UNIVERSIDADE FEDERAL DO MARANHÃO

Programa de Pós-Graduação em Ciência da Computação

José Nunes de Oliveira Neto

BDD-I: Especificações de Uso de Behavior-Driven Development no Desenvolvimento de Jogos Independentes

São Luís
2019

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

José Nunes de Oliveira Neto

BDD-I: Especificações de Uso de Behavior-Driven Development no Desenvolvimento de Jogos Independentes

São Luís
2019

José Nunes de Oliveira Neto

BDD-I: Especificações de Uso de Behavior-Driven Development no Desenvolvimento de Jogos Independentes

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Maranhão como requisito parcial para a obtenção do grau de MESTRE em Ciência da Computação.

Orientador: Davi Viana dos Santos

Doutor em Informática – UFMA

Co-orientador: Rafael Fernandes Lopes

Doutor em Engenharia Elétrica – UFMA

São Luís

2019

José Nunes de Oliveira Neto

BDD-I: Especificações de Uso de Behavior-Driven Development no Desenvolvimento de Jogos Independentes

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por José Nunes de Oliveira Neto e aprovada pela comissão examinadora.

Aprovada em 19 de Julho de 2019

BANCA EXAMINADORA

Davi Viana dos Santos (orientador)

Doutor em Informática – UFMA

Rafael Fernandes Lopes (co-orientador)

Doutor em Engenharia Elétrica – UFMA

Luis Jorge Enrique Rivero Cabrejos

Doutor em Informática – UFMA

Eveline de Jesus Viana Sá

Doutora em Engenharia Elétrica e Computação – IFMA

Rodrigo Pereira dos Santos

Doutor em Engenharia de Sistemas e Computação – UNIRIO

Ao desenvolvimento de jogos no Brasil...

Resumo

A indústria de jogos lucrou, em 2018, cerca de 137,9 bilhões de dólares ao redor do mundo. A cada dia, novos jogos são publicados, o que torna este mercado bastante competitivo. Os jogos podem ser criados por grandes empresas ou podem ser desenvolvidos por equipes com poucos recursos e que objetivam inovação e originalidade, comumente conhecidos como jogos independentes. Independentemente do tamanho da organização e/ou equipe, a complexidade de desenvolvimento inerentes à produção deste tipo de específico de software é similar. Para entregar jogos com qualidade, os profissionais precisam realizar testes durante a produção de seus jogos independentes. Esses testes visam analisar características específicas deste tipo de software, como grau de diversão e aspectos de jogabilidade. No decorrer dos anos, práticas de desenvolvimento surgiram com objetivo de melhorar a qualidade do desenvolvimento de software, como exemplo o *Behavior-Driven Development* (BDD). Até o presente momento, poucos estudos apresentam relatos sobre seus benefícios e limitações durante o uso de BDD no ciclo de desenvolvimento de jogos. Desta forma, esta pesquisa visa analisar como o BDD pode ser empregado como prática no desenvolvimento de jogos. Para isso, investigou-se como as atividades de teste são realizadas e quais aspectos de testes de jogos são explorados durante o desenvolvimento de jogos independentes. Em seguida, realizou-se um estudo de caso de uso de BDD durante o desenvolvimento de jogos. Os dados obtidos forneceram base para construção do conjunto de especificações do *Behavior-Driven Development for Indies* (BDD-I), apresentado neste trabalho e formado pelo mapeamento de ciclos de desenvolvimento, pela identificação dos aspectos de testes que podem ser trabalhados com o uso da prática e por um guia para ajudar estes desenvolvedores de jogos independentes a especificar funcionalidades e cenários utilizando a Linguagem Ubíqua do BDD.

Palavras-chave: Engenharia de Software. Testes. Desenvolvimento Dirigido a Comportamentos. Desenvolvimento de Jogos Independentes.

Abstract

The gaming industry in 2018 generated about 137.9 billion dollars around the world. Everyday, new games are launched, which makes this market quite competitive. The games can be created by the great AAA industry or can be developed by teams with few resources and that aim at innovation and originality, commonly known as independent games. Regardless of the size of the organization and/or team, the complexity of development inherent in the production of this type of software specific is similar. To deliver quality games, developers need to conduct tests during the independent games production. The tests aims to analyze specific characteristics of this type of software, such as degree of fun and aspects of gameplay. Over the years, development practices have emerged with the aim of improving the quality of software development, such as Behavior-Driven Development (BDD). Until now, few studies have reported on the benefits and limitations during the use of BDD in the game development cycle. In this context, this research aims to analyze how the BDD can be used as a practice in game development. To do so, it was investigated how test activities are performed and what aspects of game testing are explored during the development of independent games. Next, a case study on the use of BDD was conducted during the development of games. The collected data provided a basis for the construction of the Behavior-Driven Development for Indies (BDD-I) specification, presented on this work and consisted of a mapping of both game and bdd development cycles, the identification of test objectives that can be worked out using the practice, and a guide to help the independent game developers specify functionalities and scenarios using the BDD Ubiquitous Language.

Keywords: Software Engineering. Testing. Behavior-Driven Development. Indie Game Development.

Agradecimentos

Agradeço a Deus pela força de cada dia, o qual uso para me mover em direção aos meus objetivos e sonhos.

Aos meus avós. À minha avó Itaci, pelos cuidados e ensinamentos de vida. À minha avô Elizabeth, por todo o carinho. Ao meu avô José Nunes, em memória, pelos conselhos, aprendizado e boas conversas.

À minha família, por terem priorizado minha educação. Aos meus irmãos, pelos incentivos e momentos de descontração.

Ao meu orientador, Davi Viana dos Santos, por ter encarado comigo o tema proposto. Sua parceria, cordialidade, conhecimento e apoio foram cruciais para que que se pudesse chegar até aqui.

Aos meus professores da UFMA e do IFMA, pelo ensinamento e conhecimentos em ciência da computação.

Aos desenvolvedores de jogos independentes do Maranhão e do Brasil, pela dedicação e tempo para fornecer dados para esta dissertação. À AMAGames por permitir a divulgação e veiculação de pesquisas que contribuíram com esta dissertação.

À Eveline Sá, pelas contribuições e conhecimentos compartilhados ao longo de nossa parceria no IFMA. À Fábio Lima, Daniel Torres, Eliberto Castro e Gustavo Henrique, por permitirem meu afastamento para mestrado e assumido minhas responsabilidades no IFMA neste período. Aos meus sócios na Ops! Game Studio, Kassio Sousa e Artur Pinheiro, pela contribuição com dados para esta dissertação.

À Thamires Ayres pelos incentivos, compreensão e torcida, motivando-me para a conclusão desta dissertação.

"All your bases are belong to us."

Zero Wing

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ATDD *Acceptance Test-Driven Development*.

BDD *Behavior-Driven Development*.

BDD-I *Behavior-Driven Development for Indies*.

CVDS Ciclo de Vida de Desenvolvimento de Software.

DDD *Domain-Driven Design*.

GDD *Game Design Document*.

GT *Grounded Theory*.

IBJD Indústria Brasileira de Jogos Digitais.

RUP *Rational Unified Process*.

SBC Sociedade Brasileira de Computação.

TDD *Test-Driven Development*.

Sumário

1 Introdução

A indústria de jogos, em 2018, movimentou cerca de 137,9 bilhões de dólares ao redor do mundo[1]. Esse valor demonstra o poder que essa indústria tem na economia mundial. Assim, por ser um mercado promissor, torna-se bastante concorrido. Em 2015, uma média de 500 jogos foram lançados a cada dia na AppStore[2]. Em 2018, na Steam, uma das maiores lojas virtuais de jogos para computadores, foram lançados aproximadamente 25 jogos por dia[3]. Ao considerar outras plataformas como consoles e portáteis, estes números podem ser bem maiores.

Uma das razões que favoreceram estes dados no mercado atual de jogos eletrônicos foi a ascensão do desenvolvedor de jogos independentes (do inglês, *indie games*) [6]. Entende-se por jogos independentes aqueles desenvolvidos com baixo orçamento, equipes reduzidas, visando inovação e originalidade [6–8]. Dentre os fatores que possibilitaram um ambiente oportuno para que estes desenvolvedores pudessem conceber suas ideias em jogos, destacam-se a popularização das tecnologias, o surgimento de ferramentas mais simples e produtivas e a maior cobertura de lojas virtuais de dispositivos móveis e computadores [9].

Apesar destes fatores e do elevado número de jogos lançados, desenvolver jogos não é uma tarefa simples [10–12]. Do ponto de vista do desenvolvimento de software, diversos requisitos não funcionais precisam ser garantidos, como suporte a diversas configurações de hardware e plataformas, requisitos de usabilidade e diversão. Há ainda um alto grau de incerteza nos requisitos funcionais, que em geral mudam constantemente durante a produção do jogo [13]. Com toda essa complexidade, os desenvolvedores têm optado por desenvolver apenas as funcionalidades necessárias visando uma qualidade aceitável [14]. Esta situação tende a ser mais relevante no desenvolvimento de jogos independentes, uma vez que os recursos humanos e materiais são mais escassos.

[1] https://newzoo.com/insights/articles/newzoos-2018-report-insights-into-the-137-9-billion-global-games-market/. Acessado em 2019-03-05.

[2] https://venturebeat.com/2015/03/03/500-ios-games-per-day-flooded-mobile-market-is-tricky-for-indie-devs/. Acessado em 2017-11-21.

[3] `https://steamspy.com/year/2018`. Acessado em 2019-03-05.

Para atender um nível adequado de qualidade, os jogos são submetidos constantemente a diferentes tipos de testes durante sua produção [15]. Cada teste possui um objetivo específico, em geral áreas de conhecimento e aspectos característicos de jogos, como balanceamento, compatibilidade e usabilidade [16]. A realização do conjunto de testes permite o funcionamento correto do jogo para as funcionalidades as quais foi projetado, garantindo eficiência e maturidade durante as várias etapas de produção do mesmo [15].

1.1 Motivação

Práticas de desenvolvimento têm surgido com objetivo de caracterizar melhor os testes durante o desenvolvimento de software [17], como por exemplo o Desenvolvimento Dirigido a Testes (em inglês, *Test-Driven Development* (TDD)) [18] e o Desenvolvimento Dirigido a Comportamento (do inglês, *Behavior-Driven Development* (BDD)) [19].

Em se tratando de TDD, Xu e Rajlich [20] relatam os benefícios da prática, como o aumento no número de casos de testes, criação de código mais limpo e mais coeso. LLopis e Houghton [21] relatam o uso da prática na empresa de desenvolvimento de jogos High Moon Studios, na qual atribuem uma melhoria na qualidade dos jogos desenvolvidos, bem como maior documentação, melhor projeto de código, *feedback* instantâneo e diminuição de riscos nas mudanças de projeto. Keith [22] e Boeira [23] utilizam a prática na produção de jogos para garantir uma maior qualidade no produto final.

BDD se apresenta com o estado da arte do TDD [4]. Ao expandir os conceitos de TDD e Design Dirigido por Domínio (em inglês, *Domain-Driven Design* (DDD)) [24], BDD auxilia o desenvolvimento de softwares onde as funcionalidades são complexas ou com alto grau de incerteza daquilo que se quer construir, através de um conjunto de atividades que visam minimizar estes problemas. Dentre essas atividades, destaca-se a elaboração de testes a partir do uso da Linguagem Ubíqua. A Linguagem Ubíqua do BDD, baseada nas definições de DDD, permite a cooperação de desenvolvedores e interessados (*stakeholders*) para especificação de funcionalidades e cenários, que descrevem o comportamento do software [25]. Os cenários

são transformados em especificações executáveis que geram testes automatizados, necessários para garantir qualidade durante o ciclo de desenvolvimento, num ciclo que envolve as práticas do TDD. Durante o uso de BDD, as atividades realizadas melhoram o ciclo de desenvolvimento de software e a qualidade do produto final [17].

1.2 Problema

Apesar de haver indícios na literatura sobre os benefícios do uso de BDD, Solis e Wang [25] e Egbreghts [4] relatam, em seus trabalhos, a escassez de estudos sobre BDD no desenvolvimento de software em âmbito geral. Em se tratando de trabalhos referentes ao uso de BDD no desenvolvimento de jogos, essa quantidade é menor. Ainda encontram-se em aberto questões sobre se estes benefícios podem também ser obtidos durante o ciclo de desenvolvimento de jogos independentes.

Ampatzoglou e Stamelos [26] destacam a escassez de trabalhos na área de testes/*debugging*, dando ênfase na garantia de qualidade no desenvolvimento de jogos. Murphy-hill et al [13] descrevem que: (a) videogames são uma parte importante da indústria de software e (b) a área de teste e detecção prévia de *bugs* é fértil para pesquisas.

Desta forma, destaca-se a importância de se compreender como os desenvolvedores de jogos independentes testam seus jogos. Dentre as lacunas em aberto, é necessário identificar quais objetivos e aspectos são relevantes para estes desenvolvedores durante a prática de testes neste tipo específico de software. Ainda se faz necessário assimilar como as atividades e etapas do BDD podem ser explorados no ciclo de desenvolvimento de jogos. Ao coletar estas evidências, estas fornecem um contexto propício para que práticas como BDD possam ter suas atividades melhor aproveitadas no contexto de desenvolvimento de jogos independentes.

1.3 Objetivos

Este trabalho analisa o uso de BDD durante o ciclo de desenvolvimento de jogos independentes, permitindo a construção de um conjunto de especificações

denominado *Behavior-Driven Development for Indies* (BDD-I). Assim, tem por objetivo geral:

- Propor um conjunto de especificações de uso de BDD no ciclo de desenvolvimento de jogos independentes, observando as atividades e aspectos de teste de jogos que beneficiam os desenvolvedores durante a produção de jogos independentes.

Enumera-se ainda os objetivos específicos deste trabalho:

1. Caracterização do uso de processos ou práticas de teste e aspectos de teste explorados pelos desenvolvedores na produção de jogos independentes;
2. Mapeamento das etapas do ciclo de BDD no ciclo de desenvolvimento de jogos independentes;
3. Identificação de aspectos de teste de jogos e seu atendimento com o uso de prática do BDD;
4. Construção do BDD-I, um conjunto de especificações para instruir desenvolvedores na utilização do BDD no desenvolvimento de jogos independentes.

1.4 Método de Pesquisa

A condução deste trabalho organiza-se em quatro etapas, apresentadas abaixo e ilustradas na Figura 1.1:

1. Levantamento de literatura sobre práticas de testes e uso de BDD no desenvolvimento de jogos;
2. Realização de pesquisa de opinião (*survey*) com desenvolvedores de jogos independentes com o objetivo de compreender como eles realizam as práticas de testes e quais aspectos eles exploram ao realizar os testes durante a produção de seus jogos;

3. Análise do uso de BDD durante o ciclo de desenvolvimento de jogos através de estudo de campo com objetivo de verificar seus benefícios e limitações;

4. Produção das especificações do BDD-I, que propiciam o uso de BDD durante o ciclo de desenvolvimento de jogos independentes de forma a explorar eficientemente os aspectos de teste de jogos durante a realização de testes.

Figura 1.1: Sequência de etapas realizadas na condução desta dissertação. Fonte: Autor.

1.5 Estrutura da Dissertação

Esta dissertação organiza-se da seguinte forma:

- O Capítulo 2 compreende a fundamentação teórica com os principais conceitos e assuntos que dão base a este trabalho;
- O Capítulo 3 apresenta uma pesquisa de opinião sobre o uso de processos/práticas de testes em jogos independentes e aspectos de teste de jogos, seu planejamento, condução e resultados encontrados;
- O Capítulo 4 descreve o planejamento, condução e resultados encontrados do estudo de campo no qual se analisou o uso de BDD durante o ciclo de desenvolvimento de jogos;
- O Capítulo 5 apresenta o BDD-I, seu conceito e especificações que permitem o uso de BDD durante a produção de jogos independentes como prática na realização de testes;
- O Capítulo 6 relata as conclusões, resultados obtidos e trabalhos futuros que podem ser desenvolvidos a partir desta dissertação.

2 Fundamentação Teórica

Neste capítulo são apresentados os principais conceitos que dão base a este trabalho. Em sua primeira seção, expõe-se os fundamentos que deram forma ao BDD e seu ciclo de desenvolvimento. Em seguida, apresenta-se de fundamentos básicos sobre teste de software. Na seção seguinte, explana-se sobre as etapas de desenvolvimento e principais características da produção de jogos. Por fim, apresenta-se o processo de testes para jogos, suas principais técnicas e práticas na produção de jogos.

2.1 Behavior-Driven Development

BDD é um conjunto de práticas de engenharia de software projetadas para ajudar equipes a desenvolver e entregar mais rapidamente softwares de alto valor e alta qualidade [2]. Para o BDD, softwares de alto valor entregam as funcionalidades que realmente importam aos interessados.

BDD tem como base outras duas práticas consolidadas: Design Dirigido por Domínio (em inglês, *Domain-Driven Design* (DDD)) e Desenvolvimento Dirigido a Testes (em inglês, *Test-Driven Development* (TDD)).

O termo DDD foi definido por Eric Evans para descrever uma abordagem do desenvolvimento de software coordenada, conectando meticulosamente a implementação à um modelo que envolva os conceitos principais do negócio. O objetivo é entender e construir um modelo que atenda ao domínio do negócio. Assim, providencia uma estrutura de práticas e tecnologias para decisões de projeto de domínios complexos, aumentado o foco e a velocidade de desenvolvimento desses tipos de projetos [24].

O DDD propõe o uso de uma linguagem capaz de descrever o domínio do software de forma que todos os interessados, usuários, desenvolvedores e testadores possam entender. Essa linguagem foi simplificada e adaptada para o BDD, denominada Linguagem Ubíqua, para atender o maior número de domínios possíveis [25]. As funcionalidades e os cenários são descritos nesta linguagem.

As funcionalidades são expressas usando as palavras-chaves “Para/Em”, “Eu, como”e “Quero” (do inglês, *In*, *As* e *I want*). O Código 2.1 apresenta uma funcionalidade com base na sintaxe da ferramenta BDD SpecFlow[1]. Cada arquivo *.feature* deve conter a descrição de uma funcionalidade do software. A primeira parte da sintaxe expressa o valor da funcionalidade para o negócio. A segunda e terceira parte irão expressar o papel e a funcionalidade desejada, respectivamente.

Código 2.1: Exemplo de uma funcionalidade descrita em BDD. Fonte: Autor.

```
1 Para que o cliente possa resgatar seu dinheiro em nosso banco
2 Eu, como o cliente
3 Quero que o cliente possa realizar o saque em um terminal automatizado
```

Cada funcionalidade contém um conjunto de cenários que expressam os comportamentos das funcionalidade a serem desenvolvidas. Para a descrição de cenários, a linguagem ubíqua adota as palavras-chaves “Dado”, “Quando” e “Então” (do inglês *Given*, *When* e *Then*). O Código 2.2 apresenta um exemplo de cenário para descrever o comportamento de uma funcionalidade “Saque em espécie”. O exemplo de Código 2.1, juntamente com o exemplo de Código 2.2 e outros cenários descritos, integrariam um arquivo de descrição de uma funcionalidade para o BDD. O conjunto de funcionalidades e cenários descritos na linguagem ubíqua serão transformados em especificações executáveis através de ferramentas BDD, como Cucumber[2] e SpecFlow, para a criação de testes automatizados [27].

Código 2.2: Exemplo de um cenário descrito em BDD. Fonte: Autor.

```
1 Cenario: Saldo Insuficiente
2 Dado que o saldo disponível do cliente seja de R$ 100,00
3 Quando o cliente tentar realizar o saque de R$ 120,00
4 Entao o terminal automatizado deverá exibir uma mensagem de saldo insuficiente
5 E o saldo disponível deverá ser igual ao saldo inicial
6 E o terminal não deverá disponibilizar dinheiro ao cliente
```

TDD é uma técnica criada por Kent Back onde se desenvolve primeiro os testes, na forma de assertivas, antes de se desenvolver qualquer trecho de código funcional. Ao satisfazer os testes, o desenvolvedor então melhora o código funcional

[1] http://specflow.org/. Acessado em 2018-04-16.

[2] https://cucumber.io/. Acessado em 2018-04-16.

quando necessário. Isso o mantém em um ciclo constante de testes, desenvolvimento e refatoração, também conhecido por ciclo Vermelho-Verde-Refatora, ilustrado na Figura 2.1. Dessa forma, nenhum trecho de código funcional ficará descoberto de casos de testes correspondentes [18].

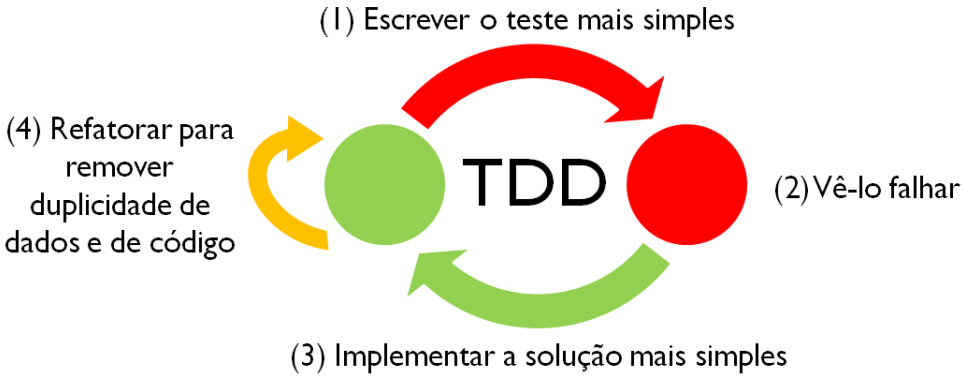

Figura 2.1: Ciclo Vermelho-Verde-Refatora. Criado por Aniche [1] e adaptado pelo Autor.

Diversas vantagens podem ser enumeradas quando se usa TDD no desenvolvimento de um software. Observa-se na literatura que softwares que utilizam TDD possuem códigos mais limpos, melhor projetados e fáceis de manter [1, 23, 28]. Testar funcionalidades contribui para uma menor quantidade de erros e defeitos, o que deixa o software com mais qualidade. Porém, muitos desenvolvedores têm dificuldades em utilizar TDD [29]. Isso em geral se deve ao fato de como responder a seguinte pergunta: "o que deve se testar?" [2]. Projetos onde os desenvolvedores têm funcionalidades com diversos detalhes facilmente deixam o desenvolvedor em dúvidas sobre o que ele deve desenvolver. Ainda nesses tipos de projetos, é comum ter uma quantidade elevada de testes de unidade, o que substancialmente aumenta a dificuldade de mantê-los.

Para o BDD, o foco dos testes não são funções, métodos ou implementações particulares, mas sim o que o software deve fazer, como ele deve se comportar. Ao se pensar em como ele deve se comportar, é mais fácil manter os esforços focados no que realmente é útil para o negócio. Nos exemplos de Código 2.3 e Código 2.4, pode-se observar a diferença ao se especificar testes nas duas abordagens. No Código 2.3, são descritos testes de unidade para as funcionalidades, sem levar em consideração sua relevância para o comportamento proposto. O Código 2.4 evidencia como se comporta a funcionalidade, deixando visível o porquê de se testar, os eventos e a saída esperada.

Código 2.3: Exemplo de classe de testes usando TDD. Fonte: Autor.

```
1 public class SaqueTest {
2     public void SaqueSaldoInsuficiente()
3     {
4         ATM atm = new ATM();
5         atm.identificarUsuario("1234");
6         float saldo = atm.consultarSaldo("1234");
7         assert( saldo, 100f );
8         string erro = atm.sacar("1234", 120f);
9         assert( erro, "saldo insuficiente" );
10        saldo = atm.consultarSaldo("1234");
11        assert( saldo, 100f );
12        assert( atm.ultimaOperacao(), 0 );
13    }
14 }
```

Egbreghts [4] e Solis e Wang [25] apresentam em seus trabalhos características consideradas essenciais ao processo de BDD: (1) Linguagem Ubíqua, (2) Especificação de Requisitos, (3) Testes de Aceitação, (4) Ferramentas, (5) Colaboração e (6) Automação. Em seu estudo, Egbreghts [4] evidencia que BDD nem sempre é implementada como um protocolo ágil, uma vez que na literatura todas as atividades do BDD não precisam ser realizadas para que o desenvolvimento obtenha sucesso. A Figura 2.2 apresenta essas características, relacionando suas afinidades com as práticas que influenciam o BDD.

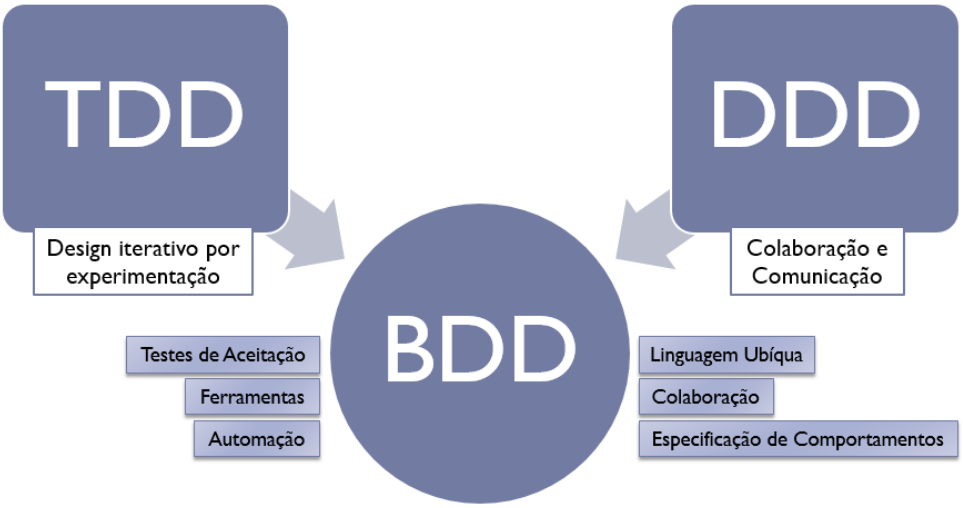

Figura 2.2: Características do BDD. Fonte: Autor.

Código 2.4: Exemplo de classe de testes usando BDD. Fonte: Autor.

```
1 public class SaqueSaldoInsuficienteScenario {
2     @Given("que o saldo disponível do cliente seja de %f", 100f)
3     public void contexto(float saldoInicial)
4     {
5         atm = new ATM();
6         atm.identificarUsuario("1234");
7         atm.definirSaldo(saldoInicial);
8         saldo = atm.consultarSaldo("1234");
9         assert( saldo, saldoInicial );
10    }
11
12    @When("o cliente tentar realizar o saque de %f", 120f)
13    public void evento(float saqueValor)
14    {
15        erro = atm.sacar("1234", saqueValor);
16    }
17
18    @Then("o terminal automatizado deverá exibir uma mensagem de %s",
19        "saldo insuficiente")
20    public void resultado1(string mensagem)
21    {
22        assert( erro, mensagem);
23    }
24
25    @Then("o saldo disponível deverá ser igual ao saldo inicial")
26    public void resultado2()
27    {
28        assert( erro, saldo );
29    }
30 }
```

Um resumo das principais atividades do BDD é apresentado na Figura 2.3. Em cada iteração do desenvolvimento do software, novas funcionalidades surgem a medida que os objetivos do negócio são cada vez mais esclarecidos. Para cada funcionalidade, um conjunto de exemplos (ou cenários) são criados para ilustrar seu comportamento durante diferentes eventos e situações. Esses cenários, descritos em uma linguagem ubíqua, são utilizados por ferramentas BDD para construção de especificações executáveis que auxiliarão desenvolvedores a construir especificações de baixo nível e o código da aplicação. As especificações executáveis e de baixo nível dão suporte para que as ferramentas BDD possam também extrair diversos artefatos relativos ao desenvolvimento do software, como documentação de desenvolvimento, validações automáticas através de testes e documentação técnica.

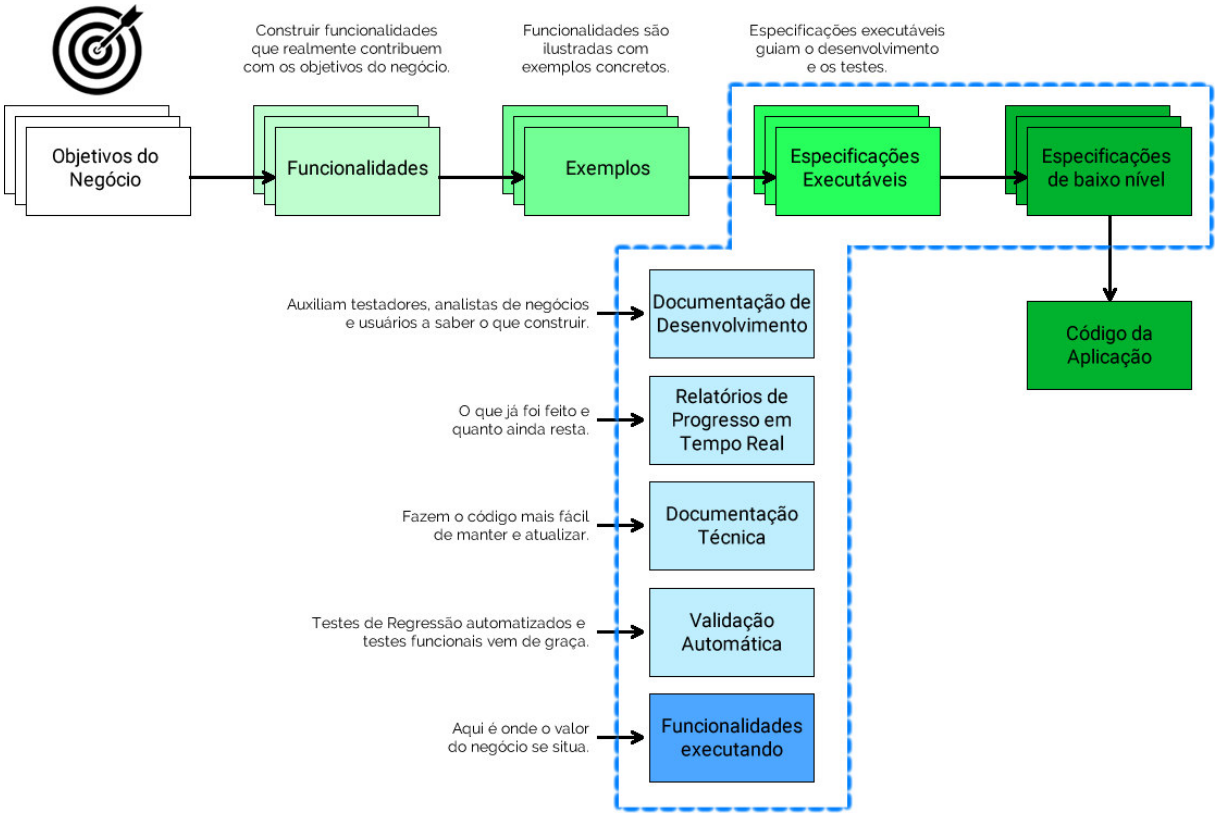

Figura 2.3: Principais atividades do BDD. Criado por Smart [2] e traduzido pelo Autor.

2.2 Teste de Software

Durante o ciclo de desenvolvimento de software [30, 31], é prevista uma etapa para realização de testes no produto. Seu objetivo é garantir que as funcionalidades desenvolvidas estão de acordo com o que foi projetado. Permitem ainda avaliar o software baseado em métricas como segurança, confiabilidade, portabilidade, manutenibilidade, dentre outros [32,33]. Em geral, as práticas de teste de software são classificados em três categorias: testes de caixa branca, testes de caixa preta e testes de caixa cinza [32,33].

Classificação de Técnicas de Teste

Testes de caixa branca necessitam de acesso ao código-fonte do software para que possam ser planejados e executados os casos de teste. Em teoria, isso garante que um desenvolvedor possa criar todos os possíveis casos de teste para o software. Assim, teríamos num cenário ideal a cobertura do software desde níveis de unidade à níveis de sistema [34].

Testes de caixa preta, diferentemente dos testes de caixa branca, não possuem acesso ao código-fonte do software. O testador terá acesso apenas a um dispositivo de entrada e observará as saídas com os resultados baseados no que foi informado como entrada. Em geral, o testador não conhece detalhes de implementação ou projeto, e por isso cria os casos de testes a partir do documento de projeto do software. Assim, testes de caixa preta permitem que se possa testar as funcionalidades do software do ponto de vista do usuário final [35].

Testes de caixa cinza buscam o meio termo entre os testes de caixa branca e os testes de caixa preta. Isso porque ainda que o software seja uma caixa-preta para o testador, este conhece limitadamente parte da estrutura e arquitetura da aplicação. Embora este tipo de testes possa ser executado por desenvolvedores, aconselha-se que sejam realizados por testadores para que se neutralize vícios e testes tendenciosos [36].

2.3 Desenvolvimento de Jogos

O desenvolvimento de jogos tem sido tratado na literatura como uma tarefa de domínio complexo [11, 26]. Os desenvolvedores de jogos precisam constantemente negociar com cenários de incerteza de funcionalidades, com a comunicação entre as equipes de produção e com diversos outros requisitos, grande parte deles não funcionais. Exemplos desses requisitos são a necessidade de suporte a diversas configurações de hardware e plataformas, boa usabilidade e excelente grau de diversão [10].

Esses desenvolvedores precisam ainda negociar com uma elevada quantidade de funcionalidades para que o jogo possa se equiparar com os demais concorrentes [37]. Para manter uma boa coordenação entre os desenvolvedores, ao longo dos anos estudos apresentaram metodologias de gestão que buscam garantir a eficiência e qualidade na produção de jogos [38–40].

O ciclo básico de produção de jogos é comumente dividido em 4 fases: pré-produção, produção, testes e pós-produção [3, 41]. Cada fase contém tarefas que podem mudar de projeto para projeto. Por exemplo, nem todos os projetos devem possuir tarefas de dublagem ou gravação de movimentos por atores. A Figura 2.4 ilustra este ciclo.

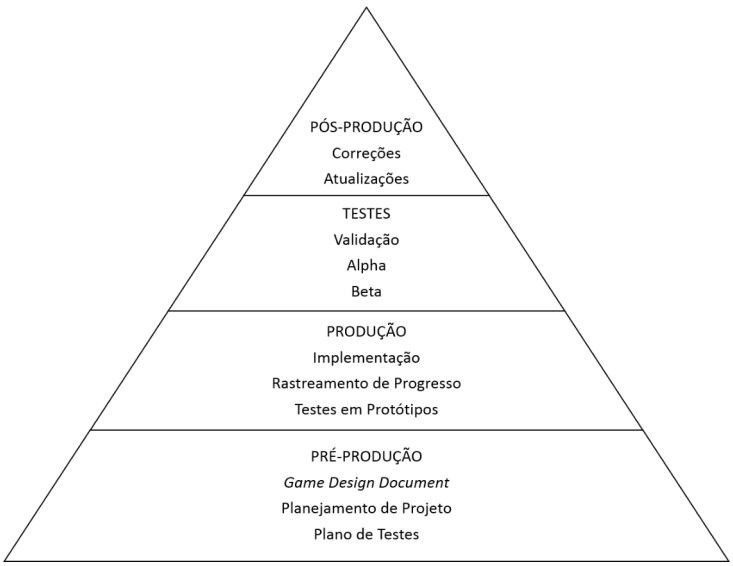

Figura 2.4: Ciclo Básico de produção de jogos. Fonte: Adaptado de Chandler [3] pelo Autor.

A **fase de pré-produção** é caracterizada pelo desenvolvimento do conceito do jogo. É nesta fase que ocorre o planejamento do projeto, dando suporte às demais

etapas de desenvolvimento e à equipe de produção na fases que se seguem. Assim, uma série de documentações são elaboradas [41]. Um destes documentos é chamado de Documento de Projeto de Jogo (em inglês, *Game Design Document* (GDD)). O GDD é o principal documento produzido, uma vez que conterá as funcionalidades iniciais idealizadas para o jogo, como história, personagens, mecânicas e jogabilidade, itens, etc. Porém, é importante frisar que este documento, em geral, sofre alterações durante todas as outras fases do ciclo. Em alguns projetos, o GDD pode até mesmo ser modificado na fase final de pós-produção. Isso se deve ao fato de que, nas fases que se seguem, as funcionalidades até então especificadas serão submetidas a constantes testes de aceitação, a fim de verificar sua adequação à proposta de valor do jogo. Funcionalidades podem ser refutadas ou mesmo novas podem ser adicionadas [42]. Outros documentos desta fase são o Plano de Testes, onde conterá informações necessárias à equipe de testes, o escopo, critérios de aceitação de fases, métodos, etc [15], e o Documento Técnico de *Design*, que descreve aspectos específicos de motor de jogo, tecnologias, ferramentas e especificações de hardware e software [42].

A **fase de produção** foca na criação de conteúdo e código para o jogo. Nesta fase, coloca-se em prática o que foi definido na fase de pré-produção. As equipes trabalham para entregar suas tarefas ao longo do cronograma estipulado. Uma vez que o desenvolvimento de jogos exige variados tipos de profissionais, cada equipe aplica as técnicas necessárias para o gerenciamento de suas tarefas. A equipe de programação se organizará de acordo com alguma metodologia de desenvolvimento de software. É possível observar equipes de desenvolvimento de jogos utilizando metodologias tradicionais como *Rational Unified Process* (RUP) ou cascata [3,41,43], mas há uma forte tendência por metodologias ágeis, como Scrum ou Lean [22,23,38]. Os primeiros testes também são executados nesta fase, ainda em nível de código. Diversos testes, como testes de unidade e testes de aceitação, são especificados e executados pela equipe de programação com o objetivo de que o jogo possa avançar para a próxima fase com o menor número de erros possíveis, diminuindo assim o trabalho da equipe de testadores. Pequenos protótipos também são liberados aos *game designers*, de forma a validar algumas mecânicas idealizadas para o jogo. Dessa forma, as funcionalidades e conceitos são atualizados durante o ciclo, refletindo assim também na atualização do GDD [3,13].

A **fase de testes** se inicia quando o jogo possui todas as suas funcionalidades implementadas, podendo ser executado do início ao fim. Em geral esta fase é subdividida em duas subfases: *Alpha* e *Beta* [15]. A subfase *Alpha* é marcada pelos testes realizados pela equipe de testes. Estes executarão os planos de testes elaborados ainda na fase de pré-produção, testando se cada funcionalidade foi implementada como fora especificada. Outros requisitos não funcionais também deverão ser testados, como diversão, desempenho do jogo nas plataformas atendidas e testes de usabilidade e experiência do usuário. A subfase *Alpha* é comumente utilizada para polimento do jogo, verificando cada detalhe de forma a garantir que o jogo fique o mais próximo do idealizado. A subfase *Beta*, por sua vez, tem o papel de identificar e corrigir *bugs*, pequenos erros ou falhas que podem surgir durante a execução do jogo. Nesta subfase também é comum que o jogo seja liberado a um grupo seleto de jogadores, geralmente pessoas que se encaixam no público-alvo do jogo, para que possam opinar sobre as funcionalidades e ajudar na identificação de *bugs*. Alguns jogos que utilizam recursos em rede utilizam esta subfase para realizar testes de comunicação, aumentando a carga de rede de servidores e verificando possíveis gargalos [15].

O jogo entra na **fase de pós-produção** após ser disponibilizado ao público a versão de lançamento, usualmente denominada versão *Gold*. É comum observar nessa fase, pacotes de correção para *bugs* que não puderam ser corrigidos a tempo antes do lançamento ou mesmo *bugs* que foram encontrados após o lançamento do jogo. Nessa fase é também recolhido o *feedback* dos jogadores, que dará subsídios a uma análise de prós e contras sobre o jogo, bem como a possibilidade da equipe em expandir o jogo para outras versões ou mesmo disponibilizar novos conteúdos. Dessa forma, aumenta-se a longevidade do jogo original [3, 15, 22].

2.4 Testes para Jogos

A construção do jogo é complexa e os riscos são altos durante a produção [11]. Além disso, as publicadoras e mantenedoras de *console* são exigentes, requisitando um padrão de qualidade. Assim, durante o ciclo de desenvolvimento de jogos, diversas etapas e técnicas de testes são realizadas [16].

O ciclo de vida de testes de jogos, que permeia pelo ciclo básico de produção de jogos, inicia pelo planejamento e projeto dos testes, em geral um documento onde conterá todos os dados de testes, testadores e condições para realização. Em seguida, há a etapa de preparação do teste, onde define-se o ambiente, os testadores recebem a versão do jogo e uma cópia dos casos de teste a serem realizados. Após, a etapa de realização dos testes é executada, onde os testadores geram um relatório para que desenvolvedores possam corrigir os defeitos e realizar novas implementações. Assim, o ciclo reinicia até que o produto esteja em uma versão estável [15].

A indústria de jogos possui diversos testes específicos [14–16, 44]. Eles abordam os mais diferentes aspectos do jogo, sendo que cada técnica pode trabalhar com um ou mais aspectos. Dentre os aspectos abordados listam-se: jogabilidade, experiência de usuário e desempenho, por exemplo. A seguir, apresenta-se um conjunto de técnicas de testes utilizadas durante o desenvolvimento de jogos.

Teste Combinatorial

Teste Combinatorial é um método para testar pares de parâmetros de entrada, combinando-os e garantindo que sejam testados ao menos uma vez. Tem por objetivo minimizar o conjunto de testes a serem realizados de acordo com a quantidade de funcionalidades cobertas pelo jogo. Os parâmetros podem ser encontrados em eventos, configurações, opções de jogabilidade, configuração de hardware, atributos de personagens, opções de personalização, etc. Valores devem ser discretizados conforme o tipo de dado a ser feito. Por exemplo, o valor da vida de um personagem em um jogo de luta pode possuir possíveis discretizações como “vida baixa”, “vida média” e “vida cheia” [15].

Diagramas de Fluxo de Teste

Diagramas de Fluxo de Teste são diagramas que representam o comportamento do jogo na perspectiva do jogador. A natureza gráfica desse tipo de teste fornece aos desenvolvedores, testadores e produtores a habilidade de revisar, analisar e prover *feedback* em testes de design. Promove modularidade e integridade. Comportamentos consistentes podem ser utilizados através de diversos jogos ou

funcionalidades, beneficiando sequências ou portabilidade para outras plataformas [15].

Testes Cleanroom

Adaptado do processo de desenvolvimento de software Cleanroom, este tipo de técnica tem como propósito medir o tempo de falha médio no decorrer de um projeto. Para isso, busca-se formas de detectar e remover defeitos mais comumente encontrados pelo jogador. Testes Cleanroom podem ser gerados por outras técnicas, tais como testes combinatórias e diagramas de fluxo de testes. É importante que os testadores conheçam ou estimem a frequência de uso das funcionalidade do jogo com o objetivo de imitar, de forma realística, como o jogador irá jogar. A frequência de uso pode ser definida de três formas: uso baseado em modo de jogo, uso baseado em tipo de jogador e uso baseado em estilo de vida real [15].

Árvores de Teste

Árvores de Teste permitem estruturar, em forma de árvores, diversos elementos e funcionalidades do jogo, sendo úteis para organizar casos de testes. O adequado conjunto de testes pode ser facilmente selecionado para um dado conjunto de funcionalidades de um jogo. Cada nó mais interno representa um conjunto de testes mais específico que os nós acima deles. São relevantes quando se precisa explorar a interação entre as regras, opções, elementos e funções do jogo. Podem ser utilizados para diferentes propósitos no teste de jogos, como documentar a relação hierárquica entre casos de teste e funcionalidades do jogo [15].

Testes Ad Hoc

Testes *Ad Hoc* permitem que os testadores estejam livres de guias, listas de verificação ou formulários. Assim, ele pode explorar de maneira investigativa o jogo. Ótimo para validação de jogabilidade e usabilidade. Existem dois tipos: teste livre, onde se permite ao testador fugir de roteiros e improvisar testes, e teste direcionado, com a finalidade de resolver um problema específico ou encontrar uma solução específica [15].

Playtesting

Playtesting é uma técnica de testes na qual é analisado o bom funcionamento do jogo em relação a todas as suas características, e explora aspectos como balanceamento de jogabilidade, grau de diversão e recepção de funcionalidades pelos jogadores [16, 45]. Para atingir este objetivo, utiliza um conjunto de técnicas e ferramentas, como uso de *analytics*, para medir e analisar o comportamento de jogadores e recuperar informações para melhorar a experiência do jogador [44].

2.5 Considerações Finais

Este capítulo apresentou os principais fundamentos que deram base à condução deste trabalho. Foram apresentados o ciclo de desenvolvimento do BDD e fundamentos gerais sobre teste de software, bem como as etapas de desenvolvimento na produção de jogos e suas principais práticas e técnicas de testes.

BDD busca auxiliar equipes a desenvolver e entregar softwares de alta qualidade e alto valor, integrando funcionalidades que realmente importam aos interessados. É baseada em duas outras práticas: DDD e TDD. Ao DDD, atribui-se a herança das características ligadas à comunicação da equipe e interessados e a descrição de comportamentos das funcionalidades do software. Ao TDD, atribui-se ao uso de práticas de desenvolvimento que objetivam a criação de testes antes da escrita de código funcional, permitindo uma melhoria na qualidade final do software. Ferramentas BDD extraem, de cenários descritos em Linguagem Ubíqua, esqueletos de códigos de testes que são utilizados para guiar os desenvolvedores na criação do software. Além disso, estas ferramentas podem extrair outras informações como relatórios de progresso e documentação técnica do software.

A indústria de desenvolvimento de jogos, em geral, divide o ciclo de desenvolvimento em quatro etapas: pré-produção, produção, testes e pós-produção. Uma série de documentos são criados durante a fase de pré-produção, sendo o principal deles o GDD, que descreve as principais funcionalidades do jogo a ser desenvolvido. Durante as etapas de produção e testes, diversas técnicas e práticas permitem que os desenvolvedores possam testar os jogos para tarefas como melhorias na jogabilidade e na busca por erros, com o objetivo de melhorar a qualidade do jogo.

Apesar deste extenso processo, é possível verificar ainda que erros são detectados nas versões de lançamento de jogos, por diversas causas. Assim, na pós-produção, é comum manter parte da equipe de testes e desenvolvimento para identificação e correção de erros que apareçam na versão final do jogo.

A indústria de desenvolvimento de jogos, para garantir um alto padrão de qualidade, investe no uso de diversas práticas e técnicas de teste. Assim, possui testes específicos que abordam diferentes aspectos do jogo, como jogabilidade, experiência de usuário ou desempenho. Alguns desses testes podem trabalhar com mais de um aspecto do jogo, como exemplo a técnica de Playtesting.

O capítulo seguinte apresenta uma pesquisa de opinião com o objetivo de expandir os principais conceitos abordados nesta fundamentação teórica, permitindo que se compreenda melhor como é realizado o processo de testes durante o desenvolvimento de jogos independentes, bem como identificar os principais aspectos que estes desenvolvedores buscam durante este processo.

3 Pesquisa de Opinião sobre Testes em Jogos Independentes

Este capítulo apresenta uma pesquisa de opinião sobre o uso de métodos e práticas de testes de software no ciclo de produção de jogos, com o objetivo de caracterizar o uso das metodologias e práticas de testes utilizadas durante o desenvolvimento de jogos independentes. Neste capítulo, relata-se o planejamento, metodologia e resultados encontrados nesta pesquisa de opinião.

3.1 Planejamento

Visando auxiliar a concepção do BDD-I, foi necessário investigar como os desenvolvedores de jogos utilizam as metodologias e práticas de testes durante o ciclo de desenvolvimento de jogos. Desta forma, planejou-se e realizou-se uma pesquisa de opinião com desenvolvedores de jogos independentes de forma a caracterizar o uso dessas práticas durante o dia-a-dia destes profissionais. Este estudo tem por base os procedimentos adotados por Fernández e Wagner em seu trabalho [46].

Surveys têm o objetivo de obter o estado corrente da prática sobre a aplicação de técnicas, métodos e abordagens em um fenômeno contemporâneo [47]. Para esta pesquisa, utilizaram-se as abordagens descritiva e exploratória [48], isto é, buscou-se identificar e analisar como os desenvolvedores aplicam diferentes atividades e práticas de teste no desenvolvimento de jogos independentes.Nas próximas seções são apresentados os detalhes do planejamento, condução e análise dos resultados desta pesquisa.

3.1.1 Questões de Pesquisa da Pesquisa de Opinião

O objetivo deste estudo é caracterizar o uso de teste de software durante o desenvolvimento de jogos independentes. Espera-se compreender como os profissionais utilizam testes de software em jogos e quais aspectos analisam ao realizar

os testes. Com as descobertas, pretende-se observar características que favoreçam os desenvolvedores de jogos independentes, fornecidas pelo uso do BDD.

Para que se possa alcançar este objetivo, é necessário compreender o quanto o desenvolvedor de jogos independentes está preocupado com o uso de testes durante a produção de jogos (QP1). A partir deste ponto, é preciso analisar quais e o quanto as práticas de testes estão envolvidos neste processo (QP2). Por fim, busca-se quais aspectos estes desenvolvedores almejam encontrar ao realizar testes durante a produção (QP3). As questões de pesquisa são apresentadas na Tabela 3.1.

Tabela 3.1: Questões de pesquisa da pesquisa de opinião sobre uso de testes em jogos independentes.

QP1	Os desenvolvedores de jogos independentes se preocupam com a realização de testes durante o desenvolvimento de jogos?
QP2	Que práticas de testes são adotadas durante o desenvolvimento de jogos independentes?
QP3	O que os desenvolvedores de jogos independentes buscam ao testar seus jogos?

3.1.2 Instrumento

Nesta pesquisa foi aplicado um questionário de 14 questões, com questões fechadas e questões abertas. As questões fechadas foram utilizadas para uniformizar as respostas e sintetizar, de forma mais simples, a informação requerida. Ainda assim, disponibilizou-se nas questões fechadas um campo que permitisse que o participante inserisse sua própria resposta, caso não se sentisse contemplado pelas opções apresentadas. Apresentou-se as questões do tipo fechada de duas formas: Múltipla Escolha (ME), onde o participante poderia escolher apenas uma das diversas opções apresentadas, e Caixa de Seleção (CS), onde o participante poderia escolher todas as opções as quais ele se identificasse. As questões abertas permitiram aos participantes descrever seu próprio relato, possibilitando dados mais completos sobre a aplicação de teste de software no desenvolvimento de jogos independentes.

As questões Q1 a Q7, da categoria demografia, buscam entender o perfil dos desenvolvedores, as quais foram baseadas no trabalho de Kasurinen et al. [49] e

adequadas ao contexto de jogos independentes. As questões Q8 e Q9, da categoria processo de desenvolvimento, visam fazer o participante refletir sobre processo de desenvolvimento durante a gestão e organização de seus projetos de produção de jogos independentes. A questão Q8 apresenta, como opções de resposta, uma lista de processos de desenvolvimento relacionados na literatura [50–52]. As questões Q10 a Q14, da categoria uso de testes, buscam identificar atividades, práticas e processos de teste que são executados durante a produção de jogos independentes. Além disso, essas questões visam obter informações sobre como é feito este processo e que aspectos os participantes buscam ao realizar os testes. A questão Q13 apresenta um vasto número de práticas e métodos de testes apresentados na literatura [15, 16, 35, 53]. As questões estão apresentadas na Tabela A.1 do Apêndice A.

Inicialmente, o questionário foi avaliado por um pesquisador doutor em Engenharia de Software e com experiência na execução de pesquisas de opinião, com cinco anos de execução deste tipo de estudo no contexto de Engenharia de Software. Para garantir a relevância da informação produzida pelas perguntas, o formulário foi ajustado e aplicado primeiramente com 10 desenvolvedores experientes da indústria de jogos independentes. Por fim, após mais uma fase de ajustes no formulário, passou-se para a etapa de coleta de dados.

3.1.3 Coleta de Dados

Para a coleta de dados desta pesquisa de opinião, utilizou-se a ferramenta Google Forms[1] para produzir um questionário *online* e permitir a distribuição nos canais de comunicação do público-alvo deste estudo: profissionais/autônomos desenvolvedores de jogos independentes. As questões da pesquisa de opinião foram redigidas no idioma português e o formulário foi compartilhado no Brasil entre os dias 13 de janeiro de 2019 a 15 de fevereiro de 2019.

A pesquisa de opinião foi distribuído nos canais de comunicação de associações de desenvolvedores de jogos do Brasil, em páginas e grupos de redes sociais como Reddit, Facebook, Twitter e LinkedIn, em grupos como “r/gamesEcultura”, “Indie Developers Brazil” e “Game Developers BraSil”. Além disso, foi veiculada em grupos de mensagem instantânea do Whatsapp, Telegram e

[1] `https://www.google.com/forms/about/`. Acessado em 2019-03-18.

Discord, como “Indies BR”, “Indie Dev Brazil” e “INDBR”. Foram cobertos ainda canais de comunicação listados no II Censo da Indústria Brasileira de Jogos Digitais (IBJD) [54]. Nos canais de comunicação que possuíam alta atividade e volume de postagens, foi realizado ao menos duas postagens convidando os usuários a participar, respeitando a política de *spam* de cada comunidade. Distribuiu-se ainda a pesquisa de opinião por *e-mail*, utilizando a lista “Jogos Eletrônicos” da Sociedade Brasileira de Computação (SBC) e por meio de convite individual a 19 desenvolvedores independentes de renome da comunidade brasileira. A Tabela B.1 no Apêndice B deste trabalho apresenta a lista de todos os canais de comunicação utilizados nesta pesquisa de opinião.

3.1.4 Análise de Dados

Os dados obtidos foram submetidos a análise quantitativa e qualitativa. Na análise quantitativa dos dados, utilizou-se estatística descritiva. Para melhorar a análise pelos pesquisadores e facilitar a divulgação dos resultados, criou-se gráficos a partir dos dados quantificáveis.

Analisou-se de forma qualitativa as questões abertas Q9, Q11 e Q14 apoiado com os métodos definidos pela *Grounded Theory* (GT). GT é um método científico que auxilia pesquisadores a gerar teorias de forma indutiva geralmente a partir de dados de texto não estruturados, como exemplo transcrição de entrevistas, artefatos e notas de campo [55]. Tais procedimentos tem sido amplamente aplicados em pesquisas sobre desenvolvimento de jogos [38, 49, 56].

Stol et al. [57] apresentam três versões da GT. Este trabalho optou por seguir as definições da versão Straussiana, uma vez que suas influências filosóficas são pragmáticas e abordam o interacionismo simbólico, considerando que a realidade é construída através da interação entre atores e que esta permanece na linguagem e comunicação destes [58]. Na GT Straussiana, três etapas de codificação são definidas: (i) codificação aberta, onde fragmenta-se as informações de forma a gerar dados categorizados ou codificados; (ii) codificação axial, onde relaciona-se e desenvolve-se as categorias e códigos encontrados na fase anterior; e (iii) codificação seletiva, a qual busca originar um código ou categoria central que serve de junção a todas as demais categorias, gerando teoria [59].

Para a GT, uma teoria só deve ser gerada quando esta alcança a saturação teórica. Saturação teórica é o ponto onde novos dados coletados não geram mais revisões ou novas interpretações acerca dos componentes da teoria [57]. Porém, apesar da proposta da GT ser de gerar novas teorias, Strauss e Corbin explicam que as etapas podem ser realizadas de acordo com o objetivo da pesquisa [60]. Apenas as etapas de codificação aberta e codificação axial foram suficientes para que se pudesse encontrar respostas para às questões de pesquisa e satisfazer assim os objetivos desta pesquisa de opinião.

Para apoio na análise qualitativa, utilizou-se como ferramenta o software ATLAS/ti[2]. Desenvolvido pela *Scientific Software Development*, o software foi escolhido, uma vez que contempla diversas ferramentas como auditoria, módulos de gerenciamento de documentos primários e códigos [61], auxiliando assim o pesquisador durante as etapas da GT. As Figuras 3.1 e 3.2 ilustram exemplos das etapas de codificação aberta e codificação axial, respectivamente. Na Figura 3.1, o texto à esquerda é analisado e trechos relevantes para a pesquisa são marcados e criados rótulos/códigos. Após essa atividade, os códigos são relacionados entre si (Figura 3.2). Esses relacionamentos também são criados com base nas informações das respostas dos participantes.

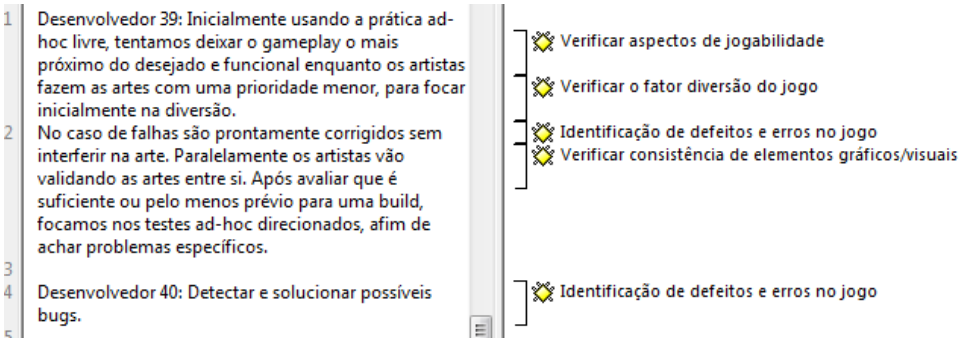

Figura 3.1: Exemplo de processo de Codificação Aberta no software ATLAS/ti. Fonte: Autor.

[2] `https://atlasti.com/`. Acessado em 2019-03-19.

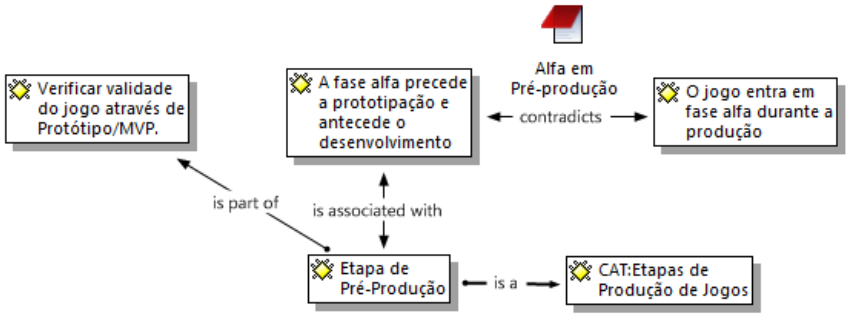

Figura 3.2: Exemplo de processo de Codificação Axial no software ATLAS/ti. Fonte: Autor.

3.2 Resultados

Ao final da etapa de coleta de dados, a pesquisa de opinião obteve 94 respostas. Destas, duas respostas foram identificadas como duplicadas e foram excluídas da fase de análise. Das 92 respostas restantes, segmentou-se os dados coletados quanto à opção informada na questão Q1. Nesta, obteve-se um total de 73 participantes (79,35%) que responderam "Sim", 11 participantes (11,96%) que responderam "Não" e oito participantes (8,70%) que responderam "Não sei responder". A Figura 3.3 apresenta o perfil dos dados coletados quanto à segmentação mencionada. Considerou-se, para etapa de análise, apenas os dados dos participantes que responderam "Sim" à questão Q1. Os resultados são apresentados nas subseções a seguir.

3.2.1 População

A Tabela 3.2 apresenta a distribuição dos dados por tempo de experiência no desenvolvimento em jogos independentes. Do total de 73 participantes, apenas sete participantes (9,59%) possuíam menos de um ano de experiência. Verificou-se que os dados coletados retratam, em sua maior parte (63,02%), participantes com no mínimo três anos de experiência, sendo 18 participantes (24,66%) com mais de seis anos de experiência.

O cargo mais indicado dentre os participantes é o de "Programador", com 45,21%, seguido de 20,55% de "Game Designer" e 10,96% de "Artista 2D/3D". Os demais cargos identificados são apresentados na Figura 3.4.

Figura 3.3: Perfil dos participantes quanto a autodeclaração de desenvolvedor independente. Fonte: Autor.

Tabela 3.2: Perfil dos participantes por tempo de experiência no desenvolvimento de jogos.

Tempo de Experiência	Quantidade	Percentual
Menos de 1 ano	7	9,59%
Entre 1 e 3 anos	20	27,40%
Entre 3 e 6 anos	28	38,36%
Mais de 6 anos	18	24,66%
Total	73	100%

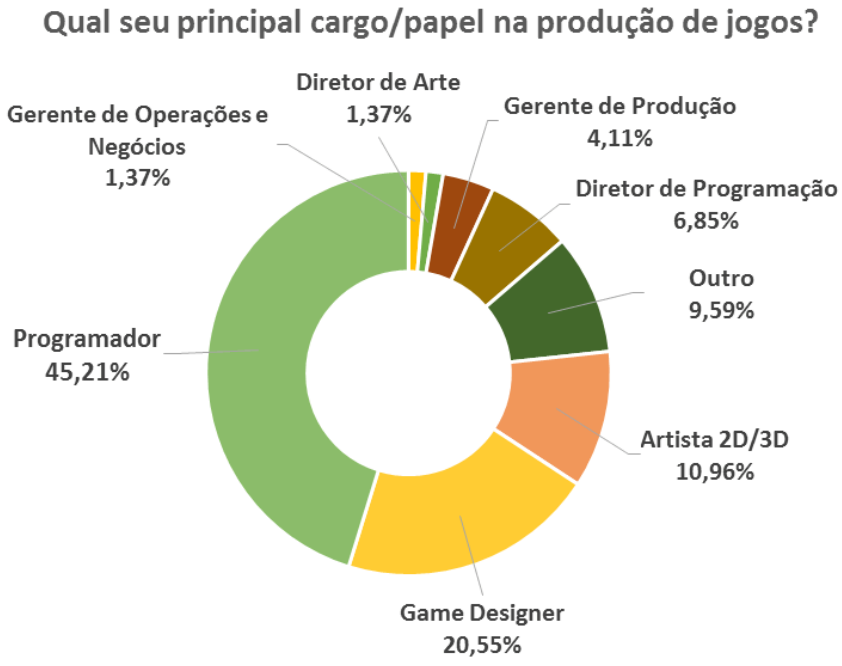

Figura 3.4: Perfil dos participantes por cargo. Fonte: Autor.

A partir dos dados coletados, 22 participantes (30,14%) afirmaram estar desenvolvendo seu primeiro jogo. Cabe ressaltar que um jogo independente pode demorar mais de um ano para ser produzido e publicado oficialmente. Adicionalmente, 38 participantes (52,06%) afirmam possuir no mínimo dois jogos publicados. A Tabela 3.3 sumariza estes dados.

Tabela 3.3: Faixas em quantidade de jogos publicados pelos participantes.

Faixas de Jogos Publicados	Quantidade	Percentual
Desenvolvendo o primeiro projeto	22	30,14%
1 jogo publicado	15	20,55%
Entre 2 e 5 jogos publicados	29	39,73%
Entre 6 e 10 jogos publicados	5	6,85%
Mais de 10 jogos publicados	4	5,48%
Total	73	100%

Em relação ao tamanho do time de desenvolvimento de jogos independentes, verificou-se que 23 participantes responderam que desenvolvem seus jogos sozinhos e 28 responderam que desenvolvem com até três pessoas, como sumarizado na Tabela 3.4. Isso corrobora com a afirmação de que jogos independentes são, em geral, desenvolvidos por pequenas equipes [7].

Tabela 3.4: Tamanho do time de desenvolvimento.

Faixas de Tamanho de Time	Quantidade	Percentual
Apenas o desenvolvedor	23	31,51%
Entre 1 e 3 pessoas	28	38,36%
Entre 4 e 7 pessoas	13	17,81%
Entre 7 e 20 pessoas	8	10,96%
Mais de 20 pessoas	1	1,37%
Total	73	100%

As plataformas-alvo mais escolhidas dentre os participantes foram computadores Windows/Mac/Linux com 62 respostas (84,93%), *smartphones* e *tablets* Android e Windows Phone com 47 respostas (64,38%), seguido de Navegadores Web com 24 respostas (32,88%). Dentre os canais de distribuição, os mais utilizados pelos

participantes são as lojas virtuais de dispositivos móveis com 41 respostas (56,16%), seguido de lojas virtuais para computadores com 40 respostas (54,59%) e sistema de distribuição próprio com 22 respostas (30,14%). Alguns participantes apontaram também *sites* como Itch.io[3] e GameJolt[4], que foram classificados como lojas virtuais para computadores. As Figuras 3.5 e 3.6 apresentam as plataformas-alvo e os canais de distribuição, respectivamente, mais utilizados pelos participantes. Nesses dois aspectos (plataformas-alvo e canais de distribuição), os participantes podiam marcar mais de uma opção no formulário.

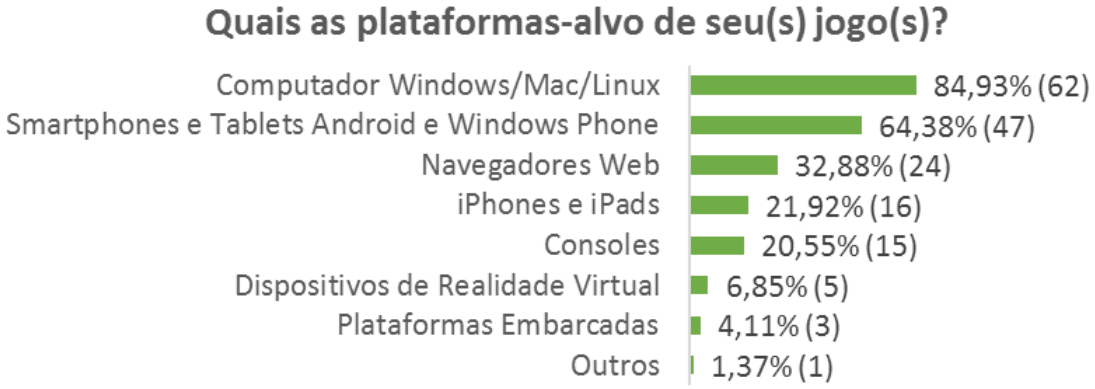

Figura 3.5: Plataformas-alvo utilizados pelos participantes. Fonte: Autor.

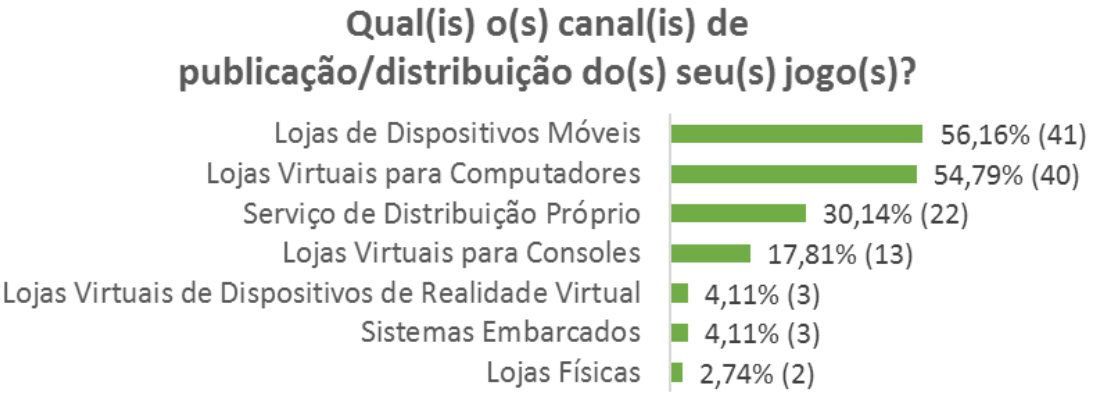

Figura 3.6: Canais de distribuição utilizados pelos participantes. Fonte: Autor.

3.2.2 Resultados das Questões de Pesquisa

Os tópicos a seguir apresentam os achados deste estudo em relação às questões de pesquisa. Discute-se ainda estes resultados em comparação a trabalhos da literatura.

[3]Disponível em `https://itch.io/`.

[4]Disponível em `https://gamejolt.com/`.

Uso de testes por desenvolvedores de jogos independentes (QP1)

A questão de pesquisa QP1 busca entender se os desenvolvedores se preocupam com o uso de técnicas de testes durante o ciclo de desenvolvimento de jogos independentes. Para identificar aspectos sobre esta questão, perguntou-se qual a relação média de tempo entre a implementação e os testes durante a produção do jogo através da Q12.

Os dados coletados apresentam que 37 participantes (50,68%) utilizam entre 1% e 25% do tempo de produção com tarefas de teste e 30 participantes (41,10%) utilizam entre 26% e 50% do tempo de produção com tarefas de teste. Apenas dois participantes (2,74%) afirmaram não utilizar testes durante o tempo de produção. Isso indica que há uma preocupação dos participantes em realizar testes durante a produção dos jogos independentes. A Figura 3.7 sumariza a distribuição das respostas quanto à relação média de tempo entre a implementação e o uso de testes.

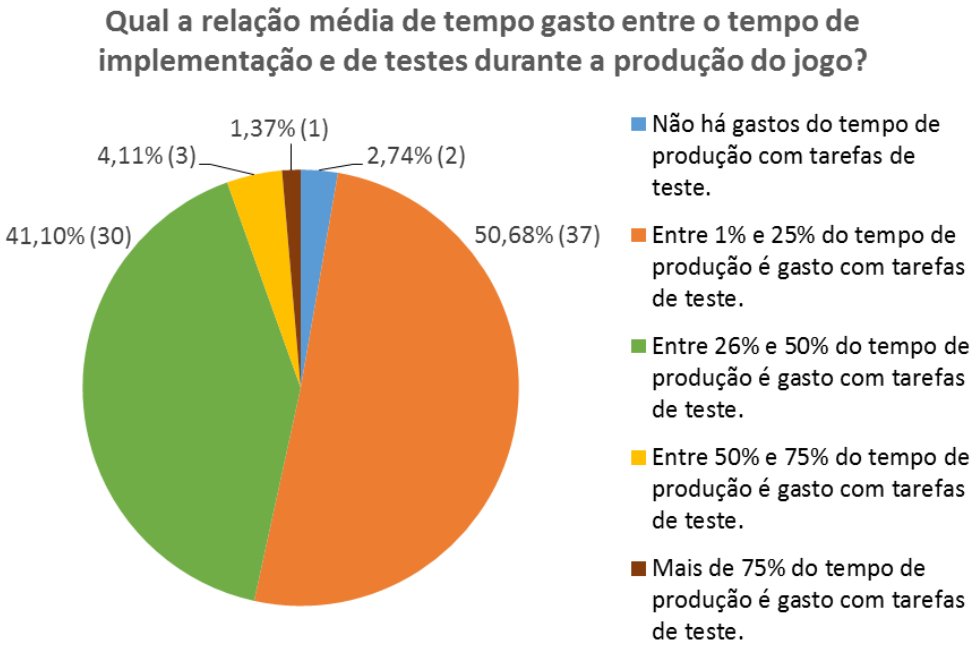

Figura 3.7: Relação média de tempo gasto entre implementação e tarefas de teste na produção de jogos independentes. Fonte: Autor.

A literatura apresenta que as atividades de teste perpassam por todas as etapas do ciclo de desenvolvimento, evidenciando a importância dessas atividades durante o ciclo de desenvolvimento. Para Chandler [3], as tarefas de testes iniciam desde a concepção do jogo, com o plano de testes, e continuam durante todas as demais etapas de desenvolvimento. Isso é demonstrado no trabalho de Ramandan e Widyani

[62], os quais apresentam um guia para conduzir o ciclo de vida do desenvolvimento de jogos, onde todas as etapas de produção se configuram num ciclo onde testes são aplicados constantemente. Aleem et al. [11] apontam que a etapa de testes possui um importante papel em cada etapa da fase de desenvolvimento e que seu gerenciamento é importante durante todo o processo de desenvolvimento de jogos.

Na amostra, 67 participantes (91,78%) afirmaram realizar em média uma quantidade inferior ou igual a 50% de tempo de produção gastos com tarefas de testes. Levantou-se assim a hipótese de que os desenvolvedores de jogos independentes se preocupam pouco com a realização de testes. Buscou-se verificar se a falta de recursos humanos poderia ser um destes fatores. Assim, calculou-se estatisticamente a correlação entre o tamanho da equipe e a média de tempo de produção gasto com tarefas de testes utilizando coeficiente de relação de Pearson, onde obteve-se P=0,126291166. Este resultado indica uma fraca correlação entre os dados coletados. Embora acredite-se que equipes menores tendam a ter mais limitações para realizar testes durante a produção dos jogos independentes, outros fatores, como a escassez de recursos financeiros e a dificuldade de gerenciar este tipo de atividade [14], podem implicar numa menor quantidade de atividades relativas a testes na produção de jogos independentes, os quais não puderam ser observados com as informações obtidas na pesquisa de opinião.

Buscou-se investigar também qual a correlação entre experiência do desenvolvedores de jogos independentes e o tempo de produção gasto com tarefas de teste. Na análise, classificou-se como experiência a quantidade de jogos lançados e o tempo que estes participantes tinham na atividade de desenvolvimento de jogos, onde valores mais altos indicavam maior grau de experiência. Novamente, calculou-se estatisticamente estas correlações utilizando coeficiente de relação de Pearson. Para as variáveis tempo de experiência e tempo de produção gasto com tarefas de teste, obteve-se P=0,196912824, o qual indica uma fraca correlação. Para as variáveis jogos publicados e tempo de produção gasto com tarefas de teste, obteve-se P=0,426259244, o qual indica também uma fraca correlação.

Em relação a quem se dedica à organização e execução dos testes durante a produção, 47 participantes (64,38%) afirmaram “Todo o time”. Assume-se assim que os participantes que também informaram se encarregar sozinhos na produção dos jogos acumulam mais esta função. Sobre os demais cargos que compartilham esta função,

11 participantes (15,07%) afirmaram que os testes são organizados por programadores e oito participantes (10,96%) afirmaram que a organização e execução dos testes são realizadas pela comunidade/jogadores. Apesar da nossa pesquisa apresentar este resultado, é possível encontrar indícios na literatura que a gestão dos testes fica sob responsabilidade da equipe de desenvolvimento quando os testes são executados pelo público-alvo do jogo e demais membros da comunidade [14,44]. A Tabela 3.5 sumariza estas respostas.

Tabela 3.5: Perfis que se dedicam à organização e execução de testes.

Perfil	Quantidade	Percentual
Todo o time	47	64,38%
Programadores	11	15,07%
Comunidade / Jogadores	8	10,96%
Game Designers	4	5,48%
Gerente de Projeto	2	2,74%
Equipe de *Quality Assurance* (QA)	1	1,37%
Total	73	100%

Práticas de testes adotadas durante o desenvolvimento (QP2)

A questão de pesquisa QP2 visa identificar e analisar as práticas e testes utilizadas pelos profissionais no desenvolvimento de jogos independentes. Assim, listou-se na questão Q13 diversos tipos e práticas de teste conhecidos pela literatura para que os participantes pudessem indicar quais utilizavam. Como resultado, identificou-se que 34 participantes (46,58%) indicaram utilizar a técnica *playtesting* durante a produção de seus jogos. Este valor é quase duas vezes maior que a quantidade de indicações da segunda prática de testes mais indicada pelos participantes, que é “Testes de Usabilidade", com 19 indicações (26,03%). *Playtesting* é objeto de estudo como prática de testes no desenvolvimento de jogos [63–65], e mais recentemente no teste de jogos independentes [44, 45], apresentando-se como uma forma efetiva de recurso para integração no ciclo de desenvolvimento de estúdios independentes [44]. Choi et al. [45] apontam a necessidade de se estruturar os métodos do processo de *playtesting* de forma a cobrir a experiência do jogo e possibilitar que

jogos se tornem mais inovadores. A Figura 3.8 apresenta estes resultados bem como os demais processos/práticas utilizadas pelos participantes.

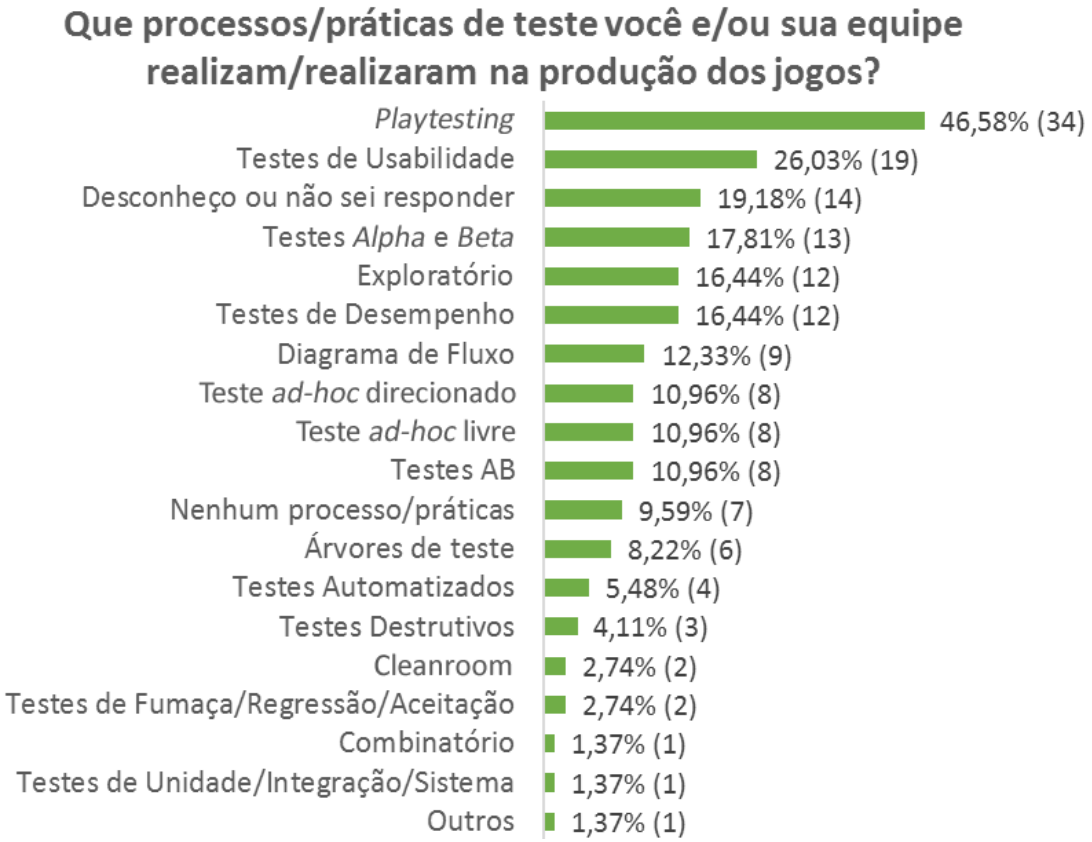

Figura 3.8: Práticas/processos de testes utilizados pelos participantes. Fonte: Autor.

Embora quase metade dos participantes tenham indicado realizar a técnica de *playtesting*, sete participantes (9,59%) afirmaram não realizar nenhum processo/práticas de testes. Ao analisar individualmente cada uma das respostas destes sete participantes, observou-se que apenas um deles havia indicado que não gasta tempo de produção com tarefas de testes durante o desenvolvimento de seus jogos (Q12). Além disso, ao responder a questão Q14, todos informaram os aspectos que buscam ao realizar tarefas de testes na produção de jogos. Apesar dessas informações serem contraditórias, elas podem indicar que estes participantes utilizam práticas próprias ou ainda não reportadas na literatura.

Outro observação se refere às categorias de práticas de testes utilizadas, conforme apresentadas na Seção 2.2. Técnicas de caixa branca são menos utilizadas pelos participantes em relação ao uso de caixas pretas. Testes de Unidade/Integração/Sistema, Testes de Fumaça/regressão/Aceitação, Testes Automatizados, são utilizados por apenas sete participantes. Dentre as implicações

da falta de uso de testes de caixa branca durante a implementação do código-fonte, pode-se levantar a hipótese de que alguns erros e defeitos do código-fonte só serão detectados em fases posteriores da produção, fazendo com que os desenvolvedores precisem refatorar funcionalidades já implementadas [66]. Por fim, apenas um participante da pesquisa de opinião apresentou um processo/prática de testes que não havia sido listado dentre as opções da questão Q13. Este participante informou que realiza ainda testes para analisar desempenho e complexidade de certos algoritmos que o mesmo utiliza em seus jogos. Durante a análise deste estudo, a prática de testes informada pelo participante foi categorizada nos resultados como "Outros".

Buscou-se verificar a quantidade de processos/práticas de testes diferentes que os participantes utilizam e categorizou-se em três faixas, apresentadas na Tabela 3.6. Cada faixa representa a quantidade de tipos de processos/práticas utilizados pelos participantes. Excluiu-se dessa análise os participantes que informaram "desconhecer ou não saber responder" (14 participantes) e "não utilizar nenhum processo/práticas" (7 participantes), conforme observado na Figura 3.8, totalizando 52 participantes. Dessa amostra, verifica-se que 18 participantes (34,62%) utilizam apenas um tipo de teste, enquanto nove participantes (17,30%) dois tipos de processos/práticas de testes diferentes. Por fim, 25 participantes (48,08%) relataram utilizar três ou mais tipos diferentes de teste. Para Schultz e Bryant [15], as técnicas de testes podem possuir finalidades variadas, sendo utilizadas em diferentes fases da produção de um jogo. Kasurinen e Smolander [14] apontam que os desenvolvedores de jogos não focam apenas em tarefas de testes de natureza de software, mas também em testes para aspectos como experiência de usuário, usabilidade ou mecânicas de jogos. Aleem et al. [11] afirmam em seu trabalho que desenvolvedores devem considerar uma maior variedade de testes durante a fase de produção para garantir a entrega contínua de jogos com qualidade ao mercado. Ainda que a variedade de processos/práticas de testes diferentes seja encorajada durante a produção do jogo independente, é preciso salientar que desenvolvedores de jogos independentes operam, em sua maioria, com baixos recursos [7]. Processos/práticas de testes que possam abordar mais de um aspecto, como o *playtesting*, configuravam-se com maior aplicabilidade neste cenário em relação ao uso de outros processos/práticas. Por fim, Murphy-hill et al. [13] relatam a preferência dos desenvolvedores em testes com humanos a testes automatizados,

devido às constantes mudanças de requisitos durante o projeto e a requisitos subjetivos como diversão e experiência de usuário.

Tabela 3.6: Quantidade de processos/práticas de testes diferentes utilizados pelos participantes que realizam testes.

Categoria	Quantidade de Participantes	Percentual
Um tipo de processo/prática de testes	18	34,62%
Dois tipos de processos/práticas de testes	9	17,30%
Três ou mais tipos de processos/práticas de testes	25	48,08%
Total	52	100%

Essa pesquisa também buscou identificar quais metodologia(s), processo(s) ou prática(s) os desenvolvedores utilizam durante a produção de jogos independentes. A Figura 3.9 apresenta os resultados em relação a este aspecto. Dentre aqueles que indicaram utilizar alguma metodologia, observa-se um maior uso de metodologias ágeis, com 26 participantes (35,62%) indicando Scrum e 21 participantes (28,77%) indicando Kanban. Estudos tem explorado Scrum e seus benefícios durante a produção de jogos [22, 67, 68], e alguns tem apontado a maior utilização desta metodologia pelos desenvolvedores [38, 56, 69]. Ainda assim, 26 participantes (35,62%) apontaram não utilizar nenhuma metodologia/processo definido e oito participantes (10,96%) afirmaram desconhecer ou não saber responder que metodologia/processo/práticas utilizam. Cinco participantes (6,85%) afirmaram utilizar uma metodologia própria. Destes, dois participantes relataram utilizar uma metodologia híbrida com uso parcial de outras metodologias ágeis, como por exemplo o uso de quadros do Kanban e simplificação de Scrum. A citação abaixo evidencia a afirmação de um desses participantes. Os outros três participantes não deixaram explícitos detalhes sobre as metodologias utilizadas.

> *Normalmente, em JAMs [maratonas de desenvolvimento de jogos com curto espaço de tempo], faço/fazemos um briefing seguido de um protótipo com a ideia definida, para então passar por um QA [quality assurance] de avaliação de mecânicas e "diversão", para posteriormente sim continuar o desenvolvimento até termos um alpha jogável, um QA para estudo de bugs e termos uma beta finalizada. O processo é o mesmo em jogos de fora, porém seguindo pelas metodologias Kanban e sprints do scrum - Participante 58*

> *Acredito que trabalhe com alguma versão simplificada do Scrum, pois sou um dev solo. Meu processo se baseia na criação de um cronograma de todo*

> *o projeto. A partir dele faço definições de tarefas semanais visando atender os "pontos chaves"definido no cronograma geral. Caso alguma tarefa fique atrasada, eu revejo sua prioridade ou tento cumpri-la dentro da necessidade do projeto. - Participante 63*

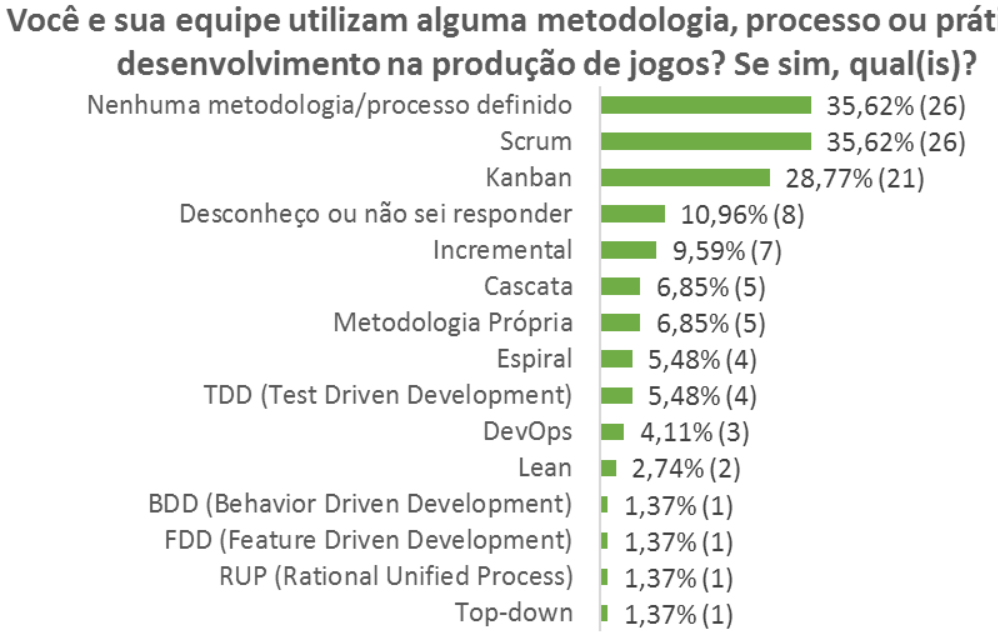

Figura 3.9: Metodologias, processos e práticas utilizadas pelos participantes. Fonte: Autor.

Para entender melhor como os processos/práticas de teste estão inseridos no processo de desenvolvimento de jogos, pediu-se, de maneira facultativa, que os participantes fornecessem detalhes de como ocorre o processo de produção de seus jogos. A partir dos dados coletados, analisou-se todas as atividades do processo de produção de jogos relatadas pelos participantes. Em seguida, categorizou-se essas atividades com as etapas de pré-produção, produção, testes e pós-produção definidas por Chandler [3] e apresentadas na subseção 2.3. Ao analisar as atividades de teste relacionadas as etapas, os participantes afirmaram que: (i) na etapa de pré-produção, um participante relatou a realização da fase *Alpha* antes de iniciar a etapa de produção; (ii) na etapa de produção, há a realização das fases *Alpha* e *Beta*. A maioria das afirmações analisadas descrevem que a fase *Alpha* ocorre nesta etapa. Adicionalmente, executa-se testes ao final desta etapa, de forma a identificar e corrigir erros no jogo antes de iniciar a etapa de pós-produção. Porém, alguns participantes não relataram uma distinção clara entre esta etapa e a etapa de testes; (iii) na etapa de testes, um participante relatou que há a realização de atividades de polimento do jogo; e, por fim, (iv) na etapa de pós-produção, não foram identificadas afirmações sobre atividades de testes. A Tabela 3.7 sumariza os resultados encontrados das atividades relativas

a testes informadas pelos participantes durante as diferentes etapas de produção do jogo. Já a Figura 3.10 apresenta uma visão detalhada das atividades de teste relatadas pelos participantes e sua afinidade com as etapas de desenvolvimento de jogos de Chandler [3], conforme apresentadas no Capítulo 2.

Tabela 3.7: Observações sobre as tarefas de testes relatadas durante a produção do jogo.

Etapas	Observações
Pré-produção	• Realização da fase *Alpha* antes de iniciar a etapa de produção.
Produção	• A maioria das tarefas relacionadas a testes encontram-se nesta etapa. • Realização das fases *Alpha* e *Beta*. • Realização de testes ao final desta etapa, de forma a identificar e corrigir erros no jogo antes de iniciar a etapa de pós-produção. Porém, não há distinção entre esta etapa e a etapa de testes. • Aplicação de testes ainda em nível de código-fonte, como teste unitário.
Testes	• Realização de atividades de polimento do jogo.
Pós-produção	*Não identificou-se relatos sobre testes nesta etapa.*

Em relação aos dados coletados sobre as ferramentas para gerenciamento de testes, relatórios de erro e *feedback* utilizadas pelos participantes, constatou-se que 53 destes (72,60%) não utilizam nenhuma ferramenta de gerenciamento de testes. O uso de ferramentas de testes é essencial para gerenciar o progresso de tarefas e garantir qualidade durante o desenvolvimento do jogo [3,16]. Existem ferramentas disponíveis no mercado para este tipo de tarefa, como as ferramentas de código-aberto MantisBT[5] e Bugzilla[6]. Ao negligenciar a utilização de uma ferramenta para o gerenciamento de testes, estes participantes correm o risco de serem pouco rigorosos em suas observações sobre os resultados dos testes executados.Os processos/práticas de testes apontados como mais utilizados (Figura 3.8) exigem um certo rigor e uma melhor estruturação para que os resultados sejam satisfatórios e melhorem a qualidade dos jogos. O profissional pode correr, ainda, o risco de não ter o registro de informações desde o registro do erro até as tratativas de resolução e fechamento deste erro.

[5]Disponível em `https://www.mantisbt.org/`.
[6]Disponível em `https://www.bugzilla.org/`.

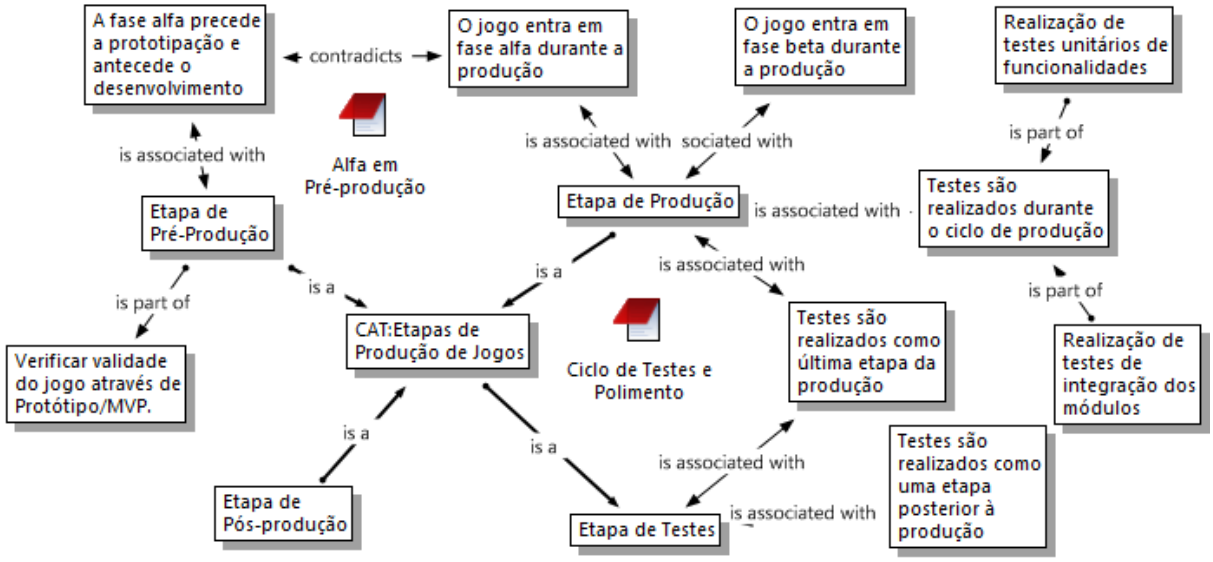

Figura 3.10: Visão detalhada das atividades de testes no processo de desenvolvimento de jogos dos participantes. Fonte: Autor.

Dentre os participantes que informaram ferramentas para gerenciamento de testes, percebeu-se que alguns descreveram também a forma como estes documentam seu processo de testes. A Tabela 3.8 apresenta as formas de gerenciamento de teste relatadas. A Figura 3.11 fornece uma visão mais detalhada da análise das ferramentas utilizadas, apresentando as ferramentas mais utilizadas e classificando-as pela natureza de sua forma de gerenciamento. Apenas cinco participantes informaram utilizar ferramentas específicas para gerenciamento de testes, onde estes citaram Github[7], Gitlab[8] e GamerTrials[9]. Destas ferramentas, a GamerTrials se configura como uma plataforma web para apoio no gerenciamento de testes em jogos independentes. O desenvolvedor recebe *feedback* sobre seu jogo de usuários da plataforma, onde estes recebem bonificações por testar os jogos disponíveis. Dentre as ferramentas adaptadas, destacam-se as ferramentas que utilizam quadros/post-its, como Trello[10], Jira[11] e Hacknplan[12], e as ferramentas que auxiliam com planilhas, como Microsoft Excel[13] e Google Spreadsheets[14].

[7]Disponível em `https://about.gitlab.com/`.
[8]Disponível em `https://github.com/`.
[9]Disponível em `https://www.gamertrials.com/`.
[10]Disponível em `https://trello.com/`.
[11]Disponível em `https://br.atlassian.com/software/jira`.
[12]Disponível em `https://hacknplan.com/`.
[13]Disponível em `https://products.office.com/pt-br/excel`.
[14]Disponível em `https://www.google.com/sheets/about/`.

Tabela 3.8: Formas de gerenciamento de testes apontadas pelos participantes.

Formas de Gerenciamento de Teste	Ferramentas Citadas
Quadros/post-its	Trello, Jira, Hacknplan
Ferramentas especializadas	GamerTrials, Github, Gitlab
Recursos de sistemas de publicação/distribuição de jogos	Voodoo.io
Formulários	Google Forms
Planilhas	Microsoft Excel, Google Spreadsheets
Documentos de texto	Google Docs
Bibliotecas e ferramentas de analytics	GameAnalytics, Unity Analytics
Fóruns e grupos em redes sociais	Facebook
Mensagens de e-mail	

Aspectos analisados ao realizar testes (QP3)

A questão de pesquisa QP3 visa identificar quais aspectos os desenvolvedores de jogos independentes buscam avaliar durante a realização dos processos/práticas de testes. Na produção do jogo, os testes avaliam aspectos, tais como grau de diversão ou quantidade de erros, para garantir o funcionamento correto e eficiente do jogo e suas funcionalidades [15]. Desta forma, melhora-se a qualidade do jogo ou auxilia-se nas tomadas de decisões do projeto [16].

Investigou-se que aspectos os desenvolvedores de jogos independentes buscam ao realizar os processos/práticas de testes em seus jogos. Ao analisar os dados coletados, foram identificados 16 aspectos relatados pelos participantes. Dentre os aspectos, destaca-se a identificação de erros e defeitos no jogo, a avaliação de aspectos de jogabilidade e a verificação do grau de diversão do jogo, descritos a seguir. A Tabela 3.9 apresenta as observações dos aspectos encontrados.

Os relatos sobre a identificação de erros e defeitos no jogo tratam da busca por *bugs* e falhas nas funcionalidades que foram previstas durante o planejamento do jogo. Alguns participantes mencionaram buscar por: (i) erros de movimentação e interação entre objetos; (ii) erros produzidos pela integração com *plugins* e kits

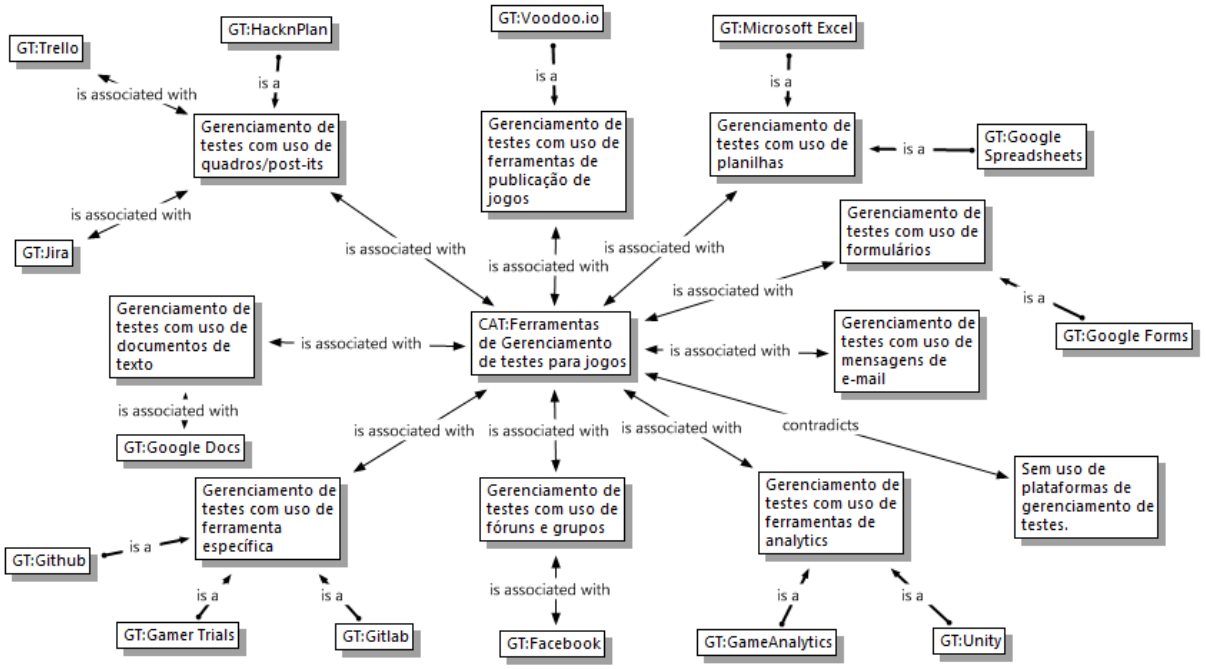

Figura 3.11: Visão detalhada das ferramentas relatadas pelos participantes. Fonte: Autor.

de desenvolvimento de software; e (iii) erros de lógica e de inteligência artificial de inimigos e personagens.

> *Erros em função de plugins/SDKs e defeitos de colisões em cenários - Participante 12*
>
> *Buscamos encontrar falhas de Gameplay [jogabilidade], tentamos encontrar formas de burlar as regras e maneiras de otimizar ao máximo o código do projeto para aumentar o público alvo. - Participante 69*

Entende-se por jogabilidade o conjunto de regras, desafios e mecânicas do jogo, bem como as respostas e estados dos elementos do jogo às ações que o jogador realiza durante o ato de jogar [70]. Dentre os relatos sobre jogabilidade, os participantes afirmaram se preocupar, além de aspectos gerais de jogabilidade, com o quanto uma mecânica é relevante e funcional, se uma mecânica é necessária ou se há problemas com os controles.

> *Gameplay fluída, a facilidade do jogador para entender os desafios propostos no jogo, descobrir os bugs existentes que atrapalham a gameplay, e saber se o jogador está realmente se divertindo com o jogo. - Participante 55*

Apesar de ser um fator subjetivo, a diversão é um dos principais requisitos atendidos pelos desenvolvedores na construção do jogo [13]. Segundo Levy e Novak [16], os principais objetivos de um testador são encontrar, replicar e relatar *bugs*.

Como objetivos secundários dos testadores, os autores elencam também verificar quais *bugs* foram corrigidos e garantir que o jogo é divertido. Os participantes relataram que, ao realizar testes, verificam o quão divertido está o jogo e suas mecânicas, além de verificar se comportamentos inesperados de funcionalidades podem se tornar diferenciais para proporcionar diversão.

> *Se o público alvo está se divertindo e se frustrando com a experiência nos momentos desejados. - Participante 26*
>
> *A diversão é fator principal. Por mais que algum elemento específico acabe tendo resultados diferentes do esperados, se divertidos, devem ser adicionados. - Participante 32*

Tabela 3.9: Aspectos buscados pelos participantes ao realizar testes.

Aspectos	Descrição
Erros e defeitos	Falhas produzidas pela programação do jogo, que podem afetar a mecânica, física, sistema de colisão, inteligência artificial, dentre outros.
Desempenho	Obter dados para analisar o desempenho e estabilidade do jogo na plataforma-alvo, com a finalidade de otimizar o código-fonte.
Jogabilidade	Regras, mecânicas e desafios são verificados com objetivo de identificar sua relevância, funcionalidade e fluidez.
Diversão	Observar se o jogo proporciona diversão ao jogador através de sua jogabilidade, enredo, aspectos visuais e sonoros, dentre outros.
Balanceamento	Observar se os desafios e tarefas do jogo oferecem uma boa curva de aprendizado, balanceando ainda desafio e diversão.
Novos elementos e mecânicas	Receber sugestões de novas mecânicas e elementos pelos jogadores, bem como aperfeiçoar as mecânicas e elementos existentes.
Projeto de níveis (*Level Design*)	Obstáculos, caminhos, posição de inimigos, desafios, dentre outros, de mapas e/ou fases são verificados para eventuais ajustes.

Tabela 3.9 – *Continuação da página anterior.*

Aspectos	Descrição
Clareza e facilidade do jogo e suas mecânicas	Observar se características de jogabilidade, como mecânicas, desafios e missões, estão claras para o jogador, garantindo eficiência e facilidade de uso.
Verificar se o jogo é intuitivo	Observar a facilidade do jogador em entender os desafios propostos e o que deve fazer, considerando a experiência antecedente do jogador.
Acessibilidade	Garantir que a navegação, interação e operação do jogo sejam acessíveis por jogadores, atendendo a necessidades especiais destes.
Experiência de usuário	Observar a percepção do jogador enquanto interage com o jogo, examinando suas emoções e respostas físicas e psicológicas.
Grau de interesse em continuar jogando	Observar como o jogador se envolve com o jogo e suas mecânicas, evitando frustrações e estimulando-o em um determinado ritmo.
Impressões sobre a estética sonora	Obter opinião dos jogadores sobre suas percepções e sensações em relação aos sons e efeitos sonoros do jogo.
Impressões sobre a estética visual	Obter opiniões dos jogadores sobre suas percepções e sensações em relação aos elementos visuais e de interface do jogo.

Após analisar os dados qualitativos, buscou-se categorizá-los em "Concepção e Planejamento", "Implementação" e "Interface e Interação". A Figura 3.12 apresenta os aspectos relatados pelos participantes, bem como sua afinidade com as categorias propostas.

A categoria "Concepção e Planejamento" reúne os aspectos ligados às mecânicas, enredo, missões, níveis, recompensas, itens e demais características que fornecem a essência do jogo. Dentre os aspectos relatados pelos participantes nesta categoria, destacam-se o balanceamento do jogo, a jogabilidade, a verificação do

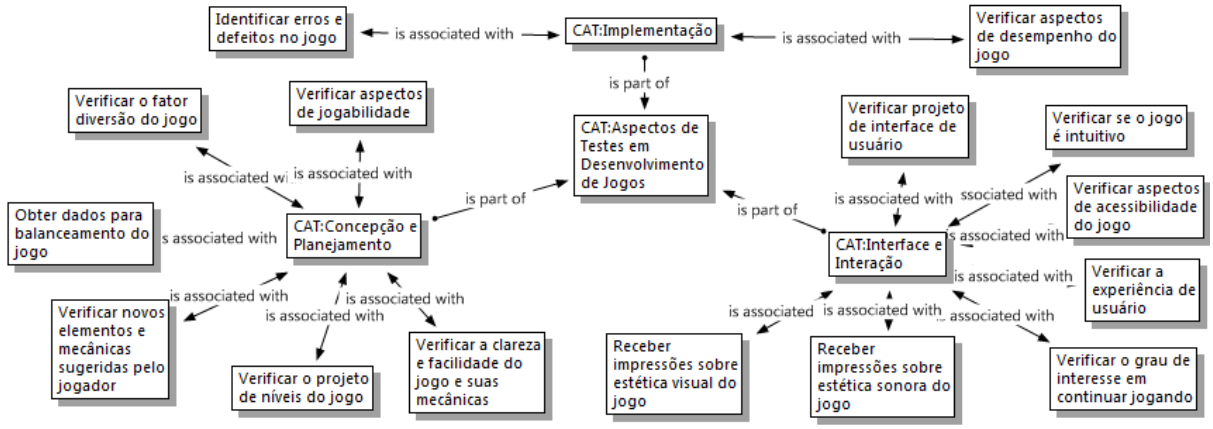

Figura 3.12: Aspectos relatados pelos participantes ao realizar testes. Fonte: Autor.

projeto de níveis, o grau de diversão, bem como receber sugestões de novos elementos e mecânicas.

A categoria "Implementação" abrange aspectos relativos a características inerentes à codificação do software do jogo, uma vez que é um produto de natureza computacional [13]. Dentre essas características, evidenciam-se rotinas de bibliotecas gráficas, implementação de algoritmos, inteligência artificial e a física do jogo. Os participantes relataram buscar como aspectos a análise do desempenho do jogo nas plataformas-alvo e a verificação de defeitos e erros de programação.

A categoria "Interface e Interação" abrange aspectos relacionados ao projeto de interface e ao design de interação entre o jogador e o jogo, bem como as percepções obtidas a partir destes. Nesta categoria, foram reunidos os seguintes aspectos dentre os relatados: observar a experiência do usuário, verificar o projeto de interface de usuário e acessibilidade, receber impressões sobre estética visual e sonora do jogo, analisar o grau de interesse do jogador em permanecer jogando, bem como identificar o quão intuitivo é o jogo.

Lewis et al [71] apresentam uma taxonomia para possíveis falhas, divididas em falhas temporais e não-temporais. As falhas temporais necessitam do conhecimento do estado anterior do contexto antes da falha, enquanto que falhas não-temporais pode ser inspecionadas a qualquer momento. Levy e Novak [16] apresentam nove categorias para erros: visual, áudio, projeto de nível, inteligência artificial, física, estabilidade, desempenho, rede e compatibilidade. Categorizar erros e *bugs* permite que os desenvolvedores possam construir uma forma mais sistemática para testar [71], bem como compreender as possíveis causas que os provocaram [16].

Os trabalhos mencionados tratam, em sua maioria, de falhas e erros de implementação. Isso significa dizer que, do ponto de vista do desenvolvedor, essas classificações não contemplam categorizar aspectos como a clareza e facilidade de uma mecânica, o balanceamento do jogo ou se o jogo é intuitivo, por exemplo. Kasurinen e Smolander [14] categorizaram, em três áreas, os objetivos das atividades de teste de sete empresas de desenvolvimento de jogos: (i) mecânicas de jogo; (ii) aspectos técnicos; e (iii) experiência de usuário. Eles observaram que os itens relacionados a experiência de usuário e usabilidade foram considerados mais importantes que os aspectos de qualidade geral de software, como confiabilidade e corretude das funcionalidades. Porém, em seu trabalho, Kasurinen e Smolander não forneceram detalhes sobre como construíram suas categorias e quais aspectos as compõem, fornecendo apenas generalizações dos objetivos de teste encontrados pelos autores.

3.2.3 Ameaças à validade

Wohlin et al. [48] enumeram quatro principais tipos de ameaça a validade: conclusão, interna, construção e externa. Neste estudo, buscou-se executar algumas ações com o propósito de mitigar ou minimizar as potenciais ameaças a validade. Para as mitigações, utilizou-se como base o trabalho de Ghazi et al [72].

Com relação à validade de conclusão, observou-se a definição da amostra na pesquisa. Pode haver um risco de não conseguir generalizar os resultados devido à amostra utilizada e o seu tamanho. Nesse contexto, foi feito o uso de contatos em fóruns de desenvolvimento de jogos no Brasil, cobrindo-se os canais de comunicação utilizados pelos desenvolvedores de jogos, utilizando a lista de canais apresentados no II Censo da Indústria Brasileira de Jogos Digitais (IBJD) [54].

Considerando a validade interna, um dos aspectos que pode impactar nos resultados é a seleção dos participantes. No entanto, os mesmos foram alcançados através de canais previamente utilizados em outras pesquisas. Além disso, ainda há a possibilidade dos participantes não responderem inteiramente ao questionário. Nesse contexto, foram tomadas algumas medidas como: (a) desenvolvimento de um questionário curto, com o intuito de diminuir a evasão dos participantes e obter o maior número de respostas possíveis; e (b) apresentação de um termo de consentimento livre e esclarecido apresentado no início do formulário da pesquisa de

opinião. Com estas medidas, espera-se impactar na confiabilidade dos resultados e evitar a percepção de que a pesquisa não tem benefícios para os participantes. Além disso, outra ameaça está relacionada ao projeto do questionário, visto que a qualidade das questões pode resultar no não entendimento do questionário. Nesse contexto, o questionário foi revisado por um pesquisador experiente na aplicação de pesquisas de opinião.

Com relação à validade de construção, por se tratar de uma pesquisa exploratória, não houve hipóteses pré-estabelecidas ou suposição de existência de uma teoria. Quanto à análise qualitativa de dados, existe uma ameaça uma vez que as codificações aberta e axial são processos criativos e interpretativos. Para diminuir o viés, estas codificações foram revisadas por outro pesquisador experiente em análises qualitativas, uma vez que a ferramenta utilizada mantém o registro dessas codificações. Desta forma, todo o processo de análise desta pesquisa é estritamente rastreável aos dados coletados.

Finalmente, com relação às ameaças da validade externa, as informações foram coletadas de profissionais da área de desenvolvimento de jogos digitais, com o intuito de entender aspectos de teste de software. Adicionalmente, descartou-se os dados dos participantes que citaram não desenvolver jogos independentes. Todavia, não foi possível cobrir toda a população de profissionais da indústria de jogos independentes.

3.3 Considerações Finais

Este capítulo apresentou uma pesquisa de opinião sobre uso de práticas de testes no desenvolvimento de jogos independentes. A pesquisa de opinião buscou caracterizar como e quais métodos e prática de testes são utilizados durante o ciclo de desenvolvimento dos jogos pelos desenvolvedores.

Para produzir jogos independentes e que sejam competitivos no mercado mundial, os profissionais precisam atender a um adequado nível de qualidade. Assim, durante a produção desses jogos independentes, é necessária a realização de testes para garantir que determinados aspectos do jogo (grau de diversão, execução correta das funcionalidades, etc) atendam ao nível espero pelos usuários. Neste trabalho,

executou-se uma pesquisa de opinião com o objetivo de identificar quais e como os processos e práticas de testes são utilizados durante o desenvolvimento dos jogos independentes.

Após a análise de 73 respostas de profissionais de jogos independentes, identificou-se que há uma preocupação com a realização de testes durante o desenvolvimento de seus jogos independentes. Entretanto, ainda há profissionais que desconhecem ou não executam nenhum processo/prática de testes durante a produção dos jogos. As técnicas de testes mais utilizadas apontadas pelos participantes são *Playtesting*, Testes de Usabilidade e realização de etapas de testes *Alpha* e *Beta*. Além disso, observou-se a ausência do uso de plataformas ou ferramentas de gerenciamento de testes. Isto pode indicar que estes participantes realizam seus testes de forma pouco rigorosa em suas observações sobre os resultados dos testes executados.

Categorizou-se formas de gerenciamento de testes a partir das ferramentas utilizadas, observando-se uma maior disposição para aplicação de ferramentas que utilizam quadros e para ferramentas especializadas. Em relação aos aspectos avaliados pelos participantes ao realizar testes, destacam-se a identificação de defeitos e erros, a verificação de aspectos de jogabilidade e a análise do grau de diversão do jogo. A partir da análise das características e afinidades dos aspectos relatados, estes foram agrupados em três categorias: (i) “Concepção e Planejamento”; (ii) “Implementação”; e (iii) “Interface e Interação”.

Ao analisar estes resultados, percebeu-se ainda que, das técnicas de testes utilizadas pelos desenvolvedores, poucas eram utilizadas em tempo de implementação do código-fonte do jogo. Técnicas como esta são conhecidas como técnicas de caixa branca e permitem aos desenvolvedores minimizar a quantidade de defeitos e erros em etapas posteriores da produção. Uma vez que os participantes citaram como um dos principais aspectos ao realizar testes a identificação de defeitos e erros, aumenta-se a necessidade de se utilizar práticas de desenvolvimento que atenda melhor este aspecto em específico. Isso contribui para que estudos sobre o uso BDD no desenvolvimento de jogos independentes possam ser realizados, uma vez que a prática utiliza testes unitários e de integração durante a implementação do código-fonte do software a ser desenvolvido.

O capítulo seguinte apresenta um estudo de campo sobre a prática BDD durante o desenvolvimento de jogos independentes. Desta forma, pretende-se verificar sua utilização durante o ciclo de desenvolvimento de jogos independentes e entender como seu uso pode favorecer a produção destes. Assim, se possibilitará posteriormente identificar como os aspectos de testes podem ser trabalhados com o uso de BDD, bem como listar especificidades que permitam utilizar BDD observando as características apresentadas neste capítulo.

4 Estudo de Campo de BDD no Desenvolvimento de Jogos

Este capítulo apresenta um estudo exploratório sobre o uso do BDD no ciclo de produção de jogos, com a finalidade de identificar as características que favoreçam ou prejudiquem sua utilização durante o ciclo e dar base à prática BDD-I proposta nesta dissertação. Assim, este capítulo relata o estudo de campo realizado, sua metodologia e resultados encontrados.

4.1 Planejamento

Estudos de caso tem sido conduzidos [73–75] com o objetivo de aprofundar o conhecimento humano acerca do desenvolvimento de jogos, uma vez que é uma área multidisciplinar, sujeita a diferentes condições alheias ao seu contexto [56]. Estudos de caso são métodos experimentais que objetivam investigar fenômenos contemporâneos em seus contextos, oferecendo uma abordagem onde não se precisa restringir os limites entre o objeto de estudo e seu ambiente [48, p. 57–58]. Paralelamente ao uso do termo "estudos de caso", estudos de campo vem sendo utilizado também na literatura para expressar técnicas de coleta de informação básica para desenvolver teorias ou entender práticas [5].

Desta forma, idealizou-se um estudo de campo para identificar indícios sobre o uso de BDD durante o processo de desenvolvimento de jogos independentes. Para isso, foram seguidas as diretrizes definidas por Runeson e Höst [76] para condução e relato de estudos de caso em engenharia de software. Para os fins do estudo, definiram-se duas unidades de análise: (a) desenvolvimento de jogo por profissionais e (b) desenvolvimento de jogo por desenvolvedores com pouca experiência. Estas unidades de análise foram planejadas com o objetivo de fornecer diferentes visões acerca das questões de pesquisa, apresentadas na Tabela 4.1.

Tabela 4.1: Questões de pesquisa do estudo de campo de BDD.

QP1	É possível utilizar BDD como prática no desenvolvimento de jogos?
QP2	O uso da linguagem ubíqua favorece a comunicação entre desenvolvedores de jogos?
QP3	Há dificuldades em especificar os comportamentos das funcionalidades de jogos? Quais?
QP4	Que funcionalidades de jogos não é possível especificar como comportamentos?

Para a unidade de análise A, os participantes deveriam receber um treinamento sobre BDD e seu ciclo no contexto do desenvolvimento de software. Os participantes então deveriam cooperar para criar um jogo de temática livre, em um ciclo de desenvolvimento apoiado com uso do BDD, simulando as etapas de pré-produção, produção e testes do desenvolvimento de jogos definidas por Chandler [3]. Os participantes dessa unidade deveriam ser desenvolvedores de jogos com mais de 3 anos de experiência

A unidade de análise B deveria consistir de uma oficina, guiada por um tutor, com apoio do BDD na construção de um jogo. Durante a oficina, os participantes seriam incentivados a criar novos comportamentos ao jogo, apoiados com as práticas do BDD. Os participantes deveriam ser desenvolvedores com pouca experiência no desenvolvimento de jogos.

Em ambas as unidades de análise, os participantes deveriam utilizar o motor de jogo Unity[1]. A ferramenta foi escolhida por ser amplamente utilizada por desenvolvedores de jogos independentes[2] e por apresentar um plugin que permite o uso da prática do BDD com o desenvolvimento de jogos. Assim, definiu-se a Unity versão 2017.3, utilizando C# como linguagem de programação para os *scripts* do jogo. A ferramenta BDD planejada para o estudo de campo foi o *plugin* "BDD Extension Framework for Unity Test Tools"[3], que utiliza a biblioteca "Unity Test Tools"[4] para execução dos códigos de teste.

[1]Disponível em `https://unity.com/`.

[2]`https://www.pcgamer.com/uk/what-indie-developers-think-of-unity-in-2018/`

[3]Disponível em `http://www.huddimension.co.uk/`.

[4]Disponível em `https://bitbucket.org/Unity-Technologies/unitytesttools/`.

4.1.1 Coleta de Dados

Em um estudo de campo, é importante utilizar diversas fontes de dados a fim de aumentar a precisão do estudo experimental, objetivando a triangulação [76]. Stake [77] apresenta quatro tipos de triangulação: fonte, observador, metodologia e teoria.

Para garantir a triangulação de dados deste estudo de campo, foram utilizados quatro técnicas de coleta de dados: entrevista, anotações de campo, gravação em vídeo e análise de artefatos. Lethbridge et al. [5] classificam as diferentes técnicas de coleta de dados em níveis. O primeiro grau são métodos diretos onde o pesquisador está em contato com os indivíduos e coleta dados em tempo real. O segundo grau apresenta métodos indiretos onde o pesquisador coleta dados sem realmente interagir com os indivíduos. O terceiro grau apresenta métodos que analisam artefatos de trabalhos e outros dados disponíveis. A tabela 4.2 apresenta os métodos de coleta utilizados e seus níveis segundo a classificação de Lethbridge et al. [5].

Tabela 4.2: Métodos de Coleta utilizados e níveis segundo Lethbridge et al. [5].

Método de Coleta	Nível
Entrevista	Primeiro Grau
Anotações de Campo	Primeiro Grau
Gravação em vídeo	Segundo Grau
Artefatos e Código-fonte	Terceiro Grau

Para coleta de dados utilizando a técnica de entrevista, optou-se por realizar uma entrevista semi-estruturada [78]. Planejou-se então um questionário de perguntas a serem aplicadas ao final do experimento, com base nas questões de pesquisa do estudo. A ordem das perguntas deveria ser seguida tal qual o planejado, mas os pesquisadores poderiam realizar novas questões caso necessitassem de mais informações acerca dos dados. O questionário final aplicado na entrevista encontra-se apresentado na tabela C.1 do Apêndice C.

Uma vez que o processo de desenvolvimento de jogos utilizando BDD e a quantidade de participantes nas duas unidades de análise seriam diferentes, planejou-se aplicar a entrevista com os participantes da unidade de análise A diretamente com

o pesquisador, sendo esta gravada em formato de vídeo. Para os participantes da unidade de análise B, o questionário deveria ser aplicado por meio de formulário eletrônico utilizando a ferramenta Google Forms[5].

Durante a execução da unidade de análise A, os participantes deveriam ser gravados realizando suas atividades. Dessa forma, os pesquisadores poderiam reproduzir novamente diálogos que ocorressem durante o processo. Isso também deveria permitir que anotações de campo pudessem ser reanalisadas sobre diferentes contextos. Durante a execução da unidade de análise B, apenas anotações de campo seriam utilizadas, devido a impossibilidade técnica de realizar a gravação em vídeo, em razão do maior número de participantes desta unidade de análise.

Durante o estudo de campo, os participantes de ambas as unidades de análise deveriam produzir especificações de comportamentos utilizando a linguagem ubíqua do BDD, além de classes de teste que representariam estes comportamentos. Todos estes artefatos deveriam ser coletados.

4.1.2 Análise de Dados

Para a análise de dados, optou-se em utilizar os mesmos procedimentos adotados na análise do *survey* "Testes em Jogos Independentes" (Capítulo 3), descritos em Subseção 3.1.4 desta dissertação. Assim como na análise do *survey* mencionado, não houve necessidade de realização da etapa de codificação seletiva definida pela *Grounded Theory* (GT), pois os resultados desejados foram alcançados ao conduzir a etapa de codificação aberta e a etapa de codificação axial [60]. As anotações de campo encontram-se no Apêndice D e os artefatos produzidos encontram-se no Apêndice E.

4.2 Condução

O estudo de campo ocorreu conforme o planejamento definido inicialmente. As duas unidades de análise deste estudo de campo produziram jogos utilizando BDD, a Linguagem Ubíqua e a ferramenta BDD definida. Foram criados dois documentos de apoio para as unidades de análise: o primeiro documento apresenta o ciclo do BDD

[5]Disponível em `https://www.google.com/forms/about/`

e a especificação da Linguagem Ubíqua; o segundo documento apresenta um guia sobre como utilizar a ferramenta "BDD Extension Framework for Unity Test Tools" com o motor de jogo Unity. Durante a execução de ambas as unidades de análise, os participantes poderiam consultar estes documentos sempre que necessitassem.

4.2.1 Unidade de Análise A

A unidade de análise A contou com a participação de dois profissionais de uma empresa de desenvolvimento de jogos do estado do Maranhão, graduados em Ciência da Computação. Eles já atuaram na produção de mais de dez jogos e possuem experiência de mais de três anos na área. Um dos participantes possuía outra atividade econômica, na qual desenvolvia *apps* e páginas web. Ambos não conheciam BDD, nem haviam utilizados antes outras práticas como TDD e DDD.

O processo de desenvolvimento do jogo iniciou pelo *brainstorming*. Os participantes se basearam nas mecânicas de outro jogo popular chamado Reigns[6] para a construção do jogo alvo deste estudo de campo. No jogo idealizado, de gênero estratégia, o jogador assume o papel de um desenvolvedor de jogos que precisa administrar seu estúdio de desenvolvimento através de uma série de decisões que podem afetar áreas da empresa. O objetivo é equilibrar as diferentes áreas, evitando que cheguem a 0 (zero). Após o *brainstorming*, o desenvolvimento do jogo passou pelas etapas de pré-produção, produção e testes, em um intervalo de dez horas de duração.

Os participantes escolheram duas das principais funcionalidades do jogo para o processo de desenvolvimento: mecânica de decisões e gerenciamento de áreas. Estas funcionalidades foram descritas utilizando a Linguagem Ubíqua do BDD. Eles descreveram, em texto plano, oito cenários que contemplavam os comportamentos do ciclo básico do jogo idealizado. Estes cenários foram mapeados para classe de testes na ferramenta BDD, com o uso de atributos na linguagem C# (recurso semelhante a *Annotations* da linguagem Java).

Ao final do desenvolvimento, conduziu-se a entrevista com os dois participantes. Durante a entrevista, percebeu-se a necessidade de se compreender um pouco mais sobre a ferramenta BDD utilizada. Assim, solicitou-se aos participantes que indicassem fatores limitantes da ferramenta que impossibilitaram que a execução

[6]Mais informações sobre o jogo em http://reignsgame.com/.

de testes fosse realizada com mais facilidade (questão 9 do questionário apresentado no Apêndice C). A entrevista foi gravada em vídeo e as respostas dos desenvolvedores foram transcritas em texto para a etapa de análise.

4.2.2 Unidade de Análise B

A unidade de análise B contou com doze participantes, dentre eles alunos de graduação do curso de Ciência da Computação e mestrandos em Ciência da Computação. Destes participantes, dez já haviam desenvolvido ao menos um jogo anteriormente.

O processo de desenvolvimento do jogo nesta unidade de análise teve um escopo reduzido. Uma vez que o jogo deveria ser conduzido por um tutor, a etapa de pré-produção foi realizada previamente. Assim, a execução consistiu apenas das etapas de produção e testes, onde estas foram realizadas em quatro horas de duração.

O jogo desenvolvido, de gênero esporte, era baseado nas mecânicas de um jogo de boliche. O jogador teria como objetivo derrubar dez pinos devidamente organizados com uma bola a ser arremessada a uma determinada distância destes. Este jogo foi escolhido uma vez que o jogo de boliche é popular, facilitando com que os participantes pudessem construir novos cenários a partir das regras do jogo.

Inicialmente, foi ministrado um treinamento aos participantes sobre o motor de jogo Unity e a ferramenta "BDD Extension Framework for Unity Test Tools". Após o treinamento, eles receberam um conjunto de quatro cenários, em Linguagem Ubíqua, que descreviam comportamentos da funcionalidade "Arremesso da bola", os quais foram implementados juntamente com o instrutor. Em seguida, os participantes foram convidados a criar novos cenários de forma a expandir as funcionalidades do jogo. Por fim, os participantes responderam ao questionário (Apêndice C) e disponibilizaram, em campo específico, os novos cenários sugeridos por eles com o uso da Linguagem Ubíqua.

4.3 Resultados

Os dados coletados no estudo de campo, tais como transcrições de entrevista e questionário, códigos-fonte e artefatos de especificação de cenários, foram importados e analisados utilizando o software ATLAS/ti. A Figura 4.1 apresenta uma visão condensada dos resultados deste estudo de campo. Os tópicos a seguir apresentam estes resultados em relação às questões de pesquisa deste estudo.

Uso de BDD no desenvolvimento de jogos (QP1)

Os participantes relataram que BDD permitiu que pudessem planejar o desenvolvimento do jogo. Ao especificar os comportamentos antes de iniciar qualquer linha de código-fonte do jogo, eles tiveram uma visão melhor das mecânicas do jogo a ser desenvolvido.

> *"Concordo, eu concordo! Tipo, é que tu fez o planejamento de antemão do que seria o jogo né?" - Participante 1 - Unidade de Análise A*
>
> *"Planejamento rápido, digamos assim?" - Participante 2 - Unidade de Análise A*

Ao realizar o planejamento de um jogo, é comum a divisão em objetivos e tarefas menores que completam o todo [3, 22]. Os participantes apontaram que, ao utilizar BDD, puderam também organizar em objetivos, onde as funcionalidades viraram tarefas e os cenários forneciam as especificações das funcionalidades, de forma que pudessem ser implementadas de maneira direcionada.

> *"É. Então a gente consegue mesmo fazer o jogo já pensando no que foi feito antes e tendo um enfoco já em objetivos né? Tipo, o objetivo é fazer aqueles comportamentos que a gente tinha pensado antes. Então, fica bacana pra direcionar então." - Participante 2 - Unidade de Análise A*

Foi apontado pelos participantes a semelhança de BDD com outras metodologias. North [19] não apresenta BDD como uma metodologia, mas como um conjunto de práticas adaptáveis à outras metodologias e processos. Para dois participantes, a prática do BDD se assemelhou com a metodologia Scrum. Essa semelhança pode ocorrer uma vez que a especificação de cenários e funcionalidades em BDD são baseadas nas estórias de usuários muito utilizadas pelas metodologias ágeis [19].

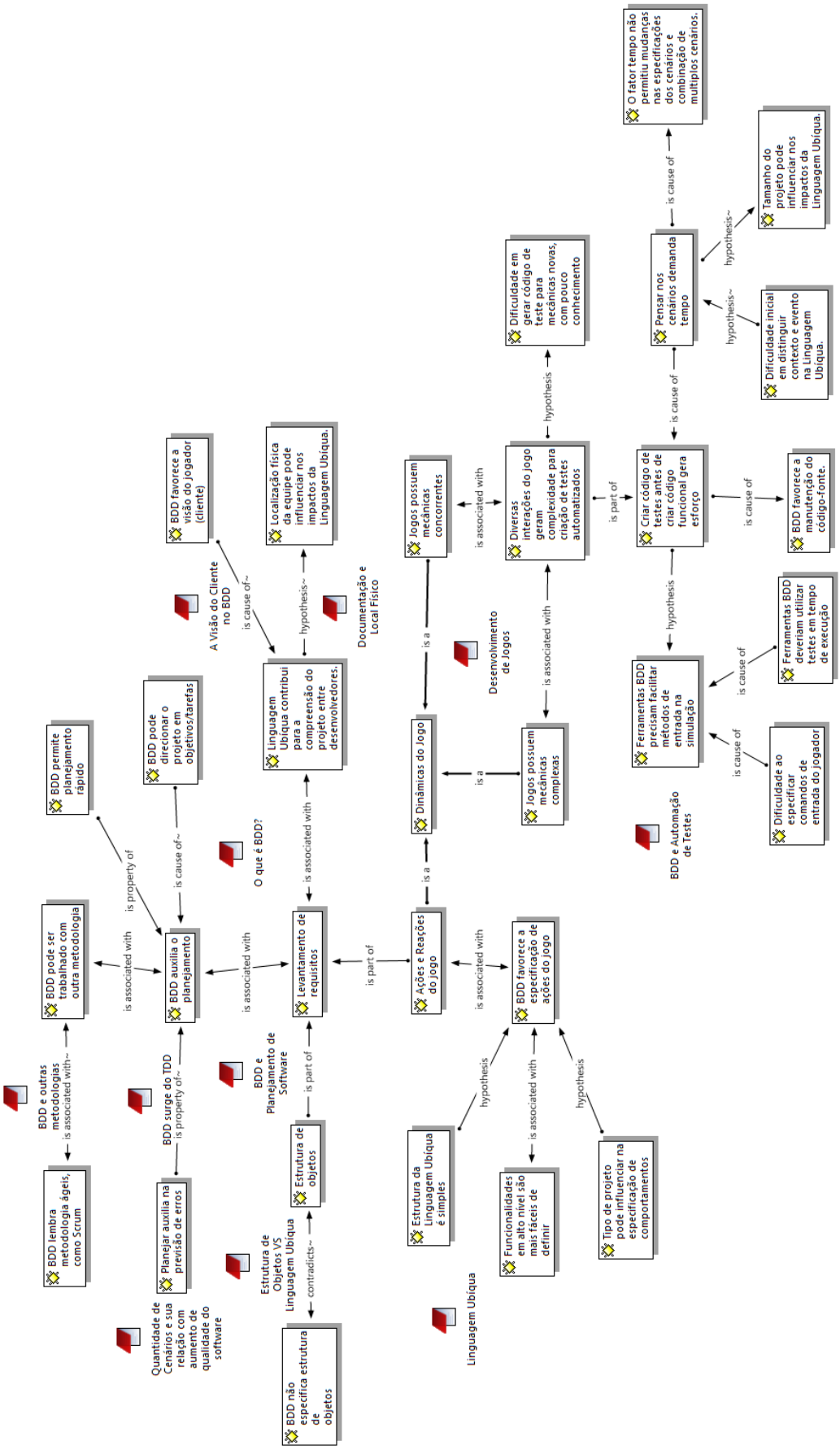

Figura 4.1: Visão Condensada dos Resultados deste estudo de campo. Fonte: Autor.

> *"Eu pensaria tipo... sei lá, juntar isso do BDD pra fazer um pré-planejamento e juntar isso com Scrum, por exemplo, daí esses cenários todos se tornariam cartões do método Scrum. Aí tu ia conseguir construir quais eram os cenários, direcionar o que tem de ser feito em si... as ações que tem de ser programadas e isso gerariam demandas pros cards, que são coisas um pouco mais alto nível." - Participante 1 - Unidade de Análise A*

> *"Sim, pois é um método útil, assim como o scrum e outros existentes" - Participante 13 - Unidade de Análise B*

Outro relato dos participantes foi que o uso do BDD auxilia na previsão de erros. O BDD surge como uma proposta de ajudar a melhorar a qualidade do software, aperfeiçoando o benefícios das técnicas TDD e *Acceptance Test-Driven Development* (ATDD), bases do BDD [25]. Durante a especificação dos cenários das funcionalidades, o praticante de BDD descreve exemplos reais para as especificações. Isso contribui porque fornece ao desenvolvedor dados de forma que ele implemente as funcionalidades de acordo com o que realmente se espera como saída [79].

> *"...Então nessa parte eu achei que ajudou bastante por exemplo... tem uma ação X no sistema que vai desencadear três, quatro reações diferentes. Então quando tu faz um sistema desses, um pensamento desses antes de implementar, tu já tem algumas coisas que tu tem em mente que pode dar errado. Então tu já consegue meio que prever aquilo, com certeza pode dar outros erros mas pelo menos a maior parte deles tu já consegue prever, porque tu já pensou antes então tu já até programa já pensando naquilo pra evitar acontecer." - Participante 1 - Unidade de Análise A*

Foi apontado ainda que o uso de BDD favorece a manutenção do código-fonte do jogo. Para os participantes, BDD deixa o código mais organizado e com os testes, modificações podem ser realizadas de forma segura. Uma vez que os desenvolvedores especificam cenários com dados reais, eventuais modificações no código permitirão perceber facilmente se algo não se comporta como deveria.

> *"...ajuda muito a fazer o código do jogo já pensando em todas as funcionalidades o que torna o código mais organizado e mais otimizado." - Participante 4 - Unidade de Análise B*

> *"...a implementação fica mais organizada para futuras correções e updates." - Participante 11 - Unidade de Análise B*

Uso da Linguagem Ubíqua (QP2)

Sobre o uso da Linguagem Ubíqua e seus benefícios quanto à comunicação dos desenvolvedores, os participantes relataram que especificar requisitos do

jogo utilizando a Linguagem Ubíqua contribuiu de forma satisfatória para a compreensão do projeto entre os desenvolvedores. Eles puderam especificar juntos as funcionalidades e observavam de maneira objetiva como cada uma deveria se comportar. Buscou-se durante o estudo de campo verificar o quanto eles interagiam com as especificações do projeto. Nas observações de campo, o documento de requisitos foi atualizado poucas vezes durante o processo. Quando indagados sobre o motivo de não haver sido atualizado o documento de requisitos durante o processo, os participantes apontaram a questão da localização física, que permitia que eles pudessem modificar eventuais características do projeto apenas comunicando verbalmente ao outro participante. Outro apontamento por parte dos participantes foi que isso não se mostrou necessário uma vez que "o que se foi planejado no início do desenvolvimento permaneceu até o final do desenvolvimento".

> *"Também pelo fato da gente ta aqui um do lado do outro, então quando tinha qualquer coisa a gente já falava direto e..." - Participante 2 - Unidade de Análise A*
>
> *"É, teve isso também." - Participante 1 - Unidade de Análise A*

Funcionalidades de Jogos e Comportamentos (QP3 e QP4)

Os participantes especificaram, em Linguagem Ubíqua, as ações e reações que contemplam as dinâmica dos jogos desenvolvidos. A estrutura da linguagem foi descrita como simples pelos participantes e que isso favoreceu que pudessem especificações em pares as ações do jogo. Porém, não foi possível colher evidências que possam verificar se a Linguagem Ubíqua favoreça todos os tipos de jogos.

Os participantes precisaram especificar a estrutura dos objetos a serem implementados. Uma parte do levantamento de requisitos é analisar estruturalmente como cada entidade é no sistema. Assim como no desenvolvimento de aplicações OO, no desenvolvimento de jogos também especifica-se objetos e atributos. BDD é sobre especificar comportamentos, como cada objeto deve se comportar em um contexto e o que se espera de saída caso um determinado evento aconteça. A Linguagem Ubíqua proposta originalmente por Dan North não contempla a especificação estrutural de objetos [80]. Para Evans [24], idealizador do DDD, a linguagem ubíqua não se limita a uma só forma de comunicação: uso de diagramas e outras ferramentas podem ser utilizadas. Para sua técnica, North define a linguagem ubíqua de forma a especificar

comportamentos que servirão de base para criação dos testes de aceitação. Durante o estudo de campo, os participantes foram encorajados a utilizarem a Linguagem Ubíqua, porém não limitou-se apenas o seu uso nem induziu-se que os participantes utilizassem outras formas para facilitar a especificação. Na unidade de análise A, onde os participantes especificaram os requisitos em Linguagem Ubíqua, estes descreveram a estrutura de objetos e seus atributos utilizando texto plano, com uso de identação.

> *"Err, eu senti dificuldades em especificar algumas... um elemento como objeto, digamos assim. Err, os cenários e ações, ok! Mas tem algumas ações que precisam de interação de alguns objetos. Eu senti dificuldades até porque eu não conhecia tão bem. Se eu deveria ter um lugar dizendo como aquele objeto é, estruturalmente... Então, lá quando eu fosse descrever os cenários eu teria como descrever o comportamento dele baseado nisso. Foi apenas os cenários e olha, isso é tipo reativo, foi basicamente reativo. Se acontece uma ação tem que ter isso, se acontece uma ação tem que ter aquilo. Esse ponto acho que não tinha... eu não consegui fazer um pouco mais estrutural, digamos assim." - Participante 1 - Unidade de Análise A*

Ao realizar o histórico dos participantes do estudo de campo, os mesmos apontaram que não utilizam a prática de realizar testes ao produzir seus jogos. Além disso, 11 dos 14 participantes do estudo de campo desconheciam a prática do TDD e nunca tinham criado testes antes de escrever o código funcional de seus jogos. Sobre realizar esta prática, os participantes afirmaram que isso demandava muito esforço. Para os participantes, jogos possuem diversas interações, mecânicas complexas e, por vezes, concorrentes. Os participantes da unidade de análise A se propuseram a desenvolver um jogo relativamente simples, porém tinham pouco conhecimento acerca da mecânica a ser desenvolvida. Este fato também que gerou maior dificuldade para que os participantes pudessem criar testes automatizados durante o estudo de campo.

> *"...Mas a parte de programar os testes eu pensaria duas vezes ainda, dependendo do tamanho do projeto. Poderia levar um tempo muito grande do desenvolvimento." - Participante 1 - Unidade de Análise A*
>
> *"...acho que a implementação em si dentro do sistema acaba tomando bastante tempo e às vezes gera mais problemas que você teria caso não tivesse né, como a gente passou bastante tempo debugando os próprios testes aqui, que a gente tava com dificuldades na estruturação." - Participante 2 - Unidade de Análise A*

Outro ponto apontado pelos participantes é que pensar nos cenários demandava muito tempo. Essa característica não favoreceu mudanças nas

especificações dos cenários e combinações de múltiplos cenários descritos para as funcionalidades. Acredita-se que essa demanda de tempo possa ter sido resultado da pouca prática de uso do BDD, uma vez que foi observado uma dificuldade em distinguir contexto de evento na linguagem ubíqua.

> *"Aí vem aquele problema que falei, de gastar muito tempo porque tu vai ter de prever todas essas combinações possíveis pra fazer o levantamento dos cenários. Aí imagina quantos cenários são... multiplica várias por várias possibilidades." - Participante 1 - Unidade de Análise A*

Sobre o uso da ferramenta BDD, ela se demonstrou bem versátil durante o experimento, porém foi questionada quanto à forma de utilização de entrada de dados. Para os participantes, houve dificuldades em mapear eventos da linguagem ubíqua como apertar um botão para as classes de testes, que acreditam que ferramentas BDD deveriam utilizar testes com entradas de dados em tempo de execução de forma que possam utilizar os periféricos. Durante a execução de um jogo, o jogador utiliza em geral periféricos como mouses, teclados e controles para controlar seu personagem. Na simulação, a ferramenta não contava com formas de favorecer a utilização de tais periféricos. Foi percebido durante o processo que os participantes não compreendiam como seria realizada a simulação de eventos de entrada através da automação de testes realizado pela ferramenta.

> *"Eu gostaria muito que ela funcionasse em tempo é... real, como eu já vi algumas em tempo de execução. Eu já vi algumas de web que faz isso. Então, eu defini um teste que é, por exemplo, arrastar um objeto, e ela ficar executando, esperar eu executar aquela ação. Eu não ter que simular com alguns casos ter de fazer... Com o drag [arrastar] do mouse, como ele não faz, eu tive de simular dentro do código. Então me gerou mais esforço ainda pra programar isso. Se eu simplesmente ó, faça o código tem que ter esse cenário aqui. Aí eu boto pra ele ficar executando, escutando a ação. Aí eu vou executar esse teste, entao ele espera que eu execute aquela ação, e depois daquela ação eu tenho um resultado, um retorno. Isso seria o ideal, do que simplesmente executar automático de uma vez, porque aí eu tenho de ficar simulando as coisas, daí o teste ficava uns testes que são inúteis porque eu tive de simular, não vai refletir uma ação real do jogador. Aí isso pra mim foi o pior problema." - Participante 1 - Unidade de Análise A*

4.4 Comparação com a Literatura

Para corroborar os resultados, compara-se estes com estudos anteriores sobre BDD e TDD no desenvolvimento de jogos e outros domínios. As comparações mais relevantes seguem nas próximas seções.

4.4.1 Planejamento de Jogos

O ciclo de desenvolvimento do BDD busca contemplar as etapas do Ciclo de Vida de Desenvolvimento de Software, apresentadas na Figura 4.2. Ao observarmos o ciclo de desenvolvimento do BDD, apresentado na Figura 4.3, três dos cinco passos do BDD encontram-se na etapa de Planejamento e Análise de Requisitos. Isso também foi apontado no trabalho de Lazar, no qual ele apresenta um modelo com as características de planejamento do BDD utilizando um diagrama UML [81]. Isso fornece indícios para compreender a proposta do uso do BDD quanto ao planejamento do software, que em geral é favorecido por práticas que facilitam a comunicação e a colaboração para criar documentações técnicas e de requisitos.

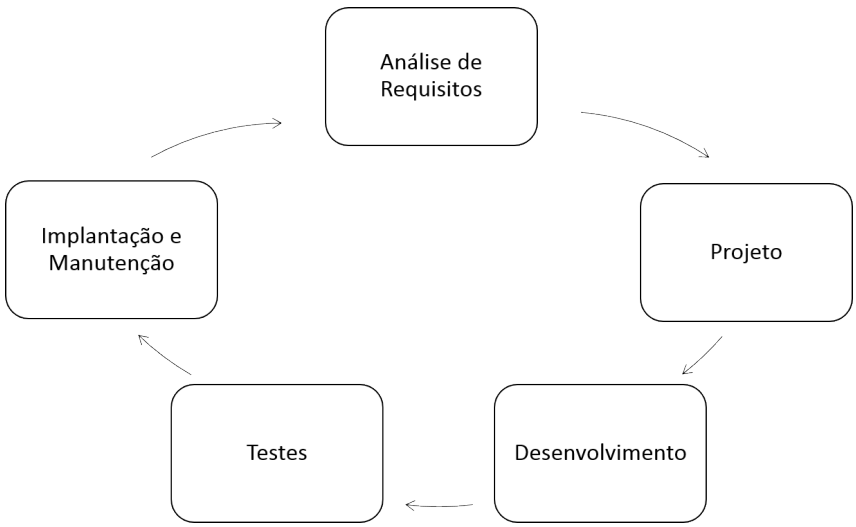

Figura 4.2: Ciclo de Vida do Desenvolvimento de Software. Adaptado de Egbreghts [4] pelo Autor.

Por não haver ainda um consenso na literatura sobre como definir BDD como metodologia, prática ou técnica, isso dificulta compará-lo com outras metodologias. Egbreghts [4] apresenta em sua revisão de literatura que, em alguns estudos, BDD é apresentado como técnicas de especificação, como técnicas para

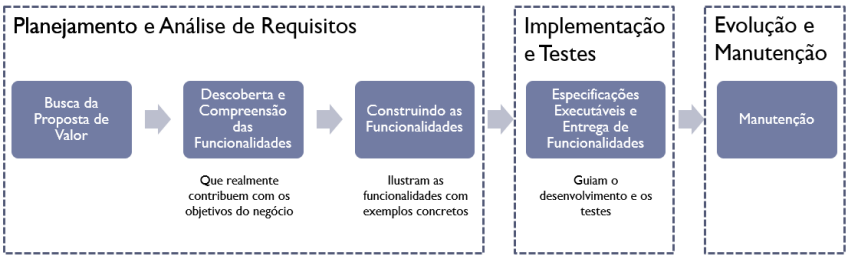

Figura 4.3: Ciclo de Desenvolvimento do BDD. Adaptado de Smart [2] pelo Autor.

projeto ou como método de desenvolvimento ágil, por exemplo. Ainda sim, estudos utilizaram BDD com outras metodologias para alcançar objetivos em determinados contextos, como educação [82], qualidade de software [83], contexto web [84]. Alguns estudos apontam o uso de Scrum com desenvolvimento de jogos [39, 67], o que permite trabalhos futuros que explorem a relação de BDD com esta metodologia no desenvolvimento de jogos.

4.4.2 Testes de Jogos

O esforço para criação de testes também foi relatado nos trabalhos [21, 85], porém os mesmos enumeram benefícios como exemplo a melhoria no projeto de código, medida de progresso e estabilidade de versão. LLopis e Houghton [21] ressaltam que a sua experiência em TDD, como qualquer outra nova técnica de desenvolvimento, é lenta no início, até que você se familiarize com a prática. Eles afirmam ainda que o custo com mudanças no código são menores quando comparado com metodologias tradicionais de desenvolvimento. Murphy-hill et al. [13] apresentam o relato de desenvolvedores que apontam que a complexidade de criação de testes automatizados para jogos é alta, relatando que testes automatizados são frágeis a frequentes mudanças, reduzindo a agilidade. Essas frequentes mudanças e o relato de grande número de mecânicas complexas e concorrentes podem também ser observadas no trabalho de Kasurinen e Smolander [49].

Um dos relatos controversos diz respeito ao modo de entrada de dados. Os participantes da unidade de análise A relataram que ferramentas automatizadas não permitem comandos de entrada realizados pelo testador durante a execução dos testes. Uma das premissas do BDD é a utilização de testes automatizados, sem a interferência de testadores ou desenvolvedores. Assim, faz-se necessário que a entrada de dados

também seja fornecida de forma automática durantes os testes. Llopis e Houghton [21], ao utilizarem TDD, criam classes dublês que simulavam os métodos de entrada de dados.

Os participantes constataram ainda que BDD ajudou-os a promover uma melhor manutenção e integração do código-fonte. Isso também foi observado no trabalho de Daylamani-zad et al. [86], onde estes utilizaram BDD para verificar o desempenho da arquitetura do framework Lu-Lu, utilizando cenários BDD como forma de verificar integração dos módulos da arquitetura proposta.

4.5 Considerações Finais

Este capítulo apresentou um estudo de campo utilizando BDD no desenvolvimento de jogos. Foram realizadas duas unidades de análises a fim de observar dois perfis de desenvolvedores utilizando as práticas de BDD durante o ciclo de desenvolvimento.

Os dados foram coletados e analisados utilizando GT, o que permitiu analisar as transcrições de entrevistas, respostas em formulário, artefatos produzidos e código-fonte dos jogos. Foram realizadas as etapas de codificação aberta e axial, o que permitiu criar um categoria de códigos que permitiram responder às questões de pesquisa deste estudo.

O uso de BDD se demonstra possível no ciclo de desenvolvimento de jogos, mas é necessário observar as especificidades inerentes ao desenvolvimento de jogos independentes. A Linguagem Ubíqua, componente-chave do BDD, possui uma estrutura que pode ser utilizada para especificação de testes de jogos. Porém, é necessário fornecer aoz desenvolvedores de jogos uma maior elucidação sobre sua sintaxe e utilização em diferentes objetivos de testes a serem trabalhados. É preciso compreender, por exemplo, que a Linguagem Ubíqua fortemente é utilizada para criação de cenários de testes. Logo, sua sintaxe não apresenta suporte para descrição de estruturas de objetos. Adicionalmente, as ferramentas BDD precisam facilitar a criação e execução de testes, para auxiliar a simulação de entrada de dados, por exemplo.

Com base nestes resultados e nos resultados de capítulos anteriores, construiu-se um conjunto de especificações que permitem utilizar BDD para utilização

no desenvolvimento de jogos independentes. Este conjunto de especificações, denominado *Behavior-Driven Development for Indies*, é apresentado no próximo capítulo, com o objetivo de permitir que desenvolvedores de jogos independentes possam mapear as etapas de BDD durante o desenvolvimento de jogos e identificar quais aspectos de teste podem ser trabalhados com o uso da prática. Por fim, apresenta-se ainda um guia de uso da Linguagem Ubíqua, criado a partir das observações e dados coletados deste capítulo.

5 *Behavior-Driven Development for Indies*

Este capítulo apresenta o *Behavior-Driven Development for Indies* (BDD-I), um conjunto de especificações do *Behavior-Driven Development* para uso por desenvolvedores na produção de jogos independentes. Inicialmente apresenta-se seu conceito e visão geral. Em seguida, explana-se sobre as especificações que permitem ao desenvolvedor compreender como as etapas do BDD são mapeadas nas atividades do ciclo de desenvolvimento do jogo. Posteriormente, apresenta-se quais aspectos de teste de jogos podem ser explorados durante o uso de BDD-I na produção de jogos independentes. Por fim, é apresentado um guia para os desenvolvedores especificarem comportamentos utilizando a Linguagem Ubíqua, com a finalidade de extrair benefícios em relação à especificação de testes.

5.1 Apresentação

Nos capítulos anteriores, foi possível compreender como os desenvolvedores de jogos independentes realizam as práticas e/ou técnicas de testes durante o ciclo de produção dos jogos. Dentre os achados, observou-se a escassez de uso de práticas de testes de em nível de código durante a produção dos jogos independentes e identificou-se os aspectos de testes que estes desenvolvedores buscam durante a realização dos testes na produção dos jogos. Paralelamente, foi realizado uma estudo de campo com o objetivo de identificar características que favoreçam ou prejudiquem a utilização de BDD durante o ciclo de produção de jogos. Foram obtidos resultados nas áreas de planejamento, comunicação e testes para jogos, que permitem propor especificações de uso da prática no desenvolvimento de jogos.

Com estes resultados, foi construído um conjunto de especificações que permitem adaptar seu uso durante o ciclo de desenvolvimento de jogos independentes para garantir que os desenvolvedores de jogos independentes possam obter benefícios no uso da prática do BDD. Denominado *Behavior-Driven Development for Indies* (BDD-I), este conjunto de especificações observam características do ciclo de desenvolvimento

de jogos independentes, como atividades realizadas e aspectos de teste explorados pelos desenvolvedores. Para isso, focou-se no trabalho de duas especificações: (i) mapear os ciclos de desenvolvimento; e (ii) apresentar aspectos de teste de jogos os quais podem ser explorados pelo BDD.

O mapeamento dos ciclos de desenvolvimento permite que se possa observar como as etapas do ciclo do BDD são executadas em adequação às diferentes fases do ciclo de desenvolvimento de jogos. O desenvolvedor pode realizar as atividades do BDD de acordo com as atividade referentes à fase em que ele se encontra no desenvolvimento de jogos. Dessa forma, benefícios do BDD, como documentos de desenvolvimento e relatórios de progresso, são fornecidos de maneira eficiente através das diversas iterações de construção de funcionalidades.

Observar características e aspectos de teste de jogos no uso da Linguagem Ubíqua permite explorar melhor a construção de cenários que expressam os comportamentos de um jogo. Desta forma, os testes que se originam a partir dos comportamentos descritos focam nos aspectos de teste pelos quais os desenvolvedores buscam explorar. Cada funcionalidade e seus cenários estão diretamente ligados a funcionalidades do jogo, permitindo ainda que artefatos como o GDD e o Plano de Testes [3, 15] possam ser fornecidos juntamente com a Linguagem Ubíqua.

A Figura 5.1 ilustra a visão geral do BDD-I. Testadores, *game designers*, desenvolvedores e demais interessados podem, com uso da Linguagem Ubíqua, definir funcionalidades e especificar diversos cenários que descrevam os seus comportamentos. As especificações podem compor seções de documentos como Planos de Testes e GDDs. Ferramentas BDD podem ainda utilizar essas especificações para fornecer esqueletos de código de testes de aceitação. Dessa forma, desenvolvedores podem construir testes que serão executados durante o desenvolvimento do jogo, observando diversos aspectos de teste.

Figura 5.1: Visão Geral do BDD-I, demonstrando suas principais atividades e atores envolvidos. Fonte: Autor.

5.2 Mapeamento dos Ciclos de Desenvolvimento

Compreender as diferentes atividades do BDD é um fator importante para que o desenvolvedor de jogos independentes possa realizar corretamente o uso da prática. Para isso, buscou-se mapear estas atividades dentro do ciclo de desenvolvimento de jogos. A partir da análise das características de ambos os ciclos, foi possível relacionar atividades similares e realizar o mapeamento. Estas características foram obtidas através de revisão de literatura (ver Capítulo 2) e da observações durante o estudo de campo realizado com profissionais (descrito no Capítulo 4).

A construção deste mapeamento tomou como base as etapas do ciclo de BDD apresentadas por Smart [2], apresentadas na Seção 2.1, e as fases do desenvolvimento de jogos proposto por Chandler [3], apresentadas na Seção 2.3. Em ambas as definições, é possível categorizá-las observando a fase correspondente no Ciclo de Vida de Desenvolvimento de Software (CVDS) (ver Figura 4.2). Para construir um modelo que compreendesse as especificidades de cada definição, foram realizadas adaptações, como por exemplo unir as fases Implementação e Testes do CVDS, para permitir que etapas do BDD e fases do desenvolvimento de jogos fossem devidamente comparadas. Algumas fases do CVDS também foram suprimidas, uma vez que

dentro das definições de Smart e Chandler, não se pode identificar as atividades que pertenciam a estas fases. O Quadro 5.1 apresenta um resumo de atividades realizadas durante a prática do BDD e do desenvolvimento de jogos em relação à etapa correspondente do CVDS. A partir destas atividades, buscou-se mapear cada etapa da prática do BDD com as fases do ciclo de desenvolvimento de jogos, observando as afinidades que haviam entre elas. A Figura 5.2 apresenta ambas as definições apresentadas por Smart e Chandler, observando a classificação do CVDS.

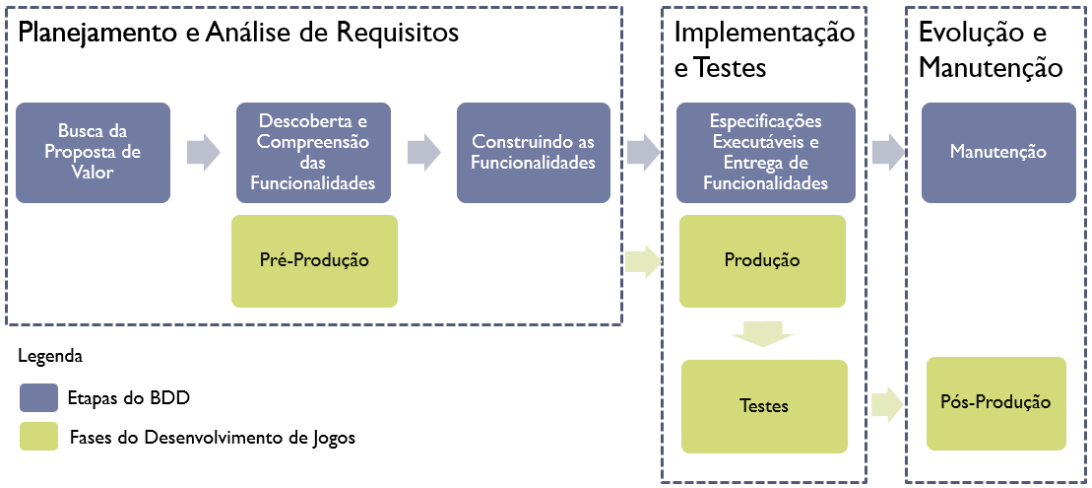

Figura 5.2: Etapas do BDD e Fases do Desenvolvimento de Jogos, do ponto de vista da engenharia de software. Fonte: Autor.

Fase de Planejamento e Análise de Requisitos

A fase "Planejamento e Análise de Requisitos" do CVDS compreende as etapas do BDD "Busca da Proposta de Valor", "Descoberta e Compreensão das Funcionalidades" e "Construindo as Funcionalidades". Nestas etapas, inicia-se pela busca da proposta de valor da aplicação e funcionalidades a serem construídas. É necessário fornecer uma visão clara daquilo que se deve construir e qual o valor das funcionalidades do software para os interessados [2, 83].

Para se descobrir e compreender as funcionalidades e seu valor, o praticantes de BDD as descrevem com os interessados utilizando a linguagem ubíqua simplificada para descrição de funcionalidades. O formato da Linguagem Ubíqua para descrever funcionalidades permite a visualização do valor de negócio, quem está interessado na funcionalidade e qual funcionalidade obtém o resultado esperado pelo interessado. Com a descrição do conjunto de funcionalidades a serem desenvolvidas,

Fases do CVDS	**Atividades**	
	Behavior-Driven Development	**Desenvolvimento de Jogos**
Planejamento e Análise de Requisitos	O que se quer construir? Qual o valor das funcionalidades do *software* para os interessados?	Uso de *brainstorming*. Como o jogo deve funcionar? Que mecânicas o público-alvo prefere?
	Descrição de funcionalidades pelos desenvolvedores e interessados utilizando a Linguagem Ubíqua.	Elaboração do Documento de *Game Design* pela equipe de desenvolvimento do jogo, com requisitos funcionais e não-funcionais.
	Especificação de Cenários com a Linguagem Ubíqua para entender como o *software* deve funcionar.	Protótipos em papel e caneta, colagens, uso de softwares e/ou artes conceituais permitem entender como será o jogo.
	Conjunto de cenários com exemplos concretos em Linguagem Ubíqua para criação de cenários de testes.	Um conjunto de testes é especificado no Documento de Plano de Testes para verifica funcionalidades em fases posteriores.
Implementação e Testes	Ferramenta BDD converte cenários descritos em Linguagem Ubíqua para esqueletos de código de testes.	As mecânicas do jogo são implementadas de forma *ad-hoc*, modularizada ou por outros critérios definidos pela equipe de desenvolvimento.
	O desenvolvimento das funcionalidades é guiado pelos testes automatizados.	Testes são realizados pela equipe de testes e/ou desenvolvimento a partir de protótipos e versões executáveis em fase alpha com objetivo de reduzir o número de erros.
	Ferramentas BDD monitoram quais funcionalidades foram desenvolvidas a partir dos testes automatizados.	Testes são realizados por testadores e parte do público-alvo a partir de versões executáveis em fase beta com objetivo de relatar *bugs*, baixo desempenho e jogabilidade.
		Builds são utilizadas de forma a validar e refinar as funcionalidades idealizadas na fase de pré-produção do jogo.
		Ao construir as funcionalidades, é utilizado *assets* substitutos (*placeholders*) até que versões maduras destes sejam disponibilizados.
Evolução e Manutenção	Alterações ou novas funcionalidades são refletidos nas funcionalidades e cenários descritos em Linguagem Ubíqua e nos testes automatizados.	Parte da equipe mantém o código para finalidades como correções de *bugs* que não puderam ser corrigidos antes do lançamento ou que foram encontrados na versão *Gold*.
		É realizado o arquivamento do código-fonte, *assets* e documentações com a finalidade de criar o *kit* de fechamento, disponível para consulta futura, em caso de criação de *patches*, atualizações, *ports*, remasterizações, traduções ou sequências do jogo.
		A equipe prepara um documento de *post-mortem* para discutir os pontos fortes e fracos do projeto, com objetivo de melhorar futuros projetos.

Tabela 5.1: Quadro resumido de atividades do BDD e do desenvolvimento de jogos observando o CVDS. Fonte: Autor.

parte-se para construção dos cenários. Especificar uma boa quantidade de exemplos contribui com uma maior qualidade do software a ser desenvolvido [27], uma vez que serão utilizados na criação de testes de aceitação.

No ciclo de desenvolvimento de jogos, a fase "Planejamento e Análise de Requisitos" do CVDS compreende a fase de pré-produção. Ao utilizar atividades do BDD no ciclo de desenvolvimento de jogos, a documentação necessária é também favorecida com o uso da Linguagem Ubíqua. Uma vez que os desenvolvedores irão cooperar definindo os comportamentos das funcionalidades, estes por sua vez também definirão as mecânicas, exemplificando-as melhor através do detalhamento de cenários de exemplos concretos. Com isso, seções que abordam mecânicas e regras em GDDs podem ser totalmente ou parcialmente atendidos. O formato da Linguagem Ubíqua para cenários obedece a estrutura básica de um caso de testes definida pela ISO/IEC/IEEE 29119-3[1], isto é, apresenta contexto inicial, passos para execução do teste e resultados esperados. Com isso, alguns comportamentos definidos com uso de BDD podem ser repassados para a equipe de testes, para que sejam também verificados como casos de testes.

Fase de Implementação e Testes

A fase "Implementação e Testes" do CVDS compreende a etapa do BDD "Especificações Executáveis e Entrega de Funcionalidades". A partir dos cenários e critérios de aceitação definidos, os praticantes BDD utilizam um conjunto de ferramentas para conversão da linguagem ubíqua em testes automatizados [87]. Os testes automatizados guiam o desenvolvedor a implementar as funcionalidades. Além disso, eles funcionam como documentação técnica, permitindo que se verifique quais funcionalidades já foram implementadas e quais estão pendentes.

No ciclo de desenvolvimento de jogos, a fase "Implementação e Testes" do CVDS compreende as fases de produção e testes. Durante estas fases, as mecânicas são transformadas em código executável. Ferramentas BDD dão suporte à criação de classes de teste a partir das funcionalidade e comportamentos descritos em Linguagem Ubíqua. A partir da transformação de comportamentos em códigos de teste, os casos de testes guiam a implementação do código-fonte, que nasce então testado. Isso

[1]https://www.softwaretestingstandard.org/

também permite um melhor projeto na arquitetura, uma vez que durante o ciclo do BDD, o código-fonte é refatorado constantemente, possibilitando que características como coesão e acoplamento possam ser atendidas.

Durante a produção do jogo, ferramentas BDD podem gerar relatórios de progresso, contabilizando quais mecânicas e regras já foram implementadas. Isso é possível uma vez que estas ferramentas podem verificar quais cenários já foram implementados e retornam sucesso como resposta à execução dos testes automatizados. Com isso, os desenvolvedores podem monitorar o projeto e identificar quanto falta para conclusão do mesmo.

Os casos de testes descritos em BDD também servem para guiar os testadores durante as fases *Alpha* e *Beta*. Nestas fases, o jogo é submetido a um exaustivo número de testes. Testes descritos durante a construção dos comportamentos utilizando BDD podem ser descartados nestas fases, uma vez que já foram cobertos durante a produção. Assim, o desenvolvedor pode se preocupar então com casos de testes que não foram pensando ou não puderam ser cobertos nas etapas anteriores com BDD. Casos de testes mais especializados podem também ser gerados a partir de combinações e/ou modificações de casos de testes descritos anteriormente, cobrindo com mais precisão o jogo e suas funcionalidades.

Fase de Evolução e Manutenção

A fase de "Evolução e Manutenção" do CVDS compreende a etapa do BDD "Manutenção". Uma vez que houver quaisquer alterações nas funcionalidades e cenários descritos em BDD, adaptações ou novos testes de aceitação deverão refletir essas modificações. Ao executar novamente os testes, identifica-se com mais facilidade as tarefas necessárias para a manutenção e modificação do software.

No ciclo de desenvolvimento de jogos, a fase "Evolução e Manutenção" do CVDS compreende a fase pós-produção. Modificações em mecânicas e regras do jogo são também relatadas em GDD e Planos de Testes, fazendo com que estes permaneçam atualizados. Da mesma forma, essas modificações deverão ser especificadas nos comportamentos e testes anteriormente criados com o uso do BDD. Isso permitirá que ferramentas BDD informem qualquer inconsistência ou erro decorridos dessas modificações nas demais funcionalidades existentes do jogo, fazendo com que os

desenvolvedores as realizem com confiança e segurança. Novas funcionalidades no jogo deverão também ser descritas utilizando BDD, permitindo manter a consistência entre todas as funcionalidades já existentes.

Erros de lógica e implementação identificados após o lançamento do jogo necessitam também de atualizações de casos de testes e especificações do BDD. Isso pode ocorrer uma vez que cenários específicos podem não terem inicialmente sido contemplados. Desta forma, os desenvolvedores devem adicionar então novos cenários observando o comportamento que forneça casos de testes para resolução do erro a ser tratado.

Por fim, toda a documentação produzida, tais como descrições em Linguagem Ubíqua, classes de testes e relatórios de progresso gerados pela ferramenta BDD utilizada, deve ser arquivada junto aos *assets*, código-fonte e demais documentações específicas do desenvolvimento de jogos. Desta forma, consegue-se permitir que remasterizações, traduções ou sequencias, por exemplo, possam ser viabilizadas.

5.2.1 Prova de Conceito com Profissionais

Para fundamentar as proposições iniciais acerca do mapeamento de ciclos entre as etapas do BDD e as atividades do ciclo de desenvolvimento de jogos para o BDD-I, idealizou-se uma prova de conceito. Para Wohlin et al [48, p. 12], pesquisas de opinião podem ser utilizadas antes de se apresentar uma nova técnica e coletar opiniões, por exemplo. Desta forma, organizou-se uma pesquisa de opinião, de caráter exploratório, com profissionais desenvolvedores de jogos independentes de forma a verificar o grau de concordância com as proposições apresentadas no mapeamento de ciclos.

Planejamento

Para aplicação da pesquisa, foram mapeados profissionais da cidade de São Luís - MA, que já haviam participado da produção de ao menos um jogo de forma independente. Para a aplicação da pesquisa, os profissionais convidados deveriam receber um material de suporte que abordasse o ciclo do BDD e suas características.

Como instrumento, utilizou um questionário com cinco seções, onde cada seção apresentava uma determinada etapa do BDD com atividades do ciclo de desenvolvimento de jogos correspondentes à fase do CVDS. Para cada atividade apresentada, o participante deveria responder o grau de concordância, em escala de cinco pontos do tipo Likert, demonstrando a equivalência com as atividades do BDD realizadas durante a etapa em questão. Quando o participante não concordasse com uma atividade, ele seria convidado a responder porque não concordou, em uma questão do tipo aberta. Por fim, em cada seção, convidava-se o participante a contribuir com outras atividades que poderiam ser relacionadas e que não constavam no questionário, em uma questão do tipo aberta. O questionário é apresentado na Tabela F.2 do Apêndice F. Os dados obtidos pelo questionário deveriam ser analisados de forma quantitativa e de forma qualitativa, com uso de GT tal qual definido na Seção 3.1.4.

Execução

Para a coleta de dados, utilizou-se a ferramenta Google Forms para produção e distribuição do questionário de forma *online*. O questionário foi redigido em idioma português e seu *link* foi compartilhado por e-mail para 10 profissionais que se encaixavam no público desta pesquisa. Além do questionário, os profissionais receberam um material, com duas páginas, onde abordou-se BDD, suas etapas e características, criado com a ferramenta Google Docs. Este material deu suporte aos profissionais que poderiam consultá-lo a qualquer momento durante o preenchimento do questionário.

Durante a análise de dados, os dados obtidos foram submetidos a análise quantitativa e qualitativa. Como ferramenta de apoio, utilizou-se o software de planilhas Microsoft Excel. Na análise quantitativa, os dados foram contabilizados, analisados e tratados em forma de gráficos em barras, para posterior interpretação dos resultados.

Resultados

Foram obtidas respostas de 10 profissionais experientes no desenvolvimento de jogos independentes. A Figura 5.3 apresenta a distribuição

destes participantes de acordo com o tempo de experiência no desenvolvimento de jogos, onde visualiza-se que mais de 50% dos participantes possui no mínimo três anos de experiência.

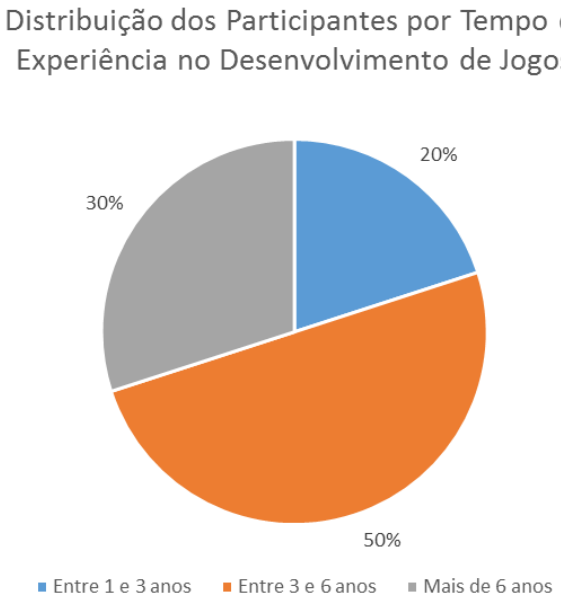

Figura 5.3: Distribuição de participantes por tempo de experiência. Fonte: Autor.

A relação de afinidade de atividades com a etapa "Busca da Proposta de Valor", indicada pelos participantes, é apresentada na Figura 5.4. Os participantes concordaram com o uso de *brainstorming* para definir como o jogo funcionará e que mecânicas deverá possuir de acordo com a preferência do público-alvo. A criação e elaboração do GDD a partir desta etapa foram também tidas como em acordo pelos participantes, embora um participante tenha discordado afirmando que "este é um documento muito pesado para se escrever quando ainda estamos tentando descobrir o que vamos produzir e para quem". Sobre atividades de prototipação, um dos participantes acredita que deva ser realizada em etapas posteriores, pois utiliza-se para "uma validação prévia da Funcionalidade". Por fim, não houve unanimidade sobre construção de casos de testes nesta etapa, sendo indicado por um participante que o mesmo "não visualiza como planejar os testes nem criação de documento para tal finalidade, visto que ainda está no campo das ideias".

A relação de afinidade de atividades com a etapa "Descoberta e Compreensão das Funcionalidades", indicada pelos participantes, é apresentada na Figura 5.5. Os participantes concordaram que é dado prosseguimento no uso de *brainstorming* nesta etapa, auxiliando na construção das funcionalidades. Além disso, a

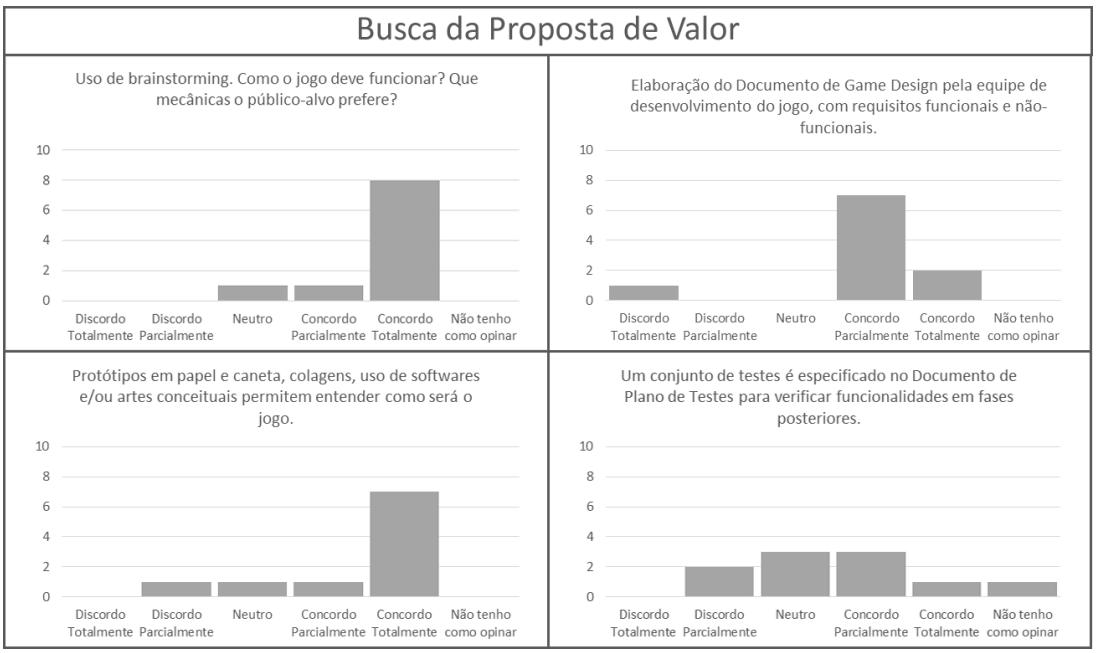

Figura 5.4: Respostas sobre concordância das atividades da fase de pré-produção de jogos em relação à etapa do BDD "Busca da Proposta de Valor". Fonte: Autor.

elaboração do GDD e o uso de prototipação auxiliam na construção do projeto do jogo, as quais os participantes também concordaram. Porém, os participantes novamente não concordaram entre si sobre a construção de casos de testes nesta etapa. Como demais atividades sugeridas que poderiam ter afinidades com os objetivos desta etapa, os participantes sugeriram "definição de público-alvo" e "elaboração dos requisitos junto ao cliente/interessado do projeto".

A relação de afinidade de atividades com a etapa "Construindo as Funcionalidades", indicada pelos participantes, é apresentada na Figura 5.6. Esta é a última etapa a fazer parte da fase de "Planejamento e Análise de Requisitos" do CVDS. Para esta etapa, os participantes concordam que já há bem menos uso de *brainstorming* e atividades relacionadas à especificação de requisitos em GDD. Para um dos participantes, "uma vez que já se constrói as funcionalidades, acredito que etapas fundamentais de concepção e documentação já devem ter sido realizadas de forma bem consolidada". É uma etapa caracterizada pelos participantes com amplo uso da prototipação, sendo acordado por todos os participantes. Apesar de um participante discordar da criação de casos de testes nesta etapa, boa parte dos participantes concordam que este tipo de atividade tem afinidade com esta etapa.

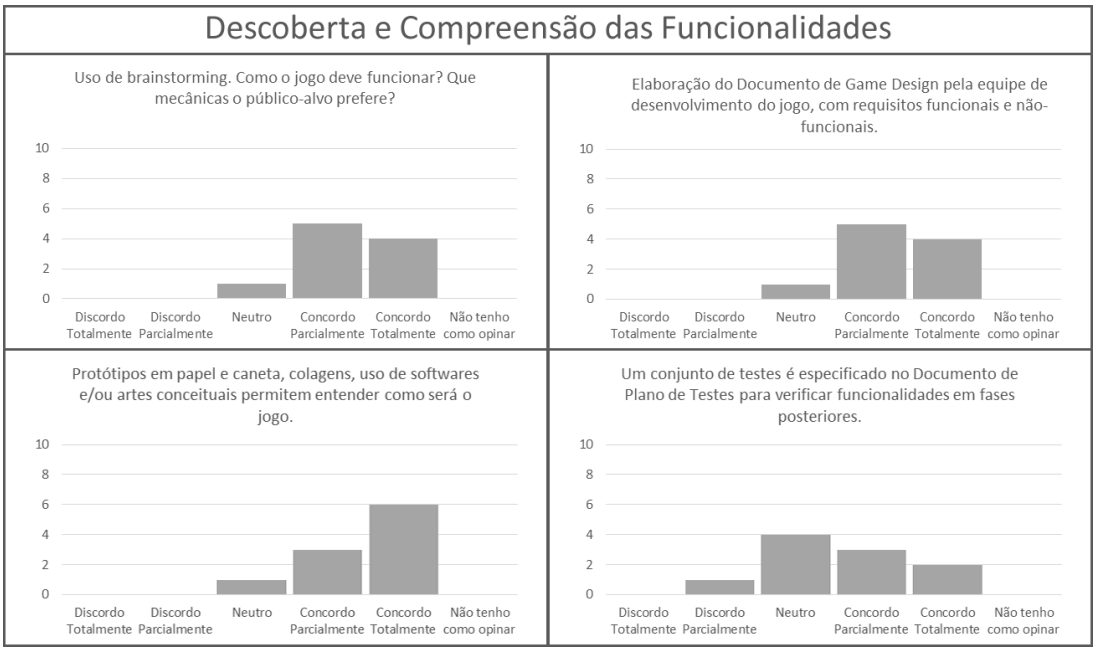

Figura 5.5: Respostas sobre concordância das atividades da fase de pré-produção de jogos em relação à etapa do BDD "Descoberta e Compreensão das Funcionalidades". Fonte: Autor.

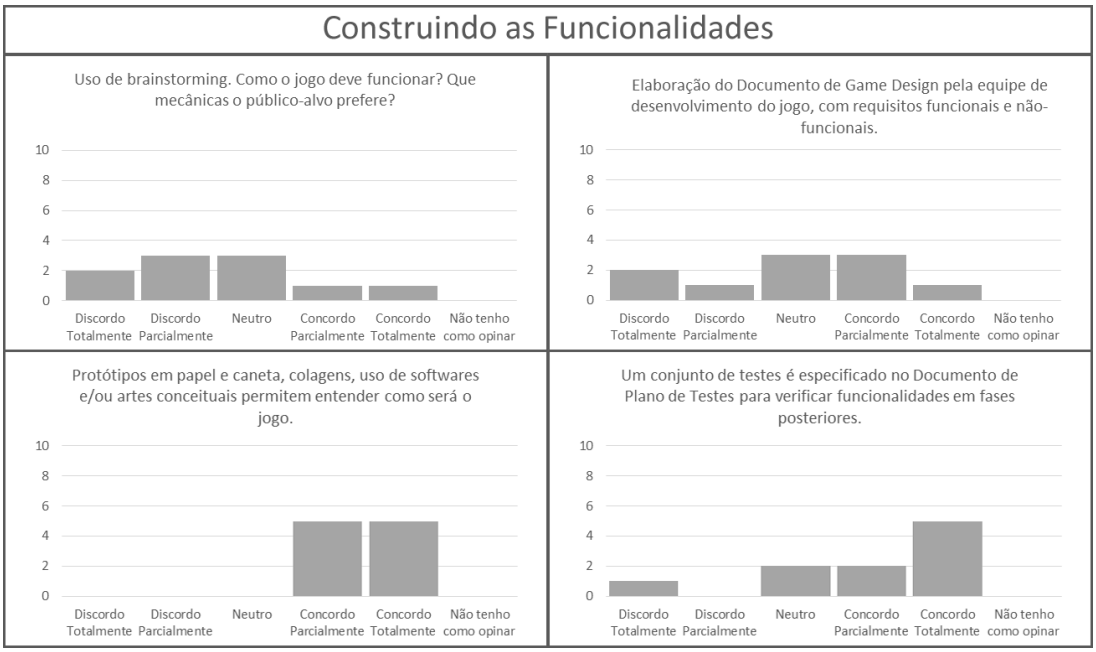

Figura 5.6: Respostas sobre concordância das atividades da fase de pré-produção de jogos em relação à etapa do BDD "Construindo as Funcionalidades". Fonte: Autor.

A relação de afinidade de atividades com a etapa "Especificações Executáveis e Entrega de Funcionalidades", indicada pelos participantes, é apresentada na Figura 5.7. Esta etapa está inserida na fase de "Implementação e Testes" do CVDS, a qual pode apresentar um alto número de atividades inseridas nesta etapa. Em relação às atividades enumeradas para este estudo, os participantes não apresentaram discordância com esta etapa, o que sugere que elas podem ser utilizadas no BDD-I tal qual especificado no mapeamento de ciclos de desenvolvimento. Dentre outras características desta etapa que foi mencionada pelos participantes, está a "disponibilização aos jogadores de versões em alpha com uma amostra do que seria o jogo desenvolvido no estágio final" onde, segundo o participante, seria uma forma de atrair o público para uma melhor construção do jogo junto aos interessados.

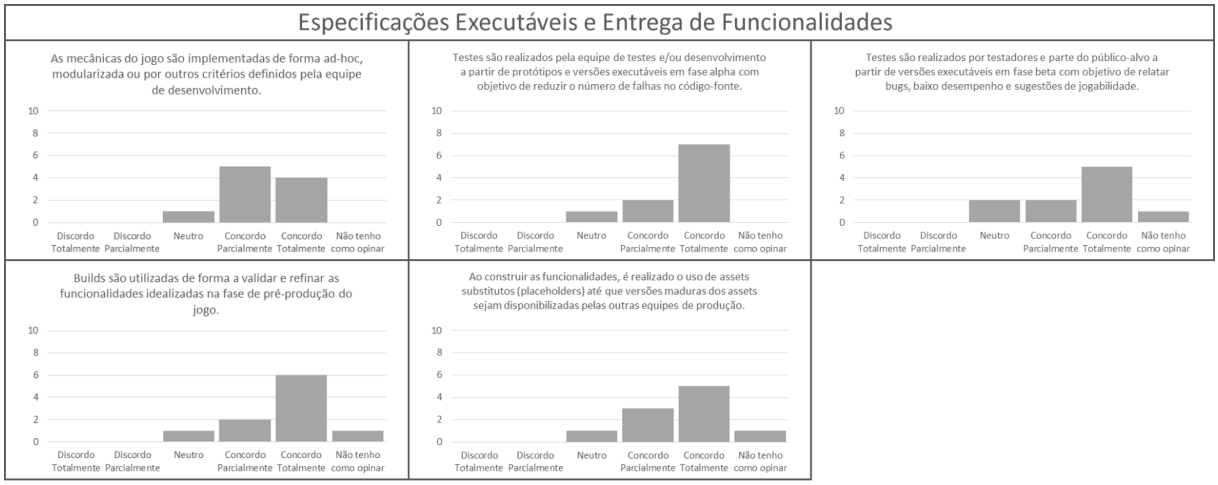

Figura 5.7: Respostas sobre concordância das atividades das fases de produção e testes de jogos em relação à etapa do BDD "Especificações Executáveis e Entrega de Funcionalidades". Fonte: Autor.

Por fim, a relação de afinidade de atividades com a etapa "Manutenção", indicada pelos participantes, é apresentada na Figura 5.8. Esta etapa compreende a fase de pós-produção do ciclo de desenvolvimento de jogos, e à etapa de "Evolução e Manutenção" do CVDS. Nesta etapa, os participantes novamente concordaram com as atividades apresentadas no mapeamento do BDD-I, com um alto grau de concordância entre eles. Os participantes não fizeram contribuições de atividades e características que pudessem ser adicionadas ao mapeamento.

Estes resultados fornecem indícios para reforçar as observações e proposições realizadas no Mapeamento de Ciclos de Desenvolvimento do BDD-I. Os participantes desta pesquisa de opinião puderam contribuir elencando novas

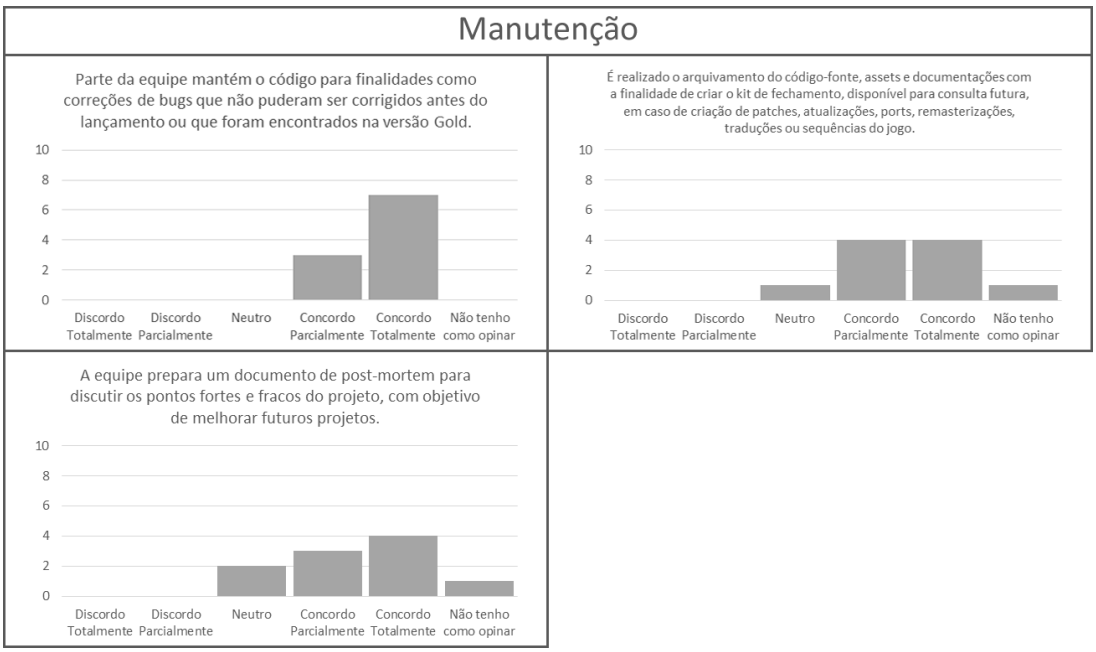

Figura 5.8: Respostas sobre concordância das atividades da fase de pós-produção de jogos em relação à etapa do BDD "Manutenção". Fonte: Autor.

características não idealizadas anteriormente e que deram base para construção do mapeamento apresentado anteriormente. Além disso, os indícios apresentados pelos participantes permitem que se compreenda como as atividades do ciclo do desenvolvimento de jogos tem afinidade com as etapas do BDD. Embora algumas dessas atividades não sejam beneficiadas diretamente pelo BDD-I, como por exemplo o uso de *placeholders* (recursos temporários) durante a produção do jogo, elas permitem que os desenvolvedores que já praticam BDD possam compreender melhor como adaptar a técnica ao ciclo de desenvolvimento de jogos.

5.3 Aspectos de Testes do Ciclo do BDD-I

No Capítulo 3, foram identificados 16 aspectos que os desenvolvedores de jogos independentes buscam ao realizar testes na produção dos jogos, apresentados na Tabela 3.9. Determinar estes aspectos é importante, uma vez que técnicas/processos de testes podem investigar vários destes aspectos durante sua realização [16].

Atribui-se, ao BDD, o tratamento de erros e defeitos como um de seus principais benefícios (apresentados no Capítulo 2, Seção 2.1) durante a produção do software. Para Kasurinen e Smolander [14], os desenvolvedores de jogos possuem

prioridades diferentes em testes em relação aos desenvolvedores de software. Desta forma identificou-se quais os aspectos de testes, dentre aqueles apontados pelos desenvolvedores independentes, que a prática do BDD-I pode auxiliar durante sua utilização no desenvolvimento de jogos.

Para a análise dos aspectos de testes trabalhados pelo BDD-I, buscou-se indícios na literatura (ver Capítulo 2) e nos dados obtidos durante o estudo de campo com uso de BDD (ver Capítulo 4). Assim, identificou-se que o BDD-I pode trabalhar com cinco dos 16 aspectos de teste (Jogabilidade, Balanceamento, Projeto de Níveis, Erros e Defeitos, Desempenho), as quais integram as categorias "Concepção e Planejamento" e "Implementação".

Aspectos de Testes de "Concepção e Planejamento"

A categoria "Concepção e Planejamento" reúne os diversos aspectos ligados às mecânicas, elementos e características de jogos. Foram identificados três aspectos nesta categoria que podem ser trabalhados com o uso do BDD-I: Jogabilidade, Balanceamento e Projeto de Níveis (*Level Design*).

Os aspectos de jogabilidade podem ser trabalhados com uso do BDD-I através da especificação, em Linguagem Ubíqua, das funcionalidades e cenários dos comportamentos do jogo, conforme observado no Código 5.1. Estes comportamentos podem derivar de regras, mecânicas, enredo, missões, níveis, recompensas, itens, desafios, dentre outros, do jogo. Ao descrever estes comportamentos, os desenvolvedores irão indicar como estes reagem a determinadas ações e condições no jogo, permitindo que se possuir construir um conjunto de casos de teste necessários para uma melhor implementação destas. Ao definir exemplos, os desenvolvedores poderão ter uma melhor visão do que se deseja obter como resultado.

Código 5.1: Exemplo de cenário para descrição de uma mecânica de jogo. Fonte: Unidade de Análise A, Capítulo 4.

```
1 Cenario: Mecânica de arrastar e soltar
2 Dado que a carta esta entre o ponto inicial e o ponto final
3 Quando o jogador arrasta a carta para o lado e solta
4 Entao a carta volta para o ponto inicial
```

> *(...) Então levantar um pouco da quais eram as dinâmicas do jogo, quais eram as ações que eram possíveis e o que aquilo ali ia impactar. Então, se eu ia fazer uma certa ação, o quê era esperado que ela fosse executar. (...) - Participante 1 - Estudo de Campo Capítulo 4*

Os aspectos de balanceamento do jogo buscam oferecer ao público-alvo uma boa curva de aprendizado, com objetivo de equilibrar desafio e diversão. É um aspecto de caráter subjetivo, pois depende de habilidades motoras e cognitivas do jogador. Porém, é possível especificar comportamentos que forneçam elementos para equilibrar diversão e desafio, definindo características base para que os desenvolvedores possam realizar ajustes com foco no balanceamento. Com a especificação de exemplos concretos com BDD-I, essas características definem parâmetros para ser trabalhados durante testes com o público-alvo. Estes parâmetros são então trabalhados de forma a equilibrar diversão e desafio. O Código 5.2 apresenta um exemplo de um cenário que prevê parâmetros (quantidade de pinos e pontuação extra) de uma mecânica com objetivo de desafiar ao mesmo tempo que recompensa o jogador, o que pode diretamente garantir a diversão do mesmo. Ao derrubar todos os pinos do jogo, os jogadores são recompensados com uma pontuação extra.

Código 5.2: Exemplo de cenário para balancear a mecânica, promovida por uma regra para garantir diversão. Fonte: Unidade de Análise B, Capítulo 4.

```
1 Cenario: Pontuação deve receber um bônus em caso de strike
2 Dado que o jogador arremessou a bola
3 E o modo tentativa foi finalizado
4 E o número de pinos da tentativa era igual a 10 pinos
5 Quando o jogador derruba todos os 10 pinos
6 Entao sua pontuação final deve ser de 20 pontos
```

Os aspectos relacionados ao projeto de níveis (*Level Design*) investigam o projeto de configuração de fases e mapas do jogo. O *Level Design* é uma importante atividade de *Game Design* que está preocupado em determinar como elementos (obstáculos, caminhos, posição de inimigos, desafios, itens) estão organizados dentro de um mapa ou fase para auxiliar o aprendizado do jogador e promover o desafio progressivo. O uso de BDD-I pode ajudar os desenvolvedores ao permitir que estes possam especificar cenários que verifiquem a organização destes elementos no jogo, em um determinado contexto. Estes elementos podem variar durante o jogo, em

resposta a interações do jogador, por exemplo. Desta forma, estes comportamentos pode ser mapeados em casos de teste para guiar o desenvolvedor a construir as fases e mapas tal qual especificado. O Código 5.3 apresenta um cenário que demonstra como deve ser a configurações de pinos ao iniciar uma partida de um jogo de boliche.

Código 5.3: Exemplo de cenário para testar posicionamento de obstáculos do *Level Design*. Fonte: Unidade de Análise B, Capítulo 4.

```
1 Cenario: Configuração de Pinos
2 Dado que o jogador iniciou uma partida
3 E é a primeira vez que o modo tentativa será iniciado na partida atual
4 Quando o modo tentativa for iniciado
5 Entao os 10 pinos precisam estar posicionados de forma triangular
6 E devem estar em pé
```

Aspectos de Testes de "Implementação"

A categoria "Implementação" reúne os aspectos ligados a características de software do jogo, relativos aos recursos digitais produzidos para compor o software: código-fonte, imagens e músicas em formato digital, dentre outros. Foram identificados dois aspectos nesta categoria que podem ser trabalhados com o uso do BDD-I: (i) Erros e Defeitos; e (ii) Desempenho.

Os aspectos relacionados a erros e defeitos estão ligados ao projeto de codificação do jogo, que pode afetar mecânicas, sistema de colisão, física, dentre outros. A criação de casos de testes a partir dos comportamentos especificados ajudar a restringir que a execução do código-fonte do jogo em determinados contextos possa provocar resultados inesperados. É importante ressaltar que quanto mais casos de testes eficientes possam ser cobertos, maiores as chances de que funcionalidades possam funcionar de maneira inesperada. LLopis e Houghton [21] apresentam os benefícios do uso da criação de casos de testes antes de desenvolver o código-fonte do jogo. Eles relatam diminuir as chances de comportamentos inesperados ao tentar prever erros que possam acontecer antes mesmo de implementar as funcionalidades. No Capítulo 4, isso também foi constatado pelos participantes, apresentados nas citações abaixo.

> *(...) Então nessa parte eu achei que ajudou bastante por exemplo... tem uma ação X no sistema que vai desencadear três, quatro reações diferentes. Então quando tu faz um sistema desses, um pensamento desses antes de implementar, tu já tem algumas coisas que tu tem em mente que pode dar errado. Então tu já consegue meio que prever aquilo, com certeza pode dar outros erros mas pelo menos a maior parte deles tu já consegue prever, porque tu já pensou antes então tu já até programa já pensando naquilo pra evitar acontecer. - Participante 1 - Estudo de Campo, Capítulo 4*
>
> *Sim, pois facilita a compreensão de como o sistema funciona e quais são as respostas esperadas - Participante 12 - Estudo de Campo, Capítulo 4*

Os Códigos 5.4 e 5.5 apresentam exemplos de cenários que descrevem comportamentos com o objetivo de prevenir erros, uma vez que descrevem com mais riqueza de detalhes o resultado esperado a partir de certas condições.

Código 5.4: Exemplo de cenario para definir comportamento de tela. Fonte: Unidade de Análise A, Capítulo 4.

```
1 Cenario: Iniciar o jogo
2 Dado que a tela do jogo carrega
3 Quando inicia a aplicação
4 Entao deve mostrar 3 áreas no topo da tela
5 E mostra texto de pontuação com zero
6 E mostra texto com descrição da questão, acima de carta de ação
7 E mostra carta de ação com imagem do personagem
8 E mostra a pergunta que deve ser respondida, abaixo da carta de ação
```

Código 5.5: Exemplo de cenário para prever comportamento de mecânica de jogo. Fonte: Unidade de Análise B, Capítulo 4.

```
1 Cenario: Modificando a Posição da Bola para a Direita
2 Dado que o jogador encontra-se definindo a posição de arremesso da bola
3 Quando o jogador pressiona o botão Seta Direita
4 Entao a posição da bola deve se movimentar para o lado direito em relação à camera
5 E a nova posição deve ter diferença de 1 unidade em relação a posição anterior
```

Os aspectos relacionados a desempenho visam analisar como o jogo se comporta quanto ao uso de recursos de hardware (processamento, memória, dentre outros) na plataforma-alvo. Os desenvolvedores buscam otimizar o código-fonte para garantir a estabilidade no uso desses recursos, para que o jogador tenha a melhor experiência possível. Daylamani et al. [86] utilizou cenários BDD para testar

o framework de jogos multijogador para redes sociais Lu-lu. Eles coletaram dados de jogadores e utilizou estes dados para executar em cenários de testes especificados em BDD. Um destes cenários é apresentado no Código 5.6, para testar o desempenho de personalização do jogador. Os resultados dos testes em seu trabalho foram satisfatórios para a plataforma-alvo na qual realizaram os testes. Desta forma, os desenvolvedores de jogos independentes podem também especificar cenários do BDD-I que avaliem o desempenho do jogo observando métricas relativas ao uso de recursos de hardware que podem ser obtidas com funções no código-fonte do jogo. O Código 5.7 apresenta um exemplo de cenário para testar o desempenho de um determinado jogo quanto a quantidade de objetos na tela. No exemplo, o objetivo é criar um teste que verifica quantos quadros por segundo[2] a plataforma-alvo suporta ao executar uma determinada funcionalidade.

Código 5.6: Exemplo de cenário para teste de desempenho do framework Lu-Lu. Fonte: Traduzido de Daylamani et al. [86].

```
1 Cenario: Personalização melhora a jogabilidade
2 Dado que a personalização melhora a lealdade do jogador
3 E a personalização melhora a capacidade de decisão do jogador
4 Quando o jogador é profiled [criação do perfil]
5 E o jogador recebe incentivos
6 Entao a jogabilidade do jogador melhora
```

Código 5.7: Exemplo de cenário para testar desempenho de uma funcionalidade. Fonte: Autor.

```
1 Cenario: Criação de onda de inimigos na fase
2 Dado que a fase foi carregada corretamente
3 Quando o uma nova onda de inimigos for disparada
4 Entao surge <NUM_INIMIGOS> na fase
5 E a contagem de quadros por segundo renderizados é <NUM_FPS>
6
7 NUM_INIMIGOS | NUM_FPS
8 200          | 120
9 400          | 80
10 600         | 60
```

[2]Unidade de medida utilizada por placas gráficas para quantificar quantas vezes por segundo é processado e exibido as imagens no monitor de um computador.

5.4 Especificando Comportamentos com BDD-I

Um dos componentes-chave do BDD é sua Linguagem Ubíqua [4, 25, 27]. Seu uso permite que atividades como comunicação entre interessados, especificação de funcionalidades, criação de artefatos de desenvolvimento e automação de testes possam se correlacionar no BDD. E é a partir dela que expressamos os comportamentos das funcionalidades.

Ao definir a Linguagem Ubíqua, North [19] visou garantir a abrangência de seu uso em diversos domínios no desenvolvimento tradicional de software. Uma vez que o desenvolvimento de jogos possui características que os torna significativamente diferente do desenvolvimento tradicional de software [13], se faz necessário observar essas características na construção dos comportamentos das mecânicas e regras de jogos.

Para que os desenvolvedores possam especificar as funcionalidades e os comportamentos dos jogos, o BDD-I apresenta um guia de uso da Linguagem Ubíqua. Desta forma, o desenvolvedor de jogos independentes pode especificar melhor casos de testes a partir das especificações realizadas em Linguagem Ubíqua.

Funcionalidades

Durante o desenvolvimento de jogos independentes, entende-se como funcionalidades as diversas mecânicas e regras que o jogo possui. Por exemplo, o avatar do jogador pode executar as funcionalidades "Atacar" e "Abrir Baú". Durante a fase de pré-produção do jogo, a equipe de desenvolvimento descreve no GDD como estas funcionalidades devem executar [3]. Por sua vez, estas funcionalidades, em geral, se transformam em estórias de usuário para guiar as tarefas do time de programação [22].

No Capítulo 4, os desenvolvedores apontaram que uma dos principais benefícios que observaram ao utilizar BDD foi o gerenciamento de tarefas. As funcionalidades e seus cenários guiaram os desenvolvedores durante a execução do estudo de campo, permitindo que eles tivessem uma maior gestão de sua produção.

Nesse sentido, os desenvolvedores, ao utilizar BDD-I, devem especificar as funcionalidades utilizando o formato proposto pela Linguagem Ubíqua. Isso permitirá que eles possam cooperar ao documentar essas funcionalidades em seções do GDD onde antes descreveriam as funcionalidades do jogo em texto plano. Desta forma, o time de programação e o time de testes ganham tempo para trabalhar na criação dos casos de testes necessários para garantia de qualidade na produção do jogo. O Código 5.8 apresenta o modelo a ser utilizado na especificação de funcionalidades do jogo.

Código 5.8: Template para especificar Funcionalidades na Linguagem Ubíqua.

```
1 Funcionalidade: <título>
2 Para <atingir um objetivo>
3 Eu, como <interessado>
4 Quero <funcionalidade>
```

A primeira parte da estrutura de uma funcionalidade o objetivo da funcionalidade, buscando evidenciar porque está se faz necessária ao jogo. Estes objetivos podem ser o mais diversos, dependendo da funcionalidade a ser explorada. Por exemplo, uma determinada funcionalidade "Pular" de um personagem pode ter o objetivo de permitir que o jogador possa resolver desafios. Já uma funcionalidade "Exibir Ícones de Interação Social" possa ter como objetivo aumentar a diversão ou permitir que o jogador possa se comunicar com outros jogadores. É importante que os desenvolvedores possam identificar este objetivo de forma clara, porque a partir dele, ele pode elencar prioridades. Por exemplo, pode ser interessante para a equipe de produção que dê prioridade em desenvolver funcionalidades que tem por objetivo resolver desafios à outras que tem por objetivo permitir a interação social.

A segunda parte da estrutura de uma funcionalidade especifica quem esta interessado na funcionalidade ou qual papel ela exerce na utilização desta. Em desenvolvimentos de software tradicional, o software a ser desenvolvido pode permitir a interação de vários papeis e usuários dentro de um sistema [2]. No desenvolvimento de jogos, em geral, não há esta distinção de papeis uma vez que, visto como um produto de software, jogos tem por usuário final o jogador. No entanto, em jogos educacionais, funcionalidades podem ter diferentes interessados, como tutor e aprendiz. Em se tratando de jogos independentes, este recai no âmbito geral de termos como interessado o jogador. Outros papeis e interessados podem ser módulos

ou outros sistemas do próprio jogo, como um banco de dados que pode necessitar de uma funcionalidade específica para integração com o jogo.

A terceira parte da estrutura de uma funcionalidade especifica a funcionalidade que será descrita através dos cenários. O jogador deverá especificar então uma afirmação que generalize a funcionalidade e que faça sentido para que se possa criar os cenários que descreverão o comportamento da funcionalidade. Um exemplo de afirmação para se encaixar nesta parte seria "o personagem possa executar a ação de pular". Desta forma, o desenvolvedor pode identificar com maior facilidade o que é a funcionalidade e como ele poderia mapear isso no código-fonte do jogo. Por exemplo, para desenvolvedores que desenvolvem em paradigma de orientação a objetos, esta afirmação de funcionalidade forneceria pra ele uma método (pular) de uma classe a ser implementada (personagem). O Código 5.9 apresenta um exemplo de especificação da funcionalidade "Identificar Jogador", pela qual um jogador poderá se identificar utilizando um sistema externo de autenticação a ser integrado com o jogo.

Código 5.9: Exemplo de Funcionalidades na Linguagem Ubíqua observando as especificações do BDD-I.

```
1 Funcionalidade: Identificar Jogador
2 Para que o jogador possa registrar seu progresso
3 Eu, como o jogador
4 Quero poder me identificar ao entrar no jogo
```

Cenários

Para cada funcionalidade descreve-se, através dos cenários, como estas se comportam. Estes, por sua vez, são estruturados de forma a dar uma visão para o desenvolvedor em três partes: (i) contexto, onde apresenta-se as pré-condições para que se possa executar o comportamento; (ii) evento, indicando um ou mais gatilhos que induzem a forma como o comportamento irá se manifestar; e (iii) saída, onde apresenta-se a sequencia de ações e estados esperados do comportamento em relação ao evento. O Código 5.10 apresenta o modelo de especificação de cenários da Linguagem Ubíqua.

Código 5.10: Template para especificar Cenários na Linguagem Ubíqua.

```
1 Cenario: <título>
2 Dado <um contexto>
3 Quando <algo acontecer>
4 Entao <o que se espera como saída>
```

Este formato beneficia os desenvolvedores de jogos independentes uma vez que sua estrutura possui uma semelhança com os modelos de caso de testes apresentados pela ISO/IEC/IEEE 29119-3 e pelo modelos para casos de testes para jogos apresentados na literatura [3, 16]: o contexto, no BDD-I, é semelhante ao que se descreve nas pré-condições e dependências de testes; o evento, no BDD-I, é semelhante ao que se descreve nos passos de teste; e por fim a saída é semelhante ao que se descreve como resultados esperados de um caso de teste. Assim, ao especificar os comportamentos de uma funcionalidade, os desenvolvedores também estão especificando alguns dos casos de testes que podem compor seções do documento de Plano de Testes.

A primeira parte da estrutura de um cenário refere-se ao contexto. Em essência, o contexto deve expressar todas as pré-condições para que um determinado comportamento possa acontecer. Por exemplo, para uma determinada funcionalidade "Pular" de um personagem, um dos possíveis comportamentos seria especificar em que condições o personagem é capaz de executar o pulo. Assim, para que ele pule, é necessário que o personagem esteja em cima de uma plataforma ou objeto sólido. Esta pré-condição se refere então a capacidade de execução deste comportamento, tendo em vista que para que todo o comportamento execute ao apertar o comando referente à ação de pular, esta pré-condição estar satisfeita. Outro exemplo de contexto para um comportamento de uma funcionalidade de um inimigo pode ser que o inimigo esteja em um estado de alerta. Assim, ao se satisfazer estas condição, quando o avatar do jogador estiver no alcance do raio de visão do inimigo, o comportamento esperado pode ser executado.

A segunda parte da estrutura de um cenário refere-se ao evento. O evento pode ser uma ou mais ações que precisam ocorrer para que o comportamento possa executar da forma esperada. No desenvolvimento de um jogo com BDD-I, são exemplo de eventos o jogador apertar um determinado botão no controle ou o avatar do jogador

entrar no raio de visão de um inimigo. Em uma funcionalidade "Pular", um evento para definir comportamentos deste funcionalidade pode ser pressionar o botão 'A' do controle. Outros exemplos de eventos que podem ocorrer em jogos e serem utilizados ao especificar comportamentos são quando um quadro (*frame*) do jogo é renderizado, quando um relógio interno alcança determinado tempo ou quando um novo estado de jogo é executado.

A terceira parte da estrutura de um cenário refere-se à saída. A saída de um cenário pode ser classificado como o ponto mais importante da descrição do comportamento. Isso porque ela representa o estado interno dos objetos, módulos e sistemas do jogo após um determinado evento ocorrer. É com estas informações que podemos descrever qual a reação de uma funcionalidade a determinado estímulo ou ação. Novamente, no exemplo de comportamentos da funcionalidade "Pular" de um personagem, poderíamos ter, como resultados esperados após o evento de execução de um determinado botão do controle, o incremento da posição vertical do objeto do avatar, a troca de quadros relativos a animação de pulo do personagem e por fim a mudança do estado interno do objeto para "pulando". Para o comportamento de uma funcionalidade "Abrir Baú", poderíamos ter, como resultadores esperados para um evento de apertar o botão 1 do controle, a animação de abertura do baú e a adição de um determinado item ao inventário do jogador. O Código 5.11 apresenta os comportamentos descritos para uma funcionalidade "Pular".

5.5 Considerações Finais

Este capítulo apresentou o *Behavior-Driven Development for Indies* (BDD-I), um conjunto de especificações do BDD para uso por desenvolvedores de jogos independentes. As especificações buscam garantir que os benefícios do uso de BDD também possam ser alcançados durante a produção de jogos independentes. BDD-I surge como uma opção de processo/prática de testes para jogos independentes uma vez que suas atividades podem ser adaptadas para atender ao ciclo de produção de jogos onde recursos humanos são escassos.

Apresentou-se como as atividades do ciclo de desenvolvimento de jogos e as etapas do BDD estão relacionadas, através do mapeamento de ciclos de

Código 5.11: Exemplo de Cenários na Linguagem Ubíqua observando as especificações do BDD-I.

```
1 Cenario 1: Pular a partir do solo
2 Dado que o personagem do jogador está tocando em uma plataforma
3 Quando o jogador apertar o botão X
4 Entao a animação de pulo deve tocar
5 E a posição do personagem deve incrementar em 10 unidades por segundo
6 E o personagem encontra-se no estado interno ``Pulando''
7
8 Cenario 2: Alcançar altura máxima
9 Dado que o personagem do jogador está no estado ``Pulando''
10 Quando a posição vertical do personagem for igual a 20 unidades a partir
11 do ponto inicial de pulo
12 Entao a animação de queda deve tocar
13 E a posição do personagem deve diminuir em 10 unidades por segundo
14 E o personagem encontra-se no estado interno ``Caindo''
```

desenvolvimento. O ciclo do BDD apresentado por Smart [2] e as fases do desenvolvimento de jogos apresentadas por Chandler [3] foram mapeadas com base nas características de produção de jogos independentes, fornecendo especificações de forma a facilitar a compreensão do uso de BDD na produção de jogos independentes. As proposições iniciais deste mapeamento foram ainda validadas por profissionais de desenvolvimento de jogos, as quais contribuíram para a versão final apresentada neste capítulo.

Abordou-se ainda neste capítulo os aspectos de testes que podem ser trabalhados pelo BDD-I. Buscou-se na literatura e no estudo de campo realizado com BDD, dados para identificar quais aspectos de testes são providos pelo uso da prática. Foram observados cinco aspectos de testes: Erros e Defeitos, Desempenho, Jogabilidade, Balanceamento e Projeto de Níveis. Apresentou-se detalhes sobre cada aspectos, com uso de citações de profissionais e da literatura e exemplos de como cenários podem ser descritos utilizando a Linguagem Ubíqua. Ao ter a possibilidade de abordar cinco aspectos de testes, o uso de BDD-I se demonstra uma alternativa vantajosa em relação a outras técnicas de teste utilizadas durante a produção de jogos independentes.

A prática do BDD-I se apresenta como uma solução interessante para os desenvolvedores de jogos independentes. Através da Linguagem Ubíqua, temos uma forma eficiente de especificar comportamentos do jogo com colaboração de outros profissionais. A partir destes comportamentos especificados, testes podem ser criados para guiar a implementação do jogo. O código-fonte nasce testado, uma vez que cada cenário origina um caso de teste, que por sua ver aborda um comportamento do jogo. A documentação criada em Linguagem Ubíqua pode compor seções de GDDs e Plano de Testes. Outras documentações técnicas obtidas através de ferramentas BDD-I podem fornecer um monitoramento melhor do progresso do jogo e seu estado atual. Desta forma, foi proposto neste capítulo um conjunto de especificações para uso da prática do BDD no que tange a criação e execução de testes que observam as necessidades do desenvolvedor de jogos independentes.

6 Conclusão

Este capítulo apresenta as conclusões e resultados obtidos por este trabalho. Em seguida, elenca-se um conjunto de trabalhos futuros oriundos desta dissertação. Por fim, é apresentado os artigos publicados e artigos em fase de produção para eventos e periódicos, oriundos desta dissertação.

6.1 Considerações Finais

O crescente mercado de desenvolvimento de jogos permitiu que novas oportunidades se configurassem na criação de jogos. Neste contexto, favorecidos pelo grande acesso a tecnologias e conhecimento, os desenvolvedores de jogos independentes surgiram, fomentando ainda mais o mercado com jogos inovadores e autorais. Porém, estes desenvolvedores operam de forma diferente do que se observa na grande indústria de jogos. Em geral, desenvolvedores de jogos independentes negociam com escassez de recursos humanos e financeiros. Desta forma, metodologias e processos de testes precisam ser adequados e propostos para atender às especificidades inerentes a este tipo de desenvolvedor.

O *Behavior-Driven Development for Indies* (BDD-I) surge como um conjunto de especificações que permite a prática de testes para aumentar a qualidade no desenvolvimento de jogos independentes, através do uso do BDD. Ao compreender como os desenvolvedores de jogos independentes realizam testes, investigou-se as práticas e os aspectos que eles buscam ao realizar testes. Foi possível assim analisar e identificar como a prática do BDD pode ser utilizada durante o ciclo de desenvolvimento de jogos independentes. O BDD-I apresenta um mapeamento de ciclos de desenvolvimentos do BDD e da produção de jogos. Este mapeamento permite que os desenvolvedores que desejam utilizarBDD-I possam conhecer como as atividades do desenvolvimento de jogos relacionam-se com as etapas do BDD. Os aspectos de testes que podem ser trabalhados pelo BDD-I permitem que os desenvolvedores possam tirar maior proveito da prática do BDD, fornecendo a eles

mais dados durante a realização de testes e melhorando assim a qualidade dos jogos por eles desenvolvidos. Especificar comportamento com uso da Linguagem Ubíqua do BDD-I fornecem casos de testes que expressam funcionalidades de jogos. Estas especificações podem, além de servir de casos de teste para as funcionalidades idealizadas, compor seções específicas de documentos como GDD e Plano de Testes.

Assim, elenca-se como as principais contribuições deste trabalho:

1. A investigação do estado da arte do BDD e suas ferramentas, do desenvolvimento de jogos independentes e suas práticas de testes.
2. A identificação de processos/práticas de testes utilizadas pelos desenvolvedores de jogos independentes e dos aspectos que estes buscam ao realizar testes através de pesquisa de opinião.
3. A apresentação do BDD-I, conjunto de especificações de uso de BDD no contexto de desenvolvimento de jogos independentes, observando as características e limitações da produção desses jogos, através de mapeamento e guia de uso da Linguagem Ubíqua por desenvolvedores de jogos independentes.

6.2 Trabalhos Futuros

Por se tratar de uma prática recente, com pouco relato na literatura sobre seu uso no desenvolvimento, o uso do BDD precisa ser investigado nos mais diferentes contextos. A adaptação proposta, BDD-I, fornece um conjunto de especificações para uso por desenvolvedores de jogos independentes que permite que diversas oportunidades sejam fornecidas a partir deste estudo inicial. Dentre estas oportunidades, elenca-se:

1. Explorar o uso de BDD-I em diferentes contextos do desenvolvimento de jogos, variando experimentos e estudos de casos com jogos de diferentes gêneros, com a finalidade de observar características não observadas nesta dissertação;
2. Fornecer especificações para construção de ferramentas BDD especializadas para uso pelos desenvolvedores de jogos independentes;

3. Comparar o BDD-I com outras técnicas de testes com a finalidade de identificar e trabalhar novos aspectos que possam ser atendidos pelo uso da Linguagem Ubíqua e dos casos de testes do BDD-I;

4. Buscar formas de especificar funcionalidades e cenários no BDD-I para que possam ser compartilhados entre os desenvolvedores com objetivo de que possam contribuir entre si para melhorar a qualidade de seus jogos.

6.3 Trabalhos Publicados

Esta dissertação teve um artigo submetido e aceito para o XXXIII Simpósio Brasileiro de Engenharia de Software (SBES 2019), intitulado "Há espaço para Teste de Software no Desenvolvimento de Jogos Independentes? Um survey com profissionais da Indústria de Jogos", no qual apresenta a condução e resultados do *survey* relatado no Capítulo 3.

Outro artigo está sendo produzido para o períodico *Entertainment Computing Elsevier*. Inicialmente intitulado "BDD-I - *Improving Quality on Indie Games*", o artigo abordará as contribuições feitas por esta dissertação pelo uso da prática do BDD no desenvolvimento de jogos independentes.

Referências Bibliográficas

[1] ANICHE, M. *Test-Driven Development: Teste e Design no Mundo Real*. [S.l.]: Editora Casa do Código, 2014. v. 1.

[2] SMART, J. F. *BDD in Action: Behavior-driven development for the whole software lifecycle*. [S.l.]: Manning, 2015.

[3] CHANDLER, H. M. *The Game Production Handbook*. MA, USA: Jones & Bartlett Publishers, 2009. ISBN 9781934015407.

[4] EGBREGHTS, A. A literature review of behavior driven development using grounded theory. 2017.

[5] LETHBRIDGE, T. C.; SIM, S. E.; SINGER, J. Studying software engineers: Data collection techniques for software field studies. *Empirical software engineering*, Springer, v. 10, n. 3, p. 311–341, 2005.

[6] PARKER, F. Indie game studies year eleven. In: *DiGRA Conference*. [S.l.: s.n.], 2013.

[7] GARDA, M. B.; GRABARCZYK, P. Is every indie game independent? towards the concept of independent game. *Game Studies*, v. 16, n. 1, 2016.

[8] LIPKIN, N. Examining indie's independence: The meaning of"indie"games, the politics of production, and mainstream cooptation. *Loading...*, v. 7, n. 11, 2013.

[9] ZAMBON, P. S. As vantagens de ser independente: inovação e criatividade na indústria brasileira de jogos digitais e suas influências no processo produtivo. *Metamorfose*, v. 2, n. 1, 2017.

[10] KASURINEN, J.; PALACIN-SILVA, M.; VANHALA, E. What concerns game developers? a study on game development processes, sustainability and metrics. In: *2017 IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM)*. [S.l.: s.n.], 2017. p. 15–21.

[11] ALEEM, S.; CAPRETZ, L. F.; AHMED, F. Critical success factors to improve the game development process from a developer's perspective. *Journal of Computer Science and Technology*, Springer, v. 31, n. 5, p. 925–950, 2016.

[12] PETRILLO, F. et al. What went wrong? a survey of problems in game development. *Computers in Entertainment (CIE)*, ACM, v. 7, n. 1, p. 13, 2009.

[13] MURPHY-HILL, E.; ZIMMERMANN, T.; NAGAPPAN, N. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In: *Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA: ACM, 2014. (ICSE 2014), p. 1–11. ISBN 978-1-4503-2756-5. Disponível em: <http://doi.acm.org/10.1145/2568225.2568226>.

[14] KASURINEN, J.; SMOLANDER, K. What do game developers test in their products? *International Symposium on Empirical Software Engineering and Measurement*, 09 2014.

[15] SCHULTZ, C. P.; BRYANT, R. D. *Game Testing: All in One*. 3rd. ed. USA: Mercury Learning & Information, 2016. ISBN 1942270763, 9781942270768.

[16] LEVY, L.; NOVAK, J. *Game development essentials: Game QA & testing*. [S.l.]: Cengage Learning, 2009.

[17] CISNEROS, L. A. et al. An experimental evaluation of itl, tdd and bdd. In: THINKMIND. *ICSEA 2018, The Thirteenth International Conference on Software Engineering Advances*. [S.l.], 2018. p. 20–24.

[18] BECK, K. *Test-driven development: by example*. [S.l.]: Addison-Wesley Professional, 2003.

[19] NORTH, D. Behavior modification: The evolution of behavior-driven development. *Better Software*, v. 8, n. 3, 2006.

[20] XU, S.; RAJLICH, V. Empirical validation of test-driven pair programming in game development. In: IEEE. *Computer and Information Science, 2006 and 2006 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse. ICIS-COMSAR 2006. 5th IEEE/ACIS International Conference on*. [S.l.], 2006. p. 500–505.

[21] LLOPIS, N.; HOUGHTON, S. Backwards is forward: Making better games with test-driven development. In: *Game Developers Conference*. [S.l.: s.n.], 2006.

[22] KEITH, C. *Agile game development with Scrum*. [S.l.]: Pearson Education, 2010.

[23] BOEIRA, J. N. *Lean Game Development: Desenvolvimento Enxuto de Jogos*. 1st. ed. SP, Brazil: Casa do Código, 2017. ISBN 9788555192647.

[24] EVANS, E. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321125215.

[25] SOLIS, C.; WANG, X. A study of the characteristics of behaviour driven development. In: IEEE. *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*. [S.l.], 2011. p. 383–387.

[26] AMPATZOGLOU, A.; STAMELOS, I. Software engineering research for computer games: A systematic review. *Information and Software Technology*, Elsevier, v. 52, n. 9, p. 888–901, 2010.

[27] PYSHKIN, E.; MOZGOVOY, M.; GLUKHIKH, M. On requirements for acceptance testing automation tools in behavior driven software development. In: *In Proc. of the Central & Eastern European Software Engineering Conf. Russia (CEE-SEC (R)*. [S.l.: s.n.], 2012.

[28] FILHO, M. C. et al. Um estudo de caso sobre o aumento de qualidade de software em projetos de sistemas de informação que utilizam test driven development. 2012.

[29] CAUSEVIC, A.; SUNDMARK, D.; PUNNEKKAT, S. Factors limiting industrial adoption of test driven development: A systematic review. In: IEEE. *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*. [S.l.], 2011. p. 337–346.

[30] BALAJI, S.; MURUGAIYAN, M. S. Waterfall vs. v-model vs. agile: A comparative study on sdlc. *International Journal of Information Technology and Business Management*, v. 2, n. 1, p. 26–30, 2012.

[31] ALSHAMRANI, A.; BAHATTAB, A. A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model. *International Journal*

of Computer Science Issues (IJCSI), International Journal of Computer Science Issues (IJCSI), v. 12, n. 1, p. 106, 2015.

[32] LI, E. Y. Software testing in a system development process: A life cycle perspective. *Journal of Systems management*, John Carroll University, School of Business, v. 41, n. 8, p. 23–31, 1990.

[33] TUTEJA, M.; DUBEY, G. et al. A research study on importance of testing and quality assurance in software development life cycle (sdlc) models. *International Journal of Soft Computing and Engineering (IJSCE)*, v. 2, n. 3, p. 251–257, 2012.

[34] SYAIKHUDDIN, M. M. et al. Conventional software testing using white box method. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, v. 3, n. 1, p. 65–72, 2018.

[35] DIVYA C D, B. T. S. A brief overview of software testing techniques and test automation. In: *Proceedings of the International Journal For Technological Research In Engineering*. [S.l.: s.n.], 2016. v. 3, n. 5, p. 1002–1006. ISSN 2347-4718.

[36] ACHARYA, S.; PANDYA, V. Bridge between black box and white box–gray box testing technique. *International Journal of Electronics and Computer Science Engineering*, v. 2, n. 1, p. 175–185, 2012.

[37] FILHO, V. V.; RAMALHO, G. L. Deepening the understanding of mobile game success. *Proceedings of the XIII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2014.

[38] POLITOWSKI, C. et al. Software engineering processes in game development: a survey about brazilian developers' experiences. *Proceedings of the XIV Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2016.

[39] POSVOLSKI, A. S. et al. Agigame: Proposta de uma metodologia híbrida para desenvolvimento de jogos. *Proceedings of the XIII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2014.

[40] SALES, R. Proposta de método para gestão ágil da visão no desenvolvimento de jogos digitais. *Anais do XII Simpósio Brasileiro de Jogos e Entretenimento Digital*, 2013.

[41] ALEEM, S.; CAPRETZ, L. F.; AHMED, F. Game development software engineering process life cycle: a systematic review. *Journal of Software Engineering Research and Development*, SpringerOpen, v. 4, n. 1, p. 6, 2016.

[42] NOVAK, J. *Game Development Essentials: An Introduction*. 2nd. ed. [S.l.]: Delmar Learning, 2007. ISBN 1418042080.

[43] PERUCIA, A. S. et al. *Desenvolvimento de Jogos Eletrônicos: Teoria e Prática*. 2nd. ed. Brazil: Editora Novatec, 2007. ISBN 978-85-7522-122-8.

[44] MIRZA-BABAEI, P.; MOOSAJEE, N.; DRENIKOW, B. Playtesting for indie studios. In: ACM. *Proceedings of the 20th International Academic Mindtrek Conference*. [S.l.], 2016. p. 366–374.

[45] CHOI, J. O. et al. Playtesting with a purpose. In: ACM. *Proceedings of the 2016 annual symposium on computer-human interaction in play*. [S.l.], 2016. p. 254–265.

[46] FERNÁNDEZ, D. M.; WAGNER, S. Naming the pain in requirements engineering: A design for a global family of surveys and first results from germany. *Information and Software Technology*, Elsevier, v. 57, p. 616–643, 2015.

[47] GHAZI, A. N. et al. Survey research in software engineering: Problems and mitigation strategies. *IEEE Access*, IEEE, v. 7, p. 24703–24718, 2019.

[48] WOHLIN, C. et al. *Experimentation in Software Engineering*. [S.l.]: Springer Science & Business Media, 2012. ISBN 978-3-642-29043-5.

[49] KASURINEN, J.; MAGLYAS, A.; SMOLANDER, K. Is requirements engineering useless in game development? In: SPRINGER. *International Working Conference on Requirements Engineering: Foundation for Software Quality*. [S.l.], 2014. p. 1–16.

[50] SOMMERVILLE, I. Software engineering 9th edition. *ISBN-10*, v. 137035152, 2011.

[51] WAZLAWICK, R. *Engenharia de software: conceitos e práticas*. [S.l.]: Elsevier Brasil, 2013. v. 1.

[52] LEACH, R. J. *Introduction to software engineering*. [S.l.]: CRC Press, 2016.

[53] ZEIN, S.; SALLEH, N.; GRUNDY, J. A systematic mapping study of mobile application testing techniques. *Journal of Systems and Software*, Elsevier, v. 117, p. 334–356, 2016.

[54] SAKUDA, L. O.; FORTIM, I. O. Ii censo da indústria brasileira de jogos digitais. *Ministério da Cultura: Brasília*, 2018. Disponível em: <http://www.tinyurl.com/censojogosdigitais>.

[55] GLASER, B. G.; STRAUSS, A. L.; STRUTZEL, E. The discovery of grounded theory; strategies for qualitative research. *Nursing research*, LWW, v. 17, n. 4, p. 364, 1968.

[56] O'HAGAN, A. O.; O'CONNOR, R. V. Towards an understanding of game software development processes: A case study. In: SPRINGER. *European Conference on Software Process Improvement*. [S.l.], 2015. p. 3–16.

[57] STOL, K.-J.; RALPH, P.; FITZGERALD, B. Grounded theory in software engineering research: a critical review and guidelines. In: IEEE. *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. [S.l.], 2016. p. 120–131.

[58] CHAMBERLAIN-SALAUN, J.; MILLS, J.; USHER, K. Linking symbolic interactionism and grounded theory methods in a research design: From corbin and strauss' assumptions to action. *Sage Open*, SAGE Publications Sage CA: Los Angeles, CA, v. 3, n. 3, p. 2158244013505757, 2013.

[59] CORBIN, J.; STRAUSS, A. et al. Basics of qualitative research: Techniques and procedures for developing grounded theory. Thousand Oaks, CA: Sage, 2008.

[60] STRAUSS, A.; CORBIN, J. M. *Grounded theory in practice*. [S.l.]: Sage, 1997.

[61] BANDEIRA-DE-MELLO, R.; CUNHA, C. Operacionalizando o método da grounded theory nas pesquisas em estratégia: técnicas e procedimentos de análise com apoio do software atlas/ti. *Encontro de Estudos em Estratégia*, v. 1, p. 2003, 2003.

[62] RAMADAN, R.; WIDYANI, Y. Game development life cycle guidelines. In: IEEE. *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. [S.l.], 2013. p. 95–100.

[63] ELADHARI, M. P.; OLLILA, E. M. Design for research results: experimental prototyping and play testing. *Simulation & Gaming*, SAGE Publications Sage CA: Los Angeles, CA, v. 43, n. 3, p. 391–412, 2012.

[64] KORHONEN, H. Comparison of playtesting and expert review methods in mobile game evaluation. In: ACM. *Proceedings of the 3rd International Conference on Fun and Games*. [S.l.], 2010. p. 18–27.

[65] WINN, B.; HEETER, C. Resolving conflicts in educational game design through playtesting. *Innovate: Journal of Online Education*, v. 3, n. 2, 2006.

[66] EHMER, M.; KHAN, F. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications*, v. 3, 06 2012.

[67] GODOY, A.; BARBOSA, E. F. Game-scrum: An approach to agile game development. *Proceedings of SBGames*, p. 292–295, 2010.

[68] LAUBISCH, A.; CLUA, E. Scrum4games: Uma aplicação do scrum para projetos de games focada em game design. *Simpósio Brasileiro de Jogos e Entretenimento Digital*, p. 178–187, 2010.

[69] KOUTONEN, J.; LEPPÄNEN, M. How are agile methods and practices deployed in video game development? a survey into finnish game studios. In: SPRINGER. *International Conference on Agile Software Development*. [S.l.], 2013. p. 135–149.

[70] DJAOUTI, D. et al. A gameplay definition through videogame classification. *International Journal of Computer Games Technology*, Hindawi Publishing Corp., v. 2008, p. 4, 2008.

[71] LEWIS, C.; WHITEHEAD, J.; WARDRIP-FRUIN, N. What went wrong: a taxonomy of video game bugs. In: ACM. *Proceedings of the fifth international conference on the foundations of digital games*. [S.l.], 2010. p. 108–115.

[72] GHAZI, A. N. et al. Survey research in software engineering: Problems and mitigation strategies. *IEEE Access*, IEEE, v. 7, p. 24703–24718, 2019.

[73] CALLELE, D.; NEUFELD, E.; SCHNEIDER, K. Requirements engineering and the creative process in the video game industry. In: IEEE. *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*. [S.l.], 2005. p. 240–250.

[74] TRAN, M. Q.; BIDDLE, R. Collaboration in serious game development: a case study. In: ACM. *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*. [S.l.], 2008. p. 49–56.

[75] NASCIMENTO, L. M.; ALMEIDA, E. S. de; MEIRA, S. R. de L. A case study in software product lines-the case of the mobile game domain. In: IEEE. *34th Euromicro Conference Software Engineering and Advanced Applications*. [S.l.], 2008. p. 43–50.

[76] RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, Springer, v. 14, n. 2, p. 131, 2009.

[77] STAKE, R. E. *The art of case study research*. [S.l.]: Sage, 1995.

[78] ROBSON, C. *Real world research: A resource for social scientists and practitioner-researchers*. [S.l.]: Oxford: Blackwell, 2002.

[79] DIEPENBECK, M. et al. Towards automatic scenario generation from coverage information. In: IEEE. *Automation of Software Test (AST), 2013 8th International Workshop on*. [S.l.], 2013. p. 82–88.

[80] WANDERLEY, F.; SILVERIA, D. S. da. A framework to diminish the gap between the business specialist and the software designer. In: IEEE. *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*. [S.l.], 2012. p. 199–204.

[81] LAZAR, I.; MOTOGNA, S.; PÂRV, B. Behaviour-driven development of foundational uml components. *Electr. Notes Theor. Comput. Sci.*, v. 264, n. 1, p. 91–105, 2010.

[82] SOUZA, P. L. de et al. Combining behaviour-driven development with scrum for software development in the education domain. 2017.

[83] BRAAMS, S. Developing a software quality framework for low-code model driven development platforms based on behaviour driven development methodology. 2017.

[84] LOPEZ-PELLICER, F. J. et al. Behaviour-driven development applied to the conformance testing of inspire web services. In: *Connecting a Digital Europe Through Location and Place*. [S.l.]: Springer, 2014. p. 325–339.

[85] TOLL, D.; OLSSON, T. Why is unit-testing in computer games difficult? In: IEEE. *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*. [S.l.], 2012. p. 373–378.

[86] DAYLAMANI-ZAD, D.; ANGELIDES, M. C.; AGIUS, H. Lu-lu: A framework for collaborative decision making games. *Decision Support Systems*, Elsevier, v. 85, p. 49–61, 2016.

[87] SOEKEN, M.; WILLE, R.; DRECHSLER, R. Assisted behavior driven development using natural language processing. In: SPRINGER. *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. [S.l.], 2012. p. 269–287.

A Questionário Aplicado no *Survey*

Tabela A.1: Perguntas do questionário do *survey*.

Categoria	#	Pergunta	Tipo de Questão
Demografia	Q1	Você se considera um desenvolvedor de jogos independente?	Fechada (ME)
	Q2	Quanto tempo de experiência você possui na produção de jogos?	Fechada (ME)
	Q3	Qual seu principal cargo/papel na produção de jogos?	Fechada (ME)
	Q4	Quantos jogos você já publicou?	Fechada (ME)
	Q5	Quantas pessoas trabalham/trabalharam com você na produção de jogos?	Fechada (ME)
	Q6	Quais as plataformas-alvo de seu(s) jogo(s)?	Fechada (CS)
	Q7	Qual(is) o(s) canal(is) de publicação/distribuição do(s) seu(s) jogo(s)?	Fechada (CS)
Processo	Q8	Você e sua equipe utilizam alguma metodologia, processo ou prática de desenvolvimento na produção de jogos? Se sim, qual(is)?	Fechada (CS)
	Q9	Poderia nos fornecer detalhes de como ocorre o processo de desenvolvimento de jogos?	Aberta
Testes	Q10	Durante a produção de seu(s) jogo(s), quem se dedica à organização e execução dos testes?	Fechada (ME)

Tabela A.1 – *Continuação da página anterior.*

Categoria	#	Pergunta	Tipo de Questão
	Q11	Utilizam alguma plataforma para gerenciamento de testes, relatórios de erro e feedback? Se sim, qual(is)?	Aberta
	Q12	Qual a relação média de tempo de implementação e de testes durante a produção do jogo?	Fechada (ME)
	Q13	Que processos/práticas de teste você e/ou sua equipe realizam/realizaram na produção dos jogos?	Fechada (CS)
	Q14	Ao realizar testes em jogos, que tipo de aspectos você julga ser mais importante descobrir?	Aberta

B Canais de Comunicação Utilizados no *Survey*

Tabela B.1: Canais de comunicação do *survey*.

Nome	**Tipo de Canal**	**Quantidade de Usuários**[1]	**Data de Publicação**
365 Indies	Discord		2019-01-27
AMAGames	Discord	24	2019-01-14
AMAGames	Whatsapp	115	2019-01-13
AMAGames Grupo	Facebook	297	2019-01-13
AMAGames Página	Facebook	983	2019-01-14
Boteco Gamer	Facebook	6800	2019-01-27
Boteco Indie	Discord		2019-01-27
Criação e Desenvolvimento de Jogos	Facebook	1821	2019-02-04
Desenvolvedores Brasileiros de Games	Facebook	1511	2019-02-04
Desenvolvimento de Jogos - Brasil	Facebook	5972	2019-01-28
Finalistas GJ+ 2018	Whatsapp	76	2019-01-13
Game Development Brazil	Facebook	1692	2019-02-04
Game Development Brazil - Grupo	Facebook	2255	2019-02-04
Game Jam+	Discord		2019-01-27
Game Jam+ Organizadores	Whatsapp	27	2019-01-13
Gamedev Nordeste	Whatsapp	13	2019-01-13
GameDevelopers BraSil	LinkedIn	43	2019-01-27
GAMinG - Associação Mineira de Jogos	Facebook	1246	2019-02-04
GDBR	Facebook	849	2019-02-04
GGDevCast	Discord		2019-01-27

Tabela B.1 – *Continuação da página anterior.*

Nome	**Tipo de Canal**	**Quantidade de Usuários**[2]	**Data de Publicação**
Global Game Jam	Discord		2019-01-27
IGDA Recife	Facebook	658	2019-02-04
INDBR	Discord	154	2019-01-13
Indie Dev Brazil	Discord		2019-01-27
Indie Developers Brasil	Facebook	43415	2019-02-04
Indie Game Developer Brazil	Facebook	2398	2019-01-27
Indies Brasil	Whatsapp	257	2019-01-13
Indies Brasil	Telegram	139	2019-01-13
Indústria de Jogos Grupo	LinkedIn	150	2019-01-13
jogos-l@sbc.org.br	Lista de E-Mails		2019-02-01
Nordevs Grupo	Facebook	520	2019-01-13
PONG - Potiguar Indie Games	Facebook	286	2019-02-04
r/gamedevbr	Reddit	30	2019-01-31
r/gamesEcultura	Reddit	5100	2019-01-31
RING - Desenvolvedores de jogos do Rio de Janeiro	Facebook	1733	2019-02-04
Unity BR	Discord	Mais de 100 membros	2019-01-13
Unity Brasil	Facebook	18984	2019-02-08
Unreal Engine 4 Brasil	Facebook	9352	2019-02-04

C Questionário Aplicado no Estudo de Caso

Tabela C.1: Perguntas do questionário do estudo de caso.

Questionário de Perguntas do Estudo de Caso	
Q1	O que foi possível especificar utilizando o BDD?
Q2	Você sentiu dificuldades em especificar os comportamentos das funcionalidades do jogo? Quais?
Q3	Ao especificar os comportamentos das funcionalidades do jogo, o que não foi possível especificar?
Q4	Descrever os comportamentos ajudou a melhorar detalhes de implementação?
Q5	A linguagem ubíqua favoreceu a comunicação entre vocês?
Q6	Durante o processo, com que frequência foi necessário atualizar a descrição dos comportamentos?
Q7	Se foi necessário atualizar a descrição dos comportamentos, como você avalia este processo? O que foi atualizado/descrito?
Q8	Você usaria BDD novamente para especificar jogos? Justifique.
Q9	Da ferramenta utilizada, o que vocês diriam que foi uma limitação muito grande para que se pudessem executar os testes com mais facilidade?

D Anotações de Campo da Unidade de Análise A

Perfil dos Participantes

- 2 participantes
- Experiência em jogos entre 1 e 3 anos.
- Participante 1 (6 jogos), Participante 2 (3 jogos)
- Se conheciam antes, são sócios de uma empresa de jogos.
- Trabalharam juntos antes, com jogos.
- Não conheciam BDD, TDD ou DDD
- Participante 1 não desenvolve outros softwares que não seja jogos.
- Participante 2 tem um outro emprego onde desenvolve apps e sites.
- Ambos possuem nível superior em Ciência da Computação.

Condução do Experimento

O experimento foi gravado a partir da etapa "Construção do Jogo: Brainstorming". Todo o material necessário e ambiente de produção já estava previamente configurado para os participantes.

Introdução ao Desenvolvimento Dirigido por Comportamentos (BDD)

- Durou mais que o previsto.

- Participantes não se identificaram muito com os problemas citados.
- Nunca trabalharam com testes, tive de explicar conceitos básicos como teste de unidade, integração e aceitação.
- Não compreenderam o conceito de TDD, a apresentação ficou muito rasa quanto a desenvolver testes antes de qualquer implementação.
- Por nunca terem tido nem experiência com testes de unidade, não conheciam o método assert(). Foi preciso explicar um pouco mais as classes de testes do TDD.
- A comparação de uma classe de testes do TDD e outra do BDD aparentou para os participantes que dava mais trabalho do que realizar testes normais.

Desenvolvimento Dirigido a Comportamentos com Unity

- Durou mais que o previsto.
- No meio da apresentação, foi solicitado pra ser algo mais prático. Então abri o projeto de exemplos nos computadores dos participantes para que eles pudessem acompanhar melhor os slides.
- Houve alguns erros de interpretação na documentação da ferramenta que ficaram no slide. Acabamos descobrindo essas coisas durante a apresentação.

Construção do Jogo: Brainstorming

- Os participantes optaram por fazer um jogo simples e no qual eles já haviam pensado previamente antes. O brainstorming se limitou a definirem o escopo do que conseguiriam fazer durante a duração do experimento.
- O jogo desenvolvido é baseado na mesma mecânica de um popular jogo chamado Reigns, disponível para venda na Steam. É um jogo de estratégia, situado em um mundo medieval fictício, o jogador assume o papel do monarca que governa um reino aceitando ou rejeitando sugestões de conselheiros.
- A temática do jogo desenvolvido foi empreendedorismo. As questões que surgiam abordavam problemas comuns a um empreendedor.

Construção do Jogo: Especificação das funcionalidades e comportamentos

- Os participantes escolheram as duas funcionalidades principais do jogo a ser desenvolvido: mecânicas de swipe e gerenciamento de áreas.
- Os participantes descreveram juntos os 8 cenários que contemplavam o ciclo básico do jogo a ser desenvolvido: Iniciar o jogo, Arrastar para o lado, Arrasta e solta, Resposta sobre uma carta, Vitória, Pontuação, Derrota, Reiniciar Partida.
- Os participantes, ao descrever os cenários, indagaram o formato e a diferença do "Dado que" para "Quando".
- Foi disponibilizado para eles os slides da apresentação sobre BDD.
- As funcionalidades e cenários foram escritos em texto plano, no formato txt.

Construção do Jogo: Implementação

- Os participantes utilizaram a game engine Unity 2017.3.1 para construir o jogo, com as bibliotecas para testes BDD Extension Framework e Unity Test Tools.
- Utilizaram o sistema de versionamento GIT. Para compartilhamento do projeto, usaram o serviço de repositórios em nuvem BitBucket.
- Os participantes criaram primeiro as classes de testes que representavam todos os cenários do jogo.
- Optaram por usar o componente estático StaticBDDComponent, indicando como motivação a pouca reutilização de métodos para testes dos cenários.
- Tiveram muitas dúvidas sobre a criação das classes de teste do BDD Extension Framework. Constantemente recorreram a documentação online e ao instrutor.
- Em alguns cenários, optaram por implementar primeiro o código funcional e depois adaptar às classes de teste.
- Os participantes passaram muito tempo implementado a mecânica swipe do jogo.

- Os participantes tiveram dificuldade em entender como seria realizado os testes automáticos quando necessitasse de um evento especifico que necessitaria de uma entrada de teclado ou mouse, por exemplo.

- Esta fase durou mais de 8 horas, o que gerou cansaço nos participantes, que no final, começaram a fazer código "ruim" apenas para satisfazer os testes.

Fatores Limitantes do Experimento

- Os participantes nunca utilizaram rotinas de testes antes e desconheciam os termos utilizados. Foi necessário ambientá-los muito mais quanto a testes de software.

- Houve resistência quanto à prática do "teste primeiro, implemente depois". Alguns cenários não foram implementados utilizando esta prática. Necessidade de reforçar o TDD quando for fazer estes experimentos.

- O número de casos de cenários descritos foi relativamente baixo, apesar de englobar em sua totalidade todos os trechos de código de funcionalidades implementadas. Não houve cobertura de cenários de falha, apenas cenários de casos de sucesso ou de passos esperados corretamente pelo usuário.

- Pela necessidade interferir o mínimo no experimento, não foram dados muitos exemplos para não induzir a implementação, mas acredito que a pouca experiência dos participantes quanto a implementar a mecânica desejada foi um fator que limitou a riqueza de detalhes nos cenários.

- Houve muita confusão na hora de diferenciar contexto (Dado que) de evento (Quando). Apenas nas horas finais da etapa de implementação é que o conceito ficou mais claro nos participantes.

- A ferramenta escolhida possui uma documentação muita rica em detalhes de uso, mas possui exemplos muito básicos. Acredito que um exemplo mais robusto acompanhado dos slides facilita a compreensão por parte de participantes com nenhuma ou pouca vivência na área de testes.

E Funcionalidade e Cenários da Unidade de Análise A

Código E.1: Funcionalidade e Cenários da Unidade de Análise A.

```
1 Funcionalidade: Aguardar na tela de jogo, por ação do usuario
2 ----------------------------------------------------------------
3 - ao mostrar a tela de fim de jogo, o que deve estar presente na tela
4
5 Cenario 1: Iniciar o jogo
6 Dado que A tela do jogo carrega
7 Quando Inicia a aplicação
8 Entao Deve mostrar 3 áreas no topo da tela
9 E Mostra texto de pontuação com zero
10 E Mostra texto com descrição da questão, acima de carta de ação
11 E Mostra carta de ação com imagem do personagem
12 E Mostra a pergunta que deve ser respondida, abaixo da carta de ação
13
14 Cenario 2: Arrastar para o lado
15 Dado que O jogador arrasta a carta para o lado
16 Quando Entre o ponto inicial e o ponto final
17 Entao A carta inclina para o lado
18 E Se desloca horizontalmente para o lado que está sendo arrastada
19 E Fica parada onde está sendo segura
20
21 Cenario 3: Arrasta e solta
22 Dado que O jogador arrasta a carta para o lado e solta
23 Quando Entre o ponto inicial e o ponto final
24 Entao A carta volta para o ponto inicial
```

```
25 Cenario 4: Resposta sobre uma carta
26 Dado que O jogador arrasta a carta para o lado
27 Quando Até o fim
28 Entao Modifica os valores das áreas, de acordo com a resposta
29 E A carta arrastada sai da tela
30 E Uma nova carta aparece em seu lugar
31
32 Cenario 5: Vitória
33 Dado que O jogador responde todas as 5 questões
34 Quando Não zera nenhuma área
35 Entao Mostra tela de vitória por cima da tela de jogo
36 E Mostra botão de reiniciar a partida
37
38 Cenario 6: Pontuação
39 Dado que Modifica os valores das áreas
40 Quando O jogador não zera uma das áreas
41 Entao Incrementa contador de questões na tela
42
43 Cenario 7: Derrota
44 Dado que Modifica os valores das áreas
45 Quando O jogador zera uma das áreas
46 Entao Mostra tela de derrota por cima da tela de jogo
47 E Mostra botão de reiniciar a partida
48
49 Cenario 8: Reiniciar Partida
50 Dado que Ao finalizar uma partida
51 Quando O jogador clica no botão de voltar
52 Entao Esconde tela de final de jogo
53 E Mostra tela de jogo
```

F Questionário Aplicado na Prova de Conceito do BDD-I

Os tipos de questões apresentadas neste questionário são: (i) Múltipla Escolha (ME), onde o participante pode escolher apenas uma das diversas opções apresentadas; (ii) Escala Likert (LK) de 5 pontos, onde o participante deve escolher dentre as opções dentre um grau de concordância variando de "Discordo Totalmente" a "Concordo Totalmente"; e (iii) Aberta (A), onde o participante pode redigir sua resposta em texto livre.

Tabela F.1: Seções do questionário aplicado na Prova de Conceito do BDD-I.

Seção	Etapas do BDD segundo Smart [2]
I	Busca da Proposta de Valor
II	Descoberta e Compreensão das Funcionalidades
III	Construindo as Funcionalidades
IV	Especificações Executáveis e Entrega de Funcionalidades
V	Manutenção

Tabela F.2: Perguntas do questionário do mapeamento do BDD-I.

Seção	#	Pergunta	Tipo
A	Q1	Quanto tempo de experiência você possui como desenvolvedor/programador na produção de jogos?	ME
I	Q2	De acordo com a sua experiência com desenvolvimento de jogos, assinale sua opinião quanto a afinidade das características abaixo com a etapa "Busca da Proposta de Valor"apresentado por [Smart 2015]?	LK
	Q3	Se você discordou de alguns dos itens anteriores, por favor nos informe as razões/motivos.	A

Tabela F.2 – *Continuação da página anterior.*

Seção	#	Pergunta	Tipo
	Q4	Existem características no desenvolvimento de jogos não listadas acima que você considera com alto grau de afinidade com a etapa "Busca da Proposta de Valor"apresentado por [Smart 2015]? Se sim, descreva-a abaixo.	A
II	Q5	De acordo com a sua experiência com desenvolvimento de jogos, assinale sua opinião quanto a afinidade das características abaixo com a etapa "Descoberta e Compreensão das Funcionalidades"apresentado por [Smart 2015]?	LK
	Q6	Se você discordou de alguns dos itens anteriores, por favor nos informe as razões/motivos.	A
	Q7	Existem características no desenvolvimento de jogos não listadas acima que você considera com alto grau de afinidade com a etapa "Descoberta e Compreensão das Funcionalidades"apresentado por [Smart 2015]? Se sim, descreva-a abaixo.	A
III	Q8	De acordo com a sua experiência com desenvolvimento de jogos, assinale sua opinião quanto a afinidade das características abaixo com a etapa "Construindo as Funcionalidades"apresentado por [Smart 2015]?	LK
	Q9	Se você discordou de alguns dos itens anteriores, por favor nos informe as razões/motivos.	A
	Q10	Existem características no desenvolvimento de jogos não listadas acima que você considera com alto grau de afinidade com a etapa "Construindo as Funcionalidades"apresentado por [Smart 2015]? Se sim, descreva-a abaixo.	A
IV	Q11	De acordo com a sua experiência com desenvolvimento de jogos, assinale sua opinião quanto a afinidade das características abaixo com a etapa "Especificações Executáveis e Entrega de Funcionalidades"apresentado por [Smart 2015]?	LK

Tabela F.2 – *Continuação da página anterior.*

Seção	#	Pergunta	Tipo
	Q12	Se você discordou de alguns dos itens anteriores, por favor nos informe as razões/motivos.	A
	Q13	Existem características no desenvolvimento de jogos não listadas acima que você considera com alto grau de afinidade com a etapa "Especificações Executáveis e Entrega de Funcionalidades"apresentado por [Smart 2015]? Se sim, descreva-a abaixo.	A
V	Q14	De acordo com a sua experiência com desenvolvimento de jogos, assinale sua opinião quanto a afinidade das características abaixo com a etapa "Manutenção"apresentado por [Smart 2015]?	LK
	Q15	Se você discordou de alguns dos itens anteriores, por favor nos informe as razões/motivos.	A
	Q16	Existem características no desenvolvimento de jogos não listadas acima que você considera com alto grau de afinidade com a etapa "Manutenção"apresentado por [Smart 2015]? Se sim, descreva-a abaixo.	A