinado número de tarefas, quanto mais níveis são avançados, mais acesso a tarefas eles possuem.
- **Progressão:** também conhecido como barras de progresso e etapas. Fornece orientação ao jogador sobre seu avanço no ambiente.
- **Ponto:** também conhecido como pontuações, pontos de experiência, pontos de habilidade, etc. É uma maneira simples de fornecer *feedback* as ações dos jogadores. Ponto é o conceito mais básico encontrado em quase todos os aplicativos gamificados.
- **Estatísticas:** também conhecido como informações e dados. Está relacionado a informação visual fornecida pelo ambiente, por exemplo, quantas tarefas eles concluíram ou estatísticas gerais sobre o ambiente.

Em relação aos elementos de jogos apresentados, a falta do conceito reconhecimento pode levar ao jogador um estado de frustração, pois suas interações não estão sendo reconhecidas como algo importante. Quanto ao nível e a progressão, são considerados elementos relevantes, pois a falta de níveis pode levar o estudante a pensar que não avançou em suas habilidades e a falta de progressão um sentimento de frustração e ansiedade. Por fim, as estatísticas são apresentadas em quase todos ambientes educacionais, e a ausência de informação faz com que o estudante se sinta desorientado (TODA et al., 2019).

2.2.2 Ecologia

Esse contexto está relacionado ao ambiente em que a gamificação está sendo implementada. Seus elementos são: chance, escolha imposta, economia, raridade e pressão do tempo. A falta deles fazem com que o ambiente pareça monótono, pois não possui elementos que produzam interações com os jogadores (TODA et al., 2019).

- **Chance:** caracterizada por aleatoriedade, sorte ou probabilidade. O conceito está relacionado a aleatoriedade de um determinado evento ou resultado, por exemplo, o estudante pode obter um número de pontos após concluir uma tarefa.
- **Escolha imposta:** também conhecida como escolha, julgamento e caminhos. O conceito se aplica quando o jogador precisa tomar uma decisão para o avanço no ambiente de jogo, um exemplo, apresentar ao estudante dois conteúdos diferentes e faça com que ele escolha um ou outro.
- **Economia:** também conhecida como transações, mercado, câmbio. Esse conceito é relacionado a qualquer transação que possa ocorrer no ambiente, por exemplo, troca de pontos por vantagem por conteúdo dentro do ambiente.
- **Raridade:** também conhecido como itens limitados, coleção, exclusivos. Esse conceito está relacionado a recursos limitados no ambiente que podem estimular os jogadores através de um objetivo específico.
- **Pressão do tempo:** representado por cronômetros de contagem ou relógios. Esse conceito utiliza o tempo para pressionar as ações dos estudantes no ambiente, porém é considerado um dos elementos mais irrelevantes devido ao fato de desmotivar o jogador.

A dimensão ecológica está relacionada a conceitos que atuam como propriedades do ambiente que podem ser implementadas de maneira sutil para envolver os jogadores a seguir um comportamento desejados. Porém, a maioria dos elementos devem ser implementadas com cuidado, pois podem afetar drasticamente as interações dos estudantes. O primeiro elemento de jogo, chance, é diretamente afetado pela sorte dos jogadores. Já a economia pode ser atraente a eles se estiver relacionada à compra de vantagens. Em um ambiente em que ocorre a falta do elemento escolha imposta, pode levar o estudante a sentir que suas ações não são significantes, ao mesmo tempo que a liberdade excessiva pode permitir que eles realizem ações indesejadas. Quanto à raridade, sendo por adição de eventos limitados que recompensam com determinado item exclusivo, pode envolver os estudantes, mas a presença de recursos raros e suas restrições pode desmotivar outros. Por último, a pressão do tempo: em sua ausência pode levar o estudante a se sentir entediado no ambiente, pois eles podem não se sentir desafiados ou pressionados a concluir uma tarefa. Um exemplo de utilizar a pressão do tempo de maneira saudável é fornecer flexibilidade (TODA et al., 2019).

2.2.3 Social

Essa dimensão está relacionada às interações entre os estudantes apresentadas no ambiente. Dentre os elementos estão: competição, cooperação, reputação, pressão social. A falta de elementos sociais pode isolar os estudantes, uma vez que eles não consigam interagir com os outros estudantes (TODA et al., 2019).

- **Competição:** caracterizada por placares, jogador disputando contra outro jogador, quadro de líderes, etc. É um conceito vinculado a um desafio do jogador enfrentar outro para atingir um objetivo comum, por exemplo, usando placares com base na pontuação, níveis, conquistas.
- **Cooperação:** também conhecida como trabalho em equipe, grupos, etc. É um conceito que os jogadores devem colaborar para alcançar um determinado objetivo em comum. Exemplos de cooperação são tarefas nas quais grupos interagem uns com os outros e são reconhecidos por essas interações.
- **Reputação:** também conhecida como classificação, status. Está relacionada aos títulos que o jogador pode ganhar e acumular-se no ambiente. Representam um status social, porém não reflete necessariamente nas habilidades do jogador.
- **Pressão social:** também conhecido como pressão dos colegas ou missões do time. O conceito está relacionado as interações sociais que exercem pressão sobre o jogador

Os elementos de jogos nessa dimensão são responsáveis por conectar os estudantes e influenciar seu comportamento em relação a uma tarefa. O conceito de concorrência pode criar um ambiente em que os estudantes tentam superar seus colegas para obter um certo prêmio, porém, tem um enorme potencial para desmotivá-lo quando seu desempenho não é o esperado. Já a cooperação é vista como uma adição positiva na maioria dos ambientes educacionais, embora sua implementação possa ser complexa. A ausência da mesma pode levar ao isolamento, causando a desmotivação ou desengajamento do estudante, porém, quando utilizada, ela pode levar os estudantes a compartilhar conhecimento e trabalhar mais para evitar comprometer sua equipe. A reputação está relacionada ao status social que os estudantes podem adquirir no ambiente, por exemplo, o melhor estudante do curso. A falta dela é semelhante a falta de reconhecimento, ou seja, o estudante pode sentir que suas ações não são importantes dentro do ambiente. Por último, a pressão social é geralmente considerada como um dos elementos mais irrelevantes, mas pode ser útil se utilizada corretamente, por exemplo, para persuadir um estudante com alta pontuação a incentivar um colega com desempenho ruim (TODA et al., 2019).

2.2.4 Pessoal

Está relacionada aos estudantes que estão usando o ambiente, tais como: sensação, objetivo, quebra-cabeça, novidade e renovação. A falta desses elementos pode fazer com que o estudante se sinta desmotivado, pois o sistema não fornece um objetivo concreto a ele (TODA et al., 2019).

- **Novidade:** também conhecida como atualização, surpresa, mudanças, etc. Esse conceito está relacionado a atualização no ambiente, com o objetivo de adicionar novos conteúdos, informações ou até mesmo novos elementos de jogo. É responsável por manter os jogadores dentro do ambiente afim de evitar estagnação.

- **Objetivo:** representados por missões, marcos, etc. Está relacionado a fornecer ao jogador um objetivo para executar as tarefas, por exemplo, obter uma determinada pontuação em uma tarefa concluída.
- **Quebra-cabeça:** representados por desafios, quebra-cabeça reais, tarefas cognitivas. Conceito relacionado às atividades de aprendizagem implementadas no ambiente, pois o foco é fornecer um desafio para o jogador. Este conceito está presente em todos os ambientes educacionais, através de questionários ou desafios.
- **Renovação:** também conhecido como reforços, vida extra, etc. Esse conceito está relacionado com a escolha de refazer uma tarefa, devido a um erro cometido pelo jogador ou pela própria escolha, é uma das propriedades que torna o jogo divertido.
- **Sensação:** representada por estimulação visual ou sonora. Está relacionada ao uso de sentidos dos jogadores para melhorar a experiência, pode ser feita através de interfaces dinâmicas, realidade virtual ou realidade aumentada.

Entre as dimensões apresentadas, está é a mais comum em jogos e ambientes gamificados, pois ela está diretamente relacionada ao estudante que usa o ambiente. Em relação a cada elemento, o objetivo é o mais importante a ser usado em ambientes educacionais gamificados e está em todos eles, já que o foco principal é fazer com que o estudante aprenda ou pratique um conceito, embora seja cauteloso para não incentivar comportamentos indesejados. A falta de objetivo pode desorientar e confundir o estudante. Em um cenário onde ambientes estáticos podem comprometer o processo de aprendizagem, a novidade pode ajudar nesse processo, porém é difícil de ser implementado. Quanto ao quebra-cabeça é representado por desafios e tarefas cognitivas que são comuns em qualquer ambiente educacional, sendo que a falta dele pode desmotivar o estudante. A renovação é um conceito presente em quase todos os ambientes educacionais, já que os estudantes podem refazer uma tarefa que cometeram erros ou apenas por querer lembrar de um conceito. Embora a renovação não seja sempre apresentada como um elemento de gamificação, a falta dela geralmente torna o aprendizado mais difícil, o que pode desmotivar os estudantes. Finalmente, a sensação é geralmente apresentada como uma interface agradável e atraente para os estudantes (TODA et al., 2019).

2.2.5 Ficção

É a dimensão mista que está relacionada aos jogadores (através da narrativa) e ao ambiente (através do *Storytelling*), vinculando sua experiência ao contexto. A falta de ambos elementos causa a perda de imersão dentro do ambiente de jogo (TODA et al., 2019).

- **Narrativa:** também conhecido como decisões implícitas. Essa experiência é influenciada por escolhas implícitas feitas pelos jogadores, por exemplo, ao interagir com outros jogadores, retribuir com um sinal de agradecimento.

- ***Storytelling*:** pode ser vista como áudio, caixa de textos, etc. É assim que a história do ambiente é apresentada ao jogador, através de textos ou voz. É altamente usada como ferramenta para apoiar a narrativa dentro de um ambiente.

O elemento narrativa está relacionada à interação do estudante com o sistema. Quando projetada corretamente, pode ajudar o estudante a se concentrar no conteúdo, e não nos elementos do jogo ao seu redor. Porém, na sua ausência, pode prejudicar o envolvimento dos estudantes ao conteúdo a ser aprendido. Já a *storytelling* é uma maneira de materializar a narrativa, usando técnicas com o auxílio de texto e áudio, estabilizando como a história é contada. A falta da mesma pode levar a confusão de contexto, fazendo com que os estudantes não vejam uma razão para executar determinada tarefa do ponto de vista da gamificação (TODA et al., 2019).

2.3 Jogos para ensino de programação

Os estudantes costumam enfrentar dificuldades em cursos de ciência da computação, principalmente com a parte da programação. Esses desafios são caracterizados por cursos entediantes e difíceis de entender. Além disso, consideram os métodos de ensino utilizados como desinteressantes e os exercícios fornecidos não são o suficiente para praticar os conceitos aprendidos. Uma proposta para aliviar esses problemas é a incorporação de jogos educativos ou jogos de propósito em cursos de programação de computadores (MONCLAR et al., 2018).

O ponto crítico dos jogos com propósito de aprendizagem de programação é a relação entre o jogo e o conteúdo educacional, ou seja, se o jogo é atraente, divertido e encoraja o jogador a progredir, então ele vai aprender as habilidades do jogo e vai absorver uma grande quantidade informação. A maioria dos jogos analisados inclui um cenário que visa cobrir uma unidade específica da área de programação, enquanto alguns jogos (em menor número) cobrem múltiplos objetivos de aprendizado e unidades teóricas (MONCLAR et al., 2018).

Foi realizada uma análise em banco de dados acadêmicos, como Web of Science, Scopus, CiteSeer e Google Scholar e buscas na AppStore, Steam e Google Play por termos "jogos"e "programação"e foram encontrados 26 jogos com o propósito de ensinar programação. Eles foram classificados em duas categorias: jogos para crianças e adolescentes e jogos para universitários. A primeira possui um foco introdutório para o aprendizado da programação. Já a segunda, apresenta jogos com um teor ou linguagem mais desenvolvida para os estudantes de nível superior (MONCLAR et al., 2018).

Os jogos para crianças e adolescentes possuem interfaces mais infantis e ilustradas, na maioria das vezes lidando com linguagem de bloco, e histórias menos elaboradas. As abordagens encontradas na maioria deles tem como foco o desenvolvimento do pensamento computacional para resolução de problemas. Já os jogos para universitários possuem foco maior em sintaxe e linguagem de *script*, mesmo aqueles mais introdutórios. Todos tentam envolver os estudantes em universos fictícios para que sejam capazes de compreender fundamento de programação (MONCLAR et al., 2018).

De maneira geral, a maior parte dos jogos educativos encontrados não se preocupam com o ensino da linguagem, mas sim, com o apoio ao aprendizado. Isso mostra que é desejável o acompanhamento por um professor com conhecimentos da linguagem de programação. Mesmo assim, os jogos apresentam soluções válidas para aumentar o desempenho de quem quer aprender a programar. No entanto, eles se concentram em conteúdos específicos de aprendizado e suportam atividades limitadas de programação, além da maioria dos jogos rodarem localmente, o que neutraliza o componente de engajamento social (MONCLAR et al., 2018).

Vale ressaltar que as abordagens atuais não contemplam suficientemente teste de software e que nenhum jogo encontrado para ensino de programação também tinha como foco ou como exercício o aprendizado de testes.

2.4 Jogos para ensino de teste de software

2.4.1 Code Defenders

O Code Defenders é uma aplicação *web* que implementa uma abordagem de gamificação para teste de mutação, utilizando o *framekwork* JUnit para a criação de testes automatizados na linguagem de programação Java. O jogo estimula a criação de conjuntos de casos de teste de qualidade (que distinguem mutantes do programa correto) e de mutantes difíceis de serem distinguidos do programa correto (requisito de teste) ou funcionalmente equivalentes (mutante equivalente). Para isso, os jogadores podem assumir um dos seguintes papéis: atacante, que pretende criar mutantes, que são modificações sutis do programa em teste. E defensor, que escreve testes que revelam essas modificações e matam esses mutantes. Ambas as funções incentivam e treinam o entendimento dos *bugs* de software e como eles são detectados pelos testes (ROJAS; FRASER, 2016c).

O teste de mutação é uma técnica de engenharia de software que visa orientar o desenvolvimento do código de teste e avaliar sua qualidade. Dado um programa em teste, o teste de mutação envolve a criação automatizada de um conjunto de variantes do programa, chamado mutantes, que diferem do original por pequenas mudanças sintáticas (ROJAS; FRASER, 2016c). A premissa do programador competente apresenta que o programador é bom e, se comete erros, são erros simples (pequenas mudanças sintáticas ou semânticas). E a premissa do efeito de acoplamento está relacionada ao fato de erros simples serem responsáveis por falhas do software. Na prática, os desenvolvedores de software tendem a escrever programas quase corretos, porém quando ocorrem pequenas mudanças sintáticas no programa, elas podem ser representadas por falhas complexas (ROJAS; FRASER, 2016c). O conjunto resultante de mutantes pode ser usado para avaliar a qualidade dos testes existentes, medindo a quantidade de mutantes detectados. Um teste mata um mutante quando o resultado de sua execução no programa original é diferente do resultado de sua mutação. Mutantes que não são mortos podem orientar na geração de testes. Dado um cenário em que um mutante seja semanticamente equivalente ao programa original, não existe teste que possa diferenciá-lo do programa original, estabelecendo-se assim como um mutante equivalente. Identificar um mutante equivalente é o melhor

que decidir se dois programas são iguais. De modo geral, sabemos que esse problema é indecidível. Logo, a detecção de mutantes equivalentes geralmente fica a cargo do desenvolvedor ou de heurísticas capazes de detectar alguns casos de equivalência (ROJAS; FRASER, 2016c).

Para criar um novo jogo, o primeiro recurso disponível para a escolha do jogador é especificar uma classe de teste, escolhendo-o em uma lista predefinida de exemplo ou fazendo o envio de uma classe diferente. O segundo recurso é o modo de jogo: duelo (um atacante vs um defensor) ou batalha (time de atacantes vs time de defensores). Vale ressaltar que, no caso de times, os membros de cada time têm visibilidade de tudo que seus membros criaram (casos de teste ou mutantes). Por fim, são configurados o número total de rodadas e o nível de dificuldade, fácil ou difícil (ROJAS et al., 2017). No nível fácil, os mutantes são totalmente mostrados aos defensores, enquanto que, no nível difícil, existe o ocultamento de informações: atacantes não tem acesso aos casos de teste (mas tem acesso aos dados de cobertura do programa sob teste) e defensores não tem acesso à alteração do código feita no mutante, mas da linha em que está localizada a alteração no programa sob teste (ROJAS; FRASER, 2016b).

Iniciado o jogo, apresenta-se uma visão correspondente ao papel e nível de dificuldade escolhidos pelo jogador. As Figuras 2.1 e 2.2 são as visões para o atacante nos níveis fácil e difícil, respectivamente. O atacante visualiza o código da classe Java em teste, com destaque das cores para indicar a cobertura do conjunto de teste dos defensores e assim editar o código com o objetivo de criar mutantes difíceis de serem mortos. Além disso, é possível ver uma lista dos mutantes que ainda estão vivos, os que foram mortos e os considerados equivalentes. O código possui ícones para rotular os locais e o status dos mutantes atacantes (vivos, mortos ou equivalentes). Se o modo de jogo estiver no fácil, conforme mostra a Figura 2.1, o atacante consegue visualizar aos casos de testes dos defensores. Já no modo difícil, Figura 2.2, isso não acontece.

O defensor visualiza o código da classe sob teste à esquerda com destaque das cores para indicar a cobertura do conjunto de teste que foram criados, e os ícones para rotular os locais e o status dos mutantes atacantes (vivos, mortos ou equivalentes). Ele cria casos de teste para matar os mutantes à direita, e na parte inferior ele visualiza os casos de testes criados e os mutantes vivos, mortos e equivalentes. No modo de jogo fácil, conforme mostra a Figura 2.3 o defensor tem acesso à alteração do código feita pelo mutante, já no modo difícil, a alteração não é visível, Figura 2.4 (FRASER et al., 2019).

O jogo é realizado em rodadas de ataque e defesa, em que o atacante começa o turno enviando um novo mutante para o defensor. Então o turno é passado ao defensor, que reage com um novo teste, fechando a rodada. O jogo termina quando o número de rodadas escolhido for alcançado. Após a conclusão de cada rodada, uma análise de mutação é realizada para determinar se o teste criado matou algum mutante. Porém, quando o defensor suspeita que um mutante seja equivalente, ele pode requisitar um duelo de equivalência, desafiando o atacante a aceitar que o mutante é equivalente ou para produzir um teste para provar que não são equivalentes (ROJAS; FRASER, 2016c).

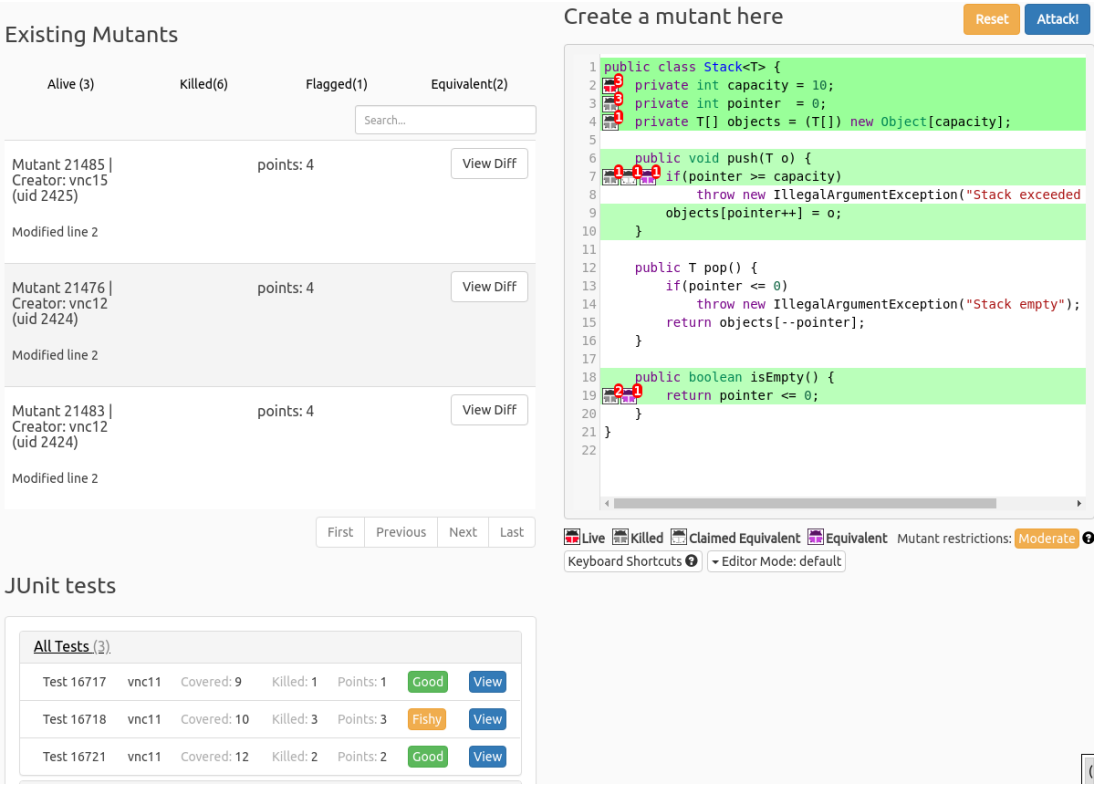

Figura 2.1. Visão do atacante no modo de jogo fácil em Code Defenders.

Fonte: Página do Code Defenders - code-defenders.org.

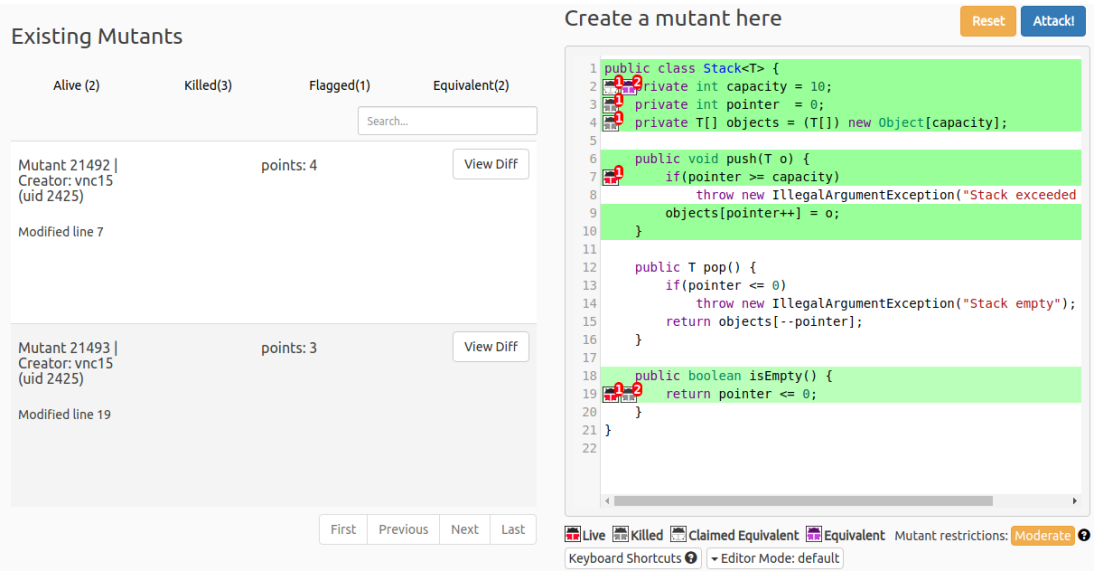

Figura 2.2. Visão do atacante no modo de jogo difícil em Code Defenders.

Fonte: Página do Code Defenders - code-defenders.org.

O Code Defenders baseia-se na competitividade para envolver os jogadores. Através de um sistema de pontuação que pode ser visto a qualquer momento do jogo, conforme apresentado na Figura 2.5, incentiva-se a produção de mutantes e de conjunto de casos de teste (CLEGG et al., 2017).

De modo geral, os atacantes vencem se produzem mutantes que sobrevivem a teste dos defensores por mais tempo. Os defensores vencem se produzem casos de teste que matam a maioria

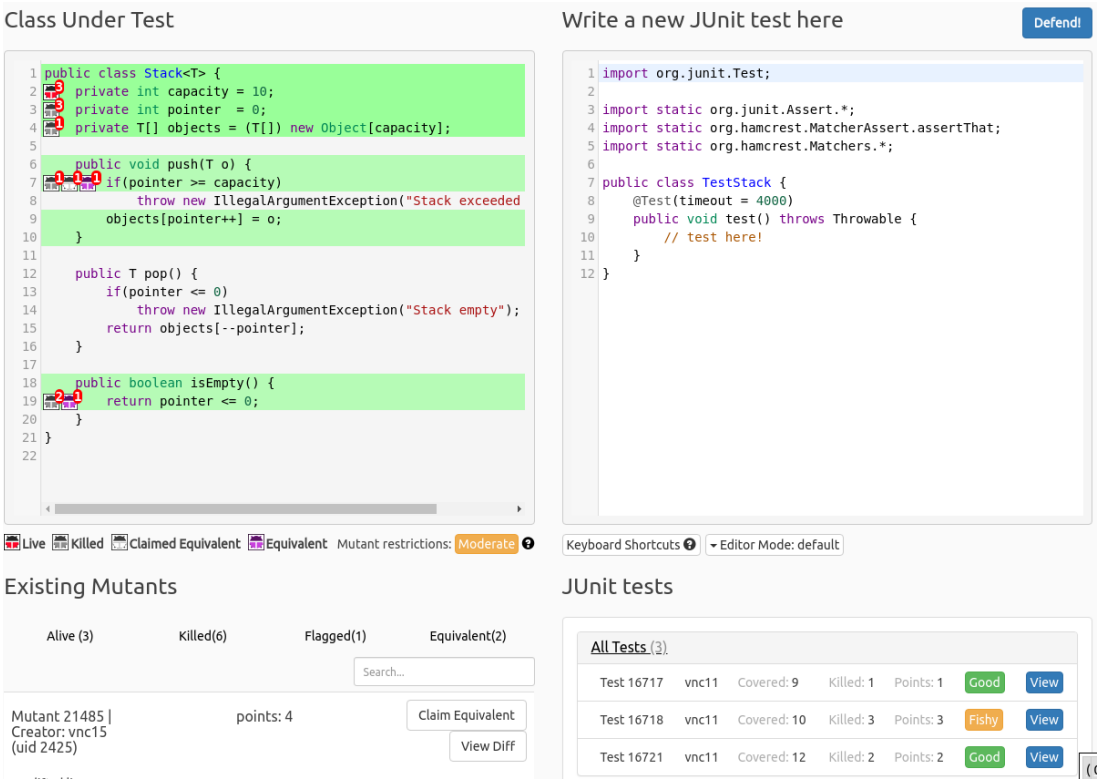

Figura 2.3. Visão do defensor no modo de jogo fácil em Code Defenders.
Fonte: Página do Code Defenders - code-defenders.org.

dos mutantes (ROJAS; FRASER, 2016a). A seguir são apresentadas as regras que pormenorizam as pontuações atribuídas aos jogadores:

- Após a criação de um novo caso de teste, o defensor ganha a quantidade de pontos acumulados daquele mutante (que não tinha sido morto por nenhum caso de teste até então).
- Após a criação de um novo mutante, se ele sobreviver, o atacante recebe um ponto para cada caso de teste existente para o qual o mutante sobreviveu.
- Quando o defensor afirma que um mutante é equivalente ao programa original:
  - Se o atacante é capaz de enviar um caso de teste que mata o mutante, ele recebe um ponto;
  - Se o atacante aceitar que o mutante é equivalente, os pontos atribuídos ao defensor e ao atacante depende do nível de dificuldade do jogo.
    - No nível fácil, o atacante mantém os pontos acumulados para esse mutante e o defensor ganha um ponto.
    - No nível difícil, o atacante perde todos os pontos acumulado para esse mutante e o defensor recebe dois pontos.

O Code Defenders foi avaliado quanto à integração em disciplinas de teste de software, observando-se efeitos positivos no engajamento e desempenho (FRASER et al., 2018; FRASER et al., 2019). Como o foco residia em teste de software, foram considerados exemplos de programas obtidos de softwares livres bem estabelecidos, buscando-se escalonamento de dificuldade e alinhamento com os objetivos da disciplina durante o semestre. Cabe destacar que os estudantes foram divididos em

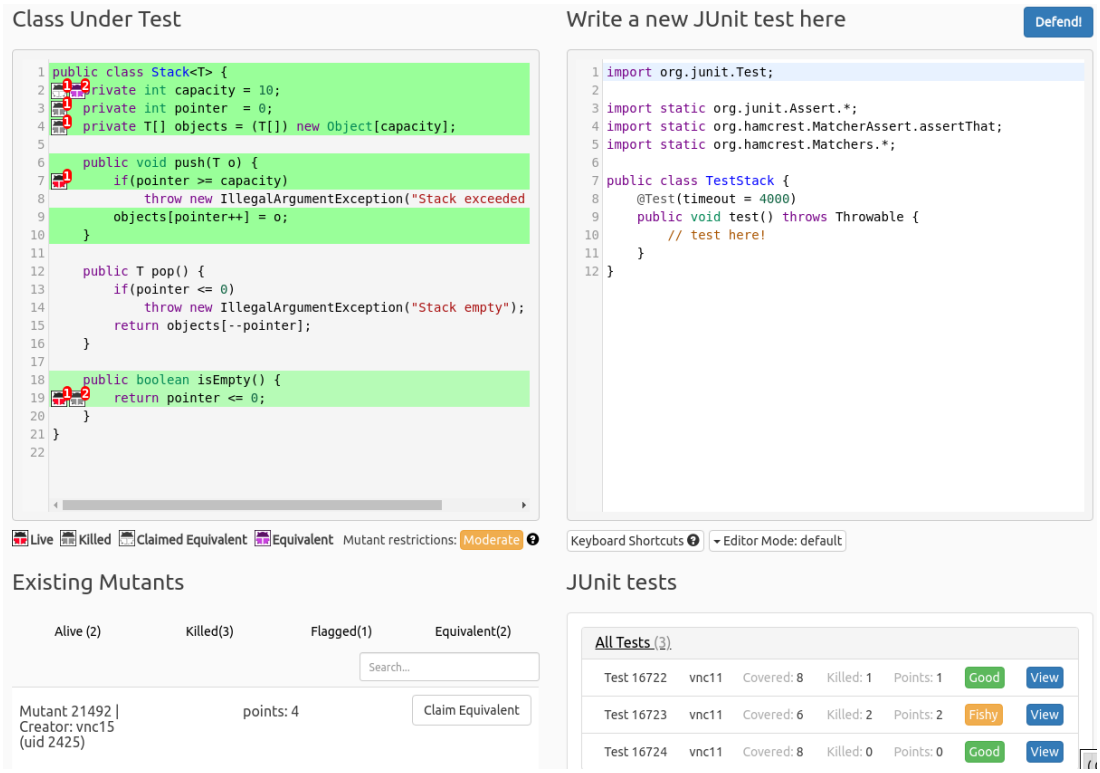

Figura 2.4. Visão do defensor no modo de jogo difícil em Code Defenders.

Fonte: Página do Code Defenders - code-defenders.org.

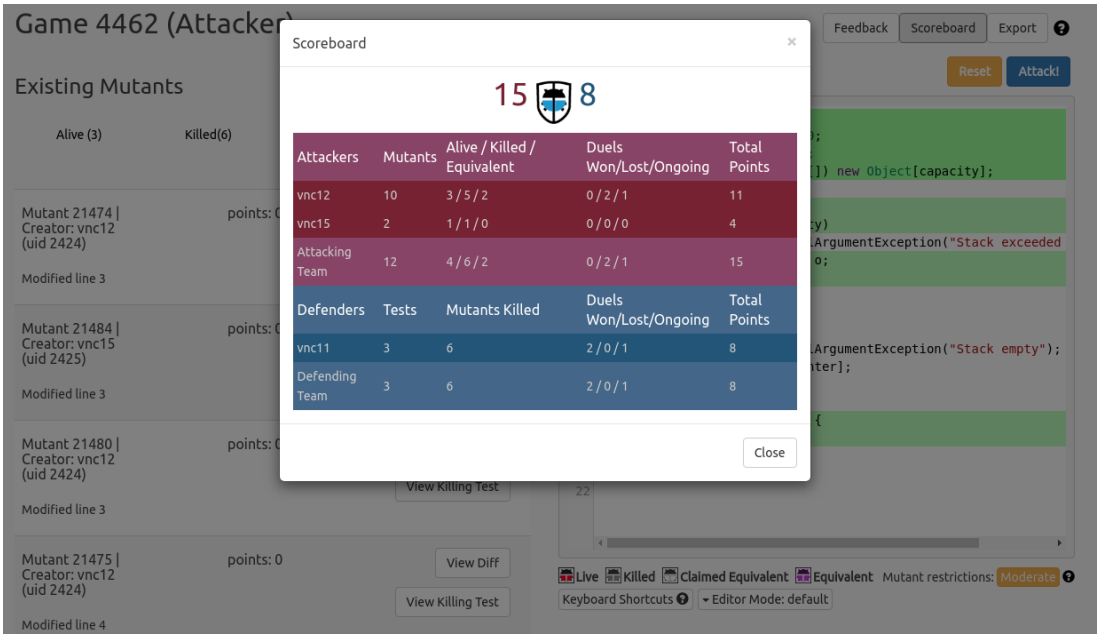

Figura 2.5. Visão do placar do jogo.

Fonte: Página do Code Defenders (https://code-defenders.org).

equipes com 3 cada, classificando-os pela pontuação semelhantes nos jogos anteriores, portanto, com habilidades semelhantes. A ideia de separar os jogadores em equipe foi constatado devido a boa dinâmica de jogo, pois em equipes menores, os mutantes e os teste demoravam mais tempo para serem criados, e em equipes maiores, havia jogadores que não acompanhavam os outros colegas

de equipe (FRASER et al., 2019). Além disso, foi implementado uma nova interface para ajudar os educadores a lidar com problemas relacionados a sala de aula, conforme a Figura 2.6 que apresenta uma visão geral dos jogos em execução, resumindo o estado do jogo. Neste exemplo, há um atacante (em vermelho) e um defensor (em azul). Para cada jogador, a visão geral mostra o número de pontos e hora da última ação, para ajudar a identificar estudantes com dificuldades (FRASER et al., 2018). Durante o jogo, os estudantes visualizavam o desempenho representados pelo mecanismo de pontuação em uma tabela, sendo assim, comparavam com os outros estudantes, estimulando-os a competição, além de desenvolver novas estratégias de jogo (FRASER et al., 2018). Ao final do jogo, os estudantes podiam observar todos os testes e mutantes criados, permitindo assim, refletir e debater entre o grupo sobre o resultado (FRASER et al., 2019).

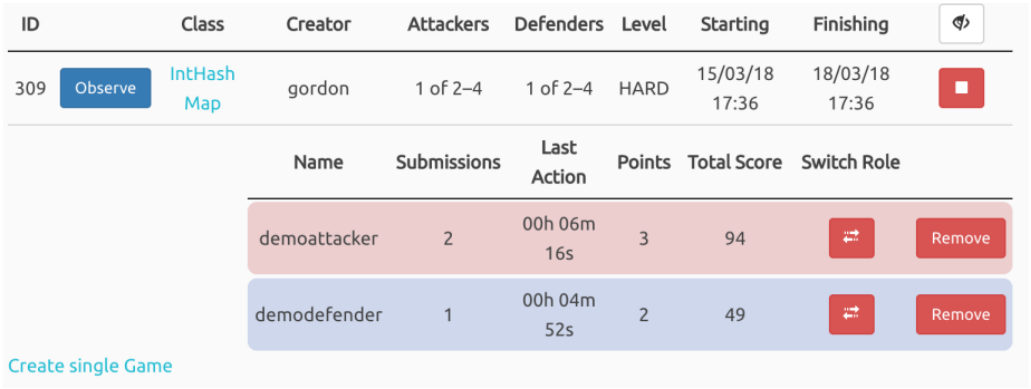

Figura 2.6. Visão do administrador referentes aos jogos em execução

Fonte: *A Preliminary Report on Gamifying a Software Testing Course with the Code Defenders Testing Game.*

Apesar de que a maioria dos testes que os estudantes fizeram foram válidos, foi encontrada uma quantidade de erros nas submissões devido a erros de compilação tanto de casos de teste quanto de alterações no programa, a casos de teste que falhavam quanto ao programa original e mutantes ou casos de teste duplicados. E outra limitação foi quanto ao duelo de equivalência, que não tem muitos instrumentos para evidenciar que tentou ser resolvido pela criação de casos de teste especificamente para o duelo e não penalizava a indicação de duelos sem fundamentos (FRASER et al., 2019). No geral, os estudantes se envolveram ativamente e gostaram de jogar, enquanto ao mesmo tempo melhoraram suas habilidades de teste (FRASER et al., 2019).

Depois de várias tentativas de sessões individuais usando o Code Defenders em vários cursos de engenharia de software (FRASER et al., 2020), foi realizado um curso opcional de teste de software aberto a estudantes da graduação e do mestrado. O curso consistia em aulas semanais de duas horas sobre teste de software de nível básico padronizado do *International Software Testing Qualification Board* (ISTQB), além de sessões semanais de exercícios apresentando as tecnologias que implementam os conceitos teóricos, incluindo JUnit. Em sessões práticas semanais de duas horas ao longo de todo o semestre, os estudantes jogaram partidas de Code Defenders. As sessões práticas consistiram na participação de cada estudante em um jogo por semana, alternando as funções de atacante e defensor

todas as semanas. No total, cada estudantes participou de 12 sessões de Code Defenders, e foram avaliados a qualidade de seus mutantes e testes ao longo do semestre (FRASER et al., 2020).

Após realizado a análise do resultado da aplicação do Code Defenders no curso, foram realizadas algumas melhorias com o objetivo de reter o engajamento dos estudantes. Foi observado que os estudantes participaram de várias sessões de jogo é que, com o tempo, alguns deles tendem a minimizar seus esforços em vez de se envolverem totalmente com os objetivos de aprendizagem. Para combater esse problema, foram adicionados vários recursos ao jogo (FRASER et al., 2020). Isso inclui algumas melhorias gerais para a experiência do usuário (por exemplo, preenchimento automático para o editor de código ou a possibilidade de exportar jogos para um formato adequado para IDEs gerais), mas também alguns recursos que visam especificamente diferentes funções do jogador. Para engajar ainda mais os defensores, foram adicionados os seguintes recursos:

- Uma vez que os estudantes têm mais experiência em escrever testes JUnit, eles tendem a enviar mais testes para o jogo. A interface do usuário do Code Defenders provou ser problemática uma vez que o número de testes tornou-se muito grande, pois se tornou um desafio navegar entre os testes existentes. Para evitar esse problema, os testes foram organizados em termos dos métodos que eles cobrem, permitindo que os jogadores pesquisem especificamente os testes de seu interesse apresentado na Figura 2.7.
- Para evitar que os defensores identifiquem mutantes como equivalentes sem tentar testá-los adequadamente, os duelos de equivalência agora só podem ser acionados quando o código mutado já está coberto por pelo menos um teste.

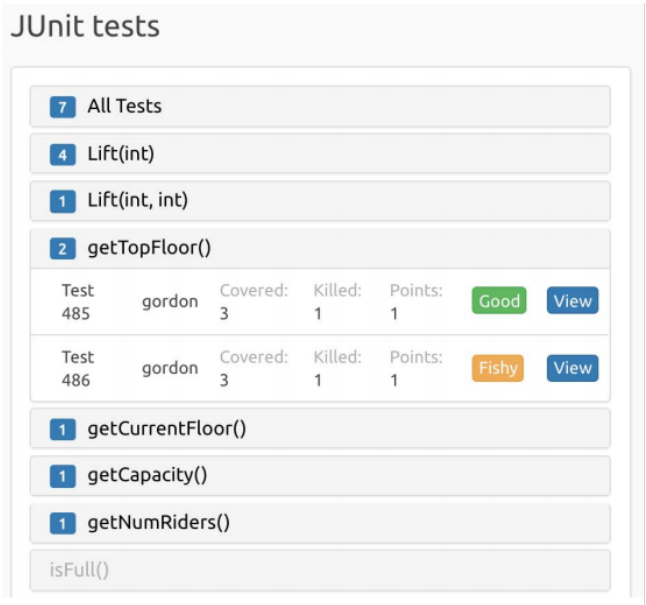

Figura 2.7. Todos os teste de um jogo apresentado em termos dos métodos da classe em teste. Fonte: (FRASER et al., 2020).

Para os atacantes, é possível criar mutantes modificando o código-fonte sem pensar nos testes existentes (FRASER et al., 2020). Para envolve-los melhor com atividades de teste, foram adicionados uma série de novos recursos:

- Para evitar que os atacantes editem código sem pensar nas consequências, foi adicionado um suporte para capturar suas intenções, apresentado na Figura 2.8. Para cada mutante que os atacantes enviam, eles precisam especificar se estão intencionalmente enviando um mutante equivalente ou um mutante que eles acreditam não ser equivalente. Eles também podem enviar um mutante e especificar que não sabem se é equivalente ou não, no entanto, o objetivo desse recurso é principalmente forçá-los a pensar sobre esse aspecto, de modo que o fato de eles selecionarem ou não o rótulo correto para seu mutante não tem efeito no jogo.
- Os defensores tendem a exigir alguns minutos antes de compreenderem adequadamente a classe em teste e como testá-la. Durante esse tempo, os atacantes geralmente criam mutantes sem saber onde os defensores estão colocando seus esforços de teste. Para evitar esse problema, foi adicionado uma funcionalidade para permitir que os jogos sejam inicializados com testes pré-existentes, de forma que os atacantes tenham que pensar imediatamente em como seus mutantes podem sobreviver dos testes existentes.

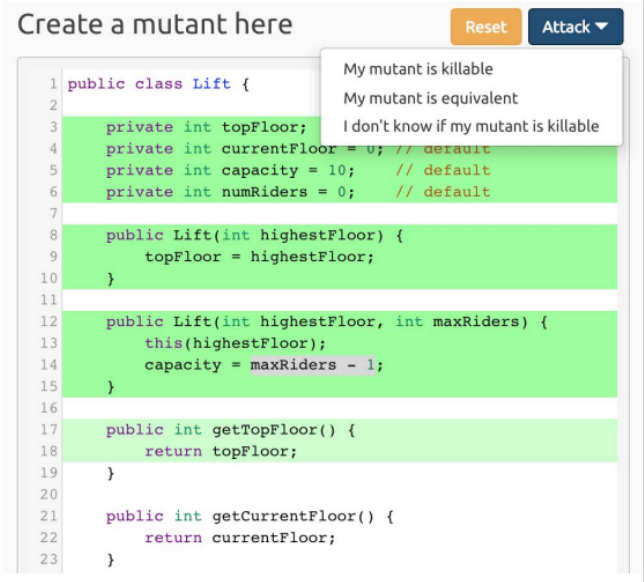

Figura 2.8. Os estudantes podem ser forçados a especificar se pretendem produzir um mutante equivalente ou não, encorajando-os a pensar em vez de criar um código desordenado.
Fonte: (FRASER et al., 2020).

Além dos aprimoramentos envolvendo os jogadores, foi aprimorada a interface administrativa do Code Defenders. Uma das funcionalidades dessa interface oferecida pela plataforma é uma série de estatísticas e análises. A Figura 2.9 mostra uma visão geral com estatísticas de cada jogador. Para avaliar os estudantes, o Code Defenders produz *killmaps*, ou seja, executa todos os testes contra todos os mutantes e, em seguida, marca os mutantes e testes como significativos ou não (ou seja, um mutante deve sobreviver a pelo menos um teste, mas pode ser eliminado, um teste deve detectar em pelo menos um mutante). O uso de *killmaps* garante que o desempenho dos estudantes não seja avaliado com base nos jogos individuais, uma vez que o resultado de um jogo depende muito dos jogadores participantes e não apenas das habilidades do jogador (FRASER et al., 2020).

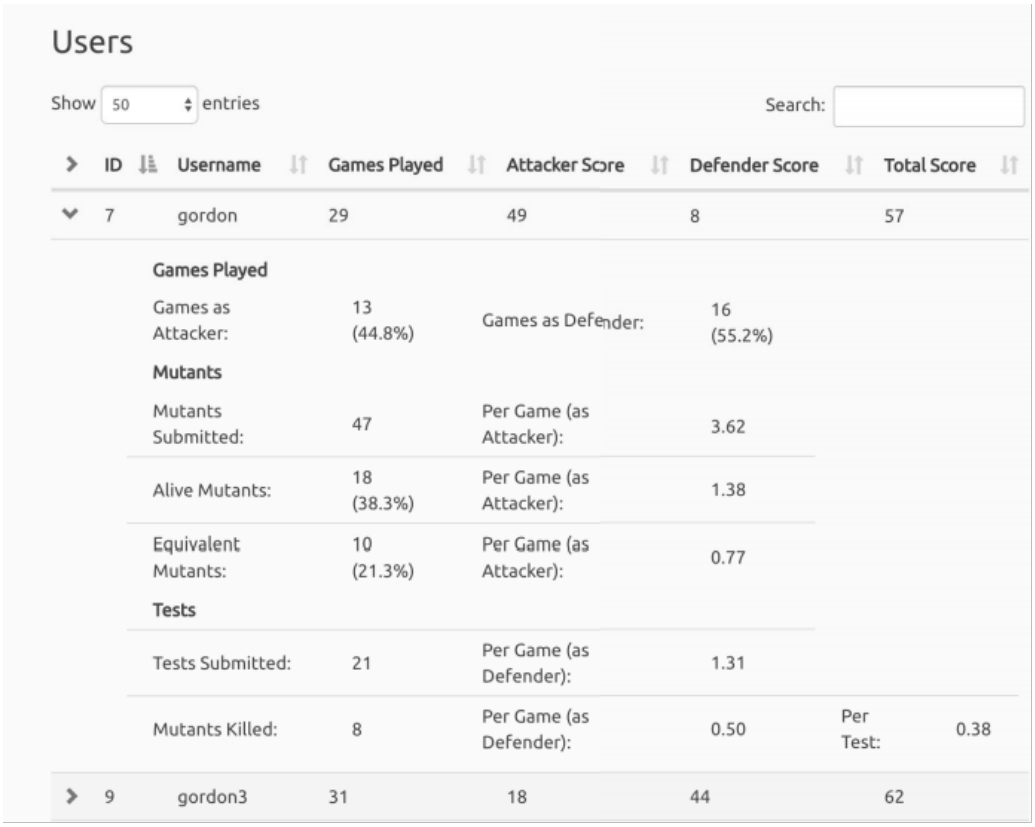

Figura 2.9. Os instrutores podem visualizar ou baixar estatísticas sobre jogadores individuais e as classes Java usadas para os jogos.

Fonte: (FRASER et al., 2020).

2.4.2 Pex4Fun e Code Hunt

Aprender a codificar através de uma plataforma gamificada é diferente de aprender escrevendo para uma especificação. Existem muitas competições em que os estudantes se empenham uns contra os outros e contra o relógio para criar uma solução para um problema definido. O Pex4Fun e o Code Hunt trazem uma experiência de aprender codificando através da diversão, que é utilizada como um ingrediente vital para acelerar o aprendizado e a obtenção de uma habilidade necessária (TILLMANN et al., 2014).

O Pex4Fun é uma plataforma *web* desenvolvida pela Microsoft Research (TILLMANN et al., 2011) para o ensino e aprendizado baseado em jogos para as linguagens de programação C#, Visual Basic e F#. O professor é responsável por criar a especificação do problema na forma de um programa e os estudantes implementam a solução, a fim de tornar o comportamento do respectivo programa igual ao do professor. O comportamento não é diretamente especificado para os estudantes, mas apresentado por casos de teste gerados a partir do código do professor, considerando uma técnica baseada em execução simbólica dinâmica (XIE et al., 2013).

O principal elemento de gamificação do Pex4Fun são os duelos de codificação, conforme mostra a Figura 2.10. Eles são flexíveis o suficiente para permitir que os professores criem vários jogos para direcionar o aprendizado a uma série de habilidades, como: programação, depuração,

resolução de problemas, teste e especificação de escrita, com diferentes níveis de dificuldade de jogo (XIE et al., 2013).

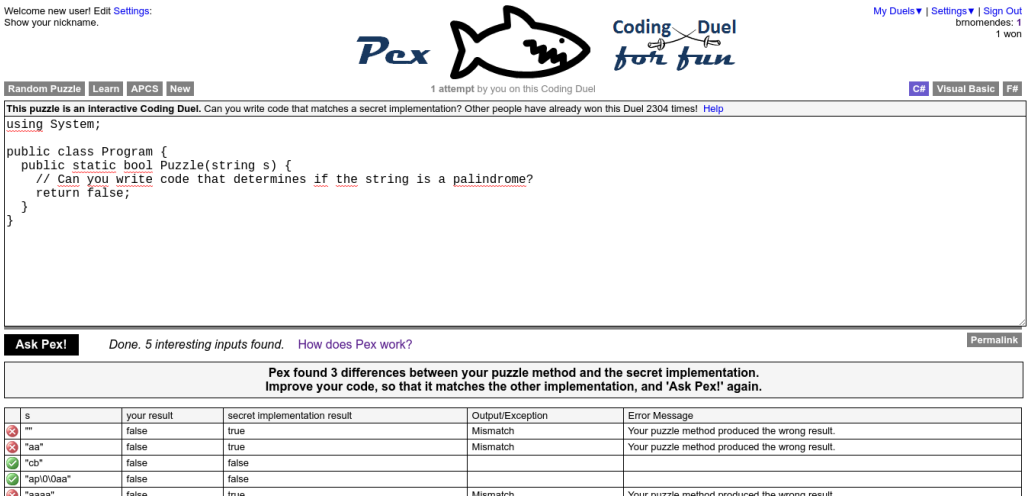

Figura 2.10. Exemplo de duelo no Pex4Fun

Fonte: Página do Pex4Fun - pexforfun.com.

Em um duelo de codificação o estudante recebe uma implementação do professor que não é visível, com os respectivo conceitos e tópicos que devem ser ensinados. Ela pode conter comentários opcionais para dar dicas ao estudante, a fim de reduzir o nível de dificuldade do jogo. Em seguida apresenta-se o código base, contendo as mesmas assinaturas dos métodos e os mesmos parâmetros do código secreto. Este código pode estar vazio, parcialmente preenchido corretamente ou incorretamente, exibido na parte superior central da interface do usuário da Figura 2.10. Os estudantes então utilizam esse código base para implementar um comportamento semelhante a implementação do professor. Durante o processo de jogo, o estudante tem a oportunidade de solicitar que a plataforma forneça o *feedback*, através da função "Ask Pex!", sobre as entradas do método que fazem com que a implementação do estudante e a secreta (do professor) tenham os mesmos ou diferentes resultados quanto à execução dos métodos. Analisando os resultados da ferramenta de geração de testes automáticos baseado em execução simbólica dinâmica, conforme implementado pela técnica Pex, na parte inferior da interface a plataforma fornece-se um pequeno número de entradas e resultados representativos dos aspectos corretos e incorretos da implementação do estudante. Como resultado, o estudante recebe um *feedback* relevante para saber quais alterações devem ser feitas a fim de obter os mesmos valores de saída do código secreto. Desta forma, iterativamente ele escreve o programa para ter o mesmo comportamento que o programa do professor utilizando as informações do Pex (TILLMANN et al., 2013).

Além do mecanismo de duelos, para envolver melhor os usuários, o Pex4fun utiliza vários recursos relacionados a dinâmica social. Por exemplo, a plataforma permite que um jogador aprenda quais duelos de codificação as pessoas já foram capazes de vencer (ou não). Ela conta também com uma classificação dos jogadores através do sistema de pontuação que visa motivar os estudantes a

continuar utilizando-a. O jogador pode visualizar as habilidades de duelo de cada jogador e compara-la com outras pessoas (TILLMANN et al., 2013).

Quanto à mecanismos de auxílio ao aprendizado, a plataforma fornece cursos virtuais abertos, juntamente com os jogos utilizados para reforçar o aprendizado dos estudantes (TILLMANN et al., 2014c). Além disso, qualquer professor pode criar um curso e compartilhar com outros estudantes. Os cursos podem agregar duelos para os alunos exercitarem o conteúdo proposto.

A partir do Pex4Fun originou-se a Code Hunt, uma plataforma *web* que disponibiliza um jogo de codificação educacional para ensinar programação nas linguagens Java e C# desde o ensino médio até a graduação (TILLMANN et al., 2014b). No Code Hunt, os duelos são divididos em setores. Cada setor tem um tema, por exemplo: aritmética, laços de repetição ou condicional. Em cada setor, há diferentes níveis, que aumentam a dificuldade do jogo. Os níveis são desafios semelhantes aos duelos do Pex4Fun, onde há um código base e o aluno deve implementar iterativamente analisando as informações do Pex (TILLMANN et al., 2014a).

O duelo de codificação é o principal tipo de jogo no Code Hunt. Ele funciona da mesma forma que no Pex4Fun, ou seja, o jogador recebe um código visível que normalmente está vazio ou uma versão incorreta/incompleta do segmento do código secreto de outro jogador. Então o jogador observa as dicas fornecidas pela plataforma para ajudá-lo no desenvolvimento do código a fim de encontrar o mesmo comportamento funcional do código secreto. A Figura 2.11 mostra a interface do usuário do jogo, onde o código visível para o jogador no duelo é exibido no lado esquerdo da interface. Depois que o jogador clica no botão "Capture Code"(mostrado na parte superior central), a plataforma gera casos de teste automaticamente pela ferramenta Pex e exibe os exemplos de comportamentos funcionais comuns e diferentes entre o código secreto e o código modificado pelo jogador, mostrado no lado direito. O aspecto do jogo é reconhecer um padrão dos casos de teste e recriar o código visível ao jogador para exibir os comportamentos esperados (XIE et al., 2015).

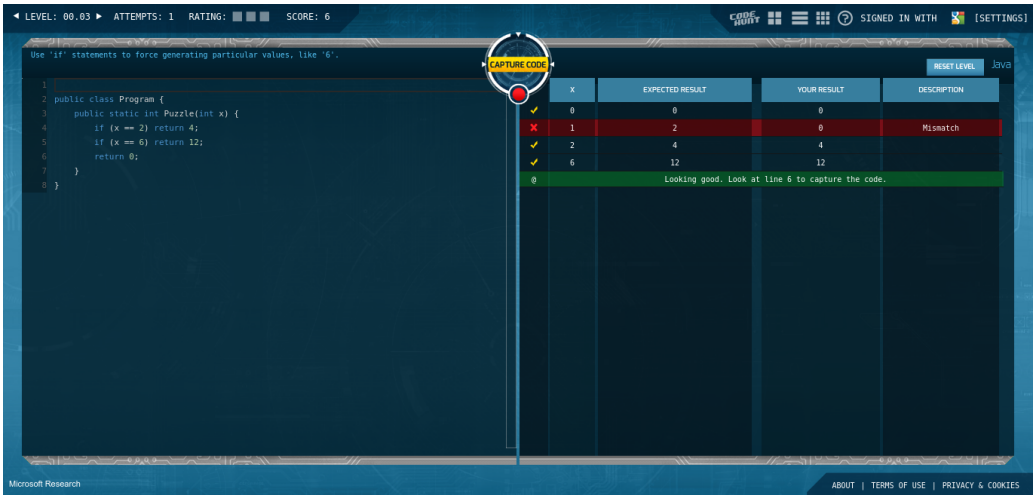

Figura 2.11. Ambiente Code Hunt

Fonte: Página do Code Hunt - codehunt.com.

O ambiente do Code Hunt possui um design baseado em jogos contando com: animações, sons e analogia a jogos de caça. Quando um jogador resolve um duelo, ele "captura o código"e lhe é retornado à quantidade de pontos obtidos, conforme apresentado na Figura 2.12. A plataforma possui uma tabela global de liderança, no qual os usuários com uma maior pontuação são exibidos em ordem decrescente pela quantidade de pontos.

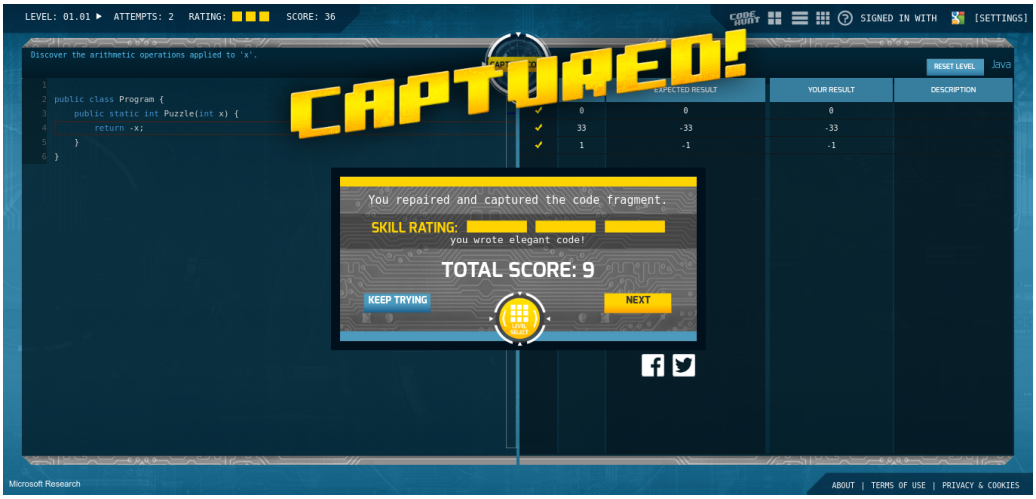

Figura 2.12. Exemplo de tela ao concluir um desafio no Code Hunt

Fonte: Página do Code Hunt - codehunt.com.

2.4.3 Testing Game: jogo educativo para a apoiar o aprendizado de teste de software

o Testing Game é um jogo educacional para ensino de teste de software, alinhando conteúdos teóricos e práticas sobre teste de software para desenvolvimento de software com qualidade. O processo de desenvolvimento foi realizado em três etapas: Revisão de Literatura e Mapeamento Sistemático, Implementação do Jogo de Teste e Avaliação do Jogo de Teste (VALLE et al., 2017b).

Na primeira etapa foram identificados os principais conceitos e pesquisas relacionadas a teste de softwares e jogos educacionais. A partir deles foi realizado um mapeamento sistemático das principais abordagens para o aprendizado de teste de software. A segunda etapa, a implementação do jogo, aconteceu apoiada a um novo mapeamento sistemático no qual identificava os mecanismos de jogos disponíveis. E a última etapa, avaliação do jogo, foi realizada por uma metodologia experimental que avalia o software desde sua primeira versão até seu uso na indústria (VALLE et al., 2017b).

O jogo é uma aplicação *web*, com o objetivo de apoiar o aprendizado de testes de software, abordando três técnicas de teste de software: Funcional, Estrutural e Baseada em Defeitos (VALLE et al., 2017b). Para sua implementação foi adotado uma estratégia incremental de teste, pois elas não devem ser utilizadas separadamente. O jogo possui 3 iterações, onde cada iteração representa uma técnica de teste e possuí um determinado número de fases. Para avançar para a próxima fase em qualquer nível de jogo, o estudante deve concluir a fase atual com sucesso. Na fase 1 de teste funcional

e de teste estrutural o jogador tem acesso a um módulo de conteúdo teórico sobre os respectivos assuntos (VALLE et al., 2017a).

O Teste Funcional, é baseado apenas na especificação do produto em teste. Os critérios desta técnica podem ser aplicados a qualquer fase do teste de software, ela identifica apenas os erros relacionadas com o mal funcionamento do software. O Teste Estrutural, é baseado na implementação do software a ser testado, isso requer a execução de partes ou componentes do software. O Teste de Mutação é o critério de teste da técnica de Teste Baseado em Defeitos, ele adiciona defeitos ao programa, que são frequentemente cometidos pelos desenvolvedores. No teste de mutação, o programa testado é alterado várias vezes, criando um conjunto de programas alternativos (mutantes). O objetivo é mostrar que o programa em teste não possui determinados tipos de defeitos (VALLE et al., 2017a).

A iteração 1 representa o primeiro nível do Testing Game, no qual são abordados os conteúdos relacionados com a técnica de teste funcional. Nesse nível há 8 fases que abordam: critérios de teste de software, especificação do programa BubbleSort, critério de particionamento em classes de equivalência, tabela de classe de equivalência, critério de análise de valor limite e conjunto de caso de teste mínimo. A interface da fase 1, mostrada na Figura 2.13, apresenta que o jogador deve arrastar os critérios para a técnica de teste que os correspondem. As técnicas consideradas estão representadas por dois retângulos. Quando o jogador arrastar um critério para a técnica correta, um sinal sonoro é realizado indicando que o jogador acertou. Caso contrário, um sinal sonoro é emitido, indicando que a ação realizada é incorreta, e o critério volta para seu local original (VALLE et al., 2017a).

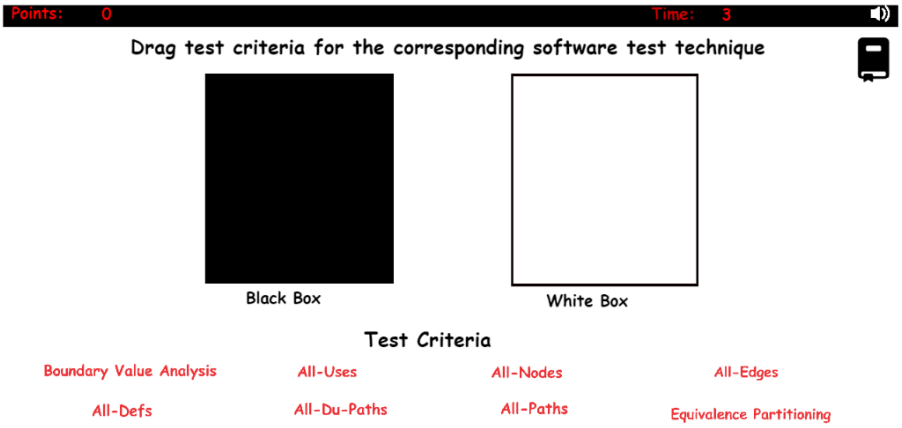

Figura 2.13. Primeira Fase do Teste Funcional.

Fonte: Valle et al. (2017a).

No segundo nível do jogo (iteração 2) são abordadas técnicas de teste estrutural, contendo 10 fases com os conteúdos relacionados com os critérios de: Todos-Nós e Todos-Usos, uso de variável (definição, uso computacional e uso predicativo), Grafo de Fluxo de Controle, Grafo Def-Uso, conjunto mínimo de teste, caminho não-executável e outros. A Figura 2.14 apresenta a interface da fase 1 do teste estrutural, onde é apresentado ao jogador o código do programa Bubble Sort. Em seguida, o jogador deve encontrar o grafo de fluxo de controle (GFC) que corresponde ao código do programa.

Para isso, ele deve eliminar os GFCs que não correspondem ao programa e eliminar os inimigos durante os desafios que são propostos (VALLE et al., 2017a).

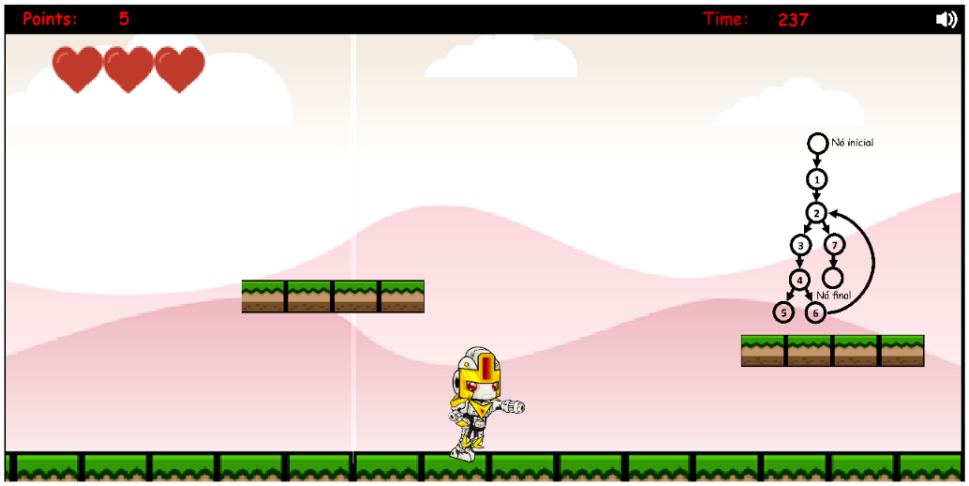

Figura 2.14. Primeira Fase do Teste Estrutural.

Fonte: Valle et al. (2017a).

O terceiro e último nível (iteração 3) aborda conteúdos relacionados com o teste baseado em defeitos, especificamente o critério de análise de mutantes, contendo 5 fases com conteúdo sobre operadores de mutação em Java, programas mutantes, mutantes equivalentes, escore de mutação e conjunto de caso de teste mínimo. Na primeira fase, conforme mostra a Figura 2.15 o jogador deve encontrar o programa mutante gerado a partir dos operadores de mutação: aritmético, atribuição, relacional e lógico. Para isso, o jogador deve eliminar todos os programas mutantes que não foram gerados por essas classes de operadores de mutação (VALLE et al., 2017a).

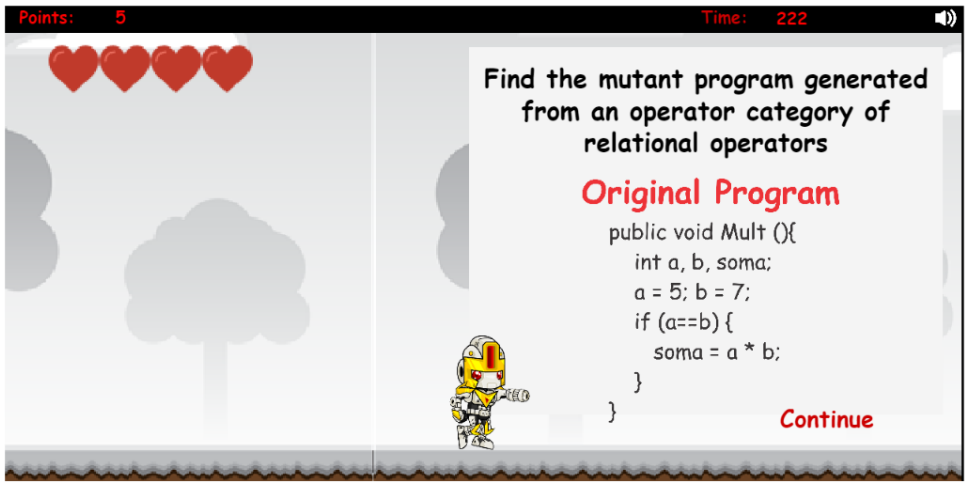

Figura 2.15. Primeira Fase do Teste de Mutação

Fonte: Valle et al. (2017a).

O Testing Game pode ser utilizado como instrumento motivador da aprendizagem de conteúdos relacionados a teste devido aos bons resultados de avaliações sobre sua usabilidade e qualidade (VALLE et al., 2017a).

2.5 Considerações finais

Seguindo a taxonomia podemos identificar os elementos de gamificação nas ferramentas apresentadas nos trabalhos relacionados, conforme a Tabela 2.1. O principal elemento utilizado para medir o desempenho do jogador no Code Defenders é o sistema de pontuação que está relacionada com a primeira dimensão (desempenho). Ele instiga e motiva os jogadores à produção de mutantes e de conjunto de casos de teste, além de definir um vencedor. Outro elemento presente da mesma dimensão são as estatísticas. Ela informa visualmente aos jogadores quais são os mutantes vivos, mortos e equivalente com suas respectivas pontuações e onde eles estão no código. A dimensão social nessa plataforma é representada por dois elementos: competição e cooperação. A cooperação está presente quando o modo de jogo da partida escolhida é a batalha (time de atacantes vs time de defensores), onde os jogadores do mesmo time interagem para conquistar seu objetivo (atacar ou defender). A competição é representada na plataforma pelo placar do jogo, apresentando a pontuação geral de cada time, e também através do modo de jogo sendo ele: jogador x jogador ou time x time. O objetivo é o elemento da dimensão pessoal e o mais importante no Code Defenders, pois através dele os jogadores podem observar que estão aprendendo com a plataforma. O jogador assume o papel de atacante ou defensor, o atacante tem como objetivo criar mutantes para passar nos casos de testes criados pelos defensores e os defensores tem o objetivo de matar esses mutantes. Apesar de apresentar esses elementos de gamificação a plataforma não apresenta nenhum elemento da dimensão de ficção e da dimensão ecológico.

Tabela 2.1. Tabela sobre os elementos presentes nas plataformas estudadas.

	Dimensões				
Ferramentas	Desempenho	Ecologia	Social	Pessoal	Ficção
Code Defenders	Pontos		Placar do jogo	Objetivo	
	Estatísticas		Batalha		
Pex4fun	Pontos		Placar do jogo	Objetivo	
			Classificação		
Code Hunt	Pontos		Placar do jogo		Storytelling
			Classificação geral		
Testing Game	Níveis	Relógio		Objetivo	
	Fases				

Fonte: Autoria própria

A segunda plataforma apresentada foi o Pex4fun, o primeiro elemento encontrado na plataforma são os pontos, no qual faz a medição de desempenho do estudante quando ele ganha um duelo de codificação. Para vencer um duelo, o estudante precisar cumprir seu objetivo, elemento da dimensão pessoal, através da implementação do comportamento semelhante ao código fornecido pelo professor. O ambiente conta uma classificação dos jogadores através do sistema de pontuação, ou seja, elemento de competição da dimensão social. E da mesma está presente o elemento reputação

onde o jogador pode visualizar as habilidades de duelo de cada jogador e compará-la com outras pessoas.

O Code Hunt foi desenvolvido a partir do Pex4Fun, com isso a plataforma contém os mesmos elementos de gamificação e alguns novos. Dentre os elementos novos está o *storytelling*, um elemento da dimensão de ficção, que traz aos duelos: animações, sons e uma analogia a jogos de caça. Através dele, os estudantes se envolvem com o conteúdo aprendido. O elemento competição foi aprimorado e agora apresenta uma tabela global de liderança, no qual os usuários com uma maior pontuação são exibidos em ordem decrescente pela quantidade de pontos.

O Testing Game é a última plataforma apresentada e a que menos contém elementos de jogos. O elemento nível é apresentado na plataforma através de um determinado número de fases pra cada um dos 3 níveis, o jogador só avança para a próxima fase quando a conclui, medindo assim seu desempenho. Para ele concluir um nível ele precisa resolver o problema proposto para cada fase, ou seja, concluir seu objetivo. Por último cada fase possui um relógio que conta o tempo que você demora para a conclusão, esse elemento está relacionado à pressão do tempo dentro da dimensão ecológica.

3 MÉTODO DE PESQUISA

O objetivo do estudo foi avaliar o efeito da integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos. Para alcançá-lo, foram definidas as seguintes questões de pesquisa:

QP1: A qualidade do ambiente para ensino de teste de software integrado à programação é satisfatória?

QP2: A integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos permite um melhor aprendizado de teste de software?

A abordagem, apresentada na Seção 3.1 acrescenta alguns elementos de gamificação ao Code Defenders e permite a criação de programas a serem testados. Ela foi avaliada em um estudo com estudantes do curso de Ciência da Computação da UTFPR de Campo Mourão que cursavam a disciplina de Engenharia de Software 1, tanto na perspectiva da qualidade dos casos de teste e do código (Seção 3.4) e de elementos de jogos associados (Seção 3.3).

3.1 Abordagem proposta

Considerando as análises das ferramentas apresentadas no capítulo anterior, o Code Defenders é a ferramenta que possui as melhores características de teste de software, devido a implementação abordar gamificação entrelaçada a uma jogabilidade intuitiva que permite aos jogadores produzir mutantes e desenvolver casos de testes.

O teste de mutação se baseia nas ideias do programador competente, em que o programador é bom e, se comete erros, são erros simples (pequenas mudanças sintáticas ou semânticas), e na premissa de acoplamento de defeitos, em que erros simples são responsáveis por falhas de software (DEMILLO et al., 1978). O Code Defenders contempla o ensino de teste de software considerando o critério de análise de mutantes, exercitando as facetas do programador competente e a acoplamento de defeitos.

Originalmente, o Code Defenders permite escolher programas do próprio banco de dados ou subir um programa completo na plataforma. Na abordagem proposta neste trabalho, o jogador poderá criar seu próprio código dentro da plataforma com o objetivo de promover o aprendizado integrado de programação na linguagem Java e em teste de software, como apresentado na Figura 3.1. Assim, são proporcionados dois momentos para exercitar diretamente habilidades de programação: na criação do programa e na criação dos casos de teste automatizados. Como na criação dos mutantes são observados erros de programação que levam a defeitos no software, indiretamente também é exercitada a habilidade de programação, com a criação de mutantes que são mais difíceis de satisfazer ou que permitem identificar erros no programa (mutante revelador de erro).

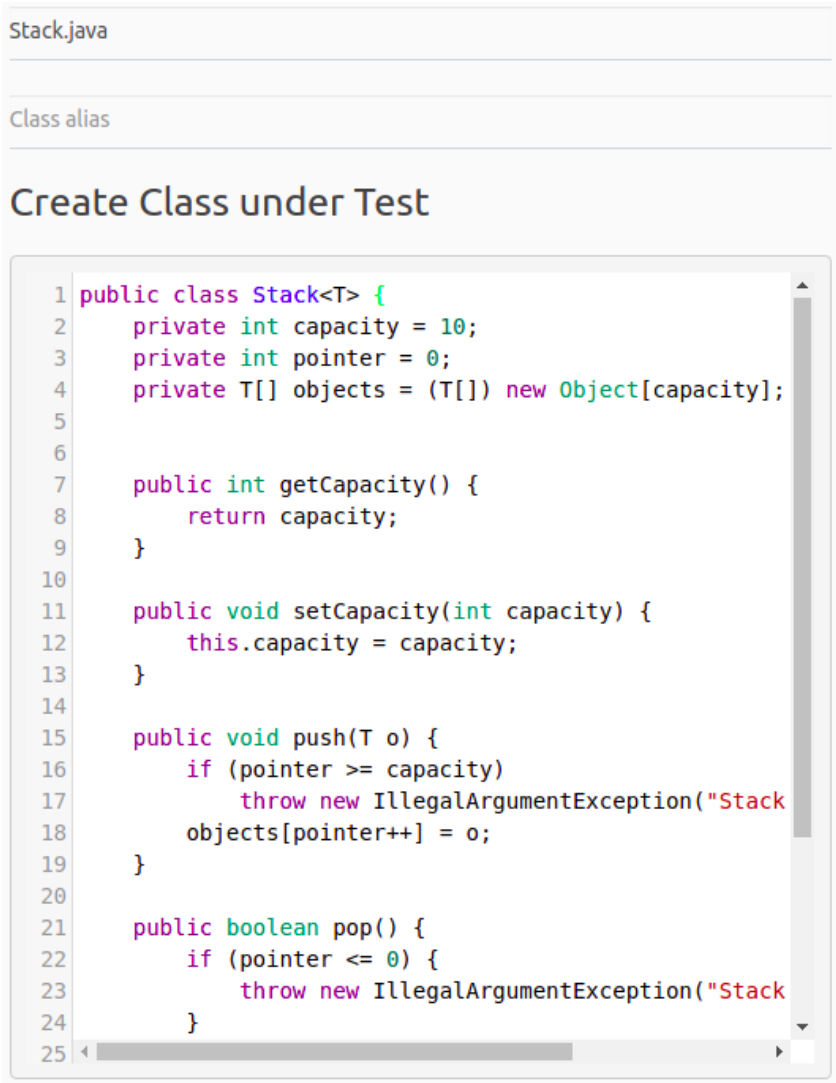

Figura 3.1. Tela de criação de código no Code Defenders.

Fonte: Autoria própria.

Para a implementação dessa nova funcionalidade ao Code Defenders, primeiramente foi adicionado um botão na página de *Create Battleground* para redirecionar o usuário para uma nova página na qual ele poderá escrever códigos em Java direto no editor de texto, conforme apresentado na Figura 3.1. O editor de texto é um código desenvolvido em JavaScript para destacar as palavras reservadas da linguagem Java. Para aprimorar a usabilidade dessa página, quando o jogador envia seu código para a plataforma, mas no seu código consta algum erro e não compila, o código completo volta a ele para que ele não perca nada do que ele fez. Para a implementação desse recurso foi criado uma classe específica que salva o código antigo do jogador. Para validar e salvar a classe enviada pelo jogador, foi criado uma classe Servlet que recebe um método HTTP POST do *front-end* com o nome da classe, apelido (se houver) e o código, a partir disso o código é compilado, e se não houver erros é salvo em um arquivo de texto com seu respectivo caminho salvo no banco de dados.

Em quesito jogabilidade, os estudantes são organizados em dois times: atacantes e defensores. Na ferramenta, não existia um limite quanto à criação de mutantes ou de casos de teste. Em

programação de computadores, sabe-se que isso pode promover a tentativa e erro, o que não proporciona uma experiência de aprendizado tão adequada quanto a reflexão em ação (EDWARDS, 2004). Para inibir esse comportamento, a abordagem proposta estabelece que cada time terá um custo para cada atividade e um custo total para ser gasto a fim de incentivar os jogadores a refletir para criar mutantes que não irão morrer tão facilmente e para criar casos de teste de maior qualidade, evitando a tentativa e erro.

Ambos atacantes e defensores terão o custo total e custo por cada atividade definidos antes do começo da partida. No caso dos atacantes, o custo de cada atividade será dobrado comparado ao defensor, para que os estudantes não criem mutantes aleatórios, mas sim mutantes que sejam mais difíceis de serem mortos. Supondo que foi definido que o custo total será 10 e o custo por atividade será 1 na partida, o atacante terá o custo de 2 pontos para cada atividade, cenário o qual é apresentado na Figura 3.2. A Figura 3.3 mostra o defensor criando um caso de teste e tendo seus pontos consumidos.

Figura 3.2. Tela do atacante após a criação de um mutante, custo total diminuído.
Fonte: Autoria própria.

Figura 3.3. Tela do defensor após a criação de um caso de teste, custo total diminuído.
Fonte: Autoria própria.

Para que nenhum dos times fiquem estagnado na partida, a cada mutante que sobrevive a cada caso de teste, o atacante receberá a pontuação para criação de mais mutantes (Figura 3.4) e para

cada mutante morto pelo caso de teste o defensor também receberá a pontuação para a criação de novos casos de teste, conforme apresentado na Figura 3.5.

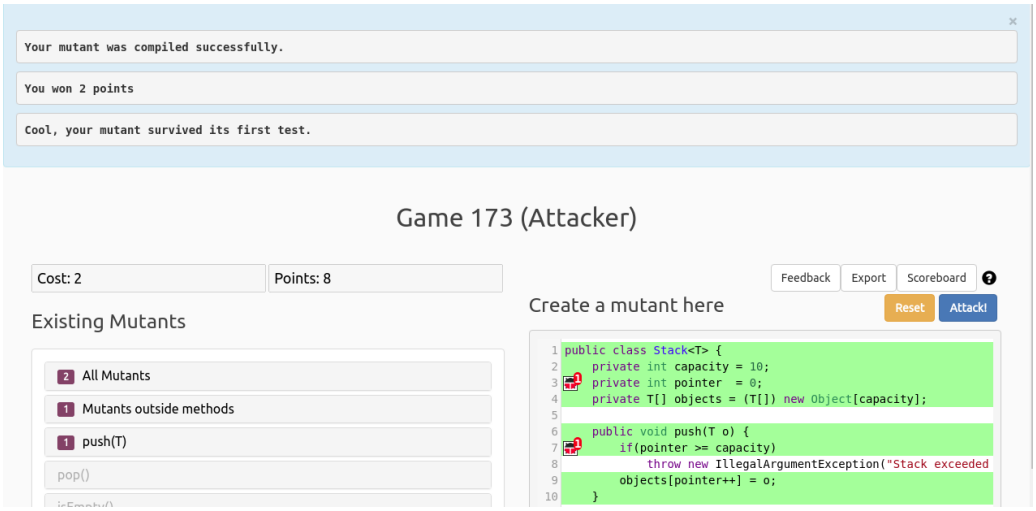

Figura 3.4. Mutante sobreviveu a um caso de teste.

Fonte: Autoria própria.

Figura 3.5. Caso de teste matou 2 mutantes.

Fonte: Autoria própria.

Os elementos de jogos citados anteriormente foram implementados primeiramente adicionando 4 novas colunas a tabela Games: custo de atividade dos atacantes, custo de atividade dos defensores, custo total dos atacantes e custo total dos defensores. Essa tabela é responsável por salvar os dados de cada partida dentro da plataforma. Estes custos são configurados pelo jogador na hora da criação da partida, porém eles possuem os valores mínimos de: 1 ponto para custo de cada atividade de atacante ou defensor e 10 pontos totais para cada time.

Para realizar a operação de diminuir ou aumentar a quantidade de pontos referente a atividade dos times, foram realizadas as alterações necessárias nas classes responsáveis por criar mutantes, criar testes de unidade, verificar se o teste de unidade matou um ou mais mutantes e verificar se um mutante sobreviveu a um ou mais casos de testes. E para finalizar essa implementação,

na página de visualização do defensor e do atacante foi adicionada à interface a visualização dos pontos de custo para que os jogadores vejam durante um jogo.

Houve também uma simplificação na interface na criação de batalha com foco na execução do estudo, pois antes haviam muitas opções de configurações desnecessárias para o estudo, que poderiam causar confusão aos jogadores. Todas essas opções foram removidas e restaram apenas as opções de: seleção da classe, custo de atividade na batalha, custo de atividade inicial e seleção do papel a qual o jogador irá jogar a partida, conforme apresenta na Figura 3.6.

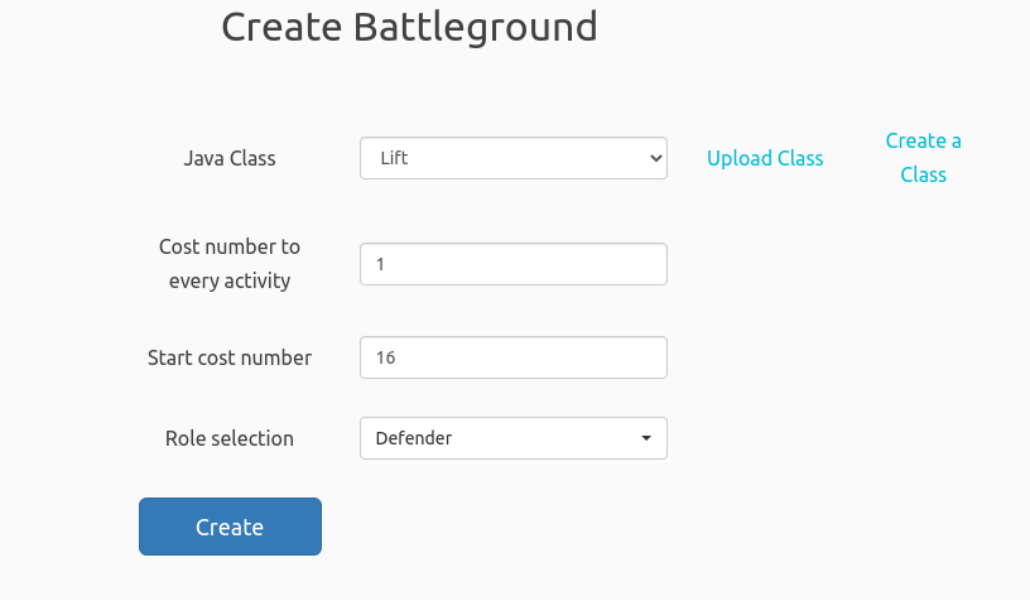

Figura 3.6. Simplificação da interface.

Fonte: Autoria própria.

O código do Code Defenders foi estudado e está disponível no repositório do GitHub[1], como citado na subseção anterior, foi implementado duas funcionalidades novas: criação de classe direto na plataforma e custo associado a cada atividade em cima de um custo total e ela foi disponibilizada para uso neste link http://dacom.cm.utfpr.edu.br/codedefenders/.

3.2 Desenho experimental

A abordagem proposta foi avaliada em um estudo de caso e sua população consistiu de estudantes do curso de Ciência da Computação. A amostra foi obtida por conveniência, através de convite a estudantes de disciplinas de Engenharia de Software 1 do curso de graduação em Ciência da Computação da Universidade Tecnológica Federal do Paraná do campus de Campo Mourão no primeiro semestre do ano letivo de 2021. Os participantes implementaram dois programas cada um, e criaram uma partida para cada implementação. A partir destes jogos criados, eles participaram dos próprios jogos e dos jogos de seus colegas, alternando no papel de atacante e defensor. Os casos

[1] https://github.com/vnc10/CodeDefenders

de teste de unidade criados pelos estudantes no papel de defensor foram analisados utilizando a biblioteca JaCoCo e PIT, assim como os códigos dos programas criados.

3.2.1 Conjunto de dados

Para a execução do estudo foi desenvolvido um conjunto de atividades contemplando problemas típicos de disciplinas de algoritmos e estruturas de dados: manipulação de caracteres (strings), números primos, ordenação (*bubble sort*), filas, filas circulares, listas encadeadas, listas encadeadas com cache e pilhas. Como critério para escolher esses problemas foram consideradas a abrangência dos conceitos requeridos e a pertinência dos problemas em si, conforme observado pela presença de programas que tratam desses tópicos em bibliotecas de qualidade reconhecida, como, no caso, o Apache Commons (Apache Software Foundation, 2019).

Os primeiros três problemas utilizam variáveis escalares e vetores de tamanhos predefinidos (arrays), como tipicamente abordado em uma disciplina inicial de programação. De modo geral, um único método é suficiente para implementar o algoritmo que resolve o problema. A primeira implementação abordada é um algoritmo de manipulação de caracteres com objetivo de transformar uma frase em caixa alta ou em caixa baixa sem a utilização de uma biblioteca externa. Seguindo para a próxima implementação, foi desenvolvido um algoritmo que verifica se o número é primo ou não. A terceiro implementação aborda o bubble sort, um algoritmo de ordenação de um vetor de números para a ordem crescente. O algoritmo ordena o vetor de forma que o percorre várias vezes e, a cada passagem o algoritmo compara a posição atual com a próxima posição. Se a posição atual for menor, é feita a troca com a próxima posição. Caso contrário, a troca não é feita e se passa para o próximo par de comparação e assim sucessivamente até que o vetor esteja ordenado.

Os demais problemas envolvem estruturas de dados dinâmicas, como visto inicialmente em disciplinas de algoritmos e estruturas de dados. Tais problemas requerem a definição de uma ou mais classes, com diversos métodos. A quarta implementação aborda a estrutura de dados fila que é do tipo FIFO (*first-in first-out*), na qual o primeiro elemento a ser inserido será o primeiro a ser retirado, ou seja, adicionam-se elementos no fim e removem-se do início. Sua classe possui os seguintes métodos:

- **Enqueue:** adiciona o elemento no fim da fila.
- **Dequeue:** remove o primeiro elemento da fila.
- **Head:** retorna o primeiro elemento da fila.
- **isEmpty:** verifica se a fila está vazia.

Seguindo a mesma abordagem da fila, foi implementada uma fila circular. A diferença entre as duas é que o primeiro nó da fila guarda o valor de memória do último nó, e o ultimo nó também guarda o valor do primeiro nó, possuindo os seguintes métodos:

- **Enqueue:** adiciona o elemento no fim da fila.
- **Dequeue:** remove o o primeiro elemento da fila.
- **isFull:** verifica se a fila está cheia.

- **isEmpty:** verifica se a fila está vazia.

A próxima estrutura de dados implementada foi a lista encadeada. Ela é uma sequência de nós, em que cada nó contém um objeto e o endereço do próximo nó. Diferente da fila, um elemento na lista pode ser removido ou adicionado em qualquer posição da estrutura. Sua classe possui os seguintes métodos:

- **Size:** retorna o tamanho da lista.
- **Add:** adiciona um elemento no fim da lista.
- **Get:** obtém um elemento por índice.
- **IndexOf:** retorna o índice do elemento da lista.
- **Remove:** remove o elemento desejado na lista.
- **isEmpty:** verifica se a fila está vazia.

O próximo algoritmo foi uma lista encadeada com cache. Como seu próprio nome diz, ela é a implementação de uma lista que armazena um cache de objetos $Node$, com o objetivo de aumentar o desempenho para a criação de objetos. Essa implementação é adequada para listas com muitos objetos em que as operações de adicionar e remover são mais custosas para o algoritmo sem cache. Como referência para esta implementação, foi reutilizado o código do Apache Commons Collection (Apache Software Foundation, 2002).

A última estrutura de dados implementada foi a pilha. Ela é uma estrutura tipo LIFO (*last-in first-out*), em que o último elemento a ser inserido será o primeiro a ser removido. Além disso, uma pilha permite o acesso apenas a um item de dados, o último inserido na estrutura. Seus métodos implementados foram:

- **GetCapacity:** retorna o tamanho da pilha.
- **SetCapacity:** altera o tamanho da pilha.
- **Push:** empilha um objeto.
- **Pop:** desempilha um objeto.
- **isEmpty:** verifica se a pilha está vazia.

Para esses problemas, foram criadas implementações dos algoritmos em Java. Quando possível, o código do programa e casos de teste foram extraídos de bibliotecas existentes, como do Apache Commons Collection para a lista encadeada com cache (Apache Software Foundation, 2002). Os dados de entrada e resultado esperado de casos de teste, quando possível, também foram obtidos de código disponível no Apache Commons. O código está disponível no repositório do GitHub.[2]

Para medir a qualidade dos casos de testes foi utilizada a cobertura de requisitos de teste considerando o fluxo de controle. Ela é uma métrica de software usada para medir quantas linhas de código e desvios são executados durante os testes. Para as respectivas implementações apresentadas anteriormente foi utilizada a biblioteca JaCoCo (HOFFMANN et al., 2009) para medir a cobertura dos testes.

[2] https://github.com/vnc10/TCC

O relatório do JaCoCo gera a porcentagem de cobertura dos testes referente às instruções e aos desvios. A cobertura de instruções fornece informações sobre a quantidade de código que foi executado ou perdido nos arquivos das classes. A cobertura de desvios calcula a cobertura para todos os desvios de execução (condicionais), contando o número total dos desvios em um método e determinando o número dos desvios executadas ou perdidas. A Tabela 3.1 apresenta a porcentagem de cobertura de instruções e de desvios dos algoritmos implementados. O único programa para o qual não foi alcançado 100% de cobertura foi referente à lista encadeada com cache, devido à reutilização de classes da biblioteca Apache Commons Collections e requisitos de teste não executáveis.

Tabela 3.1. Relatório de cobertura do testes gerado pelo JaCoCo.

Algoritmo	Cobertura de instruções	Cobertura de desvios
Manipulação de caracteres	100%	100%
Ordenação	100%	100%
Número primo	100%	100%
Fila	100%	100%
Fila circular	100%	100%
Lista encadeada	100%	100%
Lista encadeada com cache	95%	92%
Pilha	100%	100%

Fonte: Autoria própria

A cobertura de teste tradicional, como a calculada pelo JaCoCo, mede apenas quais linhas de código e desvios são executados pelos testes, não verificando se os testes são realmente capazes de detectar as falhas no código executado. A ferramenta PIT é um sistema de teste de mutação fornecendo cobertura de teste para Java (COLES et al., 2010). Ela gera mutantes automaticamente no código e, em seguida, os testes são executados. Se os testes passarem, o mutante é morto, se não o mutante sobrevive.

Foi executada a ferramenta em todos os códigos do desenho experimental e como resultado foi gerado o relatório apresentado na Tabela 3.2, com a porcentagem da cobertura de mutação para cada um. A porcentagem alcançada por cada um dos conjuntos de testes de unidade foi satisfatória, apesar de nenhum alcançar os 100%, provavelmente porque existem mutantes que são equivalentes, ou seja, ele apresenta o mesmo comportamento que o programa em teste independentemente da entrada, sendo assim não existe um caso de teste que é capaz de matá-lo. Além disso, a ferramenta não conseguiu gerar um relatório para a lista encadeada com cache devido a uma limitação técnica (PIT não conseguia encontrar e executar os casos de teste).

Considerando as medidas de cobertura e pontuação de mutação, observa-se que as implementações satisfazem os critérios de cobertura de instruções e desvios, mais simples de satisfazer, e de análise de mutantes (mais completo). Assim, nessa perspectiva, a qualidade do conjunto de casos de teste é satisfatória. Quanto ao código dos programas, ele não apresenta erros quanto aos casos de teste estabelecidos, o que sugere sua correção. Desta forma, o conjunto de dados estabelecido pode ser utilizado como referência para o estudo em questão.

Tabela 3.2. Relatório de cobertura do testes gerado pelo PIT.

Algoritmo	Cobertura de mutação	Força do Teste
Manipulação de caracteres	89%	89%
Ordenação	85%	85%
Número primo	89%	89%
Fila	88%	88%
Fila circular	84%	84%
Lista encadeada	93%	93%
Pilha	91%	91%

Fonte: Autoria própria

3.2.2 Planejamento

O estudo experimental consiste de um estudo de caso e sua população abrange estudantes do curso de Ciência da Computação. A amostra deve ser obtida por conveniência, através de convite a estudantes de disciplinas de Engenharia de Software.

Antes de apresentar o experimento, deve-se realizar a hospedagem da ferramenta Code Defenders em uma máquina da UTFPR e disponibilizá-la através de um link do DACOM. Além da parte de hospedagem, deve ser gravado um vídeo explicando todo o processo do experimento com a apresentação da ferramenta e das classes a serem implementadas pelos alunos. O começo da aula também deve contar com uma explicação para reforçar que também será gravada e disponibilizada.

O estudo experimental foi dividido em 3 etapas. Na primeira etapa os estudantes devem responder um questionário sobre a sua experiencia de programação e teste de software contendo as questões apresentadas na Tabela 3.3.

Tabela 3.3. Questionário sobre experiência do estudante

Pergunta	Opções de respostas
Quais destas linguagens de programação você utilizou?	C, C++, Java, JavaScript, Go, PHP, Python, Kotlin e outro
Quais dessas linguagens você tem mais experiências?	C, C++, Java, JavaScript, Go, PHP, Python, Kotlin e outro
Como você desenvolveu a sua experiência em programação?	Antes de ingressar no curso de BCC (autodidata), Antes de ingressar no curso de BCC (formação técnica), Durante o curso de BCC (discipilinas), Estágio, Trabalho e outro
Quais técnicas de teste de software você conhece?	Testes de unidade, Teste de integração, Teste de sistema, Teste end to end, Teste exploratório e outro
Quais técnicas de teste de software você utilizou ou utiliza	Testes de unidade, Teste de integração, Teste de sistema, Teste end to end, Teste exploratório e outro
Que ferramentas de teste você geralmente utiliza?	Questão aberta

Fonte: Autoria própria

Continuando para a parte 2 do experimento, os estudantes devem começar a implementação do código proposto para o experimento através das interfaces em Java disponibilizadas para auxilio. Após a implementação da classe, cada estudante deve criar uma partida onde ele vai participar como defensor ou atacante. A configuração da partida precisa ser direcionada para partidas no nível fácil, número do custo de atividade na plataforma em 1 e número inicial de custo em 20.

Além da participação no próprio código, cada estudante deve escolher pelo menos uma partida dos outros participantes para jogar, escolhendo o papel contrário do jogo em que previamente participou, ou seja, se jogou como atacante irá participar do outro jogo como defensor, e vice-versa. O estudante também deve ficar livre para participar de quantas partidas desejar e quais papéis preferir.

A etapa final consiste no participante responder ao formulário MEEGA+ para avaliar o uso do Code Defenders respondendo as 35 perguntas. Considerando os resultados obtidos com o MEEGA+ e com os programas e casos de teste implementados, procede-se com a análise, conforme apresentado a seguir.

3.3 Método para avaliação da qualidade dos elementos de jogos

Para este estudo, foi necessário a utilização de métodos para avaliação de jogos educacionais com o propósito de avaliar a qualidade dos elementos de jogos presente, afim de responder a QP1. O método que foi escolhido é o MEEGA+, o mais usado amplamente na prática, sendo relatado por vários estudos de diferentes autores avaliando diferentes jogos e contextos (PETRI; WANGENHEIM, 2016; PETRI; WANGENHEIM, 2017).

Avaliações de jogos educacionais visam medir se o público-alvo atingiu os objetivos definidos. Para que ela seja realizada de forma rápida e não intrusiva, a fim de não prejudicar os participantes envolvidos no estudo, opta-se pela realização de estudos de caso que permitam uma pesquisa aprofundada de um indivíduo, grupo ou evento. O modelo MEEGA+ é definido como não-experimental com pós-teste com um único grupo, ou seja, começa aplicando o jogo educacional e então um instrumento de medição (questionário) é respondido pelos estudantes para coletar dados sobre suas percepções sobre o jogo (PETRI; WANGENHEIM, 2019; PETRI et al., 2019).

O MEEGA+ (Modelo de Avaliação de Jogos Educativos) foi desenvolvido especificamente para a avaliação de jogos educativos para o ensino da computação. O modelo captura a reação dos estudantes após terem jogado o jogo, aplicando um questionário padronizado. Este modelo foi avaliado em termos de sua aplicabilidade, utilidade, validade e confiabilidade por meio de uma série de estudos de caso, além de permitir que desenvolvedores de jogos, pesquisadores e instrutores avaliem a qualidade dos jogos usados.

No MEEGA+ foram definidos itens para o instrumento de medição (questionário) conforme apresentados na Tabela 3.4 e na Tabela 3.5. O formato de resposta definido para o questionário é uma escala Likert de 5 pontos com alternativas de resposta que variam de discordo totalmente a concordo totalmente. O uso dessa escala permite expressar a opinião do estudante sob o jogo com

maior precisão, além de permitir que ele fique mais confortável para expressar sua opinião, usando um ponto neutro e assim aumentando a qualidade das respostas (PETRI et al., 2019).

Tabela 3.4. Tabela de itens do questionário do modelo MEEGA+ avaliando usabilidade.

Dimensão/Subdimensão		Item	Descrição do Item
Usabilidade	Estética	1	O design do jogo é atraente (interface, gráfico, tabuleiro, cartas, etc).
		2	Os textos, cores e fontes combinam e são consistentes.
	Aprendizibilidade	3	Eu precisei aprender poucas coisas para poder começar a jogar o jogo.
		4	Aprender a jogar este jogo foi fácil para mim.
		5	Eu acho que a maioria das pessoas aprenderiam a jogar esse jogo rapidamente.
	Operabilidade	6	Eu considero que o jogo é fácil de jogar.
		7	As regras do jogo são claras e compreensíveis.
	Acessibilidade	8	As fontes (tamanho e estilo) utilizados no jogo são legíveis.
		9	As cores utilizadas no jogo são compreensíveis.
		10	O jogo permite personalizar a aparência (fonte e/ou cor) conforme a minha necessidade.
	Proteção contra erros do usuário	11	O jogo me protege de cometer erros.
		12	Quando eu cometo um erro é fácil de me recuperar rapidamente.

Fonte: Petri et al. (2019)

A fim de orientar a aplicação do MEEGA+, o método possui um processo dividido em 5 fases: escopo, planejamento, execução, análise e apresentação, como apresentado na Figura 3.7.

Na fase do escopo o objetivo é selecionar o objeto de estudo (jogo educativo) e definir os objetivos da avaliação. Neste trabalho, o objeto de estudo foi o Code Defenders e o objetivo de avaliação é avaliar o efeito da integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos (PETRI; WANGENHEIM, 2019).

Na segunda fase do processo, são definidas onde a avaliação será realizada, o cronograma da avaliação e a elaboração do instrumento de coleta de dados. A ideia inicial era realizar a avaliação no laboratório de informática da UTFPR do campus de Campo Mourão, mas, em decorrência da pandemia do novo coronavírus, a avaliação deve acontecer de forma remota (*online*). Os estudantes participantes devem ser alunos do curso de Bacharelado em Ciência da Computação que estejam cursando a disciplina Engenharia de Software. Para a coleta de dados será utilizado o questionário modelo do MEEGA+ (PETRI; WANGENHEIM, 2019).

A terceira fase (execução) visa organizar e definir a execução da avaliação do jogo para os participantes. Nesta fase, os dados serão coletados para atingir o objetivo da avaliação. A primeira etapa da terceira fase é preparar a execução da avaliação, ou seja, devem ser convidados os estudantes para o experimento, e disponibilizar a plataforma Code Defenders online para que todos tenham acesso. Todos os participantes deverão concordar e aceitar participar da pesquisa. Antecedendo

Tabela 3.5. Tabela de itens do questionário do modelo MEEGA+ avaliando as outras dimensões.

Dimensão/Subdimensão	Item	Descrição do Item
Confiança	13	Quando olhei pela primeira vez o jogo, eu tive a impressão de que seria fácil para mim.
	14	A organização do conteúdo me ajudo a estar confiante de que eu iria aprender com este jogo.
Desafio	15	Este jogo é adequadamente desafiador para mim.
	16	O jogo oferece novos desafios com um ritmo adequado.
	17	O jogo não se torna monótono nas suas tarefas (repetitivo ou com tarefas chatas).
Satisfação	18	Completar as tarefas do jogo me deu um sentimento de realização.
	19	É devido ao meu esforço pessoal que eu consigo avançar no jogo.
	20	Me sinto satisfeito com as coisas que aprendi no jogo.
	21	Eu recomendaria este jogo para meus colegas.
Interação social	22	Eu pude interagir com outras pessoas durante o jogo.
	23	O jogo promove momentos de cooperação e/ou competição entre os jogadores.
	24	Eu me senti bem interagindo com outras pessoas durante o jogo.
Diversão	25	Eu me diverti com o jogo.
	26	Aconteceu alguma situação durante o jogo (elementos do jogo, competição, etc) que me fez sorrir.
Atenção focada	27	Houve algo interessante no início do jogo que capturou minha atenção.
	28	Eu estava tão envolvido no jogo que eu perdi a noção do tempo.
	29	Eu esqueci sobre o ambiente ao meu redor enquanto jogava este jogo.
Relevância	30	O conteúdo jogo é relevante para os meus interesses.
	31	É claro para mim como o conteúdo do jogo está relacionado com a disciplina.
	32	O jogo é um método de ensino adequado para esta disciplina.
	33	Eu prefiro aprender com este jogo do que de outra forma (outro método de ensino).
Aprendizagem percebida	34	O jogo contribuiu para a minha aprendizagem na disciplina.
	35	O jogo foi eficiente para a minha aprendizagem, em comparação com outras atividades da disciplina.

Fonte: Petri et al. (2019)

a execução do jogo, ocorrerá um treinamento dos estudantes sobre conceitos básicos de teste de software e sobre a ferramenta (em sua versão tradicional e para a versão proposta). Após a execução do jogo, será realizada a coleta de dados através de um formulário eletrônico para que os participantes respondam o questionário com base em suas percepções sobre o jogo. Após a coleta de dados, será

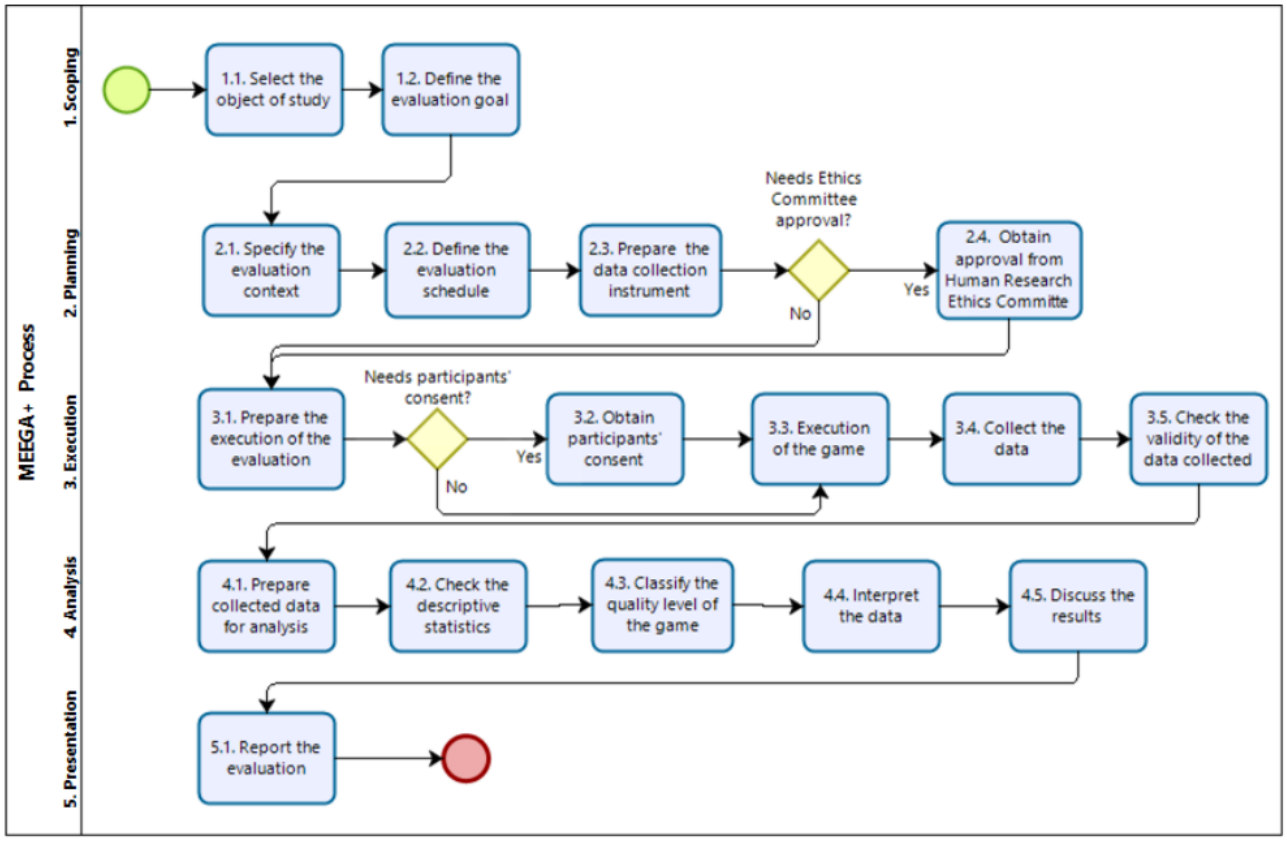

Figura 3.7. Etapas do processo do MEEGA+.

Fonte: Petri e Wangenheim (2019).

verificado se eles são razoáveis e se foram coletados corretamente. Isso abrange aspectos como se os participantes compreenderam os itens e, portanto, os responderam corretamente (PETRI; WANGENHEIM, 2019).

A quarta fase (análise) visa interpretar e analisar os dados coletados na fase de execução. A primeira etapa é preparar os dados para a análise, como usaremos a avaliação *online* essa atividade é realizada de forma automática através de planilhas. Após os dados coletados serem preparados e organizados nas planilhas de análise, os dados serão caracterizados pelo cálculo automático da estatística descritiva, porém é necessário verificar se as estatísticas foram geradas corretamente. A planilha calcula automaticamente a frequência de respostas, média e mediana para cada item do instrumento de medida. São gerados também gráficos de frequência, os quais são apresentados de acordo com os fatores de qualidade e dimensões do modelo MEEGA+. Após os dados coletados serem preparados e organizados nas planilhas de análise, esses dados podem ser utilizados para determinar o nível de qualidade do Code Defenders, classificando-o com base na escala de qualidade, definida no modelo MEEGA+. Para classificar o jogo na escala MEEGA+, é necessário utilizar o software estatístico R. Através do software é calculado a média das pontuações fornecidas de todos os participantes que avaliaram o jogo, aplicando a Teoria de Resposta ao Item (TRI). Após o cálculo da média, é realizada uma transformação desses escores em uma escala (50.15), aplicando a formula

no escore médio:

$$\Theta(50.15) = 50 + 15 * \Theta(0.1) * (\text{média das pontuações})$$

e obtendo assim o resultado. Com base neste valor final, o nível de qualidade do Code Defenders será avaliado, definido pela escala MEEGA+, descrevendo as características do jogo em relação ao seu nível de qualidade conforme a Tabela 3.6. A próxima etapa é a interpretação dos dados, essa análise é realizada seguindo os aspectos de qualidade determinados na meta de avaliação. Seguindo o modelo o Code Defenders irá ser avaliado em termos de usabilidade e experiência do jogador. Para cada fator de qualidade (usabilidade e experiência do jogador) será feito a análise de frequência de cada item do instrumento de medida, identificando o grau de concordância e / ou discordância dos estudantes. A última etapa dessa fase é a discussão de resultados, no qual será apontado as principais contribuições da utilização do jogo avaliado como estratégia instrucional para o ensino de teste de software, bem como suas oportunidades de aprimoramento (PETRI; WANGENHEIM, 2019).

A última fase (apresentação) é a responsável por reportar a avaliação, com o objetivo final da produção da monografia descrevendo detalhadamente, como a avaliação do jogo selecionado foi definida, planejada, executada e analisada (PETRI; WANGENHEIM, 2019).

3.4 Método para avaliação da qualidade dos casos de teste e dos programas

Para o método de avaliação da qualidade dos casos de teste e dos programas, devem ser considerados os casos de testes e dos programas criados durante o estudo de caso utilizando o Code Defenders pelos estudantes.

Quanto ao método para avaliação dos casos de teste, devem ser considerados os valores de cobertura de código dos testes de unidade com o do programa referência, utilizando os relatórios gerados pelo JaCoCo e pelo PIT. Para avaliar os programas criados pelos estudantes, devem ser utilizados os casos de teste de referência e também as ferramentas abordadas anteriormente. Em ambos os métodos, a avaliação compreende o número total de cobertura de linhas de código, a taxa de cobertura de mutação e a força dos testes.

Tabela 3.6. Tabela de níveis de qualidade do jogo

Nível de Qualidade	Descrição do nível
Baixa ($\theta < 42,5$)	Nesse nível, o jogo raramente proporciona interação social e dificilmente produz momentos de diversão entre os jogadores. O jogo não capta a atenção concentrada do aluno, não desperta o confiança de que aprenderá com o jogo, nem produz sentimentos de satisfação. O jogo raramente apresenta desafios, tem tarefas monótonas e não contribui para a aprendizagem dos alunos. Embora um jogo neste nível tenha uma baixa relevância para os interesses dos alunos, um aluno reconhece que o conteúdo do jogo está relacionado ao curso. Em termos de usabilidade, um jogo neste nível às vezes exibe recursos de operabilidade, que podem ter algumas regras claras e ser fáceis de jogar.
Boa ($45,5 \leq \theta \leq 65$)	Nesse nível, o jogo às vezes apresenta atividades desafiadoras, oferecendo novos desafios para os alunos. Ele fornece atenção moderadamente focada aos jogadores, embora os alunos não se esqueçam de seus arredores. Às vezes, o jogo também proporciona sentimentos de confiança e satisfação aos jogadores. Frequentemente o jogo apresenta momentos de interação social e diversão entre os jogadores. Frequentemente, o jogo é considerado relevante para os interesses dos alunos e, geralmente, os alunos reconhecem que o conteúdo do jogo está relacionado ao curso. Frequentemente, o jogo contribui de forma eficiente para o aprendizado do aluno. Em termos de usabilidade, o jogo costuma ter regras claras e é fácil de jogar, embora, normalmente, não apresente um design totalmente atraente.
Excelente ($\theta \geq 65$)	Nesse nível, o jogo é desafiador para os alunos e não tem atividades monótonas. É altamente relevante para os interesses dos alunos e fornece excelente atenção focada, satisfação, diversão e interação social. Permite que o aluno tenha certeza de que aprenderá com o jogo e contribuirá para um aprendizado eficiente do aluno. Em termos de usabilidade, o jogo apresenta excelente operabilidade e aprendizagem, ou seja, possui regras claras e é fácil de aprender a jogar. Mesmo assim, um jogo neste nível pode apresentar melhorias em termos de estética, não apresentando um design totalmente atraente.

Fonte: Petri et al. (2019)

4 RESULTADOS

4.1 Experimento

O estudo de caso foi realizado no primeiro semestre acadêmico de 2021, compreendendo os meses de agosto a setembro de 2021. Foram convidados no total 12 estudantes da disciplina Engenharia de Software 1 do curso de Bacharelado em Ciência da Computação, porém apenas 7 participaram.

A primeira parte do estudo foi realizada durante a aula síncrona com duração de 1 hora e 50 minutos. Após consentida a participação no estudo, conforme TCLE apresentado no convite, os estudantes responderam ao questionário sobre a experiência de programação e teste de software conforme apresentado no desenho experimental.

Seguindo com o estudo, cada participante acessou a plataforma do Code Defenders e criou uma conta. A partir desse momento, houve uma breve explicação sobre a pesquisa, seguida da explicação de como jogar na plataforma, criar partida, criar mutantes e testes de unidade. Durante o experimento eu fiquei disposto em todo o período do estudo para sanar dúvidas sobre a plataforma, sobre o Java, ou sobre os testes de unidade. Também foi disponibilizado um PDF e um vídeo com essas explicações para ser utilizado.

Para a primeira implementação foi proposto o desenvolvimento do *bubble sort*, algoritmo de ordenação simples que deveria ser desenvolvido durante o período de aula, assim como seus mutantes e testes de unidade. Foram disponibilizados o construtor e a interfaces dos métodos, conforme apresentado no Código 4.1.

```
1 public class BubbleSort {
2
3     private int arr[];
4
5     public BubbleSort(int array[]) {
6         arr = array;
7     }
8
9     public int[] Ordena() {
10    }
11 }
```

Código 4.1. Classe e método do BubbleSort.

Após a implementação do código cada estudante criou uma partida de sua implementação e participaram como atacante ou defensores das partidas dos colegas durante o período de aula.

Para a segunda parte do estudo, foi disponibilizada uma atividade para realização de forma assíncrona e foi proposta uma segunda implementação de desenvolver uma fila. A fila é uma estrutura

dinâmica que admite remoção de elementos e inserção de novos elementos. Ela possui a seguinte regra de operação: o primeiro objeto inserido na fila é também o primeiro a ser removido. Essa política é conhecida pela sigla FIFO (First-In-First-Out).

Essa estrutura foi desenvolvida fora do período da aula, por ser uma estrutura mais complexa do que um *bubble sort*, além de possuir teste de unidades mais complexos. Assim como no algoritmo de ordenação, foram disponibilizados o construtor e a interface com os métodos conforme o Código 4.2.

```
1  public class Queue {
2      private Node front = null;
3      private Node back = null;
4
5      private static class Node {
6          int item;
7          Node next;
8      }
9
10     /**
11      * Remove um elemento do começo da fila
12      */
13     public int dequeue() {
14     }
15
16     /**
17      * Insere um elemento no fim da fila
18      */
19     public void enqueue(int item) {
20     }
21
22     /**
23      * Retorna o primeiro elemento da fila
24      */
25     public int head() {
26     }
27
28     /**
29      * Verifica se a fila está vazia
30      */
31     public boolean isEmpty() {
32     }
33 }
```

Código 4.2. Classe e método da Fila.

Quanto aos métodos, temos que $dequeue$ é o método de remoção de elemento da lista, $enqueue$ é o método de inserção, $head$ é método que retorna o valor do primeiro nó da lista e o método $isEmpty$ retorna se a lista está vazia ou não.

Ao final dessa atividade, cada participante preencheu o formulário referente ao instrumento MEEGA+, para avaliação dos atributos de jogo.

4.2 Informações sobre os estudantes

Sete estudantes participaram do experimento. Todos responderam ao primeiro questionário sobre suas experiências com linguagem de programação e testes.

Na primeira pergunta foi questionado a eles quais linguagens de programação cada um utilizou, podendo marcar mais de uma opção de linguagem. A linguagem de programação Java, que é o foco do estudo, apareceu para apenas um participante, conforme apresentado na Figura 4.1.

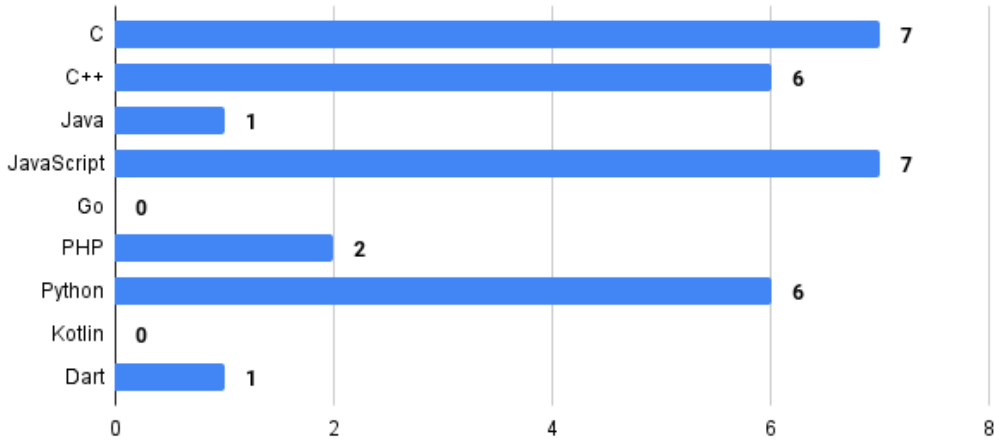

Figura 4.1. Linguagens de programação conhecidas pelos participantes.

Na segunda questão, Figura 4.2, foi questionado quais linguagens eles possuem mais experiência, podendo ser no máximo três linguagens. Neste caso, em nenhum o Java apareceu.

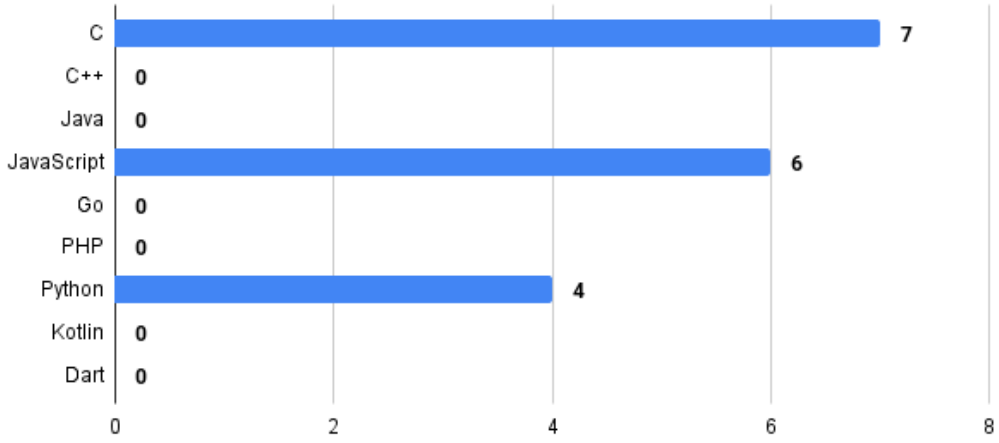

Figura 4.2. Linguagens de programação que os participantes possuem mais experiência.

Respondendo à pergunta de como os estudantes desenvolvedores sua experiência em programação, todos responderam durante o curso de Ciência da Computação. Além disso, indicaram

terem desenvolvidos outras experiências durante o estágio, trabalho, cursos externos e empresa júnior, conforme apresentado na Figura 4.3.

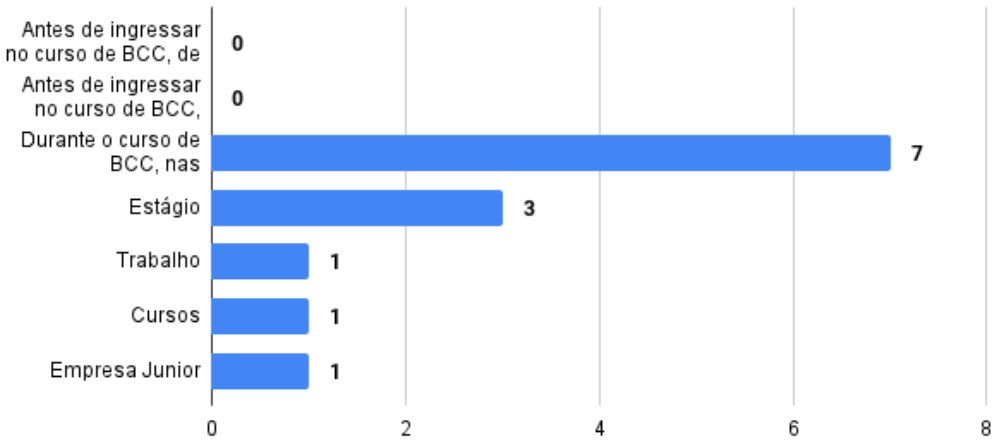

Figura 4.3. Onde os participantes desenvolveram as suas experiências em programação

Nas perguntas referentes a teste de software, foi perguntado quais técnicas eles conheciam. Teste de unidade, que é a técnica de teste utilizada neste estudo, era conhecido por 6 dos 7 estudantes, conforme apresentado na Figura 4.4. Além disso, o teste de unidade foi a técnica que eles mais utilizam ou utilizaram, representando 57,1%, conforme apresentado na Figura 4.5.

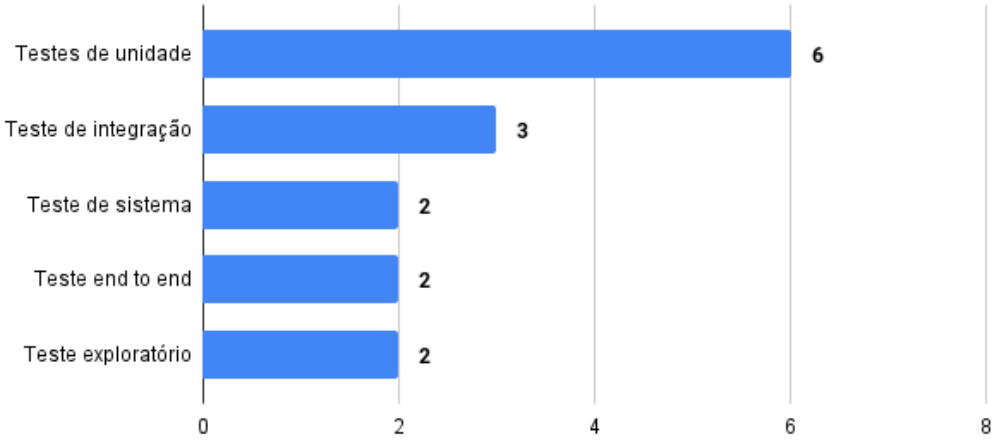

Figura 4.4. Técnicas de teste de software conhecidas pelos participantes.

Através destes números pode-se concluir que os estudantes não possuem experiência prévia com a linguagem de programação Java. Porém, quanto a não saber Java, mas saber programação, é conveniente para o estudo, dado que os erros que eles cometerão provavelmente serão alinhados com as hipóteses de programador competente e, espera-se, referentes a detalhes da linguagem (e não quanto ao entendimento do algoritmo em si). A maioria tem contato com testes de unidade, reduzindo a chance de eles terem dificuldade com a criação do caso de teste ao invés de com a técnica de análise de mutante.

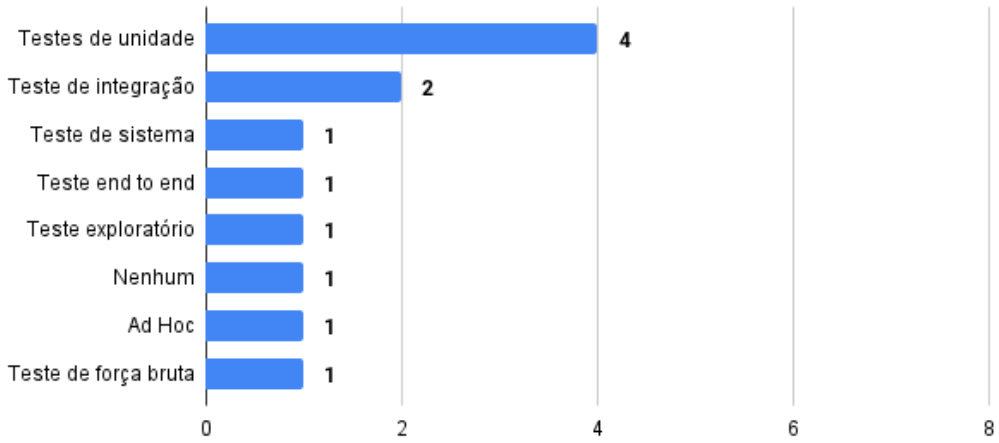

Figura 4.5. Técnicas de teste de software que os participantes possuem mais experiência.

4.3 Analise dos códigos desenvolvidos

Continuando para a segunda parte do experimento, em que os participantes tiveram que implementar o algoritmo do bubble sort e a estrutura de dados fila, apresenta-se a análise dos programas produzidos. A primeira implementação teve a participação de todos os participantes que participaram da primeira etapa, ou seja, 7 implementações, e foi realizada durante a aula. Foram utilizados os testes de unidade referência que alcançaram 100% da cobertura de código medido pelo JaCoCo, em que o primeiro teste verificava a ordenação de um vetor com mais de um elemento, e outro verificava a ordenação de um vetor com tamanho de um elemento. Assim, pode-se testar todas as implementações feitas pelos estudantes. Das 7 implementações, 4 passaram por todos os testes, ou seja, 3 algoritmos do bubble sort foram desenvolvidas de forma errada como apresentado na Tabela 4.1.

Tabela 4.1. Resultado de cada participante no Bubble Sort.

	Testes de unidade	
Bubble Sort	Ordenação	Retorna o único numero de um array
Participante 1	Passou	Passou
Participante 2	Passou	Passou
Participante 3	Arrays first differed at element [0]	Passou
Participante 4	ArrayIndexOutOfBoundsException	Passou
Participante 5	Passou	Passou
Participante 6	Passou	Passou
Participante 7	Arrays first differed at element [0]	Passou

Fonte: Autoria própria

Na segunda implementação os participantes desenvolveram a estrutura de dados no período fora da aula, como atividade. Novamente, todos os participantes realizaram essa atividade. Seguindo o mesmo método de avaliação da primeira implementação, os códigos foram testados com 6 teste de unidades que cobrem 100% do código segundo o JaCoCo que foram desenvolvidos para o desenho experimental. O primeiro teste verificava se a fila está vazia, o segundo verificava se o primeiro

elemento da fila está vazio, o terceiro verificava se quando removido um elemento a fila está vazia, o quarto verificava a operação de remover um elemento na fila, o quinto verificava a operação de adicionar um elemento na fila e o último verificava a operação de remover um elemento quando só tinha um elemento na fila.

A Tabela 4.2 apresenta os resultados de cada teste para cada participante, onde 3 participantes desenvolveram seus códigos erroneamente quebrando 4 testes de 5, representado pela sigla NPE, de $NullPointerException$. Outros 3 participantes tiveram suas implementações feitas corretamente, passando em todos os testes, sendo que um desses participantes implementou um método a mais no qual retornava o último elemento da fila. Além disso, houve um caso em que o participante não seguiu as interfaces que lhe foi entregue, porém adaptando os testes unitários para o seu código, verificou-se que seu código estava certo, só não constava o método em que a fila estava vazia e também não tinha o método onde retornava o primeiro elemento da fila.

Tabela 4.2. Resultado de cada participante na Fila.

	Participantes						
Testes	1	2	3	4	5	6	7
Verifica se a fila está vazia	Passou	Passou	Passou	Passou	Passou	Passou	Não implementado
Verifica se o primeiro elemento está vazio	Passou	Passou	NPE	NPE	Passou	NPE	Não implementado
Verifica se a fila está vazia quando utiliza a operação de dequeue	Passou	Passou	NPE	NPE	Passou	NPE	Passou
Verifica a operação de dequeue	Passou	Passou	NPE	NPE	Passou	NPE	Passou
Verifica a operação de enqueue	Passou	Passou	NPE	NPE	Passou	NPE	Passou
Verifica a operação de dequeue com um elemento na fila	Passou	Passou	NPE	NPE	Passou	NPE	Passou

Fonte: Autoria própria

No desenvolvimento do código de ordenação 4 dos 7 participantes alcançaram êxito sua implementação. Esse número também se refletiu no código da implementação da fila. Analisando o progresso dos participantes: os participantes 1, 2 e 5 implementaram ambos os códigos perfeitamente, já com os participantes 3 e 4 ocorreu o inverso, implementaram ambas de forma errônea. O participante 6 implementou o código do *bubble sort* corretamente, porém implementou errado o código da fila. Vale ressaltar que a segunda implementação é mais complexa que a primeira, o que pode ter sido um fator para ele não ter conseguido. O participante 7 não conseguiu implementar o algoritmo de ordenação corretamente, enquanto na estrutura de dados, ele a implementou corretamente, embora de forma parcial, mostrando um possível avanço no aprendizado.

4.4 Análise dos testes de unidade

A partir dos códigos implementados pelos participantes, eles mesmo criaram suas partidas, sendo uma para o código de *bubble sort* e outra para a fila no modo de jogo fácil, onde tanto o defensor consegue ver os códigos alterados para as criações de mutantes quanto o atacante consegue ver o código do teste de unidade e suas linhas de cobertura. Esse modo de jogo tem como objetivo não estragar a experiência deles durante o aprendizado de teste de unidade, pois como a maioria deles não tem experiência com a linguagem Java, o modo difícil iria dificultar o aprendizado, desmotivando-os a jogar.

Durante os jogos, o Code Defenders salva toda tentativa de criação de teste de unidade mesmo quando o teste não compila. Devido a isso, a plataforma salvou 81 tentativas, divididas em 42 para a primeira atividade e 39 para a segunda. Dessas 81 tentativas, 33 testes foram compilados e enviados para a partida, de forma que 10 foram para o *bubble sort* e 23 para a fila.

Os 10 testes de unidade referentes ao código do *bubble sort* foram desenvolvidos em 3 partidas por um participante em cada: na primeira foram desenvolvidos 5 testes, na segunda 4 e no última 1. Em todos as partidas foram alcançados 100% de cobertura de instrução e 100% de cobertura de desvio segundo o JaCoCo. Analisando os testes de unidade agrupados por participante com a ferramenta Pit, que gera um relatório a partir da geração de mutantes para o código, todos os participantes tiveram êxito na criação de seus testes e alcançaram um total de 85% de cobertura de mutação.

Já a implementação dos testes na fila foi dividida em 4 partidas, sendo 4 testes para a primeira, 9 para a segunda, 9 para a terceira e 1 para a quarta, totalizando 23 casos de teste.

Na primeira partida, todos os 4 testes de unidade tiveram sua implementação correta, porém 2 falharam. Através do JaCoCo foi medida sua cobertura de instrução, que foi apenas de 57% e 40% de para a sua cobertura de desvio, já sua cobertura de mutação alcançou 63%, segundo Pit.

Na segunda partida, dos 9 testes, 5 falharam, 1 não contém asserção e 3 passaram. Com isso o resultado dos testes que passaram foi satisfatório já que alcançou um total de 92% de cobertura de instrução, 70% de cobertura de desvio e 75% de cobertura de mutação.

Na terceira partida tivemos 2 jogadores defendendo. O primeiro jogador fez apenas um teste, no qual passou e teve um total de cobertura de instrução de 26%, cobertura de desvio de 20% e cobertura de mutação de 33%. O segundo defensor implementou 8 testes de unidade e alcançou resultados satisfatórios de cobertura: 79% para instrução, 80% para desvio e 77%. Na última partida foi realizado apenas um teste de unidade que teve um número expressivo de 57% de cobertura de instrução, 40% de cobertura de desvio e 63% de cobertura de mutação.

Para o desenvolvimento dos testes de unidade, o Code Defenders salvou todas as tentativas compiladas ou não, dessa forma foram realizadas 81 tentativas, 33 foram compiladas e enviadas as partidas. Esse número representa que 40,74% das tentativas estavam corretas, analisando alguns testes pode-se observar que os estudantes enfrentaram dificuldades em relação a linguagem de programação já a maioria deles nunca tinham programado em Java.

Em relação a qualidade dos testes de unidade desenvolvidos para o algoritmo de ordenação, apenas 3 participantes desenvolveram testes que foram enviados às partidas, porém todos eles conseguiram a cobertura completa de linha de código. Analisando os testes na segunda implementação, 4 participantes enviaram testes compilados para suas partidas, porém apenas 2 obtiveram taxas de cobertura de instrução, desvio e mutação maiores que 70%. Vale ressaltar também, que um participante obteve um número expressivo de 57% de cobertura de instrução, 40% de cobertura de desvio e 63% de cobertura de mutação com apenas um teste.

4.4.1 Evolução dos testes

Vale destacar o ponto que a plataforma salva todo progresso do jogador criando seus testes. Com isso pode-se observar quais os principais erros que os participantes cometeram. Observando um participante criando seu teste para o bubble sort, o Código 4.3 apresenta sua primeira tentativa da criação do teste.

```
1     public void test() throws Throwable {
2
3         array1 = array[1,3,2,11,4];
4         BubbleSort = new BubbleSort;
5         array2 = BubbleSort(array1).Ordena;
6         array3 = [1,2,3,4,11];
7         assertArrayEquals(array3,array2);
8
9     }
```

Código 4.3. Primeiro teste criado pelo participante.

É possível concluir que o participante, desde o princípio estava utilizando o JUnit de maneira correta para criar o teste e o que testar, porém ele estava cometendo erros de sintaxe no Java nos quais desenvolvedores que não o utilizam podem cometer. O Código 4.3 apresenta que ele estava declarando um *array* de números inteiros de forma errada, além de estar instanciando o objeto também de forma equivocada. Depois de 16 tentativas o participante conseguiu superar o problema e enviar o teste correto a partida.

Para afirmar que esse problema não foi apenas isolado, outro participante o enfrentou da mesma forma. Ele não estava instanciando a classe da forma correta, conforme o Código 4.4.

```
1     public void test() throws Throwable {
2
3         int[] valores = {43, 76, 22, 41, 9, 14, 2};
4         int[] esperado = {2, 9, 14, 22,41, 43, 76};
5
6         BubbleSort bubble = new BubbleSort();
7
```

```
8           bubble.bubbleSort(valores);
9           bubble.Ordena();
10
11          assertArrayEquals(esperado, valores);
12      }
```

Código 4.4. Teste criado de forma incorreta por outro participante.

Além disso, o participante achava que o problema era na forma de inicialização dos *arrays* de números inteiro, e continuou alterando essas inicializações, mesma estando certo, como no Código 4.5.

```
1       public void test() throws Throwable {
2
3           int valores[]  = {43, 76, 22, 41, 9, 14, 2};
4           int esperado[]  = {2, 9, 14, 22,41, 43, 76};
5
6           BubbleSort bubble = new BubbleSort();
7
8           bubble.bubbleSort(valores);
9           bubble.Ordena();
10
11          assertArrayEquals(esperado, valores);
12      }
```

Código 4.5. Teste criado de forma incorreta novamente pelo mesmo participante, porém com outro tipo de erro.

Isso representa que ele não estava prestando atenção no erro que a plataforma devolve ao jogador, quando existe erro de compilação. Além disso, ele tentou 8 vezes e desistiu, não enviando nenhum teste de unidade correto para a partida.

Analisando outro teste de unidade, agora na fila, o participante fez o teste correto, porém ele utilizou o $AssertEquals$ para um valor literal booleano, apresentado no Código 4.6.

```
1       public void test() throws Throwable {
2               Queue queue = new Queue();
3               boolean empty = queue.isEmpty();
4               assertEquals(empty,true);
5       }
```

Código 4.6. Teste utilizando assertEquals.

Porém, para criação de testes booleanos existem as asserções $assertTrue$ e $assertFalse$. Cabe destacar que, se esse teste tivesse sido realizado em uma IDE Java, ela iria sugerir esta mudança, mas vale ressaltar que houve uma falha do participante em utilizar a documentação do framework de teste.

4.5 Análise do Code Defenders

A última etapa do experimento os participantes responderam ao formulário MEEGA+ no qual contia 35 perguntas, 3 perguntas abertas perguntando sobre a experiência dentro do jogo e 32 perguntas de múltipla escolha com respostas em escala Likert, ou seja, seu formato era de discordo totalmente, discordo parcialmente indiferente, concordo parcialmente e concordo totalmente. Essas perguntas têm como objetivo avaliar a qualidade dos elementos de jogos no Code Defenders.

Para classificar o jogo na escala MEEGA+, foi utilizado um script fornecido na documentação do MEEGA+ e utilizado o software estatístico R para o cálculo. Através do software é calculado a média das pontuações fornecidas de todos os participantes que avaliaram o jogo, aplicando a Teoria de Resposta ao Item (TRI). Após o cálculo da média, foi realizada uma transformação desses escores em uma escala (50.15), aplicando a formula no escore médio:

$$\Theta(50.15) = 50 + 15 * \Theta(0.1) * (0{,}294098133)$$

$$\Theta(50.15) = 49,559$$

onde o valor 0,294098133 é a média, obtendo assim o resultado de 49,559.

Com base neste valor final, o nível de qualidade do Code Defenders foi em avaliado no nível de qualidade boa, em que o jogo às vezes apresenta atividades desafiadoras, oferecendo novos desafios para os estudantes. Ele fornece atenção moderadamente focada aos jogadores, embora os estudantes não se esqueçam de seus arredores. Às vezes, o jogo também proporciona sentimentos de confiança e satisfação aos jogadores. Frequentemente o jogo apresenta momentos de interação social e diversão entre os jogadores. Frequentemente, o jogo é considerado relevante para os interesses dos estudantes e, geralmente, os estudantes reconhecem que o conteúdo do jogo está relacionado ao curso. Frequentemente, o jogo contribui de forma eficiente para o aprendizado do estudante. Em termos de usabilidade, o jogo costuma ter regras claras e é fácil de jogar, embora, normalmente, não apresente um design totalmente atraente (PETRI et al., 2019).

Analisando as perguntas abertas sobre a experiência do jogador na plataforma, a primeira questionava qual o fator que mais gostou no jogo, e teve as seguintes respostas:

- “A parte dos ataques me prendeu mais no jogo.”
- “Achei interessante para quem cria os casos de teste, é desafiador criar o máximo de casos de teste para que nenhum erro de um atacante passe pela sua defesa.”
- “O conceito criativo.”

Na primeira resposta é possível observar que o participante preferiu jogar de atacante do que defensor. O motivo pode se entender que, quando participando de um jogo na plataforma pela primeira vez, jogar de atacante é mais tentador por ser mais fácil, porém não cumpre com o propósito do jogo, que é aprimorar a habilidade de desenvolver teste de unidade. Já a segunda resposta é o

oposto da primeira: o participante realmente se mostrou interessado em desenvolver teste de unidade dentro das partidas nos quais participou. Finalmente, a última resposta é um comentário sobre a jogabilidade na plataforma, pois a abordagem de teste de mutação pode ser vista como algo inédito para todos os participantes.

A segunda pergunta questionava aos participantes sobre o que poderia ser melhorado no jogo e teve as seguintes respostas:

- "Especificação e uma explicação melhor sobre os testes."
- "Senti um pouco de dificuldade em entender o que fazer após um ataque, na primeira vez acabei clicando em *endgame*, para voltar para a *home*."
- "Regras mais claras, talvez um exemplo prático."

A primeira resposta pode ser interpretada pela falta de material na plataforma de testes de unidades, e, realmente na plataforma não tem nada a respeito. Já pensando nesse problema antes de aplicar o experimento, foi passado aos participantes alguns materiais para se consultar durante as atividades, porém fora da plataforma, e pouco didático. Na segunda resposta o participante ficou confuso após jogar como atacante, ao invés de voltar para a *home* clicando no ícone do jogo na *side bar*, ele clicou no botão de encerrar partida, no qual estava mais visível, já que o botão de voltar para a *home* o jogador precisava adivinhar que clicando no nome do jogo iria voltar, pouco *user friendly*. Na última resposta pode se fazer uma relação com a primeira, visto que é a falta de material didático sobre as regras dos jogos, apesar de serem apresentados para eles durante a aula, ainda houve uma deficiência na parte da plataforma.

A última pergunta, não obrigatória, era se o participante gostaria de fazer algum comentário sobre qualquer ponto que desejasse. Mesmo não obrigatória, ela teve duas respostas:

- "O jogo ajuda no entendimento dos testes, mas tive um pouco de dificuldade."
- "Neste jogo, eu pensava que teria muita dificuldade, principalmente por não ter muita afinidade com Java, mas acabou sendo interessante, consegui fazer os códigos e os casos de teste, sem tanta dificuldade. Foi possível entender como fazer casos de teste e a ter mais contato com a linguagem Java."

Na primeira resposta, o participante concordou que o Code Defenders ajudou no desenvolvimento de teste de unidade, porém teve dificuldades. Já na segunda, o participante achou que teria mais dificuldades por se tratar de uma linguagem pelo qual ele não usa, porém ele superou a dificuldade e ainda destacou como pontos positivos o desenvolvimento dos testes e ter um pouco mais de prática com o Java.

4.6 Considerações finais

Após a aplicação das duas atividades envolvendo o desenvolvimento do algoritmo de ordenação e da estrutura de dados no Code Defenders para 7 estudantes da disciplina de Engenharia de Software do

curso de Ciência da Computação, como descrito neste capítulo, foram obtidos os códigos referentes às implementações das duas atividades e seus respectivos testes de unidade.

Para avaliar a qualidade do Code Defenders como ambiente para ensino de teste de software integrado à programação os participantes responderam um formulário com base no método do MEEGA+. A partir dos resultados pode-se concluir que o seu nível de qualidade é boa, apresentando às vezes atividades desafiadoras, oferecendo novos desafios para os estudantes. Frequentemente o jogo é relevante para os interesses dos estudantes e frequentemente contribui de forma eficiente para o aprendizado.

Na perspectiva do aprendizado da linguagem de programação, a maioria dos participantes logrou sucesso, conseguindo desenvolver tanto o algoritmo de ordenação quanto a estrutura de dados e também foi apontado que as atividades de teste auxiliaram no aprendizado.

No desenvolvimento dos testes de unidade, os estudantes que enviaram testes à plataforma durante a primeira parte do estudo conseguiram êxito alcançar os 100% de cobertura de instrução e desvio. Para a segunda parte utilizando um algoritmo mais complexo como o da fila, foram obtidos bons números, onde dois estudantes conseguiram cobertura relevantes de mais de 75%, outros dois estudantes conseguiram cobertura maior que 50% e apenas um com cobertura de 26%.

5 CONCLUSÕES

Para o desenvolvimento de software com mais qualidade é necessário promover o uso de técnicas de teste de software integradas com atividades de programação. Porém, os estudantes estão tendo falta de motivação no aprendizado de teste devido à dificuldade dos educadores em trazerem exercícios realistas e apresentar um curso atualizado.

Uma solução potencial é abordar gamificação junto ao aprendizado de teste de software. Considerando isso, o desenvolvimento do presente estudo possibilitou a análise de ferramentas que abordam gamificação e o aprendizado de teste de software integrado com atividades de programação. Com o resultado, o Code Defenders foi a ferramenta encontrada para avaliar o efeito da integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos.

O Code Defenders é uma aplicação Web que implementa uma abordagem de gamificação para teste de mutação, onde os jogadores podem assumir o papel de atacante ou defensor. O atacante é responsável por criar mutantes, ou seja, variações simples do programa. O defensor é responsável por criar casos de teste que matam esses mutantes. O time que tiver maior pontuação vence.

Durante o estudo foi proposta uma nova abordagem com o objetivo de trazer novas funcionalidades para o Code Defenders e posteriormente serem avaliadas. Na primeira funcionalidade proposta, os jogadores podem criar o programa Java dentro na plataforma ao invés de apenas submeter o arquivo com o código escrito. Essa funcionalidade tem como objetivo promover o aprendizado integrado de programação na linguagem Java e em teste de software. A segunda funcionalidade refere-se a jogabilidade, em que cada time terá uma pontuação de custo total para serem gastos com atividades na plataforma (criar um mutante ou criar um caso de teste). Essa funcionalidade tem como foco incentivar os jogadores a refletir para criar mutantes e criar casos de teste, evitando a tentativa e erro.

Foi criado um conjunto de dados, considerando problemas típicos de disciplinas de algoritmo e estrutura de dados, e de soluções de referência, tanto de programas quanto de casos de teste com respeito a critérios de fluxo de controle (comandos e desvios) e análise de mutantes. Esse conjunto abordou os seguintes tópicos: manipulação de caracteres (strings), números primos, ordenação (*bubble sort*), fila, fila circular, lista encadeada, lista encadeada com cache e pilha.

A partir do conjunto de dados foram selecionados dois algoritmos para serem aplicados ao estudo de caso com os estudantes do curso de Ciência da Computação na disciplina de Engenharia de Software. A primeira implementação proposta foi o *bubble sort*, no qual eles implementaram durante a aula síncrona. A segunda implementação proposta foi um algoritmo mais complexo para implementar, a fila. O estudo contou com 7 participantes e, a partir dos códigos de programa e testes de unidade, foram analisados para serem respondidas as questões de pesquisa.

Para a primeira questão de pesquisa, a qual se refere à qualidade do Code Defenders para o ensino de teste de software, foi concluído que a plataforma possui uma boa qualidade de gamificação

através do MEEGA+, apesar de obter apenas 3 respostas dos 7 participantes. Observou-se que a ferramenta às vezes apresenta atividades desafiadoras, oferecendo novos desafios para os estudantes. Porém, não foi possível concluir se as duas novas funcionalidades implementadas na ferramenta interferiram para melhorar ou não a qualidade do Code Defenders.

Para responder a segunda questão de pesquisa sobre a integração de atividades de programação no ensino de teste de software em um ambiente com elementos do jogos, verificando-se se isso permite melhorar o aprendizado de teste de software, pode-se notar, através dos códigos de programa e dos testes de unidades desenvolvidos pelo estudantes, que eles foram capazes de alcançar o objetivo proposto pela atividade. Além disso os estudantes escreveram comentários no formulário de avaliação no MEEGA+, informando que o jogo auxiliou no entendimento de teste, e também destacou-se um ponto positivo de ter contato com a linguagem Java.

Dado o objetivo do presente estudo em avaliar o efeito da integração de atividades de programação no ensino de teste de software em um ambiente com elementos de jogos, a ferramenta apresentou-se como como uma alternativa a estudantes da computação que buscam aprender ou melhorar a habilidade de desenvolvimento de testes de unidade e praticar o desenvolvimento da linguagem de programação Java. O ambiente oferece elementos de jogos para o estudante se sentir fora de uma atividade comum à sala de aula, trazendo uma experiência mais divertida a eles.

Após o estudo de caso, pode-se notar algumas limitações da ferramenta, como não permitir alteração do código do programa durante a criação da partida ou enquanto outros jogadores testam ou criam mutantes, não contabilizar erros de compilação e o editor de texto para programar ser limitado.

Para trabalho futuros podem ser implementadas algumas alterações na ferramenta com base na suas limitações como: permitir a alteração do programa sob teste, para que o estudante corrija seu código durante a partida, a fim de progredir no aprendizado da linguagem; contabilizar erros de compilação quanto ao custo para que os jogadores não fiquem tentando a criação de testes e mutantes incessantemente; e criar um agente que gera mutantes com operadores de mutação definidos conforme nível de conhecimento e competência do estudante, para que mesmo sem um atacante na partida, um defensor possa testar seus testes de unidade. Além disso, devem ser planejados estudos com mais participantes, permitindo a obtenção de mais dados e a avaliação mais rigorosa dos resultados quanto às questões de pesquisa.

REFERÊNCIAS

ANICHE, Maurício; HERMANS, Felienne; DEURSEN, Arie van. Pragmatic software testing education. In: **Proceedings of the 50th ACM Technical Symposium on Computer Science Education**. New York, NY, EUA: ACM, 2019. p. 414–420. ISBN 978-1-4503-5890-3.

Apache Software Foundation. Programa de computador, **Apache Commons Collection, Node Caching Linked List**. 2002. Disponível em: https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/list/NodeCachingLinkedList.html.

Apache Software Foundation. Programa de computador, **Apache Commons**. 2019. Disponível em: https://commons.apache.org/.

BISHOP, Judith; HORSPOOL, R. Nigel; XIE, Tao; TILLMANN, Nikolai; HALLEUX, Jonathan de. Code Hunt: Experience with coding contests at scale. In: **37th International Conference on Software Engineering**. Piscataway, NJ, EUA: IEEE, 2015. p. 398–407.

CARRINGTON, David. Teaching software testing. In: **Proceedings of the 2nd Australasian Conference on Computer Science Education**. New York, NY, USA: ACM, 1997. p. 59–64. ISBN 978-0-89791-958-6.

CLEGG, Benjamin; ROJAS, José Miguel; FRASER, Gordon. Teaching software testing concepts using a mutation testing game. In: **39th International Conference on Software Engineering: Software Engineering and Education Track**. New Jersey, NJ, EUA: IEEE, 2017. p. 33–36.

COLES, Henriy et al. **PIT**. 2010. Programa de computador. Disponível em: http://pitest.org/.

DEMILLO, Richard A.; LIPTON, Richard J.; SAYWARD, Frederick G. Hints on test data selection: Help for the practicing programmer. **IEEE Computer Society Press**, ACM, Washington, DC, EUA, v. 11, n. 4, p. 34–41, abr. 1978.

DETERDING, Sebastian; DIXON, Dan; KHALED, Rilla; NACKE, Lennart. From game design elements to gamefulness: Defining "gamification". In: **15th International Academic MindTrek Conference: Envisioning Future Media Environments**. New York, NY, EUA: ACM, 2011. p. 9–15. ISBN 978-1-4503-0816-8.

EDWARDS, Stephen H. Using software testing to move students from trial-and-error to reflection-in-action. In: . New York, NY, USA: Association for Computing Machinery, 2004. p. 26–30. ISBN 1581137982.

FRASER, Gordon. Gamification of software testing. In: **Proceedings of the 12th International Workshop on Automation of Software Testing**. Piscataway, NJ, EUA: IEEE, 2017. p. 2–7. ISBN 978-1-5386-1548-5.

FRASER, Gordon; GAMBI, Alessio; KREIS, Marvin; ROJAS, José Miguel. Gamifying a software testing course with Code Defenders. In: **Proceedings of the 50th ACM Technical Symposium on Computer Science Education**. New York, NY, EUA: ACM, 2019. p. 571–577. ISBN 978-1-4503-5890-3.

FRASER, Gordon; GAMBI, Alessio; ROJAS, José Miguel. A preliminary report on gamifying a software testing course with the code defenders testing game. In: **Proceedings of the 3rd European Conference of Software Engineering Education**. New York, NY, EUA: ACM, 2018. p. 50–54. ISBN 978-1-4503-6383-9.

FRASER, Gordon; GAMBI, Alessio; ROJAS, José Miguel. Teaching software testing with the Code Defenders testing game: Experiences and improvements. In: **1st International Software Testing Education Workshop**. [S.l.]: IEEE, 2020. p. 461–464.

HAKULINEN, Lasse. Using serious games in computer science education. In: **Proceedings of the 11th Koli Calling International Conference on Computing Education Research**. New York, NY, USA: Association for Computing Machinery, 2011. p. 83–88. ISBN 9781450310529.

HOFFMANN, Marc R.; MANDRIKOV, Evgeny; FRIEDENHAGEN, Mirko; LIBA, Radek; BECK, Christoph; JANICZAK, Brock et al. **JaCoCo – Java Code Coverage Library**. 2009. Programa de computador. Disponível em: https://www.jacoco.org/jacoco/.

HUNICKE, Robin; LEBLANC, Marc; ZUBEK, Robert. MDA: A formal approach to game design and game research. In: FU, Dan; HENKE, Stottler; ORKIN, Jeff (Ed.). **AAAI Workshop on Challenges in Game Artificial Intelligence**. Palo Alto, CA, EUA: AAAI, 2004. p. 1–5. ISBN 978-1-57735-205-1.

JESUS, Gabriela Martins de; FERRARI, Fabiano Cutigi; PORTO, Daniel de Paula; FABBRI, Sandra Camargo Pinto Ferraz. Gamification in software testing: A characterization study. In: **Proceedings of the III Brazilian Symposium on Systematic and Automated Software Testing**. [S.l.]: Association for Computing Machinery, 2018. p. 39–48. ISBN 9781450365550.

KATS, Lennart C.L.; VERMAAS, Rob; VISSER, Eelco. Integrated language definition testing: Enabling test-driven language development. In: **Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications**. New York, NY, USA: Association for Computing Machinery, 2011. p. 139–154. ISBN 9781450309400.

LUXTON-REILLY, Andrew; SIMON; ALBLUWI, Ibrahim; BECKER, Brett A.; GIANNAKOS, Michail; KUMAR, Amruth N.; OTT, Linda; PATERSON, James; SCOTT, Michael James; SHEARD, Judy; SZABO, Claudia. Introductory programming: A systematic literature review. In: **Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education**. New York, NY, USA: Association for Computing Machinery, 2018. p. 55–106. ISBN 9781450362238.

MONCLAR, Rafael Studart; SILVA, Marcelo Arêas; XEXéO, Geraldo. Jogos com propósito para o ensino de programação. In: **XVII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2018)n**. [S.l.]: SBC, 2018. p. 1–9. ISSN 2179-2259.

PETRI, Giani; WANGENHEIM, Christiane Gresse von. How to evaluate educational games: a systematic literature review. **Journal of Universal Computer Science**, J.UCS Consortium, Gras,Áustria, v. 22, n. 7, p. 992–1021, jul. 2016. ISSN 0948-695x.

PETRI, Giani; WANGENHEIM, Christiane Gresse von. How games for computing education are evaluated? a systematic literature review. **Computers & Education**, Elsevier, Países Baixos, v. 107, p. 68–90, abr. 2017. ISSN 0360-1315.

PETRI, Giani; WANGENHEIM, Christiane Gresse von. A method for the evaluation of the quality of games for computing education. In: ARAújO, Aletéia Patrícia Favacho de; SILVEIRA, Ismar Frango (Ed.). **Anais dos Workshops do VIII Congresso Brasileiro de Informática na Educação (WCBIE 2019) – Concurso Alexandre Direne de Teses, Dissertações e TCCs em Informática na Educação**. Porto Alegre, RS, Brasil: SBC, 2019. p. 951–960. ISSN 2316-8889.

PETRI, Giani; WANGENHEIM, Christiane Gresse von; BORGATTO, Adriano Ferreti. MEEGA+: Um modelo para a avaliação de jogos educacionais para o ensino de computação. **Revista Brasileira de Informática na Educação (RBIE)**, Comissão Especial de Informática na Educação (CEIE) – Sociedade Brasileira de Computação (SBC), Porto Alegre, RS, Brasil, v. 27, n. 3, p. 52–81, set. 2019. ISSN 1414-5685. Disponível em: http://www.br-ie.org/pub/index.php/rbie/.

PITEIRA, Martinha; COSTA, Carlos. Computer programming and novice programmers. In: **Proceedings of the Workshop on Information Systems and Design of Communication**. New York, NY, USA: Association for Computing Machinery, 2012. p. 51–53. ISBN 9781450312943.

ROJAS, Jose M.; FRASER, Gordon. Code Defenders: A mutation testing game. In: **9th International Conference on Software Testing, Verification and Validation Workshops – International Workshop on Mutation Analysis**. [S.l.: s.n.], 2016. p. 162–167. ISBN 978-1-5090-3675-2.

ROJAS, José Miguel; FRASER, Gordon. Teaching mutation testing using gamification. In: **European Conference of Software Engineering Education 2016 (ECSEE)**. [S.l.]: Shaker Publishing, 2016. p. 1–5.

ROJAS, José Miguel; FRASER, Gordon. Teaching software testing with a mutation testing game. In: CHURCH, Luke (Ed.). **27th Annual Workshop Psychology of Programming Interest Group 2016 (PPIG)**. [S.l.]: PPIG, 2016. p. 290–293.

ROJAS, José Miguel; WHITE, Thomas; CLEGG, Benjamin; FRASER, Gordon. Code Defenders: Crowdsourcing effective tests and subtle mutants with a mutation testing game. In: **39th International Conference on Software Engineering**. New Jersey, NJ, EUA: IEEE, 2017. p. 677–688.

SCATALON, Lilian Passos; CARVER, Jeffrey C.; GARCIA, Rogério Eduardo; BARBOSA, Ellen Francine. Software testing in introductory programming courses: A systematic mapping study. In: **Proceedings of the 50th ACM Technical Symposium on Computer Science Education**. New York, NY, EUA: ACM, 2019. p. 421–427. ISBN 978-1-4503-5890-3.

TECHNICAL Symposium on Computer Science Education. New York, NY, EUA: ACM, 2019. ISBN 978-1-4503-5890-3.

TILLMANN, Nikolai; BISHOP, Judith; HORSPOOL, Nigel; PERELMAN, Daniel; XIE, Tao. Code Hunt: Searching for secret code for fun. In: **7th International Workshop on Search-Based Software Testing**. New York, NY, EUA: ACM, 2014. p. 23–26. ISBN 978-1-4503-2852-4.

TILLMANN, N.; HALLEUX, J. De; XIE, Tao. Pex4Fun: Teaching and learning computer science via social gaming. In: **24th Conference on Software Engineering Education and Training**. Piscataway, NJ, EUA: IEEE Computer Society, 2011. p. 546–548. ISBN 9781457703492. ISSN 1093-0175.

TILLMANN, Nikolai; HALLEUX, Jonathan De; XIE, Tao; GULWANI, Sumit; BISHOP, Judith. Teaching and learning programming and software engineering via interactive gaming. In: **35th International Conference on Software Engineering**. Piscataway, NJ, EUA: IEEE, 2013. p. 1117–1126. ISBN 978-1-4673-3076-3.

TILLMANN, Nikolai; HALLEUX, Jonathan de; XIE, Tao. Transferring an automated test generation tool to practice: From Pex to Fakes and Code Digger. In: **29th ACM/IEEE International Conference on Automated Software Engineering**. New York, NY, EUA: ACM, 2014. p. 385–396. ISBN 978-1-4503-3013-8.

TILLMANN, Nikolai; HALLEUX, Jonathan de; XIE, Tao; BISHOP, Judith. Code Hunt: Gamifying teaching and learning of computer science at scale. In: **1st ACM Conference on Learning @ Scale Conference**. New York, NY, EUA: ACM, 2014. p. 221–222. ISBN 978-1-4503-2669-8.

TILLMANN, Nikolai; HALLEUX, Jonathan de; XIE, Tao; BISHOP, Judith. Constructing coding duels in Pex4Fun and Code Hunt. In: **2014 International Symposium on Software Testing and Analysis**. New York, NY, EUA: ACM, 2014. p. 445–448. ISBN 978-1-4503-2645-2.

TODA, Armando M.; KLOCK, Ana C. T.; OLIVEIRA, Wilk; PALOMINO, Paula T.; RODRIGUES, Luiz; SHI, Lei; BITTENCOURT, Ig; GASPARINI, Isabela; ISOTANI, Seiji; CRISTEA, Alexandra I. Analysing gamification elements in educational environments using an existing gamification taxonomy. **Smart Learning Environments**, Springer, v. 6, p. 6:1–6:14, dez. 2019.

VALLE, Pedro Henrique Dias; ROCHA, Rafaela Vilela; MALDONADO, José Carlos. Testing game: An educational game to support software testing education. In: MALDONADO, Jose Carlos; FERRARI, Fabiano Cutigi (Ed.). **Proceedings of the 31st Brazilian Symposium on Software Engineering**. New York, NY, EUA: ACM, 2017. p. 289–298. ISBN 978-1-4503-5326-7.

VALLE, Pedro Henrique Dias; TODA, Armando Maciel; BARBOSA, Ellen Francine; MALDONADO, José Carlos. Educational games: A contribution to software testing education. In: **2017 IEEE Frontiers in Education Conference (FIE)**. Piscaway, NY, EUA: IEEE, 2017. p. 1–8. ISBN 978-1-5090-4920-2.

VIHAVAINEN, Arto; AIRAKSINEN, Jonne; WATSON, Christopher. A systematic review of approaches for teaching introductory programming and their influence on success. In: **Proceedings**

of the Tenth Annual Conference on International Computing Education Research. New York, NY, USA: Association for Computing Machinery, 2014. p. 19–26. ISBN 9781450327558.

XIE, Tao; BISHOP, Judith; TILLMANN, Nikolai; HALLEUX, Jonathan de. Gamifying software security education and training via secure coding duels in Code Hunt. In: **2015 Symposium and Bootcamp on the Science of Security**. New York, NY, EUA: ACM, 2015. p. 26:1–26:2. ISBN 978-1-4503-3376-4.

XIE, Tao; TILLMANN, Nikolai; HALLEUX, Jonathan de. Educational software engineering: Where software engineering, education, and gaming meet. In: **3rd International Workshop on Games and Software Engineering: Engineering Computer Games to Enable Positive, Progressive Change**. Piscataway, NJ, EUA: IEEE, 2013. p. 36–39. ISBN 978-1-4673-6263-4.