Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento de um jogo educativo para o auxílio no ensino da Extreme Programming (XP)

Lucas Freitas do Rosário

MSc. Jéssica Bassani de Oliveira(Orientador)

MSc. Diogo Fernando Trevisan(Co-orientador)

Dourados - MS

2016

Desenvolvimento de um jogo educativo para o auxílio no ensino da Extreme Programming (XP)

Lucas Freitas do Rosário

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Lucas Freitas do Rosário e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Dourados, 10 de novembro de 2016.

MSc. Jéssica Bassani de Oliveira(Orientador)

MSc. Diogo Fernando Trevisan(Co-orientador)

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento de um jogo educativo para o auxílio no ensino da Extreme Programming (XP)

Lucas Freitas do Rosário

Outubro de 2016

Banca Examinadora:

Profa. MSc. Jéssica Bassani de Oliveira(Orientador)
Área de Computação - UEMS
Prof. MSc. Diogo Fernando Trevisan(Co-orientador)
Área de Computação - UEMS
Profa. Dra. Glaucia Gabriel Sass
Área de Computação - UEMS
Profa. Dra. Mercedes Rocío Gonzales Márquez
Área de Computação - UEMS

AGRADECIMENTOS

Aos meus pais, pelo amor, incentivo e apoio incondicional.

A essa universidade, seu corpo docente, direção e administração que proporcionaram essa oportunidade.

Aos meus orientadores, pelo empenho dedicado à elaboração desse trabalho.

RESUMO

É notória a dificuldade natural para o aprendizado produtivo de novos assuntos. Aliada a essa dificuldade, no meio acadêmico e profissional ainda existem diversas outras dificuldades que contribuem para o insucesso do aprendizado. Dessa forma, com um jogo eletrônico educativo utilizado como ferramenta educacional ilustrativa, a complexidade para o aprendizado eficiente da metodologia poderá ser dirimido? Para resolver essa problemática, esse estudo tem como objetivo principal o desenvolvimento de um jogo eletrônico educativo que ilustre a metodologia de desenvolvimento ágil de software *Extreme Programming* (XP), com a finalidade de dirimir ou eximir o tempo gasto com o aprendizado dessa metodologia por métodos convencionais de ensino. Especificamente, aplicar a metodologia de desenvolvimento ágil de software XP e design do jogo e implementar o jogo educativo que ilustre essa metodologia, com base no design estudado. O método do tipo experimental foi utilizado como método de procedimento, visto que a pesquisa foi moldada com o intuito de desenvolver o jogo como ferramenta auxiliadora no processo de aprendizado, buscando atingir o objetivo descrito. O desenvolvimento do jogo foi baseado em um *Game Design Document* (GDD), feito exclusivamente para esse fim. É esperado que a ferramenta possa vir solucionar a problemática apresentada, atendendo os objetivos. Ainda, como sugestão para trabalhos futuros, pode-se adicionar novos elementos no jogo e disponibilizá-lo em outras plataformas.

Palavras-chave: Jogo. Extreme Programming. Educativo.

ABSTRACT

It is notorious the natural difficult to productively learn new subjects. Coupled with this difficulty, in the academic and professional environment there are still many other problems that contribute to the failure of learning. Thus, with an educational game used as illustrative educational tool, the complexity for efficient learning of the methodology can be resolved? To solve this problem, this study aims to develop an educational game that illustrates the agile development methodology of software Extreme Programming (XP), in order to settle or exclude the time spent learning this methodology by methods conventional teaching. Specifically, apply the agile development methodology of software XP and game design and implement educational game that illustrates this methodology, based on the study of game design. The experimental type method was used as a method of procedure, because the research has been shaped in order to develop the game as a helper tool in the learning process, seeking to attain the objective described. The development of the game was based on a Game Design Document (GDD), made exclusively for this purpose. It is expected that the tool can come to solve the problem presented, It is expected that the tool can come to solve the problem presented, meeting the objectives. Still, as a suggestion for future work, can add new elements to the game and make it available on other platforms.

Keywords: Game. Extreme Programming. Educational.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

GDD	Game Design Document
TDD	Technical Design Document
UI	User Interface
XP	Extreme Programming

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 INTRODUÇÃO

Com o intuito do aprendizado nas mais variadas áreas de conhecimento, o interesse da utilização de jogos, por parte de instituições de ensino ou empresas, como ferramenta auxiliadora nesse processo tem aumentado. Sendo assim, a categoria de jogos educativos é uma ferramenta importante para esse processo de aprendizagem, uma vez que pode auxiliar na dificuldade natural do ser humano em assimilar novos assuntos.

Além da dificuldade natural de desenvolver e assimilar um novo assunto de maneira produtiva, no meio educacional ou no profissional existem outras complicações que, em conjunto com fatores aleatórios do dia-a-dia, podem influenciar negativamente no aprendizado. Ainda, a dificuldade natural é maximada se o conteúdo a ser assimilado for muito extenso e não houver exemplificações gráficas. Essa dificuldade natural pode ser dirimida ou até mesmo eximida se utilizado um jogo educativo como uma ferramenta educacional ilustrativa?

O objetivo principal desse estudo é o desenvolvimento de um jogo educativo que ilustre a metodologia de desenvolvimento ágil de software *Extreme Programming* (XP), com a finalidade de dirimir ou eximir a complexidade de aprender essa metodologia por métodos convencionais de ensino. Especificamente, aplicar a metodologia de desenvolvimento ágil de software XP e o documento de *design* do jogo e implementar o jogo educativo que ilustre essa metodologia, com base no docmento de *design* estudado.

Sendo necessária a realização do Projeto Final de Curso, componente da Matriz Curricular do curso de Ciência da Computação, esse estudo atingirá, primariamente, esse escopo. Para a realização do mesmo, como auxílio utilizou-se os recursos específicos disponíveis na universidade, como os laboratórios de computação. Para desenvolver qualquer tipo de produto com qualidade, todo o processo de desenvolvimento deve ser guiado por alguma técnica. Sendo o software um importante produto, para produzí-lo com qualidade então deve-se seguir alguma técnica de desenvolvimento, nesse caso a XP. Devido ao demasiado tempo necessário para o aprendizado de novos assuntos, espera-se que, com a utilização do jogo educativo ilustrativo desenvolvido, o aprendizado do aluno em relação à metodologia de desenvolvimento XP possa ter a mesma ou uma melhor eficácia, contudo

com menor complexidade. Esse documento é organizado da seguinte maneira:

Capítulo 1. Apresenta as dificuldades, o objetivo e itens convenientes.

Capítulo 2. Discorre sobre o referencial teórico utilizado para a idealização da ferramenta.

Capítulo 3. Apresenta a metodologia utilizada no processo de desenvolvimento.

Capítulo 4. Mostra os resultados alcançados.

Capítulo 5. Conclusão da pesquisa.

2 REFERENCIAL TEÓRICO

2.1 EXTREME PROGRAMMING

A Extreme Programming (XP) é um processo ágil de desenvolvimento de software, ou seja, é um processo que admite a imprevisibilidade, a tratando com confiança e segurança seguindo o conceito de adaptabilidade incremental. O conceito de adaptabilidade incremental só pode ser concretizado se houver *feedback* constante por parte do cliente (PRESSMAN, 2011).

Alcançando esse objetivo, o processo será incremental. A cada incremento é produzido um artefato, que nesse caso é um protótipo funcional do sistema ou parte dele. Esses incrementos devem ocorrer rapidamente, de forma que acompanhe a imprevisibilidade do projeto (PRESSMAN, 2011). Um conjunto de cinco valores servem como base para todo o processo de desenvolvimento XP: comunicação, simplicidade, feedback, coragem e respeito. "Cada um desses valores é usado como um direcionador das atividades, ações e tarefas específicas da XP".

2.1.1 ATIVIDADES METODOLÓGICAS

Na atividade metodológica de planejamento, também conhecida como jogo do planejamento, o cliente lista os requisitos para a equipe. Então, a atividade tem como objetivo primário obter informações acerca do que será implementado (PRESSMAN, 2011). Nessa fase são desenvolvidas as histórias, explicadas em detalhe na subseção 2.1.5.6, que têm como intuito descrever uma funcionalidade que será implementada no software, a partir da descrição do cliente.

Trabalhando sempre em conjunto, os clientes fazem parte da equipe. Sempre eles auxiliam na escolha das histórias a serem implementadas no final de cada incremento, ciclo semanal, entre outras tarefas. No decorrer do projeto, o cliente pode inserir novas histórias, bem como alterar as existentes e até mesmo retirar, o que implica na reformulação dos planos por parte da equipe (PRESSMAN, 2011).

Durante o decorrer desse documento será explicitado que a XP sempre procura

realizar as tarefas da maneira mais simples possível, porém de forma funcional. Logo, o projeto é realizado de forma simples também, evitando qualquer tipo de complexidade. (PRESSMAN, 2011) continua discorrendo que o projeto é como um mapa, um guia para a equipe. Ele, dentre outras finalidades, guia a implementação para uma história à medida que essa é escrita, sem acrescentar nem reduzir funcionalidades. Um projeto XP pode ser elaborado antes ou até mesmo depois de ter iniciada a codificação, visto que a cada incremento esse projeto poderá e irá ser feito, caso não tenha sido e refeito sempre.

Codificação é a atividade metodológica onde, segundo (PRESSMAN, 2011), após todas as histórias terem sido escritas, a equipe passa então para a fase de testes, antes de iniciar a codificação propriamente dita. Esses testes ocorrem no formato de unidade, testando cada história individualmente, e no formato de integração, onde cada história é testada integrando no incremento atual. Nessa fase de codificação entra a prática de programação em par (subseção 2.1.5.5), que diminui a quantidade de falhas inseridas no código, aumentam as ideias e a qualidade do código escrito. Ao terminar a codificação, o trabalho é então integrado aos de outras equipes. A integração ocorre frequentemente.

Na atividade metodológica de testes são realizados, com mais frequência, os testes de integração e validação. Previamente, antes de realizar a codificação, os testes de unidade foram realizados com o intuito de evitar retrabalho. Outro teste muito importante é o teste de aceitação, onde o incremento atual é testado, verificando se esse atende as características propostas de forma satisfatória. Os testes de aceitação são obtidos de histórias de usuários implementadas como parte de uma versão de software (PRESSMAN, 2011).

2.1.2 VALORES

Todos aqueles que trabalham com desenvolvimento de software tem um senso sobre o que, de fato, importa. Uma pessoa pode pensar que, o que realmente importa, é pensar cuidadosamente sobre todas as decisões de *design* concebíveis antes de implementar. Outra pessoa pode pensar que o que realmente importa é não ter quaisquer restrições à sua liberdade pessoal (BECK, 2004). Como cada pessoa tem uma forma de pensar diferente, todos em uma equipe podem ter diferentes sensos sobre o que irá importar para ele, não refletindo sobre o que irá importar para a equipe. Dessa forma, a XP descreve cinco valores que orientam todo o processo de desenvolvimento de software.

2.1.2.1 COMUNICAÇÃO

A comunicação é um dos valores mais importantes para a execução de um bom projeto, visto que todas as atividades são desenvolvidas após a comunicação. A comunicação, dentre outros, tem como principais objetivos: informar aos desenvolvedores os requisitos do sistema, resolver possíveis problemas ou iminência dos mesmos (BECK, 2004). A XP preza sempre a possibilidade de um diálogo na forma presencial, uma vez que dessa forma é possível uma maior compreensão das partes. Em um diálogo presencial, é possível resolver as dúvidas, problemas, possíveis ambiguidades em requisitos, entre outros assuntos, de forma mais positiva. Dessa forma também é possível absorver mais conhecimento e, portanto, entender da melhor forma possível o que o cliente ou desenvolvedor está dizendo. Entretanto, o autor continua discorrendo que, por mais que seja um dos principais valores da XP, não é garantido que o seguí-lo somente, o projeto terá alta qualidade. A XP tem como base todo o conjunto de valores, não deixando qualquer um de lado.

2.1.2.2 SIMPLICIDADE

Simplicidade diz respeito a concentrar o esforço apenas na realização do que é necessário e essencial. Os desenvolvedores, juntamente com os clientes, devem realizar as tarefas que lhe foram atribuídas de forma simples e funcional, entretanto, somente o que é preciso para hoje. O que é necessário para o amanhã não deve, a princípio, ser realizado hoje, visto que demandaria gastos financeiros, temporais e recursos humanos sem haver a real necessidade. Além de não realizar trabalho sem ser necessário, a simplicidade também implica em um sistema mais fácil para evitar ou corrigir problemas, caso surgir a eventualidade. Portanto, segundo (BECK, 2004), os envolvidos em projeto XP devem sempre se atentar em "manter as coisas simples", pois isso evita problemas e gastos desnecessários.

2.1.2.3 *FEEDBACK*

O *feedback* está ligado a uma boa comunicação, já que ele deve ser realizado com frequência e de forma correta, passando o máximo de conhecimento possível. Executado de forma recíproca, os desenvolvedores atendem o mais rápido possível as requisições dos usuários. Os usuários, por sua vez, ficam próximos dos desenvolvedores, emitindo informações, dúvidas que possam ter durante o desenvolvimento. Essencial durante todo

o decorrer do processo de desenvolvimento, o *feedback*, emitido por qualquer dos lados, desenvolvedor ou usuário, pode evitar ou descobrir eventuais problemas, erros no sistema que está sendo desenvolvido. Se o *feedback* for realizado de forma constante, a chance de encontrar eventuais erros aumenta, o que proporciona uma rápida correção, que por sua vez implica em reduzir os danos que esse possível problema poderia causar em todo o sistema. O *feedback* pode ser realizado de diferente formas: opiniões sobre as ideias do grupo; como o código pode parecer quando implementado alguma ideia; se os testes são fáceis de escrever; se os testes funcionam e, dentre outros, como a ideia funciona desde que foi implantada (BECK, 2004).

2.1.2.4 CORAGEM

Como (BECK, 2004) discorre:

> Às vezes a coragem se manifesta como um viés para a ação. Se você sabe qual é o problema, faça algo sobre isso. Às vezes a coragem se manifesta como a paciência. Se você sabe que há um problema, mas não sabe o que é, é preciso coragem para esperar o verdadeiro problema surgir distintamente.

Dentre inúmeros problemas que podem ocorrer durante o processo de desenvolvimento, dois dos mais críticos são: a mudança de requisitos e a necessidade de alterações grandes no sistema. Sendo assim, para a equipe XP é conhecido que, caso for necessário no meio do processo realizar alguma a alteração, existe um risco de essa deixar inoperante partes já concluídas do sistema. Entretanto, esse risco é enfrentado com tranquilidade, consciência e confiança, uma vez que existe uma confiabilidade da eficácia das práticas XP.

2.1.2.5 RESPEITO

O último, mas não menos importante valor é o respeito. O respeito deve partir de todos os lados, do desenvolvedor ou cliente, todos devem se respeitar de forma mútua. Na filosofia XP o cliente também é parte da equipe, portanto deve haver respeito para com ele e partindo dele. Entretanto, isso não implica que, caso não fosse da equipe, poderiam faltar respeito para com ele ou ele faltar com respeito com o restante da equipe. O conceito de respeitabilidade é bem amplo, mas pode ser resumido em saber trabalhar em equipe. Em hipótese alguma trabalhamos sozinhos, e na filosofia XP isso é ainda mais forte. Trabalhar em equipe é saber o momento de dar alguma opinião, mas também saber ouvir

as opiniões, possivelmente divergentes. Deve haver compreensão em toda a equipe, dessa forma o conceito de respeitabilidade será plenamente implantado na equipe. (BECK, 2004) continua: “Se os membros da equipe não ligam para os outros e o que estão fazendo, a XP não funciona”.

2.1.3 PRINCÍPIOS

Os princípios da XP servem como apoio para os valores, que por si só são muito abstratos. (BECK, 2004) exemplifica o princípio do humanismo, que sugere que a conversa satisfaz a necessidade humana de conexão. Conversas de forma presencial são as prioritárias numa equipe XP, visto que os dois lados podem oferecer feedback instantâneo, discussões positivas e que possam esclarecer qualquer tipo de dúvida. São vários os princípios que podem nortear uma equipe, discorridos nas subseções seguintes.

2.1.3.1 HUMANISMO

O princípio do humanismo referencia que durante todo o processo de desenvolvimento de software é necessário que haja uma completa respeitabilidade no atendimento das necessidades humanas. Além disso, deve-se reconhecer a fragilidade e talento de cada indivíduo, a fim de haver respeito, potencializando o mesmo sempre que possível. Dessa forma, atendendo a todos esses requisitos, a equipe será integrada de bons indivíduos. Algumas das necessidades humanas, citadas por (BECK, 2004), são:

- **Segurança básica:** ser livre da fome, não sofrer danos físicos e/ou psicológicos e não haver ameaças a ente queridos;
- **Realização:** a oportunidade e a capacidade de contribuir com a sociedade;
- **Participação:** a capacidade de identificar um grupo e contribuir para seus objetivos comuns;
- **Crescimento:** a oportunidade de expandir as habilidades e perspectivas;
- **Intimidade** a capacidade de compreender e ser compreendido profundamente pelos outros.

Além dessas, existem outras necessidades humanas que não precisam, necessariamente, ser atendidas no ambiente de trabalho. Outro ponto a ser considerado durante todo

o processo de desenvolvimento de software, é o fato de saber equilibrar as necessidades individuais e as necessidades da equipe.

2.1.3.2 ECONOMIA

Praticamente toda atividade na qual se investe tempo e dinheiro, é esperado que essa ofereça retorno o mais breve possível. Sendo assim, as práticas da XP trabalham para fazer com que o retorno venha mais rápido e os gastos sejam adiados. Uma boa ação nesse sentido, segundo (BECK, 2004), é sempre resolver a atividade que necessita de mais prioridade no momento, maximizando assim o valor do projeto. Considerando a natureza da XP, onde seu desenvolvimento ocorre de maneira incremental, esse estilo de projeto contribui diretamente para o princípio da economia, pois a cada incremento tem-se um produto parcialmente concluído - não inclui todas as funcionalidades do produto final, entretanto é funcional. Com um produto parcialmente concluído sendo entregue a cada incremento, o retorno pode vir muito antes que entregando somente o produto final, como acontece em outros estilos de projetos.

2.1.3.3 BENEFÍCIO MÚTUO

Benefício mútuo diz respeito a todas as práticas que norteiam a XP, todo o processo de desenvolvimento do software poder beneficiar todos os envolvidos, sejam desenvolvedores, gerentes, clientes, entre outros. (BECK, 2004) descreve como a XP resolve, de forma mutuamente benéfica, o problema de comunicação com o futuro:

- Escrever testes automatizados que possam ajudar a projetar e implementar melhor hoje. Deixar esses testes para futuros programadores usarem também;
- Refatorar de forma cuidadosa, com o intuito de remover complexidade acidental, dando satisfação, menos defeitos e tornando o código mais fácil de entender para o futuro;
- Escolher nome de um conjunto coerente e explícito de metáforas que acelere o desenvolvimento e torne o código mais claro para novos programadores.

Deve-se levar em consideração práticas que beneficiam a todos e, além disso, beneficiem o hoje e o amanhã. Existem inúmeras práticas que aliviam os problemas atuais,

mas não se importam com o amanhã, comprometendo o desenvolvimento como um todo.

2.1.3.4 AUTO SEMELHANÇA

Auto semelhança implica na reutilização de soluções previamente encontradas em outros contextos do desenvolvimento, mesmo em escalas diferentes. No desenvolvimento XP, o ritmo básico é, segundo (BECK, 2004):

- Escrever um teste que falha e então fazê-lo funcionar;
- Em um trimestre, listar os temas a abordar e, em seguida, tratá-los como histórias;
- Em uma semana, listar as histórias a abordar, escrever testes que expressem essas histórias, fazendo-os funcionar;
- Em poucas horas, listar os testes que precisam ser escritos. Escrever um teste, fazê-lo funcionar. Escrever outro teste e torná-lo funcional. Repetir até a lista estar completa.

2.1.3.5 MELHORIA

(BECK, 2004) explicita que o princípio da melhoria implica que o software em desenvolvimento deve ser melhorado de forma constante. Isso implica em deixar de tentar criar um software perfeito, mas criar o melhor possível hoje para que amanhã ele possa ser melhorado. Para que essa melhora constante ocorra durante todo o processo de desenvolvimento do software, além da natureza do projeto da XP ser incremental, o que contribui diretamente para esse princípio, deve haver também de forma constante: feedback rápido de todos os indivíduos da equipe (desenvolvedores, clientes, entre outros) e uma boa comunicação com todas as partes envolvidas no projeto. É importante a equipe ter consciência que a execução do projeto nunca será totalmente fiel a ele, contudo pode-se chegar muito próximo, utilizando sempre de melhora contínua.

2.1.3.6 DIVERSIDADE

Diversidade sugere que dentro da equipe deve haver indivíduos que possam ter diferentes pontos de vistas sobre resoluções de problemas, ideias e soluções a serem implementadas e que possam ter diferentes habilidades. Sem essas características, (BECK,

2004) diz que a equipe não será eficaz. Nesse princípio, o fato de existir duas ou mais opiniões não é um problema, mas sim uma oportunidade, onde todas as opiniões devem ser consideradas e principalmente respeitadas pelos envolvidos, convergindo então à uma solução.

2.1.3.7 REFLEXÃO

Como o próprio nome sugere, o princípio de reflexão alerta para que haja reflexão na equipe, isto é, fazer o trabalho como deve ser feito e pensar sobre e porque está fazendo tal trabalho. Além disso, (BECK, 2004) diz que não importa se houve êxito ou falha, deve haver uma reflexão sobre, não escondendo nada da equipe. Ao realizar a reflexão, além das informações que podem ser extraídas devido à característica intelectual, pode-se extrair outras informações tentando avaliar os sentimentos, as emoções sentidas ao passar por determinadas situações, sejam elas positivas e principalmente as negativas, que podem indicar problemas. Contudo, a reflexão só deve acontecer depois de ter feito algo, de ter realizado alguma tarefa, não deixando de agir para refletir. A reflexão, na XP, é misturada a ação para maximizar o *feedback*.

2.1.3.8 FLUIDEZ

(BECK, 2004) continua explicitando que as práticas utilizadas pela XP convergem para um fluxo contínuo de atividades, diferentes de outros projetos onde as atividades são distintas. Além disso, essas atividades são realizadas em pequenos lotes que, a cada incremento, é melhorado. Trabalhar com grandes lotes não é muito eficaz, visto que a implantação, a integração e o *feedback* ocorreriam com uma frequência reduzida. Por ser de natureza incremental, todo o processo de desenvolvimento de software pode, devido às ocasionais mudanças de projeto, voltar e seguir de maneira fácil, ou seja, flui de maneira fácil se comparado a outros tipos de processos de desenvolvimentos.

2.1.3.9 OPORTUNIDADE

O princípio da oportunidade ensina que mesmo quando ocorrer uma falha ou surgir algum tipo de problema durante o processo de desenvolvimento, é necessário ver como uma maneira de aprendizado. (BECK, 2004) frisa que é necessário que ocorram problemas para que se possa melhorar cada vez mais. Muitas práticas da XP convertem os problemas em

oportunidades, o autor cita dois problemas que podem ser convertidos em oportunidades utilizando uma prática da XP: se não for possível fazer planos precisos a longo prazo, então utilizando a prática de ciclo trimestral pode-se refinar os planos de prazo; e se existir algum indivíduo da equipe que produz muitos erros, então utilizando a prática de programação em par os erros devem diminuir ou deixar de existir. Não somente no processo de desenvolvimento de software, mas como em qualquer outro tipo de processos de desenvolvimento, vai surgir algum tipo de problema ou erro. Isso faz com que o processo seja melhorado cada vez mais. Mas em todos os tipos de processos deve existir uma visão consciente para transformar o problema em uma oportunidade de aprendizado.

2.1.3.10 REDUNDÂNCIA

Redundância é um princípio que implica na resolução de problemas antes dele se tornar catastrófico, comprometendo a confiança da equipe, dentre outros fatores negativos. Dessa forma, o custo de aplicar a redundância é menor, se comparado ao custo que teria ao tentar resolver o problema que se tornou catastrófico. (BECK, 2004) discorre que a redundância na XP pode ser aplicada com as próprias práticas. Como uma prática apenas não detecta uma falha ou um problema, ao se aplicar diversas práticas para vários contextos, algumas delas acabam se tornando redundantes, resolvendo o mesmo problema. Por exemplo, a programação em par e sentar-se junto podem achar ou resolver o mesmo problema. Como a redundância consome recursos humanos e financeiros, apesar de ser menos custosa que uma falha catastrófica, ela pode ser eliminada depois de ter a completa confiança, na prática, que não surgirá qualquer falha. Essa confiança só é alcançada depois de vários testes, várias implantações consecutivas sem apresentar falhas.

2.1.3.11 FALHA

Ao se deparar com algum tipo de dúvida, é permitida a falha, porque essa propicia um aumento do conhecimento sobre a causa, o que leva ao possível aprendizado da solução para o problema. (BECK, 2004) cita como exemplo alguns *designers* que tinham sempre duas ou mais soluções para um problema. Eles se sentavam e discutiam sobre as soluções, as ideias a fim de chegar numa solução final. Nesse tempo gasto, eles poderiam ter implementado duas vezes cada umas das ideias e veriam se era ou não uma solução. Mas o medo de falhar, de desperdiçar tempo de programação os impediam.

2.1.3.12 QUALIDADE

Ainda segundo (BECK, 2004), produzir software de altíssima qualidade causa mais satisfação, pois aumenta a confiança e o orgulho da equipe. Além desses fatores, existe o fator econômico: um software de alta qualidade gera muito mais lucro que um de qualidade inferior. Não saber o nível de qualidade desejado não significa que deve ficar sem trabalhar, mas como todos os outros princípios, deve-se trabalhar da melhor maneira possível, produzindo assim um produto com a melhor qualidade possível para aquele momento. No futuro, no próximo incremento, na próxima semana ou ciclo trimestral, esse produto poderá ter sua qualidade elevada.

2.1.3.13 PASSOS DE BEBÊ

Ao se deparar com alguma falha ou simplesmente a necessidade de melhorar a qualidade do que já está parcialmente pronto, muitas vezes são realizadas grandes mudanças, até mesmo pelo tempo disponível, que pode ser curto. Isso é considerado, na XP, um erro (BECK, 2004). Grandes mudanças podem gerar muitas falhas, o que dificultaria ainda mais a situação. Portanto, as mudanças devem ser feitas como o nome do princípio sugere, lentamente, uma mudança por vez. Dessa forma gera poucas falhas, que podem ser corrigidas de maneira rápida, tornando assim todo o processo seguro. Não esquecendo de sempre realizar testes, e é muito mais fácil realizar um teste em pequenos blocos de mudanças do que em grandes blocos.

2.1.3.14 RESPONSABILIDADE ACEITA

Responsabilidade aceita sugere que as responsabilidades podem somente ser aceitas, não atribuídas. Somente o indivíduo da equipe, ao receber uma atribuição de responsabilidade, é capaz de decidir se é de fato responsável ou não. Na XP, as práticas refletem responsabilidade aceita, ou seja, como (BECK, 2004) diz “a pessoa responsável por implementar uma história também é responsável pelo design, implementação e teste da mesma”.

2.1.4 PAPÉIS

Para que o desenvolvimento de software ocorra de forma eficaz, é necessária uma variedade de perspectivas, funções dentro da equipe. Os indivíduos que ocuparão essas

funções devem trabalhar em conjunto, realizando assim um projeto mais eficaz. (BECK, 2004) cita as atribuições de cada indivíduo que ocupa uma função dentro da equipe XP:

• **Testador:** Com a ajuda do cliente, elabora testes automatizados para as histórias previamente escritas, antes delas serem implementadas. No decorrer da implementação, auxilia os desenvolvedores a encontrar falhas e automatizar testes.

• **Designer de Interação:** Trabalha com o cliente, ajudando a escrever e esclarecer histórias. Além disso, analisa o uso real do sistema, com a finalidade de decidir o que o sistema precisa fazer em sequência.

• **Arquiteto:** Executa refatorações de grande escala, sempre seguindo o princípio passos de bebê. Escreve testes a nível de sistema que enfatizam a arquitetura e implementam histórias.

• **Gerente de projeto:** Facilita a comunicação dentro da equipe e fora dela, coordenando a comunicação com clientes, fornecedores, entre outros. Outra tarefa do gerente de projeto é manter os planos sincronizados com a realidade.

• **Gerente de Produto:** Escreve histórias, escolhem temas no ciclo trimestral e histórias no ciclo semanal. Ao adaptar a história ou tema para o que realmente está acontecendo, identificando as necessidades reais, o gerente de produto ajuda a equipe a decidir prioridades.

• **Executivo:** Fornece coragem, confiança e responsabilidade para toda a equipe. Articula e mantém as metas de grande escala e estimula a equipe a realizar melhorias.

• **Escritor técnico:** Escreve a documentação do projeto, de forma que suas publicações técnicas fornecem feedback para toda a equipe.

• **Usuário:** É o que fornecesse os requisitos para a equipe, quem está requerendo o produto. Ajuda a escrever as histórias, selecioná-las e tomar decisões durante o processo de desenvolvimento.

• **Programador:** Estima histórias e tarefas, quebra as histórias em tarefas, escreve testes, escreve código que implementa um recurso, automatiza um processo de desenvolvimento tedioso e melhora gradualmente a concepção do sistema.

2.1.5 PRÁTICAS

Segundo (BECK, 2004), uma equipe que trabalha com a XP irá realizar as práticas todos os dias de trabalho, tendo sempre como principal meta o benefício mútuo. Nem todas as práticas descritas nas próximas subseções são usadas sempre e simultaneamente, visto que elas devem ser adaptadas e utilizadas de acordo com o contexto atual de desenvolvimento do projeto. Entretanto, os valores nunca mudam, já os princípios podem mudar dependendo do domínio. As práticas devem ser escolhidas pela equipe, ser testadas diariamente com a finalidade de haver perfeita adequação implicando na melhoria do desenvolvimento de software.

2.1.5.1 SENTAR-SE JUNTO

A prática de sentar-se junto, como o nome sugere, diz respeito a todos os membros da equipe trabalharem de forma reunida no ambiente de trabalho, sentando próximos uns dos outros. Para isso, o espaço físico deverá ser suficiente para acomodar a todos nessas condições. Dessa forma, a comunicação será muito mais rápida e efetiva, visto que todos estarão localizados muito próximos. Contudo, não significa necessariamente que todos os integrantes da equipe ficarão um de frente para o outro, mas poderão ficar, por exemplo, em pequenos espaços que possam respeitar sua privacidade e seu espaço, mas que ainda assim facilitam a comunicação entre a equipe. Outra forma que (BECK, 2004) cita de respeitar a privacidade da equipe é limitando as horas de trabalho, assim a equipe pode ter sua privacidade em outros locais de sua preferência.

2.1.5.2 EQUIPE INTEGRAL

A equipe deve ser composta por indivíduos variados, que tenham habilidades e perspectivas diferentes, isso é o que define a prática de equipe integral. Dessa forma, a equipe terá a qualquer momento todos os recursos que precisar, aumentando a probabilidade de sucesso do projeto realizado. Sendo um dos intuitos a economia de recursos, (BECK, 2004) diz que a equipe deve ser dinâmica, do ponto de vista de ter alguém na equipe somente se for realmente necessário. Quando surgir a necessidade de um recurso e a equipe não puder disponibilizá-lo, deve-se então encontrar um indivíduo que atenda esse recurso e integrá-lo na equipe. Quando a necessidade desse recurso acabar, o indivíduo que passou a integrar a equipe não é mais necessário, podendo ser deslocado para outra função.

2.1.5.3 AMBIENTE INFORMATIVO

O ambiente de trabalho da equipe deve refletir todo o projeto, o trabalho que está sendo feito no momento. Assim, além dos indivíduos pertencentes à equipe, qualquer um que chegar no ambiente e observar por alguns instantes, pode estar inteiramente atualizado sobre o andamento do projeto (BECK, 2004). Parte da implementação dessa prática, em grande maioria, ocorre colocando os cartões contendo as histórias em uma parede, organizadas por seções que indicam as histórias já concluídas, as que estão sendo implementadas, as que estão sendo analisadas, as que irão ser implementadas, entre outras categorias convenientes para a realização do projeto.

2.1.5.4 TRABALHO ENERGIZADO

Trabalho energizado diz respeito a trabalhar dentro do limite de horas e com carga suficiente que não esgotará o indivíduo ou toda a equipe. (BECK, 2004) descreve que, caso a equipe se esgote de tanto trabalho, o rendimento poderá ficar comprometido, refletindo diretamente na qualidade do projeto. Após longas horas de trabalho sem descanso, a equipe não conseguirá produzir com a mesma qualidade como se estivesse trabalhando poucas horas e com intervalos para descanso. Então deve-se respeitar o limite físico da equipe, para ter mais produtividade.

2.1.5.5 PROGRAMAÇÃO EM PAR

Programação em par é a prática onde a escrita de todos os códigos é realizada com dois indivíduos da equipe, sentados lado a lado (SCHACH, 2009; SOMMERVILLE, 2007). O ambiente deve ser propício para acomodá-los de forma que respeite a privacidade de ambos. (BECK, 2004) ainda cita o que programadores em par fazem:

- Se mantêm mutuamente nas tarefas;
- Fazem refinamentos para o sistema;
- Esclarecem as ideias;
- Tomam iniciativas quando o parceiro está preso, diminuindo assim a frustração;
- Se responsabilizam mutuamente pelas práticas da equipe.

Programar em par não implica em ter ideias sempre em par, pode-se pensar em algumas ideias individualmente, depois discuti-las com o par. Implementar até mesmo o protótipo, mas não muito além disso, pois deve haver sempre uma ação com o par, afim de ter uma maior rapidez e diminuição da ocorrência de erros. Para que a programação em par funcione de forma adequada e produtiva, deve haver respeitabilidade do espaço individual de cada indivíduo da dupla. Também deve-se respeitar as diferenças de cada pessoa.

2.1.5.6 HISTÓRIAS

As histórias, segundo (BECK, 2004), são descrições de uma funcionalidade do sistema. Nelas são detalhadas, em breve frases e até mesmo imagens, o que deverá ser implementado. Geralmente são colocadas em uma parede em que o fluxo de movimentação de pessoas é constante, afim de começar a estimá-las desde cedo. Estimar as histórias desde cedo implica em obter o maior retorno com o menor investimento possível. Deve-se conhecer vários detalhes para poder tomar uma decisão, por isso quanto mais cedo começar a ser estimada, menor será o investimento, implicando em maior retorno.

2.1.5.7 CICLO SEMANAL

A prática de ciclo semanal consiste em planejar o trabalho uma semana por vez, evitando assim trabalhar com grandes quantidades de dados, o que aumenta a probabilidade de planejar algo errado. No início de cada semana deve haver uma reunião, (BECK, 2004) cita que nessa reunião deve acontecer:

1. Análise dos progressos alcançados até a data, incluindo como o progresso real combinou com o progresso esperado;

2. Os clientes devem pegar as histórias a serem implementadas nessa semana;

3. Quebrar as histórias em tarefas. Os membros da equipe se habilitam para as tarefas e as estimam.

Após a reunião, a semana deve ser iniciada com o trabalho focado nos testes automatizados que serão executados no término das tarefas dessa semana. Durante a semana as tarefas são implementadas e, por fim, testadas. Ao fim da semana é esperado que

as histórias estejam completamente funcionais, obtendo um progresso e então reiniciando esse ciclo.

2.1.5.8 CICLO TRIMESTRAL

Assim como o ciclo semanal, na prática do ciclo trimestral deve haver uma reunião, uma reflexão, um planejamento, porém nesse caso a cada trimestre. Nesse planejamento deve ser discutido:

- Identificação de gargalos;
- Iniciar os reparos;
- Planejar o tema ou temas para o semestre.

Nesse lapso temporal, é confortável que haja as interações externas, com fornecedores, além de reflexões da equipe e encontrar possíveis bloqueios do projeto (BECK, 2004).

2.1.5.9 *BUILD* DE DEZ MINUTOS

A compilação (*build*) de todo o sistema e a execução dos testes devem durar no máximo dez minutos. Caso a compilação demore mais que dez minutos, pode haver uma perda de *feedback*. Em três passos, (BECK, 2004) diz como compilar e executar os testes em dez minutos:

1. Se o processo não for automatizado, deve-se então automatizá-lo;

2. Compilar somente a parte que foi alterada;

3. Executar apenas os testes que cobrem a parte do sistema em risco por causa das alterações realizadas.

Compilações automáticas são mais vantajosas se comparada as que precisam de qualquer intervenção manual, devido ao fato de inevitavelmente todo o processo ficar cansativo, fazendo com que as compilações não automáticas possam ser feitas com menos frequência, resultando em mais erros e cansaço.

2.1.5.10 INTEGRAÇÃO CONTÍNUA

A integração deve ser realizada de forma contínua, pois leva mais tempo para integrar que programar. Se houver demora para que ocorra as integrações, maior será o custo quando ela ocorrer. Se demorar a ocorrer, o custo não poderá ser previsível, o que será péssimo. O estilo mais comum de integração contínua é a assíncrona. Um indivíduo checa as alterações que ele fez. Logo em seguida, o sistema de compilação percebe a mudança e começa a construir e testar. Se houver problemas, o indivíduo será notificado por e-mail, mensagem de texto, ou outro meio de comunicação (BECK, 2004; SOMMERVILLE, 2007).

2.1.5.11 PROGRAMAÇÃO "TESTE-PRIMEIRO"

Antes de desenvolver qualquer linha de código, é necessário primeiro desenvolver um teste automatizado. Nesse estilo de programação existe alguns problemas como: aumento do escopo, acoplamento e coesão. Entretanto é mais fácil encontrar uma falha, visto que é conhecido quando o teste deve ou não falhar. O problema do aumento do escopo se dá ao saber, de forma explícita e objetiva, o que o programa irá fazer. Dessa forma, pode-se cometer o erro de ter o foco diretamente à codificação. O problema pode ser contornado, caso o código for realmente necessário ser implantado, escrevendo outro teste depois de o ter implantado. Um código fracamente acoplado e altamente coeso será muito mais fácil de testar. Se o teste não ocorre de maneira fácil, ou não der para escrever um teste, então existe algum problema de projeto (BECK, 2004; SOMMERVILLE, 2007).

2.1.5.12 *DESIGN* INCREMENTAL

A prática de *design* incremental consiste em concentrar todos os esforços para priorizar as soluções, as criando da forma mais simples e funcional possível para aquele dia. Se existem problemas que não precisam ser solucionados para hoje, então não deve ser gasto recursos e tempo nele hoje. Como intuito de não elevar de forma significativa o custo do projeto, uma equipe XP tem como auxílio os testes automatizados, a melhoria contínua, entre outras práticas e princípios (BECK, 2004).

2.2 JOGOS ELETRÔNICOS

No início da ascensão dos jogos eletrônicos, os jogos do gênero educativo eram feitos, exclusivamente, para o público infantil. Com o passar dos anos, mas ainda não muito difundido, o público-alvo desses jogos começou a mudar, alcançando os adultos também, em universidades, instituições de pesquisas, entre outros locais. Os jogos educativos têm o intuito de permitir que o jogador possa adquirir conhecimentos do mundo real e aplicá-los. (NOVAK, 2012), continua salientando que além dessa definição tradicional para jogos educativos, existem outras definições ou formas de aprendizado que atuam sobre a maioria dos jogos. Recentemente, alguns jogos que não são especificamente do gênero educativo passaram a realizar essa função, devido as informações do mundo real contidas neles que, ao serem sintetizadas pelo jogador, podem ser aplicadas de forma conveniente.

2.2.1 PROCESSO DE DESENVOLVIMENTO

Para obter qualidade final em produto, esse deve ter seu processo de construção organizado. Sendo o jogo um produto, essa prática não costuma ser simples, principalmente quando se fala de um jogo educativo. O processo de desenvolvimento de jogos, assim como todos os processos monitorados, segue um conjunto de fases. Esse capítulo discorre sobre quais são e o que é realizado nas fases desse processo (NOVAK, 2012; MARTINS, G. et al, 2015; NETO, J. C. et al, 2015).

2.2.1.1 CONCEITO

A fase de conceito é a primeira do desenvolvimento de um jogo. Ela inicia quando surge alguma ideia para realizar um jogo, terminando quando as tomadas de decisões para o início do planejamento começarem a ser tomadas. O objetivo dessa fase é transformar a ideia em formato escrito, com a finalidade de todos entenderem do que se trata o jogo. Além disso, deve-se decidir com precisão sobre o que o jogo é. Como não existem muitas atividades nessa fase, a equipe é formada por poucos indivíduos, um de cada área. No final dessa etapa, é produzido como artefato um documento de conceito, que tem como principal finalidade transmitir o objetivo e o propósito do jogo previamente proposto. Esse documento é curto, contendo no máximo cinco páginas, e feito em uma semana (NOVAK, 2012).

2.2.1.2 PRÉ-PRODUÇÃO

Na fase de pré-produção é realizado todo o projeto do jogo. Estando concluído o documento de conceito, esse serve como base para a formação de dois novos documentos do final dessa fase: o "*Game Design Document*" (GDD) e o "*Technical Design Document*" (TDD). O GDD, contendo entre cinquenta a duzentas páginas, tem como objetivo guiar todo o processo de desenvolvimento do jogo, uma vez que todos os aspectos do jogo estarão completamente detalhados nele: enredo, jogabilidade, personagens, interfaces, entre outros aspectos. (NOVAK, 2012) diz que "O GDD deve especificar as regras de jogar o jogo em detalhes o suficiente para que você possa, em teoria, jogar o jogo sem o uso de um computador". Já o TDD, que é feito com base no GDD, descreve os detalhes técnicos do desenvolvimento do jogo, como por exemplo em qual motor ele irá ser desenvolvido, sendo comparado com outros motores que existem a disponibilidade, justificando a escolha. Outros objetivos do TDD são: especificar como o conceito irá ser transformado em software; quais integrantes da equipe serão designados para quais tarefas; quanto tempo irá demorar para concluir as tarefas; quais as ferramentas centrais serão utilizadas; e qual o hardware e software a ser adquirido.

2.2.1.3 PROTÓTIPO

Quando o processo de desenvolvimento do jogo chega na fase de protótipo, o objetivo principal é criar um protótipo funcional do jogo que está sendo desenvolvido. (NOVAK, 2012) explicita que a definição usual de protótipo dentro da indústria de games é "Um pedaço de trabalho do software que captura na tela a essência do que faz o jogo especial, isto é, o que diferencia do resto, e o que vai torná-lo bem-sucedido". Levando essa definição em consideração, um dos cuidados a ser tomado é o de não sofisticar o protótipo, acrescentando funcionalidades desnecessárias para o momento, melhorando características visuais, visto que isso desvia o foco de montar uma base sólida para o jogo.

2.2.1.4 PRODUÇÃO

A fase de produção é onde, de fato, o jogo é feito, idealizado. Por esse motivo é a fase mais longa, podendo durar entre seis meses a dois anos, sendo o resultado dessa fase o jogo completo. Nessa etapa um dos desastres que podem acontecer é ter um ciclo temporal de desenvolvimento muito curto, como por exemplo menos que seis meses, não dando

tempo para o término das atividades em tempo hábil, o que acarreta uma sobrecarga de trabalho. Tendo sobrecarga de trabalho, os integrantes da equipe poderão até sair do projeto, o prejudicando ainda mais. Todo o controle de tempo, cumprimento de prazos o cronograma de atividades e também o cumprimento com o orçamento previsto fica ao encargo dos gestores do projeto (NOVAK, 2012).

2.2.1.5 ALFA

Na fase alfa, segundo (NOVAK, 2012), o jogo deve estar completamente jogável, ou seja, toda a interface com o usuário e o motor devem estar totalmente funcionais. Os elementos gráficos ainda podem não ser os finais. Durante essa fase, os testes conduzidos devem assegurar que cada módulo do jogo foi testado pelo menos uma vez; criar um banco de dados de bugs e registrar os erros e os resultados de desempenho. O jogo na fase alfa é testado por pessoas fora da equipe, portanto ele deve conter os seguintes elementos:

- Um caminho de jogabilidade (início ao fim).
- Textos no idioma primário.
- Interface básica com documentação preliminar.
- Compatibilidade com configurações de hardware e software especificados.
- Requisitos mínimos do sistema testados.
- A maioria das interfaces manuais testadas.
- Projeto do manual do jogo.

2.2.1.6 BETA

Quando o jogo atende todos os requisitos da fase alfa, ele é então passado para a fase beta, onde o foco é a correção de erros. O objetivo, além de corrigir o máximo de erros possíveis, é alcançar a estabilização do projeto. O jogo é liberado por um determinado tempo para que outras pessoas fora da equipe testem, tendo assim um feedback real do que acontecerá quando o jogo estiver concluído. Entre outros objetivos, considerando os já citados, são: isolamento de todos os erros significativos e problemas de desempenho; teste completo, correção de *bugs* e problemas de desempenho; e por fim o teste em todas

as plataformas a qual foi projetado. Para sair da fase beta e avançar para a fase *gold*, (NOVAK, 2012) cita que o jogo deverá conter, de forma completa:

- Código;
- Conteúdo;
- Caminho de navegação;
- Interface de usuário;
- Compatibilidade de hardware e software;
- Compatibilidade de interface manual;
- Arte e áudio;
- Manual do jogo.

2.2.1.7 *GOLD*

Ao chegar na fase *gold* o jogo está pronto, visto que a alta administração reviu o banco de dados de *bugs* e erros feito anteriormente e considerou que o jogo poderá ser gravado em discos. Essa etapa de gravação de discos em massa leva algumas semanas, visto que são muitas unidades a serem produzidas (NOVAK, 2012).

2.2.1.8 PÓS-PRODUÇÃO

Nessa fase podem ser lançadas novas versões do jogo ou atualizações que aumentam sua longevidade. Ainda, eventualmente pode ter passado algum erro despercebido, então nessa fase são lançados possíveis pacotes de correção para esses erros ou melhorias de desempenho gerais ou em plataformas ou hardwares isolados (NOVAK, 2012).

3 METODOLOGIA

Nesse capítulo será descrito o processo de desenvolvimento dessa pesquisa, sendo explicitado qual o método de abordagem e de procedimento utilizado. Segundo (PEIXOTO, D. C. C. et al, 2015), um jogo utilizado como ferramenta auxiliadora no processo de aprendizado pode auxiliar o jogador a obter um melhor desempenho. Então, ao fim dessa pesquisa é esperado que a ferramenta possa ser utilizada como auxílio nesse processo, servindo a princípio para ilustrar os conteúdos teóricos previamente apresentados. Sendo assim, o método de abordagem utilizado foi o hipotético-dedutivo.

Como método de procedimento, foi utilizado o experimental, uma vez que a pesquisa se moldou com o intuito de desenvolver o jogo como ferramenta auxiliadora no processo de aprendizado, buscando atingir o resultado explicitado. O desenvolvimento do jogo foi baseado em um *Game Design Document* (GDD) feito exclusivamente para servir como base para a implementação do jogo.

3.1 FERRAMENTAS

Pensando na alta usabilidade dos dispositivos móveis, o jogo desenvolvido foi para uma das plataformas presentes nesses dispositivos, a plataforma Android. Se comparada a outras plataformas disponíveis atualmente no mercado, a escolhida apresenta maior facilidade na aquisição de dispositivos para realizar a implementação e testes da ferramenta. Além desse fator condicional para a escolha, outro fato notório é em relação aos custos e complexidade para a publicação do jogo na loja de aplicativos da plataforma. De acordo com análises realizadas, o custo e a complexidade para publicar um aplicativo na loja de aplicativos da plataforma Android é consideravelmente menor em relação as outras plataformas (SOLVUS, 2016). Todavia, a publicação do jogo na loja de aplicativos Android (Google Play) será analisada e, se considerada possível, realizada em um futuro.

Atendendo a esse requisito, desenvolver o jogo para a plataforma Android, o motor de jogo (*engine*) escolhido foi o Unity, na versão 5.3.3f1. Além de atender a esse requisito, a *engine* escolhida possui facilidade na manuseabilidade de suas características, possui diversas características já implementadas, contém uma loja interna onde é possível adquirir

objetos convenientes de forma gratuita, documentação oficial com vídeos explicativos, diversos fóruns entre outras facilidades que auxiliam a utilização dessa engine. Ela pode ser adquirida de quatro diferentes formas, as quais oferecem recursos diferenciados (UNITY, 2016):

- **Personal:** Possui as características necessárias para iniciantes começarem utilizar a ferramenta. Essa versão é gratuita, porém só pode ser usada se a receita anual, ganha com o jogo produzido na ferramenta, for menor ou igual a US $100 mil.

- **Plus:** Possui todas as características da versão Personal mais as necessárias para criadores sérios que querem trazer sua visão para a vida. Essa versão custa 32 euros por mês, contudo seu uso é limitado a quem receber uma receita anual, ganha com o jogo produzido na ferramenta, for menor ou igual a US $200 mil.

- **Pro:** Seguindo o mesmo princípio, possui todas as características da versão Plus mais as necessárias para criadores profissionais que desejam uma completa customização e flexibilidade. Essa versão custa 115 euros por mês e seu uso é livre, não importando a receita recebida a partir do jogo produzido na ferramenta.

- **Enterprise:** Possui todas as características da versão Pro, contudo tem o enfoque para a produção em equipe. Seu preço é sob consulta.

Como o objetivo do jogo não é arrecadar lucros de qualquer natureza e a versão Personal dessa engine possui todas as características necessárias para a implementação do jogo, essa foi a versão escolhida.

Para auxiliar na modelagem de alguns objetos que compõe o jogo, foi utilizado o software Blender, pois é de fácil utilização e já existia prévia familiarização com o mesmo.

3.1.1 DESCRIÇÃO DA *ENGINE*

Por se tratar de uma *engine* bastante ampla, no que se refere a quantidade de funcionalidades disponíveis na mesma, ela permite que sejam implementados jogos em duas ou três dimensões. De forma geral, jogos em duas dimensões, utilizam *sprites*, ou seja, imagens bidimensionais que, ao serem combinadas, provocam a sensação de movimento, animação da cena. Já nos jogos em três dimensões, existe o conceito de objeto que possui diversas características, representado um elemento da vida real ou não.

3.1.1.1 OBJETOS

Um objeto é uma representação genérica de um elemento, que pode ser da vida real ou não. Na engine, um objeto é chamado de "GameObject". Como pode ser visto na Figura 1, existem diversos objetos de diferentes grupos:

1. Lista de todos os objetos possíveis para criação.

2. Objetos de três dimensões disponíveis.

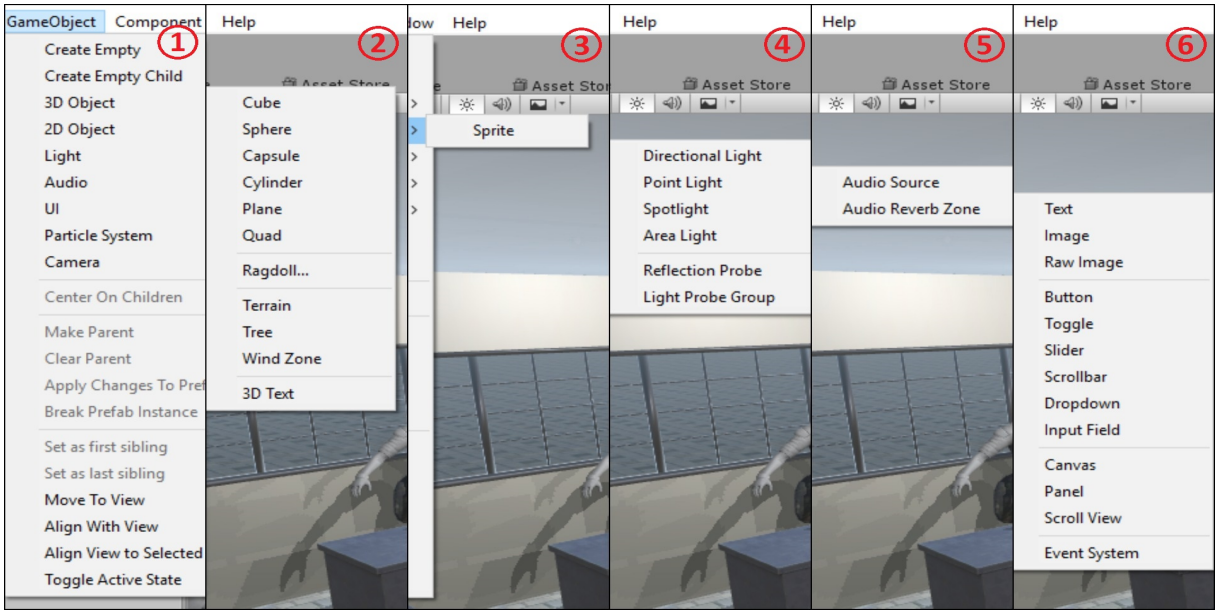
3. Objeto de duas dimensões, somente o Sprite.

4. Todos os objetos disponíveis de iluminação.

5. Objetos relacionados a áudio.

6. Objetos de interface com usuário, de maneira simples, os objetos que compõe menus.

Figura 1 – Tipos de objetos da engine



Fonte: Próprio autor

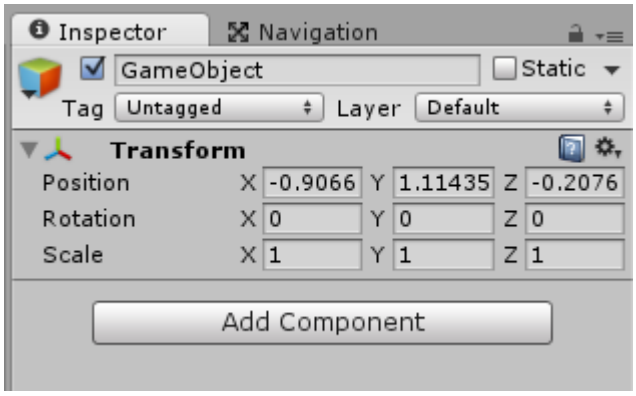
Como pode ser observado, enquanto alguns objetos são bem definidos, como por exemplo a câmera, iluminação e áudio, outros são bastante abstratos e indefinidos, deixando a cargo do desenvolvedor fazer a "especialização" de tal objeto. É importante ressaltar que é possível realizar customizações em objetos já definidos. A customização desses objetos,

tornando-os menos abstratos e mais fieis ao elemento que se deseja representar, é realizada com o segundo conceito que a engine fornece, os componentes.

3.1.1.2 COMPONENTES

Um componente descreve uma ou mais características de um objeto. Por padrão da engine, ao se criar um objeto, esse já vem com o componente "*Transform*". O componente "*Transform*", ilustrado na Figura 2, descreve as características de localização do objeto em relação a seu parentesco e sua escala de tamanho nos três eixos.

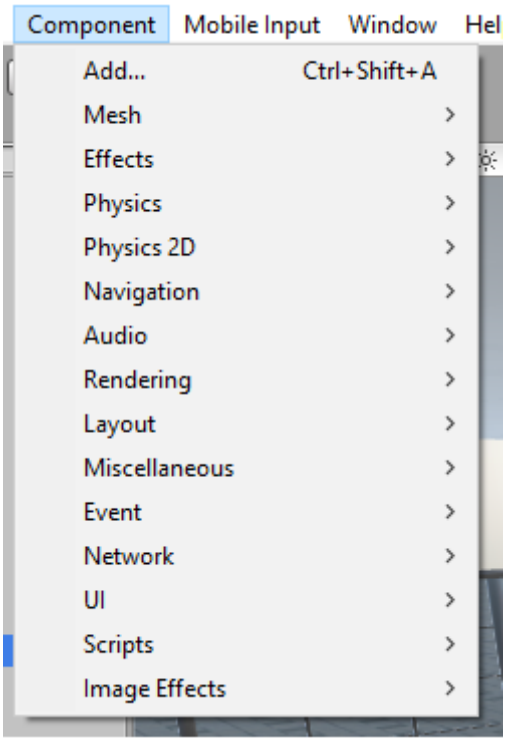
Figura 2 – Componente "*Transform*" dos objetos



Fonte: Próprio autor

Na engine, existe um grande conjunto de componentes disponíveis para inserção em um objeto. Na Figura 3 pode-se observar todos os componentes existentes agrupados: física, navegação, eventos, internet, interface de usuário, scripts, entre outros. Os scripts podem ser escritos nas linguagens C# ou JavaScript.

Figura 3 – Grupos de componentes da engine



Fonte: Próprio autor

3.1.2 UTILIZAÇÃO DA *ENGINE*

Do modo geral, os objetos usados para a construção do jogo podem ser divididos em três grandes grupos: Interface com o Usuário, Personagem e Cenário. Os objetos pertencentes a um grupo compartilham de características semelhantes, quando não são estritamente idênticos. Para controlar a comunicação entre objetos de grupos diferentes e/ou a relação deles com o tempo, foi criado um script, o qual pode ser visto no Apêndice F.

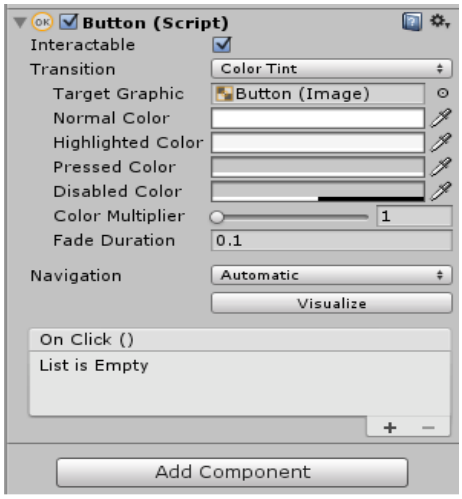
3.1.2.1 INTERFACE COM O USUÁRIO

O grupo de objetos Interface com o Usuário é o conjunto de menus dispostos na tela durante a execução do jogo, os quais permitem o jogador controlar aspectos específicos do *gameplay*. Cada menu é composto de:

- **Objetos gráficos:** Um ou mais objetos UI que a engine disponibiliza, entre eles: texto, caixa de seleção, caixa de inserção de texto, paineis, botões, entre outros, como a Figura 1, item 6 apresentou.

• **Componentes de controle:** São os *scripts* que determinam qual ação deve ocorrer ao interagir com algum elemento gráfico do menu. A engine disponibiliza automaticamente o *script* para atribuir ação ao botão (Figura 4), contudo para os outros elementos gráficos ou alguma ação diferente do padrão foi criado um novo *script*, como pode ser visto no Apêndice A. Já no Apêndice B é possível ver como foi realizada a ação de abrir e fechar um menu.

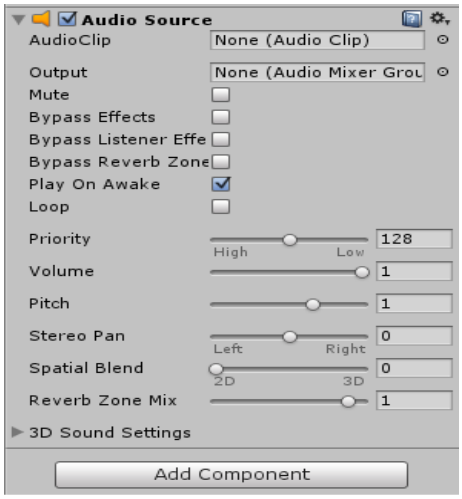
Figura 4 – Componente Script do botão



Fonte: Próprio autor

• **Componente de sonoridade:** Não está presente em todo o conjunto de menus desenvolvidos, apenas nos menus com o intuito de alertar o jogador. Observou-se que a presença de sonoridade pode aumentar o grau de alerta. A Figura 5 demonstra esse componente.

Figura 5 – Componente áudio



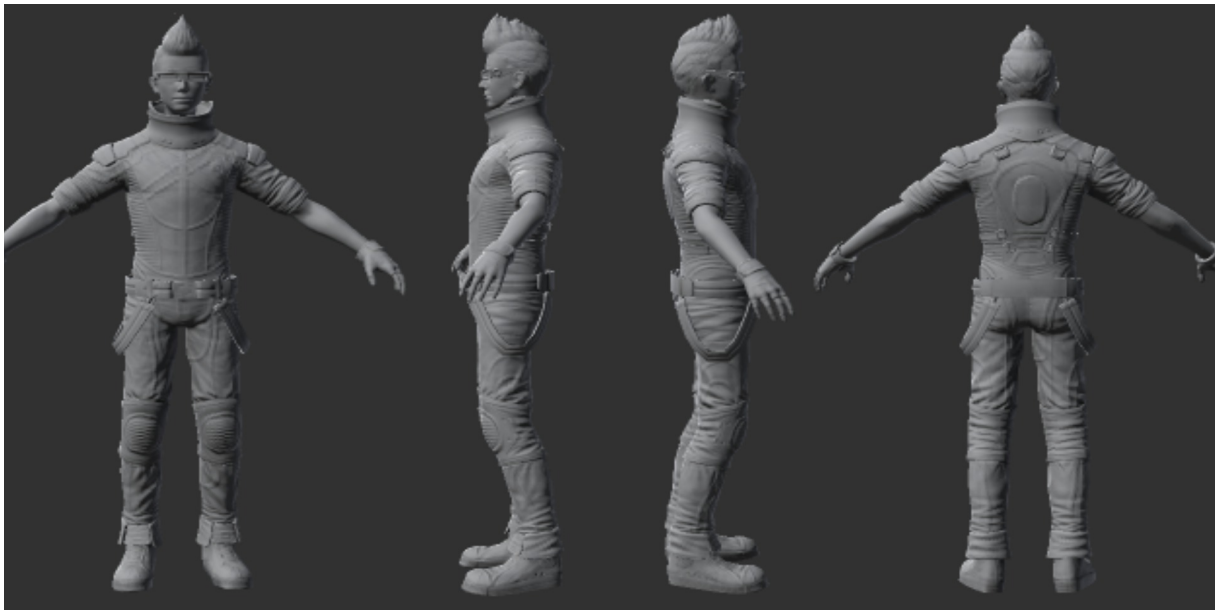
Fonte: Próprio autor

3.1.2.2 PERSONAGEM

O personagem é um objeto composto de uma série de componentes como: navegação no ambiente, colisão, entre outros componentes menos importantes para esse contexto. Os mais relevantes são:

• **Objeto gráfico:** Modelagem gráfica de uma pessoa, representando um membro da equipe. Na Figura 6 é possível ver o modelo utilizado, que vem por padrão em um dos pacotes gratuitos disponíveis na loja da engine.

Figura 6 – Modelo do personagem utilizado



Fonte: Próprio autor

• **Componentes de controle:** *Scripts* que controlam quais ações os personagens devem realizar de acordo com determinado momento do jogo. Nesse caso, o jogador não irá controlar diretamente os personagens através de comandos específicos. Os personagens são controlados automaticamente de acordo com o momento do jogo. Os *scripts* que realizam todos esses controles podem ser vistos no Apêndice C, Apêndice D e no Apêndice E.

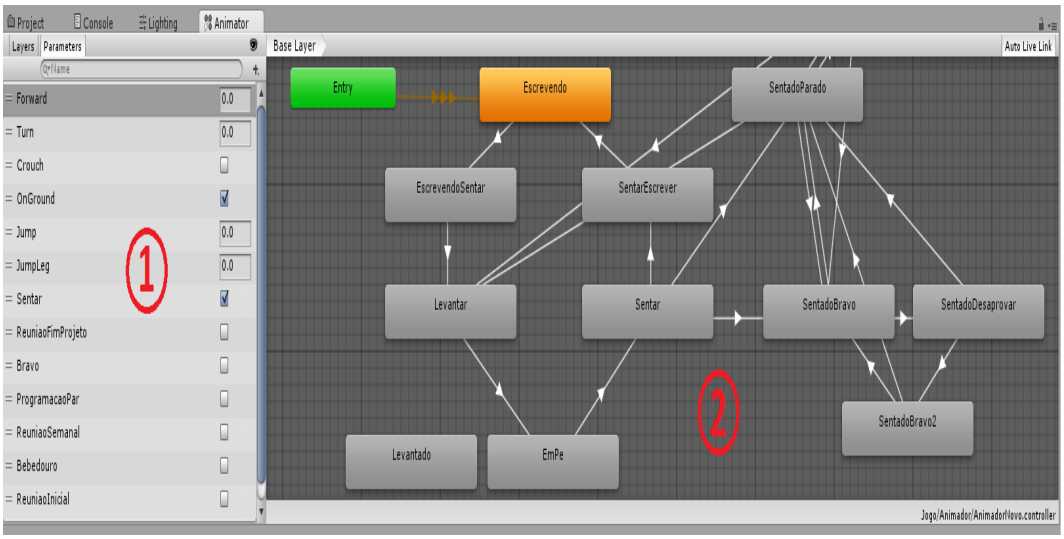
• **Componente de animação:** O personagem representa uma pessoa, ou seja, ele deve tentar seguir fielmente as características reais de uma. Portanto, os controles citados no item anterior servem para, entre outras tarefas, enviar comandos para o

Animador (Figura 7). Esse, por sua vez, é uma funcionalidade implementada na engine composta por:

- **Parâmetros:** Servem como controladores condicionais para as transições dos estados e para um conjunto de estados. Por exemplo, um personagem só irá realizar a animação de sentar se o parâmetro, do tipo booleano, nomeado "Sentar" estiver setado como "*true*". Representado na Figura 7, item 1.

- **Camadas de máquinas de estados:** Em uma camada existe uma máquina de estados. Cada estado pode ser uma única animação, de andar para frente, por exemplo. Mas um estado pode representar um conjunto de animações, gerenciadas internamente pelos parâmetros. Ainda, cada estado pode possuir várias transições para os demais, tornando a animação do personagem contínua. Representado na Figura 7, item 2.

Figura 7 – Componente animador da engine



Fonte: Próprio autor

- **Componentes de sonoridade:** Elementos sonoros para as diversas necessidades, podendo ser adicionado vários ao mesmo objeto. Auxiliam na representação das ações.

3.1.2.3 ELEMENTO CENÁRIO

Um elemento do tipo cenário é qualquer objeto inanimado que compõe o cenário, isto é, todo objeto que não for um Elemento do tipo personagem. Nessa categoria estão

objetos como: poltronas, mesas, computadores, objetos de composição (paredes, piso, teto, porta e janelas), entre outros objetos diversos. Os componentes importantes para esses elementos são:

- **Componente gráfico:** Modelos gráficos dos objetos. Na Figura 8 é possível ver os objetos pertencentes ao cenário.

Figura 8 – Objetos do cenário

Fonte: Próprio autor

- **Componente de controle:** *Script* para gerenciar determinadas ações dos objetos, como mudar de posição, desaparecer ou aparecer de acordo com a situação atual. Não são todos os elementos desse tipo que contém esse componente.

3.2 GAME DESIGN DOCUMENT

O documento utilizado como base para a implementação do jogo foi o GDD disponibilizado no Apêndice G. Para facilitar a compreensão desse documento, a Tabela 1 lista os itens que o compõe.

1 Introdução	1.1 Gameplay overview		
	1.2 Gênero, semelhanças e diferenças		
	1.3 Público alvo		
	1.4 Fluxo do jogo	1.4.1 Planejamento	1.4.1.1 Coleta de requisitos

			1.4.1.2 Escrita de histórias
		1.4.2 Projeto + Codificação + Teste	1.4.2.1 Implementação e teste de histórias
			1.4.2.2 Teste de integração
		1.4.3 Teste de aceitação	
2 Interface e Interação	2.1 Entradas		
	2.2 Saídas	2.2.1 Telas	
		2.2.2 Menus	
3 Mecânica do jogo	3.1 Mecânica Básica	3.1.1 Ações dos personagens	
		3.1.2 Pensamentos dos personagens	
	3.2 Projetos	3.2.1 Descrição	
		3.2.2 Histórias	
		3.2.3 Cronograma e progressão do jogo	
	3.3 Dificuldade		
	3.4 Condição de vitória		
4 Detalhamento técnico	4.1 Hardware		
	4.2 Software		
	4.3 Engine		
	4.4 I.A.		
5 Arte	5.1 Resumo de estilo		
	5.2 Concept Arts	5.2.1 Personagens	
		5.2.2 Cenário	
		5.2.3 Objetos	
	5.3 H.U.D. (Head Up Display)		
	5.4 Animações		
6 Som	6.1 Efeitos sonoros		
	6.2 Trilhas sonoras		
7 Referências			

Tabela 1 – Itens do Game Design Document

4 RESULTADOS

Esse capítulo irá discorrer a respeito dos resultados alcançados, ou seja, o jogo desenvolvido. De forma específica, apresentará como o jogo se apresenta no decorrer do tempo, isto é, como ele deve ser jogado, quais ações podem ser tomadas de acordo com o momento. A descrição parte do menu inicial e vai até o fim do projeto, fluxo a ser seguido.

4.1 MENU INICIAL

Na Figura 9 é demonstrado o Menu Inicial do jogo. Ele contém os seguintes botões:

• **Novo Jogo:** Carrega o menu para escolha do projeto a ser realizado no jogo.

• **Carregar jogo:** Redireciona para o menu de escolher um jogo previamente salvo. Opção será implementada no futuro.

• **Opções:** Abre o menu para alterar as configurações de intensidade do áudio. Opção será implementada no futuro.

• **Sobre:** Mostra o menu que lista informações de quem produziu o jogo, qual o propósito do jogo e dá opção para o jogador ver a ajuda sobre todo o jogo. Opção será implementada no futuro.

• **Sair:** Sai do jogo.

Figura 9 – Menu inicial do jogo

Fonte: Próprio autor

4.2 MENU NOVO JOGO

Esse submenu (Figura 10) dá a opção do jogador escolher entre três projetos, classificados de acordo com sua complexidade como: simples, médio e complexo. Contudo, o médio e complexo ficarão para implementações futuras, estando bloqueados na versão atual do jogo. O projeto simples consiste de quatro histórias, sendo cada uma delas implementadas dentro de uma semana. Então o projeto terá duração total de um mês. Após a seleção, deve-se pressionar o botão "iniciar"para o projeto começar.

Figura 10 – Menu para iniciar um novo jogo

Fonte: Próprio autor

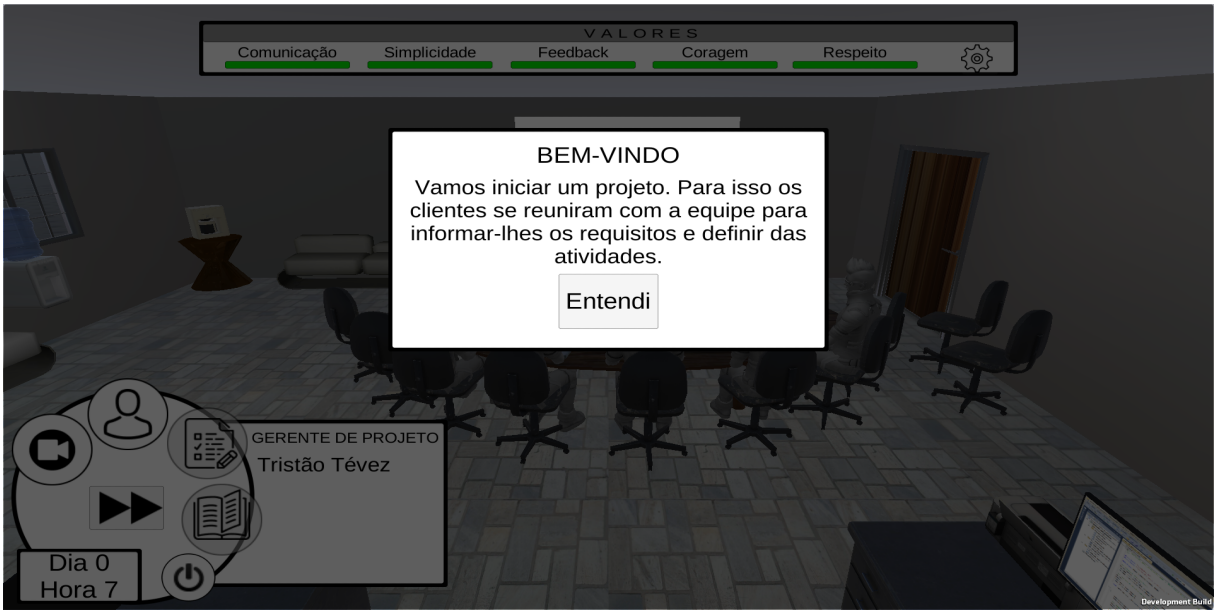
4.3 FLUXO PRINCIPAL

Considera-se como fluxo principal a sequência de passos realizadas de maneira a ter sucesso na realização do projeto, ou seja, chegar ao fim do mesmo concluído.

4.3.1 REUNIÃO INICIAL

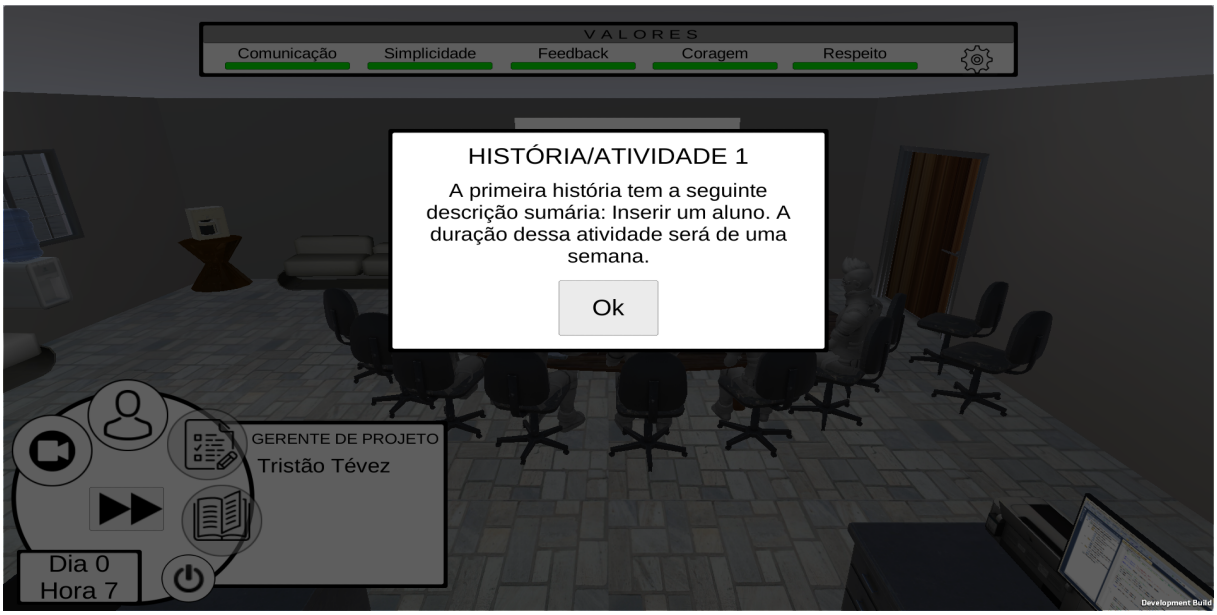
Ao iniciar o jogo, será mostrado ao jogador uma mensagem de boas-vindas (Figura 11), indicando que o que será mostrado a seguir é a reunião para coleta de requisitos, a reunião inicial. Nessa reunião o jogador não poderá interferir em nenhuma escolha, uma vez que: o cliente está passando os requisitos para a equipe e formulando, com a equipe, as histórias que serão implementadas. Portanto, serão mostradas quatro janelas (na Figura 12 é possível ver a primeira história), indicando as quatro histórias que a equipe formulou e o tempo de duração de cada.

Figura 11 – Mensagem de boas-vindas



Fonte: Próprio autor

Figura 12 – História formulada pela equipe



Fonte: Próprio autor

4.3.2 EXECUÇÃO DO PROJETO

Nessa etapa o projeto será executado, ou seja, as histórias serão implementadas de acordo com sua definição de tempo na fase anterior. A Figura 13 apresenta o menu principal, indicando qual a função de cada parte dele, sendo:

1. Botão para seleção de câmera: Esse botão pode ser utilizado para trocar entre as câmeras disponíveis em cada contexto. Durante a execução das atividades existe a possibilidade de trocar entre cinco câmeras. Já na reunião inicial e na semanal, é possível trocar entre somente duas câmeras.

2. Botão para seleção de personagem: Os personagens participantes de cada atividade (reunião inicial, execução, reunião semanal e reunião final) são diferentes. Portanto, o menu de seleção de personagens se adapta de acordo com o contexto, permitindo selecionar apenas os personagens que estão executando aquela atividade (A Figura 14 apresenta esse menu no momento da reunião inicial, podendo ver que os personagens não participantes não podem ser selecionados). Existem dois personagens com a função cliente e quatro com a função programador. Para selecioná-los, basta pressionar o botão correspondente novamente, selecionando o próximo dessa função. Os personagens das demais funções são somente uma unidade de cada.

3. Botão para visualização dos requisitos: Abre um menu mostrando quais foram os requisitos que o cliente passou à equipe (Figura 15).

4. Botão para visualização das atividades/histórias: Apresenta um menu listando todas as atividades/histórias formuladas pela equipe, junto com o cliente. É marcado qual item está sendo realizado naquele momento (Figura 16).

5. Botão para pausar o jogo: Pausa o jogo e abre um menu (Figura 17), dando possibilidade de retornar ao jogo; abrir um menu de opções (será implementado futuramente); salvar o jogo (será implementado futuramente) e sair do jogo.

6. Botão de aumento de velocidade temporal: Aumenta a velocidade temporal do jogo, fazendo com que as atividades sejam feitas com mais celeridade. Pressionando o botão várias vezes, a velocidade é aumentada até um limite. Ultrapassado esse limite, é voltado à velocidade normal.

7. Indicação do tempo atual: Apresenta o tempo atual do jogo, mostrando qual o dia que se encontra e qual a hora desse dia. Um dia tem seu início às sete horas e seu término às dezessete.

8. Indicação do personagem selecionado: Mostra o cargo e o nome do personagem selecionado atualmente.

9. Informação dos valores: Apresenta os pontos de valores atuais.

10. Configuração de itens relacionado aos valores: Abre um menu de configurações de itens que influenciam na mudança dos pontos de valores (Figura 18).

11. Quadro de atividades: Quadro onde as atividades realizadas são inseridas, deixando a equipe e o jogador informado (Figura 19).

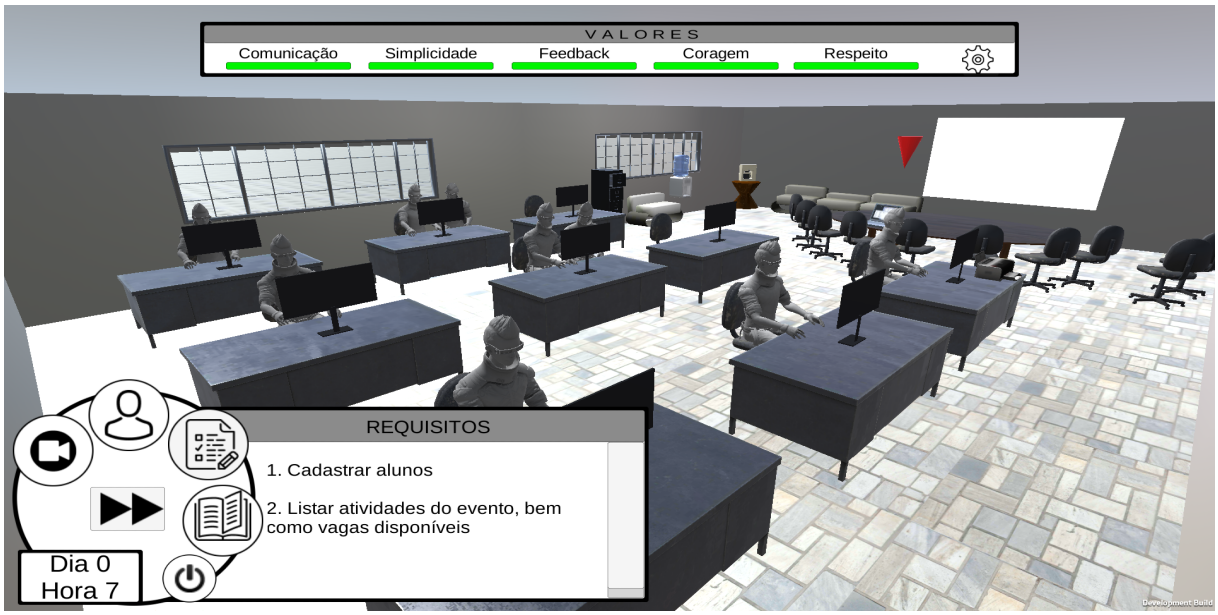
Figura 13 – Conjunto de menus durante a execução do jogo

Fonte: Próprio autor

Figura 14 – Menu de seleção de personagens

Fonte: Próprio autor

Figura 15 – Menu de requisitos



Fonte: Próprio autor

Figura 16 – Menu de atividades/histórias

Fonte: Próprio autor

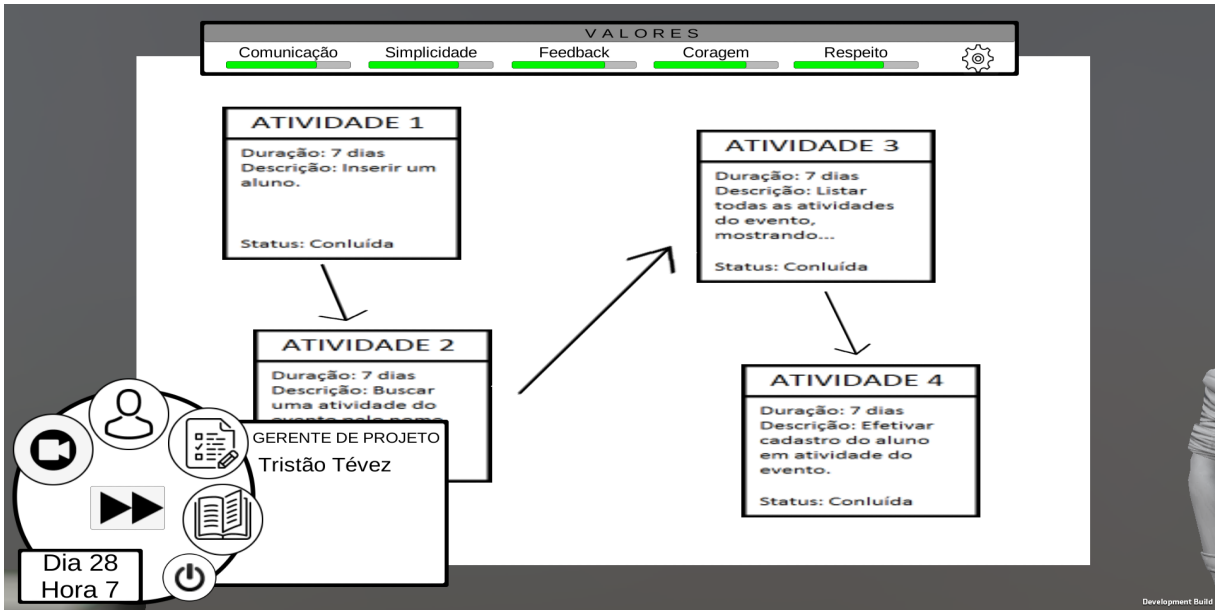
Figura 17 – Menu de pausa

Fonte: Próprio autor

Figura 18 – Menu de configuração de itens dos valores

Fonte: Próprio autor

Figura 19 – Quadro de atividades do projeto



Fonte: Próprio autor

No fim de cada dia será decrementado um valor aleatório entre 0.5 e 5.5 unidades dos pontos de todos os valores, devido a, segundo (BECK, 2004) avalia, cada pessoa pensar de maneira diferente, não refletindo sobre o que irá importar para a equipe.

No início de cada semana será apresentada uma mensagem questionando o jogador se o mesmo deseja fazer a Programação em Par (Figura 20). O jogador pode responder

positivamente, o que reduzirá o decremento aleatório do fim do dia de 0.5 até 5.5 unidades para 0.0 até 1.0 unidades. Essa diminuição do decremento dos pontos diz respeito a probabilidade menor da ocorrência de erros ao executar a programação em par (BECK, 2004; SCHACH, 2009; SOMMERVILLE, 2007). O jogador poderá pressionar o botão contendo uma "interrogação", o qual levará para o menu de ajuda para a programação em par (Figura 21). O seguinte trecho de código realiza essas operações, quando a programação em par não está sendo realizada e quando ela está:

```
if (!programacaoPar) {
   alteraCOM (-UnityEngine.Random.Range (0.5f, 5.5f));
   alteraSIM (-UnityEngine.Random.Range (0.5f, 5.5f));
   alteraFED (-UnityEngine.Random.Range (0.5f, 5.5f));
   alteraCOR (-UnityEngine.Random.Range (0.5f, 5.5f));
   alteraRES (-UnityEngine.Random.Range (0.5f, 5.5f));
} else {
   alteraCOM (-UnityEngine.Random.Range (0.0f, 1.0f));
   alteraSIM (-UnityEngine.Random.Range (0.0f, 1.0f));
   alteraFED (-UnityEngine.Random.Range (0.0f, 1.0f));
   alteraCOR (-UnityEngine.Random.Range (0.0f, 1.0f));
   alteraRES (-UnityEngine.Random.Range (0.0f, 1.0f));
}
```

Figura 20 – Menu de programação em par

Fonte: Próprio autor

Figura 21 – Menu de ajuda da programação em par

Fonte: Próprio autor

Já no fim do dia, uma mensagem no mesmo formato (Figura 22) questionará o jogador se o mesmo deseja que a equipe faça a Integração Contínua, mostrando a sua definição no menu de ajuda (Figura 23). Caso a resposta seja negativa, serão descontados, devido ao custo que essa negação pode gerar (BECK, 2004; SOMMERVILLE, 2007):

- 2.5 unidades do valor de comunição, por não informar a equipe;

- 1.5 unidades do valor de simplicidade, por ser provável que a falta de comunicação implique no surgimento de problemas, que consequentemente aumentarão a complexidade do software;

- 4.5 unidades do valor de feedback, por não informar a equipe.

Figura 22 – Menu de integração contínua

Fonte: Próprio autor

Figura 23 – Menu de ajuda da integração contínua

Fonte: Próprio autor

O trecho de código responsável por realizar essas operações é:

```
if (!respFimDia) {
   alteraCOM (-2.5f);
   alteraSIM (-1.5f);
   alteraFED (-4.5f);
}
```

Ao chegar no final da semana, uma mensagem (Figura 24) questionará o jogador se ele gostaria que a equipe realizasse a Reunião Semanal, em referência a prática de ciclo semanal, mostrando a sua definição no menu de ajuda (Figura 25). Caso a resposta seja negativa, devido ao não planejamento do trabalho realizado, aumentando a probabilidade de trabalhar com grandes quantidades de dados e a chance de planejar algo errado (BECK, 2004), serão descontados:

- 18.75 unidades do valor de comunição, por não informar a equipe;

- 11.25 unidades do valor de simplicidade, por ser provável que a falta de comunicação implique no surgimento de problemas, que consequentemente aumentarão a complexidade do software;

- 33.75 unidades do valor de feedback, por não informar a equipe.

A realização da reunião é mostrada na Figura 26.

Figura 24 – Menu de reunião semanal

Fonte: Próprio autor

Figura 25 – Menu de ajuda da reunião semanal

Fonte: Próprio autor

Figura 26 – Execução da reunião semanal

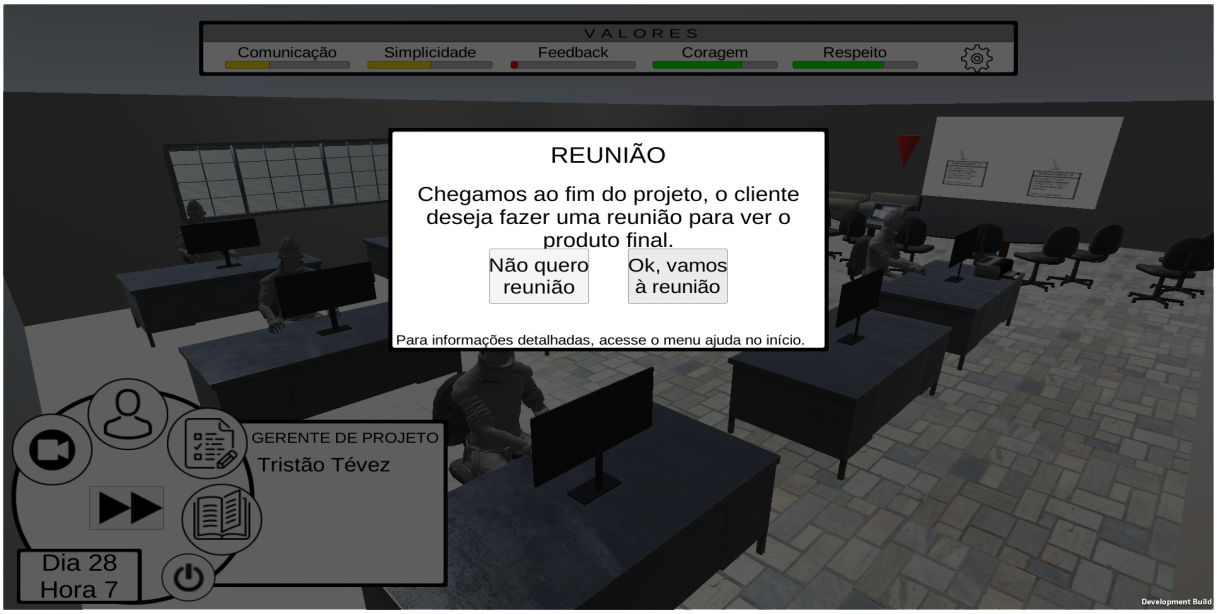
Fonte: Próprio autor

O trecho de código responsável por realizar essas operações é:

```
if (!respFimSemana) {
   alteraCOM (-18.75f);
   alteraSIM (-11.25f);
   alteraFED (-33.75f);
}
```

Por fim, ao final das quatro semanas, uma mensagem (Figura 27) questionará o jogador se ele gostaria que a equipe realizasse a Reunião Final com o cliente. Caso a resposta seja negativa, o projeto termina e volta-se ao menu inicial. Se a resposta for positiva, a reunião é realizada Figura 28.

Figura 27 – Menu de reunião final



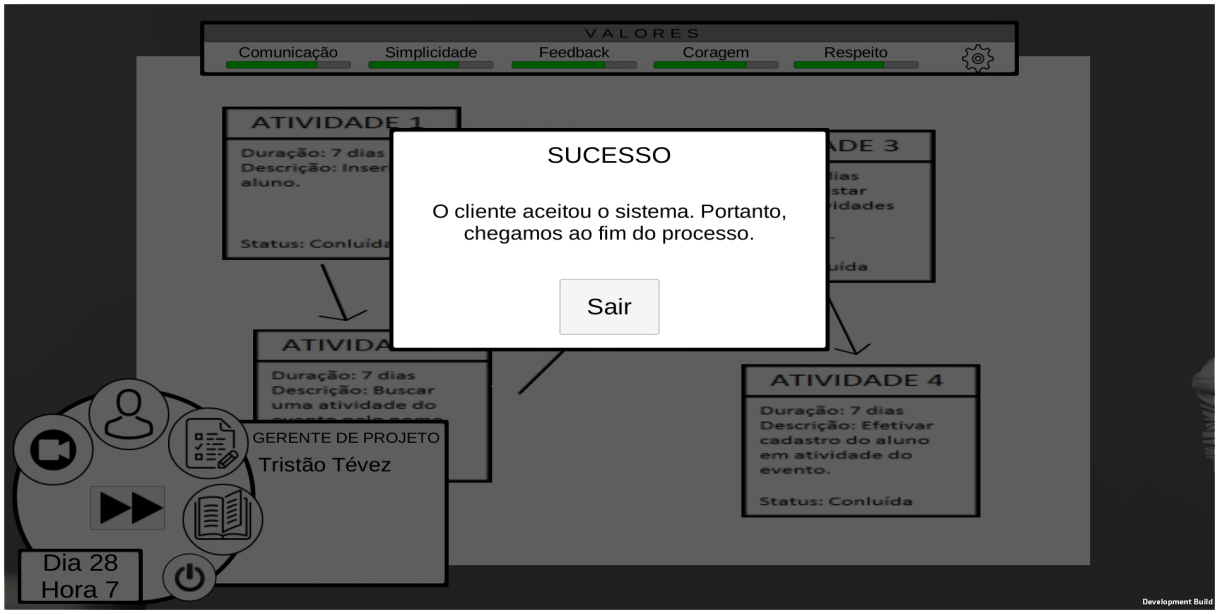
Fonte: Próprio autor

Figura 28 – Execução da reunião final

Fonte: Próprio autor

Ao final da reunião final, uma mensagem (Figura 29) indica o fim do projeto e volta-se ao menu inicial.

Figura 29 – Mensagem de confirmação do projeto



Fonte: Próprio autor

4.4 FLUXO SECUNDÁRIO

O fluxo secundário é realizado quando não se obtém sucesso na execução do projeto. O insucesso é ocasionado se qualquer um dos pontos de valores chegar a zero. A mensagem mostrada na Figura 30 aparece ao jogador, retornando ao menu inicial.

Figura 30 – Mensagem de falha no projeto

Fonte: Próprio autor

5 CONCLUSÃO

Devido a complexidade e conteúdo extenso da metodologia de desenvolvimento ágil de software XP, aliado a dificuldade natural de nós, seres humanos, aprendermos novos assuntos, um jogo utilizado como ferramenta educacional ilustrativa pode auxiliar no processo de aprendizado da mesma, como foi discorrido na pesquisa. Ainda podem coexistir com esses pontos negativos as diversas complicações do dia-a-dia dos utilizadores dessa metodologia, estando no meio educacional ou profissional.

O objetivo principal desse estudo foi o desenvolvimento de um jogo educativo, a partir de um GDD previamente idealizado, que ilustre a metodologia de desenvolvimento ágil de software XP. Esse jogo educativo tem a finalidade de dirimir ou eximir a complexidade de aprender essa metodologia somente pelos métodos convencionais de ensino. De forma específica, o objetivo foi aplicar a metodologia de desenvolvimento ágil de software XP e o documento de *design* do jogo e implementar o jogo educativo que ilustre essa metodologia, com base no documento de *design* estudado.

Foi utilizado o método de procedimento experimental, visto que a pesquisa foi idealizada com o intuito de atingir o resultado específico de desenvolver o jogo educativo como ferramenta auxiliadora no processo de aprendizado, baseado no estudo do GDD.

Com a pesquisa foi implementado o jogo baseado no GDD com o intuito de ilustrar a execução da XP. Sendo para efeitos de ilustração, o jogo apresenta os conceitos utilizados na XP, contudo o jogador deve ter um conhecimento prévio da metodologia, utilizando o jogo para fixar e exemplificar o conhecimento já obtido. Por conseguinte, a problemática objeto de estudo foi solucionada.

Apesar do jogo corresponder ao esperado para esse estudo, atendendo ao objetivo, ele pode ser melhorado em alguns aspectos, ficando a cargo de trabalhos futuros. Alguns desses aspectos que podem sofrer melhora são: implementar os modelos dos personagens, indicando os esteriótipos; aumentar o grau de interação do jogador, deixando mais decisões a cargo dele; e melhora estética do ambiente em geral.

REFERÊNCIAS

BECK, K. *Extreme Programming Explained: Embraced Change*. 2. ed. [S.l.]: Addison-Wesley, 2004. Citado 19 vezes nas páginas 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 41, 42, 43 e 45.

MARTINS, G. et al. Broad-plg: Modelo computacional para construção de jogos educacionais. *Anais do XXVI Simpósio Brasileiro de Informática na Educação*, Maceió, Brasil, 2015. Citado na página 19.

NETO, J. C. et al. Processo de desenvolvimento de software: uma análise exploratória com profissionais que desenvolvem jogos eletrônicos educacionais. *Anais do XXVI Simpósio Brasileiro de Informática na Educação*, Maceió, Brasil, 2015. Citado na página 19.

NOVAK, J. *Game Development Essentials: An Introduction*. 3. ed. [S.l.]: Cengage Learning, 2012. Citado 4 vezes nas páginas 19, 20, 21 e 22.

PEIXOTO, D. C. C. et al. Avaliação de jogos educacionais multiusuários: Uma revisão sistemática da literatura. *Anais do XXVI Simpósio Brasileiro de Informática na Educação*, Maceió, Brasil, 2015. Citado na página 23.

PRESSMAN, R. S. *Engenharia de software: uma abordagem profissional*. 7. ed. Porto Alegre: AMGH, 2011. Citado 2 vezes nas páginas 3 e 4.

SCHACH, R. S. *Engenharia de software: os paradigmas clássico e orientado a objetos*. 7. ed. [S.l.]: McGraw-Hill, 2009. Citado 2 vezes nas páginas 15 e 42.

SOLVUS. *Publicar um aplicativo na Apple Store e Google Play.* 2016. Acesso em: 25 de setembro de 2016. Disponível em: <http://solvus.com.br/4-coisas-que-voce-precisa-saber-para-publicar-um-aplicativo-apple-store-google-play/>. Citado na página 23.

SOMMERVILLE, I. *Engenharia de software*. 8. ed. [S.l.]: Pearson Addison-Wesley, 2007. Citado 4 vezes nas páginas 15, 18, 42 e 43.

UNITY. *Comparação entre os planos.* 2016. Acesso em: 25 de setembro de 2016. Disponível em: <https://store.unity.com/pt>. Citado na página 24.

APÊNDICE A – *SCRIPT* PARA GERENCIAR BOTÕES

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class Botao : MonoBehaviour {
   void Start () {
   }

   void Update () {
   }

   public void fechaJogo(){
      Application.Quit ();
   }

   public void carregaMenu(){
      SceneManager.LoadScene ("MenuPrincipal");
   }

   public void carregaJogo(){
      SceneManager.LoadScene ("Cena1");
   }

   public void retomaJogo(){
      GameObject.FindGameObjectWithTag("MenuPausa").SetActive (false);
      Time.timeScale = 1;
      AudioListener.pause = false;
   }
```

```
public void mudarVelocidade(){
   float g = Gameplay.multiplicador;
   if (g == 1000.0f) {
      Gameplay.multiplicador = 10000.0f;
   } else {
      if (g == 10000.0f) {
         Gameplay.multiplicador = 100000.0f;
      } else {
         if (g == 100000.0f) {
            Gameplay.multiplicador = 1000.0f;
         }
      }
   }
}

public void mudarCamera(){
   if ((!Gameplay.reuniaoFimProjeto) && (!Gameplay.reuniaoSemanal) &&
       (!Gameplay.reuniaoInicial)) {
      Gameplay.cameraValue = (Gameplay.cameraValue % 5) + 1;
   } else {
      if (Gameplay.reuniaoSemanal) {
         if (Gameplay.cameraValue == 7)
            Gameplay.cameraValue = 4;
         else
            Gameplay.cameraValue = 7;
      } else {
         if (Gameplay.reuniaoFimProjeto) {
            if (Gameplay.cameraValue == 6)
               Gameplay.cameraValue = 4;
            else
               Gameplay.cameraValue = 6;
         }
      }
   }
}
```

```
}
```

APÊNDICE B – *SCRIPT* PARA GERENCIAR MENUS

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class AbrirMenu : MonoBehaviour {
   void Start () {
      if (this.CompareTag ("MenuFimDia")) {
         Gameplay.objFimDia = this.gameObject;
         this.gameObject.SetActive (false);
      } else {
         if (this.CompareTag ("MenuFimSemana")) {
            Gameplay.objFimSemana = this.gameObject;
            this.gameObject.SetActive (false);
         } else {
            if (this.CompareTag ("MenuProgramacaoPar")) {
               Gameplay.objProgramacaoPar = this.gameObject;
               this.gameObject.SetActive (false);
            } else {
               if (this.CompareTag ("MenuFimJogoFalha")) {
                  Gameplay.objFimJogoFalha = this.gameObject;
                  this.gameObject.SetActive (false);
               } else {
                  if (this.CompareTag ("MenuFimJogoReuniao")) {
                     Gameplay.objFimJogoReuniao = this.gameObject;
                     this.gameObject.SetActive (false);
                  } else {
                     if (this.CompareTag ("Fade")) {
                        Gameplay.Fade = this.gameObject;
                        this.gameObject.SetActive (false);
```

```
} else {
   if (this.CompareTag ("MenuHistoriasAtividades")) {
      Gameplay.Atv1 = GameObject.FindGameObjectWithTag
          ("Atv1");
      Gameplay.Atv2 = GameObject.FindGameObjectWithTag
          ("Atv2");
      Gameplay.Atv3 = GameObject.FindGameObjectWithTag
          ("Atv3");
      Gameplay.Atv4 = GameObject.FindGameObjectWithTag
          ("Atv4");
      this.gameObject.SetActive (false);
   } else {
      if (this.CompareTag ("MenuEscolhaPersonagem")) {
         Gameplay.botaoANA =
             GameObject.FindGameObjectWithTag ("botaoANA");
         Gameplay.botaoARQ =
             GameObject.FindGameObjectWithTag ("botaoARQ");
         Gameplay.botaoCLI =
             GameObject.FindGameObjectWithTag ("botaoCLI");
         Gameplay.botaoEXE =
             GameObject.FindGameObjectWithTag ("botaoEXE");
         Gameplay.botaoDES =
             GameObject.FindGameObjectWithTag ("botaoDES");
         Gameplay.botaoGER =
             GameObject.FindGameObjectWithTag ("botaoGER");
         Gameplay.botaoPRO =
             GameObject.FindGameObjectWithTag ("botaoPRO");
         Gameplay.botaoRED =
             GameObject.FindGameObjectWithTag ("botaoRED");
         this.gameObject.SetActive (false);
      } else {
         if (this.CompareTag
             ("MenuAtividadeReuniaoInicial")) {
            Gameplay.menuAtividadeReuniaoInicial =
                this.gameObject;
```

```
                        Gameplay.textoTituloAtividadeReuniaoInicial =
                            GameObject.FindGameObjectWithTag
                            ("tituloAtividadeReuniaoInicial").GetComponent<Text>
                            ();
                        Gameplay.textoCorpoAtividadeReuniaoInicial =
                            GameObject.FindGameObjectWithTag
                            ("textoAtividadeReuniaoInicial").GetComponent<Text>
                            ();
                        this.gameObject.SetActive (false);
                    } else {
                        if (this.CompareTag ("MensagemFinalProjeto")) {
                            Gameplay.mensagemFinalProjeto =
                                this.gameObject;
                            this.gameObject.SetActive (false);
                        }
                    }
                }
            }
        }
    }
}

void Update () {
}

public void ativaDesativaMenu(){
    GameObject go;
    this.gameObject.SetActive(!this.gameObject.activeSelf);
    if (this.CompareTag ("MenuPausa")) {
        Time.timeScale = 0;
        AudioListener.pause = true;
```

```
}else{
   if (this.CompareTag ("MenuRequisitos")) {
      go = GameObject.FindGameObjectWithTag ("MenuHistoriasAtividades");
      if(go != null)
         go.SetActive (false);
   } else {
      if (this.CompareTag ("MenuHistoriasAtividades")) {
         go = GameObject.FindGameObjectWithTag ("MenuRequisitos");
         if (go != null)
            go.SetActive (false);
      } else {
         if (this.CompareTag ("MenuFimDia")) {
            Time.timeScale = 1;
            Gameplay.tempo += 3600 * 14;
         } else {
            if (this.CompareTag ("MenuFimSemana")) {
               Time.timeScale = 1;
               Gameplay.tempo += 3600 * 14;
            } else {
               if (this.CompareTag ("MenuProgramacaoPar")) {
                  Time.timeScale = 1;
               } else {
                  if (this.CompareTag ("MenuFimJogoReuniao")) {
                     Time.timeScale = 1;
                  } else {
                     if (this.CompareTag ("MenuAtividadeReuniaoInicial")) {
                        Time.timeScale = 1;
                     } else {
                        if (this.CompareTag ("MenuBoasVindas")) {
                           Time.timeScale = 1;
                        }
                     }
                  }
               }
            }
```

```
                }
            }
        }
    }
}
}
```

APÊNDICE C – *SCRIPT* PARA CONTROLE DE PERSONAGEM

```
using System;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;
using UnityEngine.UI;

namespace UnityStandardAssets.Characters.ThirdPerson{
   [RequireComponent(typeof (NavMeshAgent))]
   [RequireComponent(typeof (ThirdPersonCharacter))]
   public class ControlePersonagem : MonoBehaviour{
      private Animator animator;
      private ThirdPersonCharacter m_Character;
      private Transform m_Cam;
      private Vector3 m_CamForward;
      private Vector3 m_Move, posSentar;
      private bool m_Jump;
      private bool reuniaoInicio, reuniaoFim, reuniaoSem;
      private float tempo, limiteTempo;

      private void Start(){
         if (Camera.main != null){
            m_Cam = Camera.main.transform;
         }else{
            Debug.LogWarning(
               "Warning: no main camera found. Third person character needs a
                  Camera tagged \"MainCamera\", for camera-relative controls.");
         }
         m_Character = GetComponent<ThirdPersonCharacter>();
         animator = GetComponent<Animator>();
         if (!this.CompareTag ("Cliente1") && !this.CompareTag ("Cliente2")) {
```

```
            animator.Play ("Escrevendo", 0, UnityEngine.Random.Range (0.0f,
                1.0f));
        }
        posSentar = GetComponent<Transform> ().position;
        reuniaoInicio = false;
        reuniaoFim = false;
        reuniaoSem = false;
        tempo = 0.0f;
    }

    public void setPersonagemSelecionado(){
        ThirdPersonCharacter pSelecionado =
            ControlePersonagens.getPersonagemSelecionado();
        MeshRenderer mesh;
        GameObject go;
        Text t = null, t1 = null;
        if (pSelecionado != null) {
            mesh = pSelecionado.GetComponentInChildren<MeshRenderer> ();
            mesh.enabled = false;
        }
        go = GameObject.FindGameObjectWithTag ("TextoCargoPersonagem");
        if(go != null)
            t = go.GetComponent<Text>();
        go = GameObject.FindGameObjectWithTag ("TextoNomePersonagem");
        if(go != null)
            t1 = go.GetComponent<Text>();
        if(m_Character.CompareTag("Programador1") ||
            m_Character.CompareTag("Programador2") ||
            m_Character.CompareTag("Programador3") ||
                m_Character.CompareTag("Programador4")){
            ThirdPersonCharacter p;
            if (pSelecionado.CompareTag ("Programador1")) {
                p = GameObject.FindGameObjectWithTag
                    ("Programador2").GetComponent<ThirdPersonCharacter> ();
            } else {
```

```
            if (pSelecionado.CompareTag ("Programador2")) {
                p = GameObject.FindGameObjectWithTag
                    ("Programador3").GetComponent<ThirdPersonCharacter> ();
            } else {
                if (pSelecionado.CompareTag ("Programador3")) {
                    p = GameObject.FindGameObjectWithTag
                        ("Programador4").GetComponent<ThirdPersonCharacter> ();
                } else {
                    p = GameObject.FindGameObjectWithTag
                        ("Programador1").GetComponent<ThirdPersonCharacter> ();
                }
            }
        }
        ControlePersonagens.setPersonagemSelecionado (p);
        mesh = p.GetComponentInChildren<MeshRenderer> ();
        if(t != null)
            t.text = p.GetComponent<SetAttrsPersonagens>().getFuncao();
        if(t1 != null)
            t1.text = p.GetComponent<SetAttrsPersonagens>().getNome();
    }else{
        if (m_Character.CompareTag ("Cliente1") || m_Character.CompareTag
            ("Cliente2")) {
            ThirdPersonCharacter p;
            if (pSelecionado.CompareTag ("Cliente1")) {
                p = GameObject.FindGameObjectWithTag
                    ("Cliente2").GetComponent<ThirdPersonCharacter> ();
            } else {
                p = GameObject.FindGameObjectWithTag
                    ("Cliente1").GetComponent<ThirdPersonCharacter> ();
            }
            ControlePersonagens.setPersonagemSelecionado (p);
            mesh = p.GetComponentInChildren<MeshRenderer> ();
            if(t != null)
                t.text = p.GetComponent<SetAttrsPersonagens>().getFuncao();
            if(t1 != null)
```

```
            t1.text = p.GetComponent<SetAttrsPersonagens>().getNome();
        } else {
            ControlePersonagens.setPersonagemSelecionado (m_Character);
            mesh = m_Character.GetComponentInChildren<MeshRenderer> ();
            if (t != null)
                t.text = m_Character.GetComponent<SetAttrsPersonagens>
                    ().getFuncao ();
            if (t1 != null)
                t1.text = m_Character.GetComponent<SetAttrsPersonagens>
                    ().getNome ();
        }
    }
    mesh.enabled = true;
}

private void Update(){
    if (Gameplay.terminarReuniao) {
        animator.SetBool ("Sentar", true);
        animator.SetBool ("ReuniaoSemanal", false);
        animator.SetBool ("ReuniaoInicial", false);
        if (!this.CompareTag ("Cliente1") && !this.CompareTag ("Cliente2"))
            {
            animator.Play ("Escrevendo", 0, UnityEngine.Random.Range (0.0f,
                1.0f));
        }
        Vector3 v = new Vector3 ();
        v.Set (posSentar.x, posSentar.y, posSentar.z);
        m_Character.transform.position = v;
        m_Character.transform.rotation = Quaternion.Euler (0, 90, 0);
        Gameplay.cameraValue = 5;
        reuniaoSem = false;
        reuniaoInicio = false;
        Gameplay.qtdTerminaramReuniao++;
        if (ControlePersonagens.cliente1.activeSelf) {
```

```
            ControlePersonagens.cliente1.GetComponent<Animator> ().SetBool
                ("ReuniaoInicial", false);
            ControlePersonagens.cliente1.GetComponent<Animator> ().SetBool
                ("ReuniaoFimProjeto", false);
            ControlePersonagens.cliente1.SetActive (false);
        }
        if (ControlePersonagens.cliente2.activeSelf) {
            ControlePersonagens.cliente2.GetComponent<Animator> ().SetBool
                ("ReuniaoInicial", false);
            ControlePersonagens.cliente2.GetComponent<Animator> ().SetBool
                ("ReuniaoFimProjeto", false);
            ControlePersonagens.cliente2.SetActive (false);
        }
        if (ControlePersonagens.executivo.activeSelf) {
            ControlePersonagens.executivo.GetComponent<Animator> ().SetBool
                ("ReuniaoInicial", false);
            ControlePersonagens.executivo.SetActive (false);
        }
    }else{
        if (!Gameplay.stop) {
            if (animator.GetCurrentAnimatorStateInfo (0).IsName
                ("Escrevendo")) {
                if ((Gameplay.reuniaoFimProjeto) && (!reuniaoFim)){
                    animator.SetBool ("Sentar", false);
                    animator.SetBool ("ReuniaoFimProjeto", true);
                } else {
                    if (Gameplay.reuniaoSemanal) {
                        if ((!this.CompareTag ("Arquiteto")) &&
                            (!this.CompareTag ("DesignerInteracao"))) {
                            animator.SetBool ("Sentar", false);
                        }
                        animator.SetBool ("ReuniaoSemanal", true);
                    } else {
                        if ((Gameplay.reuniaoInicial) && (!reuniaoInicio)){
                            Vector3 v = new Vector3 ();
```

```
            Quaternion q = Quaternion.Euler (0, 0, 0);

            if (this.CompareTag ("GerenteProjeto")) {
               v.Set (0.006f, posSentar.y, 2.216f);
               q = Quaternion.Euler (0, 15.26f, 0);
               m_Character.transform.position = v;
               m_Character.transform.rotation = q;
               animator.Play ("SentadoFalandoDireita2", 0,
                   UnityEngine.Random.Range (0.0f, 10.0f));
            } else {
               if (this.CompareTag ("RedatorTecnico")) {
                  v.Set (0.5627159f, posSentar.y, 2.02f);
                  q = Quaternion.Euler (0, 16.33f, 0);
                  m_Character.transform.position = v;
                  m_Character.transform.rotation = q;
               }
            }
            Gameplay.cameraValue = 6;
            animator.SetBool ("ReuniaoInicial", true);
            reuniaoInicio = true;
         } else {
            if (Gameplay.programacaoPar) {
               if (this.CompareTag ("Programador3")) {
                  animator.SetBool ("ProgramacaoPar", true);
                  Vector3 v = new Vector3 ();
                  v.Set (-0.913f, posSentar.y, -1.567f);
                  m_Character.transform.position = v;
                  Quaternion q = Quaternion.Euler (0, 110, 0);
                  m_Character.transform.rotation = q;
                  animator.Play ("SentadoParado", 0, 0);
               } else {
                  if (this.CompareTag ("Programador1")) {
                     animator.SetBool ("ProgramacaoPar", true);
                     Vector3 v = new Vector3 ();
                     v.Set (-3.507264f, posSentar.y, -1.551249f);
```

```
                    m_Character.transform.position = v;
                    Quaternion q = Quaternion.Euler (0,
                        113.5365f, 0);
                    m_Character.transform.rotation = q;
                    animator.Play ("SentadoParado", 0, 0);
                }
            }
        }
    }
  }
}
if ((!Gameplay.programacaoPar) && (animator.GetBool
    ("ProgramacaoPar"))) {
  animator.SetBool ("ProgramacaoPar", false);
  Vector3 v = new Vector3 ();
  v.Set (posSentar.x, posSentar.y, posSentar.z);
  m_Character.transform.position = v;
  Quaternion q = Quaternion.Euler (0, 90, 0);
  m_Character.transform.rotation = q;
  animator.Play ("Escrevendo", 0, UnityEngine.Random.Range
      (0.0f, 1.0f));
}
if (reuniaoFim) {
  float prob;
  tempo += Time.deltaTime;
  if (this.CompareTag ("Cliente1") || this.CompareTag
      ("Cliente2")) {
    prob = (Gameplay.comunicacao + Gameplay.coragem +
        Gameplay.feedback + Gameplay.respeito +
        Gameplay.simplicidade) / 500.0f;
  } else {
    prob = 0.75f;
  }
  if (animator.GetBool ("Bravo"))
```

```
            limiteTempo = (UnityEngine.Random.Range (3.0f, 7.5f));
        else
            limiteTempo = (UnityEngine.Random.Range (19.0f, 26.5f));
        if (tempo >= limiteTempo) {
            animator.SetBool ("Bravo", false);
            if ((UnityEngine.Random.Range (0.0f, 1.0f)) <= 1.0f - prob)
                {
                animator.SetBool ("Bravo", true);
            }
            tempo = 0.0f;
        }
    }
    if (animator.GetCurrentAnimatorStateInfo (0).IsName ("EmPe")) {
        if (animator.GetBool ("ReuniaoFimProjeto")) {
            if (!reuniaoFim) {
                Vector3 v = new Vector3 ();
                Quaternion q = Quaternion.Euler (0, 0, 0);
                Gameplay.cameraValue = 6;
                if (this.CompareTag ("Programador1")) {
                    v.Set (2.467f, posSentar.y, 2.433f);
                    q = Quaternion.Euler (0, 314, 0);
                } else {
                    if (this.CompareTag ("Programador2")) {
                        v.Set (2.785f, posSentar.y, 2.907f);
                        q = Quaternion.Euler (0, 300.95f, 0);
                    } else {
                        if (this.CompareTag ("Programador3")) {
                            v.Set (3.527f, posSentar.y, 2.595f);
                            q = Quaternion.Euler (0, 329.59f, 0);
                        } else {
                            if (this.CompareTag ("Programador4")) {
                                v.Set (3.69f, posSentar.y, 3.257f);
                                q = Quaternion.Euler (0, 322, 0);
                            } else {
                                if (this.CompareTag ("AnalistaTeste")) {
```

```
                    v.Set (1.904f, posSentar.y, 2.097f);
                    q = Quaternion.Euler (0, 345, 0);
                } else {
                    if (this.CompareTag ("Arquiteto")) {
                        v.Set (0.006f, posSentar.y, 2.216f);
                        q = Quaternion.Euler (0, 15.26f, 0);
                    } else {
                        if (this.CompareTag
                            ("DesignerInteracao")) {
                            v.Set (1.233f, posSentar.y, 1.964f);
                        } else {
                            if (this.CompareTag
                                ("GerenteProjeto")) {
                                v.Set (2.635f, posSentar.y,
                                    5.606f);
                                q = Quaternion.Euler (0, 180, 0);
                            } else {
                                v.Set (0.5627159f, posSentar.y,
                                    2.02f);
                                q = Quaternion.Euler (0, 16.33f,
                                    0);
                            }
                        }
                    }
                }
            }
        }
    }
}
m_Character.transform.position = v;
m_Character.transform.rotation = q;
if (!this.CompareTag ("GerenteProjeto")) {
    if (this.CompareTag ("RedatorTecnico")) {
        animator.Play ("Escrevendo", 0,
            UnityEngine.Random.Range (0.0f, 10.0f));
```

```
                } else {
                    animator.Play ("SentadoParado", 0,
                        UnityEngine.Random.Range (0.0f, 10.0f));
                }
                animator.SetBool ("Sentar", true);
            } else {
                animator.Play ("FalandoEmPe1", 0, 0);
            }
            reuniaoFim = true;
        }
    } else {
        if (animator.GetBool ("ReuniaoSemanal")) {
            if (!reuniaoSem) {
                Vector3 v = new Vector3 ();
                Quaternion q = Quaternion.Euler (0, 0, 0);
                Gameplay.cameraValue = 7;
                if ((!this.CompareTag ("Arquiteto")) &&
                    (!this.CompareTag ("DesignerInteracao"))) {
                    if (this.CompareTag ("Programador1")) {
                        v.Set (-2.2f, posSentar.y, 5.57f);
                        q = Quaternion.Euler (0, 180, 0);
                        animator.Play ("SentadoFalandoEsquerda2", 0,
                            UnityEngine.Random.Range (0.0f, 100.0f));
                        animator.SetBool ("Sentar", true);
                    } else {
                        if (this.CompareTag ("Programador2")) {
                            v.Set (-1.35f, posSentar.y, 5.57f);
                            q = Quaternion.Euler (0, 180, 0);
                            animator.Play ("SentadoFalandoDireita2", 0,
                                UnityEngine.Random.Range (0.0f, 100.0f));
                            animator.SetBool ("Sentar", true);
                        } else {
                            if (this.CompareTag ("Programador3")) {
                                v.Set (-0.45f, posSentar.y, 5.57f);
                                q = Quaternion.Euler (0, 180, 0);
```

```
            animator.Play ("SentadoParado", 0,
                UnityEngine.Random.Range (0.0f, 10.0f));
            animator.SetBool ("Sentar", true);
        } else {
            if (this.CompareTag ("Programador4")) {
                v.Set (-3.4f, posSentar.y, 4);
                q = Quaternion.Euler (0, 120, 0);
                animator.Play ("FalandoBebedouro", 0, 0);
                animator.SetBool ("Bebedouro", true);
            } else {
                if (this.CompareTag ("AnalistaTeste")) {
                    v.Set (-3.4f, posSentar.y, 2.9f);
                    q = Quaternion.Euler (0, 40, 0);
                    animator.Play
                        ("SentadoFalandoEsquerda2", 0,
                        UnityEngine.Random.Range (0.0f,
                        10.0f));
                    animator.SetBool ("Sentar", true);
                } else {
                    if (this.CompareTag
                        ("GerenteProjeto")) {
                        v.Set (-2.5f, posSentar.y, 4.2f);
                        q = Quaternion.Euler (0, 180, 0);
                        animator.Play ("Bebendo", 0, 0);
                        animator.SetBool ("Bebedouro",
                            false);
                    } else {
                        v.Set (-2.5f, posSentar.y, 3.3f);
                        q = Quaternion.Euler (0, 320, 0);
                        animator.Play ("FalandoEmPe2", 0,
                            0);
                    }
                }
            }
        }
```

```
                }
            }
        }
        m_Character.transform.position = v;
        m_Character.transform.rotation = q;
        reuniaoSem = true;
      }
    }
  }
}
if (reuniaoInicio) {
  if (!ControlePersonagens.cliente1.activeSelf) {
    ControlePersonagens.cliente1.SetActive (true);
    ControlePersonagens.cliente1.GetComponent<Animator>
        ().SetBool ("Sentar", true);
    ControlePersonagens.cliente1.GetComponent<Animator>
        ().SetBool ("ReuniaoInicial", true);
    ControlePersonagens.cliente1.GetComponent<Animator> ().Play
        ("SentadoParado", 0, UnityEngine.Random.Range (0.0f,
        10.0f));
    Vector3 v = new Vector3 ();
    v.Set (1.904f, posSentar.y, 2.097f);
    ControlePersonagens.cliente1.transform.position = v;
  }
  if (!ControlePersonagens.cliente2.activeSelf) {
    ControlePersonagens.cliente2.SetActive (true);
    ControlePersonagens.cliente2.GetComponent<Animator>
        ().SetBool ("Sentar", true);
    ControlePersonagens.cliente2.GetComponent<Animator>
        ().SetBool ("ReuniaoInicial", true);
    ControlePersonagens.cliente2.GetComponent<Animator> ().Play
        ("SentadoFalandoEsquerda2", 0, UnityEngine.Random.Range
        (0.0f, 10.0f));
    Vector3 v = new Vector3 ();
    v.Set (2.467f, posSentar.y, 2.433f);
```

```
            ControlePersonagens.cliente2.transform.position = v;
            ControlePersonagens.cliente2.transform.rotation =
                Quaternion.Euler (0, 314, 0);
        }
        if (!ControlePersonagens.executivo.activeSelf) {
            ControlePersonagens.executivo.SetActive (true);
            ControlePersonagens.executivo.GetComponent<Animator>
                ().SetBool ("Sentar", true);
            ControlePersonagens.executivo.GetComponent<Animator>
                ().SetBool ("ReuniaoInicial", true);
            ControlePersonagens.executivo.GetComponent<Animator>
                ().Play ("SentadoParado", 0, UnityEngine.Random.Range
                (0.0f, 10.0f));
        }
    }else{
        if (reuniaoFim){
            if (!ControlePersonagens.cliente1.activeSelf) {
                ControlePersonagens.cliente1.SetActive (true);
                ControlePersonagens.cliente1.GetComponent<Animator>
                    ().SetBool ("Sentar", true);
                ControlePersonagens.cliente1.GetComponent<Animator>
                    ().SetBool ("ReuniaoFimProjeto", true);
                ControlePersonagens.cliente1.GetComponent<Animator>
                    ().Play ("SentadoParado", 0, UnityEngine.Random.Range
                    (0.0f, 10.0f));
            }
            if (!ControlePersonagens.cliente2.activeSelf) {
                ControlePersonagens.cliente2.SetActive (true);
                ControlePersonagens.cliente2.GetComponent<Animator>
                    ().SetBool ("Sentar", true);
                ControlePersonagens.cliente2.GetComponent<Animator>
                    ().SetBool ("ReuniaoFimProjeto", true);
                ControlePersonagens.cliente2.GetComponent<Animator>
                    ().Play ("SentadoParado", 0, UnityEngine.Random.Range
                    (0.0f, 10.0f));
```

```
                    }
                }
            }
        }
    }

    private void FixedUpdate(){
        if (m_Cam != null) {
            ;
        }
    }
  }
}
```

APÊNDICE D – *SCRIPT* PARA GERENCIAR TODOS OS PERSONAGENS

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

namespace UnityStandardAssets.Characters.ThirdPerson{
   [RequireComponent(typeof (ThirdPersonCharacter))]
   public class ControlePersonagens : MonoBehaviour {
      private static ThirdPersonCharacter personagemSelecionado = null;
      public static GameObject cliente1, cliente2, executivo;
      private AudioSource[] audios;

      void Start () {
         GameObject go;
         Text t = null, t1 = null;
         personagemSelecionado = GameObject.FindGameObjectWithTag
             ("GerenteProjeto").GetComponent<ThirdPersonCharacter> ();
         if (personagemSelecionado != null) {
            MeshRenderer mesh =
                personagemSelecionado.GetComponentInChildren<MeshRenderer> ();
            mesh.enabled = true;
         }
         go = GameObject.FindGameObjectWithTag ("TextoCargoPersonagem");
         if (go != null) {
            t = go.GetComponent<Text> ();
            if(t != null)
               t.text = "GERENTE DE PROJETO";

         }
         go = GameObject.FindGameObjectWithTag ("TextoNomePersonagem");
```

```
        if (go != null) {
            t1 = go.GetComponent<Text> ();
            if(t1 != null)
                t1.text = "Trist\~ao T\'evez";
        }
        cliente1 = GameObject.FindGameObjectWithTag ("Cliente1");
        cliente2 = GameObject.FindGameObjectWithTag ("Cliente2");
        executivo = GameObject.FindGameObjectWithTag ("Executivo");
        cliente1.SetActive (false);
        cliente2.SetActive (false);
        executivo.SetActive (false);
        audios = GetComponents<AudioSource> ();
    }

    void Update () {
        foreach (var audio in audios) {
            if (audio != null){
                if (!Gameplay.reuniaoInicial && !Gameplay.reuniaoSemanal &&
                    !Gameplay.reuniaoFimProjeto) {
                    if (audio.mute) {
                        if (audio.volume < 0.05)
                            audio.volume += 0.005f;
                        else
                            audio.mute = false;
                    }
                } else {
                    if (!audio.mute){
                        if (audio.volume > 0.05)
                            audio.volume -= 0.005f;
                        else
                            audio.mute = true;
                    }
                }
            }
        }
```

```
      }

      public static ThirdPersonCharacter getPersonagemSelecionado(){
         return personagemSelecionado;
      }

      public static void setPersonagemSelecionado(ThirdPersonCharacter p){
         personagemSelecionado = p;
      }
   }
}
```

APÊNDICE E – *SCRIPT* PARA SETAR CARACTERÍSTICAS DOS PERSONAGENS

```
using UnityEngine;
using System.Collections;

namespace UnityStandardAssets.Characters.ThirdPerson{
   [RequireComponent(typeof (ThirdPersonCharacter))]
   public class SetAttrsPersonagens : MonoBehaviour {
      string nome, funcao;
      string[] atividades;

      void Start () {
         ThirdPersonCharacter Personagem = GetComponent<ThirdPersonCharacter>();
         if (Personagem.CompareTag ("Programador1")) {
            funcao = "PROGRAMADOR";
            nome = "Godo Coutinho";
         } else {
            if (Personagem.CompareTag ("Programador2")) {
               funcao = "PROGRAMADOR";
               nome = "Alexandre Bocai\'uva";
            } else {
               if (Personagem.CompareTag ("Programador3")) {
                  funcao = "PROGRAMADOR";
                  nome = "Elisa Quadros";
               } else {
                  if (Personagem.CompareTag ("Programador4")) {
                     funcao = "PROGRAMADOR";
                     nome = "Noel Meirelles";
                  } else {
                     if (Personagem.CompareTag ("AnalistaTeste")) {
                        funcao = "ANALISTA DE TESTE";
```

```
      nome = "Isaura Bogado";
   } else {
      if (Personagem.CompareTag ("Arquiteto")) {
         funcao = "ARQUITETO";
         nome = "Solano Sequera";
      } else {
         if (Personagem.CompareTag ("DesignerInteracao")) {
            funcao = "DESIGNER DE INTERA\c{C}\'AO";
            nome = "Honorina Sardina";
         } else {
            if (Personagem.CompareTag ("GerenteProjeto")) {
               funcao = "GERENTE DE PROJETO";
               nome = "Trist\~ao T\'evez";
            } else {
               if (Personagem.CompareTag ("RedatorTecnico")) {
                  funcao = "REDATOR T\'ECNICO";
                  nome = "Vit\'oria Gon\c{c}alves";
               } else {
                  if (Personagem.CompareTag ("Cliente1")) {
                     funcao = "CLIENTE";
                     nome = "Jo\~ao Dias";
                  } else {
                     if (Personagem.CompareTag ("Cliente2")) {
                        funcao = "CLIENTE";
                        nome = "Lu\'is Ribeiro";
                     } else {
                        funcao = "EXECUTIVO";
                        nome = "T\^ania Martins";
                     }
                  }
               }
            }
         }
      }
   }
```

```
                }
            }
        }
    }
}

public string getFuncao(){
    return funcao;
}

public string getNome(){
    return nome;
}

void Update () {
}
}
}
```

APÊNDICE F – *SCRIPT* PARA GERENCIAR TODO O *GAMEPLAY*

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class Gameplay : MonoBehaviour {
   Text tempoTexto;
   public static float tempo, multiplicador, tempoReuniao,
       tempoMostrarAtividadeReuniaoInicial;
   int fimDia, atividadeAtualReuniciaoInicial;
   public static int qtdTerminaramReuniao;
   public static int cameraValue;
   bool ativarMenuFimDia, respFimDia, ativarMenuFimSemana, respFimSemana,
       respProgramacaoPar, fimReuniao;
   public static bool reuniaoInicial, reuniaoSemanal, reuniaoFimProjeto,
       programacaoPar, fadeOut, terminarReuniao, stop;
   public static GameObject objFimDia, objFimSemana, objProgramacaoPar,
       objFimJogoFalha, objFimJogoReuniao, Fade, Atv1, Atv2, Atv3, Atv4,
       mensagemFinalProjeto;
   public static float comunicacao, simplicidade, feedback, coragem, respeito;
   Light luzSol;
   GameObject QuadroA1, QuadroA2, QuadroA3, QuadroA4, QuadroL1, QuadroL2,
       QuadroL3;
   public static GameObject botaoANA, botaoARQ, botaoCLI, botaoEXE, botaoDES,
       botaoGER, botaoPRO, botaoRED, botaoRequisitos,
       botaoHistoriasAtividades, menuAtividadeReuniaoInicial;
   public static Text textoTituloAtividadeReuniaoInicial,
       textoCorpoAtividadeReuniaoInicial;
   private Slider sliderCOM, sliderSIM, sliderFED, sliderCOR, sliderRES;
   private Image imgCOM, imgSIM, imgFED, imgCOR, imgRES;
```

```
AudioSource audio;

public static Slider iniciaSlider(string tag, float value){
   GameObject go = GameObject.FindGameObjectWithTag(tag);
   if(go != null){
      Slider s = go.GetComponent<Slider> ();
      if(s != null)
         s.value = value;
      return s;
   }
   return null;
}

Color avaliaCor(Slider s){
   if (s != null) {
      if (s.value < 70) {
         if (s.value < 30) {
            return Color.red;
         } else {
            return Color.yellow;
         }
      } else {
         return Color.green;
      }
   }
   return Color.black;
}

public void alteraCOM(float num){
   comunicacao += num;
}
public void alteraSIM(float num){
   simplicidade += num;
}
public void alteraFED(float num){
```

```
    feedback += num;
}
public void alteraCOR(float num){
    coragem += num;
}
public void alteraRES(float num){
    respeito += num;
}

public void avaliaBotoesEscolhaPersonagem(bool rI, bool rS, bool rF){
    if (!rI && !rS && !rF) {
        botaoANA.GetComponent<Button> ().interactable = true;
        botaoARQ.GetComponent<Button> ().interactable = true;
        botaoCLI.GetComponent<Button> ().interactable = false;
        botaoEXE.GetComponent<Button> ().interactable = false;
        botaoDES.GetComponent<Button> ().interactable = true;
        botaoGER.GetComponent<Button> ().interactable = true;
        botaoPRO.GetComponent<Button> ().interactable = true;
        botaoRED.GetComponent<Button> ().interactable = true;
    } else {
        if (rI) {
            botaoANA.GetComponent<Button> ().interactable = false;
            botaoARQ.GetComponent<Button> ().interactable = false;
            botaoCLI.GetComponent<Button> ().interactable = true;
            botaoEXE.GetComponent<Button> ().interactable = true;
            botaoDES.GetComponent<Button> ().interactable = false;
            botaoGER.GetComponent<Button> ().interactable = true;
            botaoPRO.GetComponent<Button> ().interactable = false;
            botaoRED.GetComponent<Button> ().interactable = true;
        } else {
            if (rS) {
                botaoANA.GetComponent<Button> ().interactable = true;
                botaoARQ.GetComponent<Button> ().interactable = false;
                botaoCLI.GetComponent<Button> ().interactable = false;
                botaoEXE.GetComponent<Button> ().interactable = false;
```

```
            botaoDES.GetComponent<Button> ().interactable = false;
            botaoGER.GetComponent<Button> ().interactable = true;
            botaoPRO.GetComponent<Button> ().interactable = true;
            botaoRED.GetComponent<Button> ().interactable = true;
         } else {
            botaoANA.GetComponent<Button> ().interactable = true;
            botaoARQ.GetComponent<Button> ().interactable = true;
            botaoCLI.GetComponent<Button> ().interactable = true;
            botaoEXE.GetComponent<Button> ().interactable = false;
            botaoDES.GetComponent<Button> ().interactable = true;
            botaoGER.GetComponent<Button> ().interactable = true;
            botaoPRO.GetComponent<Button> ().interactable = true;
            botaoRED.GetComponent<Button> ().interactable = true;
         }
      }
   }
}

void Start () {
   tempo = 3600 * 7;
   multiplicador = 1000.0f;
   tempoReuniao = 0.0f;
   tempoMostrarAtividadeReuniaoInicial = 0.0f;
   fimDia = 17;
   atividadeAtualReuniciaoInicial = 1;
   qtdTerminaramReuniao = 0;
   cameraValue = 5;
   ativarMenuFimDia = true;
   respFimDia = true;
   ativarMenuFimSemana = true;
   respFimSemana = true;
   respProgramacaoPar = false;
   fimReuniao = false;
   reuniaoInicial = true;
   reuniaoSemanal = false;
```

```
reuniaoFimProjeto = false;
programacaoPar = false;
fadeOut = false;
terminarReuniao = false;
stop = false;
comunicacao = 100.0f;
simplicidade = 100.0f;
feedback = 100.0f;
coragem = 100.0f;
respeito = 100.0f;
Time.timeScale = 0;
tempoTexto = GameObject.FindGameObjectWithTag
    ("textoTempo").GetComponent<Text>();
luzSol = GameObject.FindGameObjectWithTag ("Sol").GetComponent<Light>();
sliderCOM = iniciaSlider ("ValorCOM", 100.0f);
sliderSIM = iniciaSlider ("ValorSIM", 100.0f);
sliderFED = iniciaSlider ("ValorFED", 100.0f);
sliderCOR = iniciaSlider ("ValorCOR", 100.0f);
sliderRES = iniciaSlider ("ValorRES", 100.0f);
imgCOM =
    GameObject.FindGameObjectWithTag("ValorCOMCor").GetComponent<Image>
    ();
imgSIM =
    GameObject.FindGameObjectWithTag("ValorSIMCor").GetComponent<Image>
    ();
imgFED =
    GameObject.FindGameObjectWithTag("ValorFEDCor").GetComponent<Image>
    ();
imgCOR =
    GameObject.FindGameObjectWithTag("ValorCORCor").GetComponent<Image>
    ();
imgRES =
    GameObject.FindGameObjectWithTag("ValorRESCor").GetComponent<Image>
    ();
QuadroA1 = GameObject.FindGameObjectWithTag ("QuadroA1");
```

```
    QuadroA1.SetActive (false);
    QuadroA2 = GameObject.FindGameObjectWithTag ("QuadroA2");
    QuadroA2.SetActive (false);
    QuadroA3 = GameObject.FindGameObjectWithTag ("QuadroA3");
    QuadroA3.SetActive (false);
    QuadroA4 = GameObject.FindGameObjectWithTag ("QuadroA4");
    QuadroA4.SetActive (false);
    QuadroL1 = GameObject.FindGameObjectWithTag ("QuadroL1");
    QuadroL1.SetActive (false);
    QuadroL2 = GameObject.FindGameObjectWithTag ("QuadroL2");
    QuadroL2.SetActive (false);
    QuadroL3 = GameObject.FindGameObjectWithTag ("QuadroL3");
    QuadroL3.SetActive (false);
    botaoRequisitos = GameObject.FindGameObjectWithTag ("botaoRequisitos");
    botaoHistoriasAtividades = GameObject.FindGameObjectWithTag
        ("botaoHistoriasAtividades");
    audio = GetComponent<AudioSource> ();
}

void Update () {
    Transform m_Cam = Camera.main.transform;
    Vector3 v = new Vector3 ();
    switch (cameraValue) {
    case 1:
        v.Set (-3.6f, 2.1f, -5.7f);
        m_Cam.position = v;
        m_Cam.rotation = Quaternion.Euler (20, 45, 0);
        break;
    case 2:
        v.Set (-3.6f, 2.1f, 5.7f);
        m_Cam.position = v;
        m_Cam.rotation = Quaternion.Euler (20, 135, 0);
        break;
    case 3:
        v.Set (3.6f, 2.1f, 5.7f);
```

```
            m_Cam.position = v;
            m_Cam.rotation = Quaternion.Euler (20, 225, 0);
            break;
        case 4:
            v.Set (1.2f, 1.35f, 4.4f);
            m_Cam.position = v;
            m_Cam.rotation = Quaternion.Euler (0, 0, 0);
            break;
        case 5:
            v.Set (3.6f, 2.1f, -5.7f);
            m_Cam.position = v;
            m_Cam.rotation = Quaternion.Euler (20, 315, 0);
            break;
        case 6:
            v.Set (1, 2.1f, -1);
            m_Cam.position = v;
            m_Cam.rotation = Quaternion.Euler (20, 0, 0);
            break;
        case 7:
            v.Set (0, 2.1f, 2);
            m_Cam.position = v;
            m_Cam.rotation = Quaternion.Euler (20, 315, 0);
            break;
        }
        if (!reuniaoInicial) {
            botaoRequisitos.GetComponent<Button> ().interactable = true;
            botaoHistoriasAtividades.GetComponent<Button> ().interactable = true;
        } else {
            botaoRequisitos.GetComponent<Button> ().interactable = false;
            botaoHistoriasAtividades.GetComponent<Button> ().interactable = false;
        }
        avaliaBotoesEscolhaPersonagem (reuniaoInicial, reuniaoSemanal,
            reuniaoFimProjeto);
        if ((!reuniaoFimProjeto) && (!reuniaoSemanal) && (!reuniaoInicial)){
            fimReuniao = false;
```

```
terminarReuniao = false;
stop = false;
qtdTerminaramReuniao = 0;
fadeOut = true;
tempo += (Time.deltaTime * multiplicador);
int dia = (int)(tempo / 86400),
hora = (int)(tempo / 3600) % 24,
min = (int)(tempo / 60) % 60;
tempoTexto.text = "Dia " + dia + "\nHora " + hora;
luzSol.intensity = 1.0f - ((((hora-7.0f) * 60.0f) + min) / 1200.0f);
RenderSettings.ambientIntensity = luzSol.intensity + 0.0f;
RenderSettings.reflectionIntensity = luzSol.intensity + 0.0f;
if (((dia == 0) || (dia == 7) || (dia == 14) || (dia == 21) || (dia ==
    28)) && (hora == 7)) {
   Color cAtv1 = Atv1.GetComponent<Image> ().color,
   cAtv2 = Atv2.GetComponent<Image> ().color,
   cAtv3 = Atv3.GetComponent<Image> ().color,
   cAtv4 = Atv4.GetComponent<Image> ().color;
   if (dia == 7) {
      QuadroA1.SetActive (true);
      cAtv1.a = 0.0f;
      cAtv2.a = 1.0f;
   } else {
      if (dia == 14) {
         cAtv2.a = 0.0f;
         cAtv3.a = 1.0f;
         QuadroA2.SetActive (true);
         QuadroL1.SetActive (true);
      } else {
         if (dia == 21) {
            cAtv3.a = 0.0f;
            cAtv4.a = 1.0f;
            QuadroA3.SetActive (true);
            QuadroL2.SetActive (true);
         } else {
```

```
            if (dia == 28) {
               QuadroA4.SetActive (true);
               QuadroL3.SetActive (true);
            }
         }
      }
   }
   Atv1.GetComponent<Image> ().color = cAtv1;
   Atv2.GetComponent<Image> ().color = cAtv2;
   Atv3.GetComponent<Image> ().color = cAtv3;
   Atv4.GetComponent<Image> ().color = cAtv4;
   if (dia != 28){
      if (!respProgramacaoPar) {
         if (Time.timeScale == 1) {
            objProgramacaoPar.SetActive (true);
            Time.timeScale = 0;
         }
         respProgramacaoPar = true;
      }
   }
}
if (((dia == 6) || (dia == 13) || (dia == 20)) && (hora == fimDia)) {
   programacaoPar = false;
   if (ativarMenuFimSemana) {
      if (Time.timeScale == 1) {
         objFimSemana.SetActive (true);
         Time.timeScale = 0;
      }
   } else {
      if (Time.timeScale == 1) {
         reuniaoSemanal = respFimSemana;
         if (!respFimSemana) {
            alteraCOM (-18.75f);
            alteraSIM (-11.25f);
            alteraFED (-33.75f);
```

```
            }
         }
      }
      respProgramacaoPar = false;
   }
   if ((hora == fimDia) || (hora == fimDia + 1)) {
      if (ativarMenuFimDia) {
         if (Time.timeScale == 1) {
            if (!programacaoPar) {
               alteraCOM (-UnityEngine.Random.Range (0.5f, 5.5f));
               alteraSIM (-UnityEngine.Random.Range (0.5f, 5.5f));
               alteraFED (-UnityEngine.Random.Range (0.5f, 5.5f));
               alteraCOR (-UnityEngine.Random.Range (0.5f, 5.5f));
               alteraRES (-UnityEngine.Random.Range (0.5f, 5.5f));
            } else {
               alteraCOM (-UnityEngine.Random.Range (0.0f, 1.0f));
               alteraSIM (-UnityEngine.Random.Range (0.0f, 1.0f));
               alteraFED (-UnityEngine.Random.Range (0.0f, 1.0f));
               alteraCOR (-UnityEngine.Random.Range (0.0f, 1.0f));
               alteraRES (-UnityEngine.Random.Range (0.0f, 1.0f));
            }

            objFimDia.SetActive (true);
            Time.timeScale = 0;
         }
      } else {
         if (Time.timeScale == 1) {
            if (!programacaoPar) {
               alteraCOM (-UnityEngine.Random.Range (0.5f, 5.5f));
               alteraSIM (-UnityEngine.Random.Range (0.5f, 5.5f));
               alteraFED (-UnityEngine.Random.Range (0.5f, 5.5f));
               alteraCOR (-UnityEngine.Random.Range (0.5f, 5.5f));
               alteraRES (-UnityEngine.Random.Range (0.5f, 5.5f));
            } else {
               alteraCOM (-UnityEngine.Random.Range (0.0f, 1.0f));
```

```
            alteraSIM (-UnityEngine.Random.Range (0.0f, 1.0f));
            alteraFED (-UnityEngine.Random.Range (0.0f, 1.0f));
            alteraCOR (-UnityEngine.Random.Range (0.0f, 1.0f));
            alteraRES (-UnityEngine.Random.Range (0.0f, 1.0f));
         }
         tempo += 3600 * 14;
         if (!respFimDia) {
            alteraCOM (-2.5f);
            alteraSIM (-1.5f);
            alteraFED (-4.5f);
         }
      }
   }
}
if (sliderCOM != null)
   sliderCOM.value = comunicacao;
if (sliderSIM != null)
   sliderSIM.value = simplicidade;
if (sliderFED != null)
   sliderFED.value = feedback;
if (sliderCOR != null)
   sliderCOR.value = coragem;
if (sliderRES != null)
   sliderRES.value = respeito;
GameObject[] go;
go = GameObject.FindGameObjectsWithTag ("t1");
if (go != null) {
   foreach (GameObject g in go) {
      Toggle t = g.GetComponent<Toggle> ();
      t.isOn = !ativarMenuFimDia;
   }
}
go = GameObject.FindGameObjectsWithTag ("t2");
if (go != null) {
   foreach (GameObject g in go) {
```

```
            Toggle t = g.GetComponent<Toggle> ();
            t.isOn = !ativarMenuFimSemana;
        }
    }
    if (dia == 28) {
        Time.timeScale = 0;
        objFimJogoReuniao.SetActive (true);
    }
    if ((comunicacao <= 0.0f) || (simplicidade <= 0.0f) || (feedback <=
        0.0f) || (coragem <= 0.0f) || (respeito <= 0.0f)) {
        Time.timeScale = 0;
        objFimJogoFalha.SetActive (true);
    }
} else {
    if (audio != null && !audio.isPlaying){
        audio.Play ((ulong) UnityEngine.Random.Range (0L, 10000L));
    }
    if (!fimReuniao) {
        Fade.gameObject.GetComponentInChildren<Text> ().text = "Organizando
            reuni\~ao...";
        if (reuniaoInicial) {
            tempoMostrarAtividadeReuniaoInicial += (Time.deltaTime *
                multiplicador);
            if (atividadeAtualReuniciaoInicial < 5) {
                if (tempoMostrarAtividadeReuniaoInicial >
                    UnityEngine.Random.Range (7000.0f, 10000.0f)) {
                    switch (atividadeAtualReuniciaoInicial) {
                    case 1:
                        textoTituloAtividadeReuniaoInicial.text =
                            "HIST\'ORIA/ATIVIDADE 1";
                        textoCorpoAtividadeReuniaoInicial.text = "A primeira
                            hist\'oria tem a seguinte descri\c{c}\~ao sum\'aria:
                            Inserir um aluno. A dura\c{c}\~ao dessa atividade
                            ser\'a de uma semana.";
                        break;
```

```
            case 2:
               textoTituloAtividadeReuniaoInicial.text =
                  "HIST\'ORIA/ATIVIDADE 2";
               textoCorpoAtividadeReuniaoInicial.text = "A segunda
                  hist\'oria consiste em: Buscar uma atividade do
                  evento pelo nome parcial ou total. A dura\c{c}\~ao
                  dessa atividade ser\'a de uma semana.";
               break;
            case 3:
               textoTituloAtividadeReuniaoInicial.text =
                  "HIST\'ORIA/ATIVIDADE 3";
               textoCorpoAtividadeReuniaoInicial.text = "J\'a a
                  terceira hist\'oria deve: Listar todas as atividades
                  do evento, mostrando quantidade de vagas
                  dispon\'iveis. A dura\c{c}\~ao dessa atividade ser\'a
                  de uma semana.";
               break;
            case 4:
               textoTituloAtividadeReuniaoInicial.text =
                  "HIST\'ORIA/ATIVIDADE 4";
               textoCorpoAtividadeReuniaoInicial.text = "A quarta e
                  \'ultima hist\'oria tem a seguinte descri\c{c}\~ao
                  sum\'aria: Efetivar cadastro do aluno em atividade do
                  evento. A dura\c{c}\~ao dessa atividade ser\'a de uma
                  semana.";
               break;
            }
            if (Time.timeScale == 1) {
               menuAtividadeReuniaoInicial.SetActive (true);
               Time.timeScale = 0;
            }
            atividadeAtualReuniciaoInicial++;
            tempoMostrarAtividadeReuniaoInicial = 0.0f;
         }
      } else {
```

```
            if (Time.timeScale == 1) {
               fadeOut = true;
               Color cA = Fade.GetComponent<Image> ().color;
               cA.a = 0.0f;
               Fade.GetComponent<Image> ().color = cA;
               fimReuniao = true;
            }
         }
      } else {
         tempoReuniao += Time.deltaTime;
         if (tempoReuniao > UnityEngine.Random.Range (15.0f, 20.0f)) {
            fadeOut = true;
            Color cA = Fade.GetComponent<Image> ().color;
            cA.a = 0.0f;
            Fade.GetComponent<Image> ().color = cA;
            fimReuniao = true;
         }
      }
   } else {
      Fade.gameObject.GetComponentInChildren<Text> ().text = "Terminando
          reuni\~ao...";
      if (reuniaoFimProjeto) {
         if (Time.timeScale == 1) {
            mensagemFinalProjeto.SetActive (true);
            Fade.gameObject.SetActive (false);
            Time.timeScale = 0;
         }
      }
   }
   if (Time.timeScale == 1) {
      Color c = Fade.GetComponent<Image> ().color;
      if (fadeOut) {
         if (c.a < 2.5f) {
            if (c.a >= 1.0f) {
               if (fimReuniao)
```

```
                terminarReuniao = true;
            }
            Fade.gameObject.SetActive (true);
            c.a += (Time.deltaTime * 1.0f);
            Fade.GetComponent<Image> ().color = c;
         } else {
            fadeOut = false;
         }
      } else {
         if (qtdTerminaramReuniao >= 9) {
            terminarReuniao = false;
            stop = true;
         }
         if (c.a > 0.5f) {
            Fade.gameObject.SetActive (true);
            c.a -= (Time.deltaTime * 1.5f);
            Fade.GetComponent<Image> ().color = c;
         } else {
            Fade.gameObject.SetActive (false);
            if (fimReuniao) {
               tempoReuniao = 0.0f;
               reuniaoSemanal = false;
               reuniaoInicial = false;
               fimReuniao = false;
               stop = false;
               multiplicador = 1000.0f;
               qtdTerminaramReuniao = 0;
               if (audio != null && audio.isPlaying) {
                  audio.Stop ();
               }
            }
         }
      }
   }
}
```

```
        if (imgCOM != null)
            imgCOM.color = avaliaCor (sliderCOM);
        if (imgSIM != null)
            imgSIM.color = avaliaCor (sliderSIM);
        if (imgFED != null)
            imgFED.color = avaliaCor (sliderFED);
        if (imgCOR != null)
            imgCOR.color = avaliaCor (sliderCOR);
        if (imgRES != null)
            imgRES.color = avaliaCor (sliderRES);
    }

    public void setAtivarMenuFimDia(bool value){
        ativarMenuFimDia = !value;
    }

    public void setRespFimDia(bool value){
        respFimDia = value;
    }

    public void setAtivarMenuFimSemana(bool value){
        ativarMenuFimSemana = !value;
    }

    public void setRespFimSemana(bool value){
        respFimSemana = value;
        reuniaoSemanal = value;
    }

    public void setReuniaoFimProjeto(bool value){
        reuniaoFimProjeto = value;
        if(value)
            programacaoPar = false;
    }
```

```
    public static bool getReuniaoFimProjeto(){
        return reuniaoFimProjeto;
    }

    public void setProgramacaoPar(bool value){
        programacaoPar = value;
    }
}
```

APÊNDICE G – GAME DESIGN DOCUMENT - GDD

UNIVERSIDADE ESTADUAL DE MATO GROSSO DO SUL – UEMS

GAME DESIGN DOCUMENT:
eXPlay

Desenvolvedor:
Lucas Freitas do Rosário RGM 27304

Orientação:
Prof.ª M.ª Jéssica Bassani de Oliveira

DOURADOS – MS
2016

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Esse documento mostrará os aspectos técnicos e artísticos do jogo eXPlay. Discorrerá sobre a mecânica do jogo, aspectos de jogabilidade, ferramentas de desenvolvimento, entre outros itens.

Atendendo a definição de um GDD, após o entendimento desse documento, será possível implementar de fato o jogo.

1.1 GAMEPLAY OVERVIEW

Os personagens que compõe o ambiente têm como objetivo desenvolver um software, escolhido pelo jogador dentre as possíveis opções fornecidas pelo jogo, utilizando a metodologia de desenvolvimento ágil Extreme Programming (XP). Sendo assim, o jogo irá conduzir o jogador para aprender essa metodologia, dando liberdade em determinadas condições para escolhas positivas ou negativas. Essas últimas escolhas influenciarão negativamente no desenvolvimento do projeto, o que acarretará numa possível perca no jogo.

Como o objetivo é ensinar a metodologia, em nenhum momento o jogador será induzido ao erro nem será colocado em situações em que ele deveria saber previamente qual a melhor opção a ser escolhida. O jogo será grande parte “auto jogável”, com algumas partes em que o jogador terá que realizar explicitamente alguma ação, porém em todo o processo ele poderá intervir.

O nível de dificuldade dependerá de qual projeto o usuário selecionou, visto que cada um possui uma complexidade relacionada. Contudo, todos os projetos possuem as mesmas etapas, fases de desenvolvimento, com o único intuito de ensinar de forma clara e prática como a metodologia XP trabalha.

1.2 GÊNERO, SEMELHANÇAS E DIFERENÇAS

O jogo é do gênero educativo, sendo seu principal estilo o Role-Playing Game (RPG) em terceira pessoa, uma vez que o jogo tenta simular os personagens que o jogador poderia estar fazendo o papel (Novak, 2012). Devido sua função, ensinar uma metodologia ágil de desenvolvimento de software, a ambientação ocorre em cenários internos, representando escritórios, salas de desenvolvimento e sala de reunião.

Possui semelhanças visuais com o jogo Game Studio Tycoon 2 (Figura 1), desenvolvido por Michael Sherwin (Google Play Store, 2016), o qual tem como

objetivo desenvolver jogos para diversas plataformas, o qual são desbloqueadas com o progresso no jogo. O mesmo acontece com as tecnologias de software e hardware utilizadas para desenvolver o jogo, são desbloqueadas melhorias para essas tecnologias com o decorrer do gameplay.

Figura 1. Tela do jogo Game Studio Tycoon 2.

Por ser um jogo do gênero educativo, sua diferença está justamente nesse aspecto, além de ensinar uma metodologia de desenvolvimento de software.

1.3 PÚBLICO ALVO

Esse jogo tem como público alvo estudantes da área de desenvolvimento de software, bem como desenvolvedores de empresas que querem aprender uma metodologia ágil de desenvolvimento para obter celeridade, organização, dentre outros benefícios para o processo de desenvolvimento. O jogo é uma alternativa para as metodologias convencionais de ensino, podendo aprender a metodologia de forma prática, intuitiva e divertida.

1.4 FLUXO DO JOGO

O fluxo do jogo seguirá o mesmo de um projeto, facilitando o entendimento do mesmo. Então, todos os projetos terão três fases: Planejamento, Projeto + Codificação

+ Teste e Teste de aceitação. Dessa forma, o fluxo conterá todas essas fases o número de interações definidas para cada atividade.

O jogo não induzirá o jogador ao erro, visto que deve ensinar a metodologia. Dessa forma, não cancelando ações e escolhendo as opções visivelmente convenientes e sugestivas, o jogador conseguirá atingir o objetivo final facilmente, aprendendo todas as fases da metodologia. Todavia, apesar do intuito ser ensinar a metodologia, caso o jogador faça muitas escolhas que influenciem negativamente no projeto, os pontos de valores cairão a um nível que o projeto falhará, o que fará o jogador perder o jogo.

1.4.1 PLANEJAMENTO

Sendo a primeira fase do fluxo do jogo, nela o jogador entrará na, também primeira, fase do projeto (planejamento), que inclui as atividades de coleta de requisitos e escrita de histórias em cartões. Ambas atividades serão realizadas na mesa de reunião do ambiente com integrantes específicos para cada uma.

1.4.1.1 COLETA DE REQUISITOS

A coleta de requisitos será realizada mediante reunião na mesa de reuniões do ambiente. Essa reunião será composta dos seguintes membros: **Gerente de projeto, Cliente, Executivo e Redator técnico**. Sendo obrigado ter, no mínimo, um membro de cada função.

O jogo irá conduzir a reunião de forma que o cliente apresente um requisito por vez à equipe, dando tempo suficiente para o jogador ver quais são esses requisitos, os quais serão apresentados em forma de um bloco “popup” com tempo limitado de existência. Outra maneira de ver todos os requisitos já listados pelo cliente, será pelo menu lateral “PROJETO”.

Nesse momento o jogador não terá controle sobre as ações dos personagens, sendo possível somente visualizar as informações do ambiente e dos personagens.

1.4.1.2 ESCRITA DE HISTÓRIAS

Realizada em formato de reunião após o término da coleta de requisitos, também na mesa de reuniões do ambiente, a composição dessa será de: **Designer**

de interação, Gerente de projeto, Executivo, Gerente de produto, Cliente e Redator técnico. Sendo obrigado ter, no mínimo, um membro de cada função.

O jogo apresentará uma história por vez à equipe, e consequentemente ao jogador (em forma de um bloco “popup”). Outra forma de ver todos as histórias escritas, será pelo menu lateral “PROJETO”.

Nesse momento o jogador também não terá controle sobre as ações, somente poderá visualizar informações das mesmas, do ambiente e dos personagens.

1.4.2 PROJETO + CODIFICAÇÃO + TESTE

Nessa fase serão realizadas duas atividades: implementar e testar unicamente as histórias previamente escritas e realizar o teste de integração.

1.4.2.1 IMPLEMENTAÇÃO E TESTE DE HISTÓRIAS

Cada indivíduo estará em seu devido lugar no ambiente, ocupando um espaço contendo uma mesa, cadeira, computador e demais acessórios necessários para a função. Para essa atividade serão necessários os seguintes membros, com quantidade mínima necessária: **Analista de teste (1), Arquiteto (1), Designer de interação (1), Gerente de projeto (1), Programador (2) e Redator técnico (1)**. Devido a eventuais ações que podem ocorrer durante o projeto, alguns outros membros ou outras atividades dos membros já existentes podem ser necessárias:

- Necessidade de refatoração em larga escala: **Arquiteto (1)**;
- Reduzir escopo devido a atrasos: **Gerente de produto (1)**;
- Modificar alguma história ou funcionalidade: **Cliente (1)**;
- Refatorar o sistema: **Programador (2)**.

Essa atividade se inicia com todos os membros trabalhando em seus devidos computadores, sobre todas as histórias que devem ser implementadas, cada uma em sua interação correspondente. O jogador pode cancelar alguma atividade em execução de determinado membro da equipe, influenciando no cronograma e pontos dos valores. Além disso, ele deve tomar algumas decisões durante esse processo, como realizar integração no fim do dia, escolher fazer ou não fazer as reuniões semanais e trimestrais, entre outras. Todas essas escolhas influenciam nos pontos de valores.

1.4.2.2 TESTE DE INTEGRAÇÃO

Cada indivíduo estará em seu devido lugar no ambiente, ocupando um espaço contendo uma mesa, cadeira, computador e demais acessórios necessários para a função. Para essa atividade serão necessários os seguintes membros, com quantidade mínima necessária: **Analista de teste (1), Gerente de projeto (1), Programador (2) e Redator técnico (1)**. Devido a eventuais ações que podem ocorrer durante o projeto, alguns outros membros ou outras atividades dos membros já existentes podem ser necessárias:

- Necessidade de refatoração em larga escala: **Arquiteto (1)**;
- Reduzir escopo devido a atrasos: **Gerente de produto (1)**;
- Modificar alguma história ou funcionalidade: **Cliente (1)**;
- Refatorar o sistema: **Programador (2)**.

Da mesma forma, a atividade será iniciada com todos os membros em seu devido lugar trabalhando em tarefas relacionadas a testes. O jogador também terá escolha de cancelar algumas atividades que estão sendo realizadas.

1.4.3 TESTE DE ACEITAÇÃO

Nessa atividade, o artefato da interação será mostrado ao cliente para avaliação. Sendo assim, o local do ambiente será a mesa de reuniões, onde ficarão os membros: **Designer de interação, Gerente de projeto, Gerente de produto, Redator técnico e Cliente.** Sendo obrigado ter, no mínimo, um membro de cada função.

A equipe mostrará as histórias ou funcionalidades implementadas ao cliente. Essa mostra será vista pelo jogador em formato de janela, cabendo a ele decidir quando fechar a visualização. O cliente poderá ter duas escolhas:

- Aceitar: o projeto segue para as próximas interações caso houver. Senão houver mais interações, o projeto é finalizado.
- Negar: o projeto vai para uma nova interação para a correção do que o usuário sugeriu.

Essa escolha do cliente é aleatória baseada nas escolhas do jogador, ou seja, se o jogador escolheu muitas ações negativas para o projeto, a probabilidade de o

cliente negar o artefato da interação é maior. Em contrapartida, se as escolhas forem positivas, o cliente terá grande probabilidade de aceitar o artefato da interação.

2 INTERFACE E INTERAÇÃO

Esse capítulo descreve como o jogador interage com o dispositivo de entrada, para controle e tomada de decisões, e dispositivo de saída, para visualização das informações produzidas pelo jogo e jogador.

2.1 ENTRADAS

Uma vez que o jogo é para a plataforma mobile Android, o dispositivo original para entrada utilizado será o Touch Screen do próprio aparelho. Então, todos os comandos do jogador para escolher itens dos menus iniciais, dos menus durante o gameplay, interagir com os personagens, interagir com o ambiente, e demais interações serão realizadas através do Touch Screen.

2.2 SAÍDAS

As saídas serão tanto no formato visual, através da tela do aparelho mobile, quanto no formato sonoro, através dos autofalantes do aparelho ou fones de ouvido conectados ao mesmo.

2.2.1 TELAS

A seguir estão protótipos de todas as telas de menu inicial, bem como do gameplay:

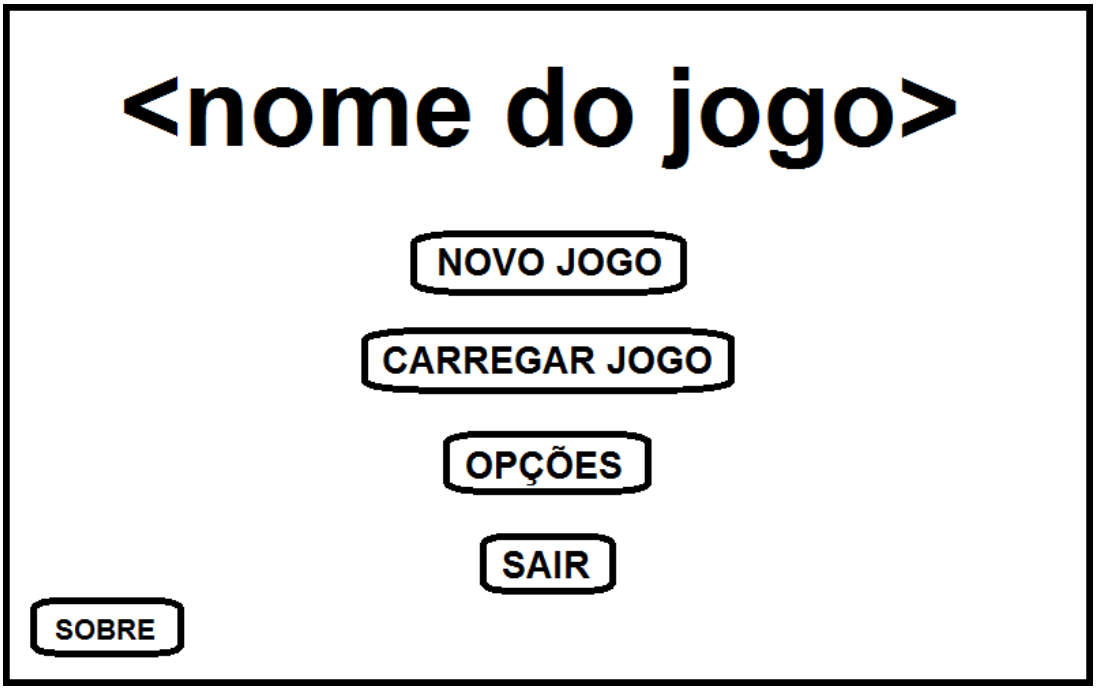


Figura 2. Menu principal do jogo.

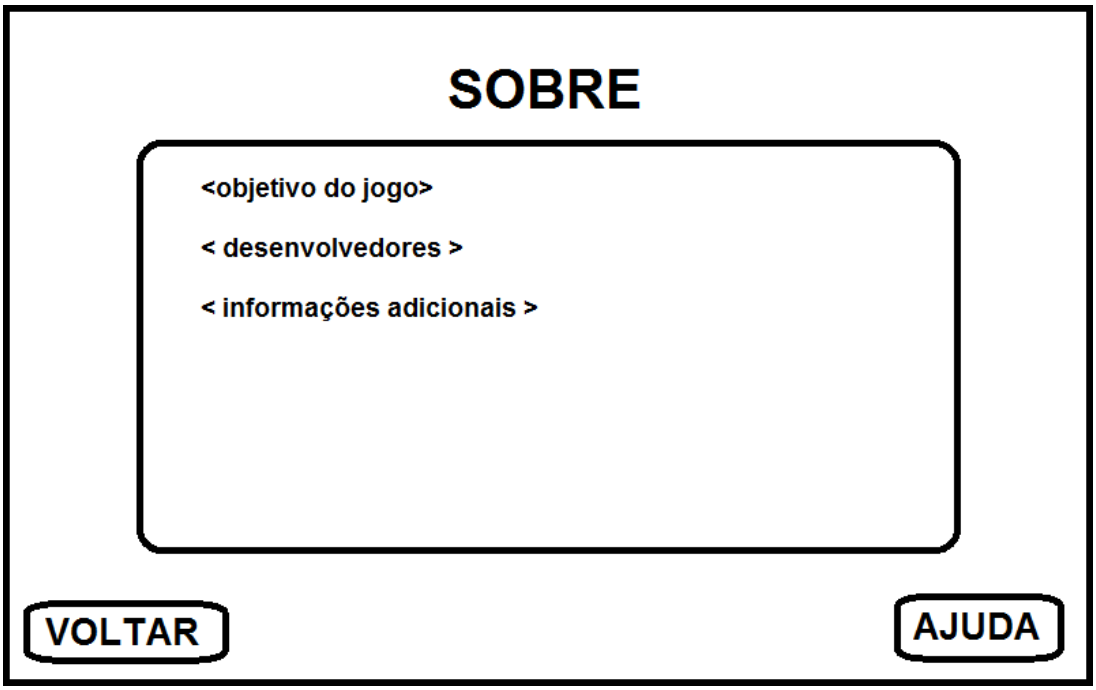


Figura 3. Tela de informações sobre o jogo e quem o desenvolveu.

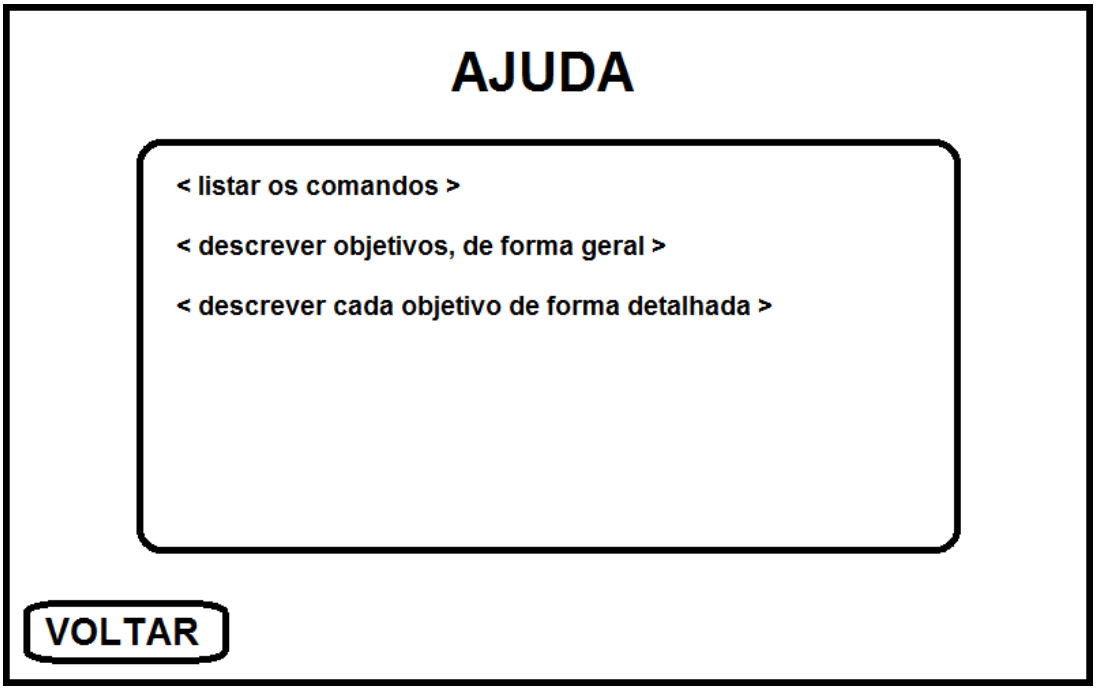


Figura 4. Tela para apresentação de ajuda de todos os aspectos do jogo.

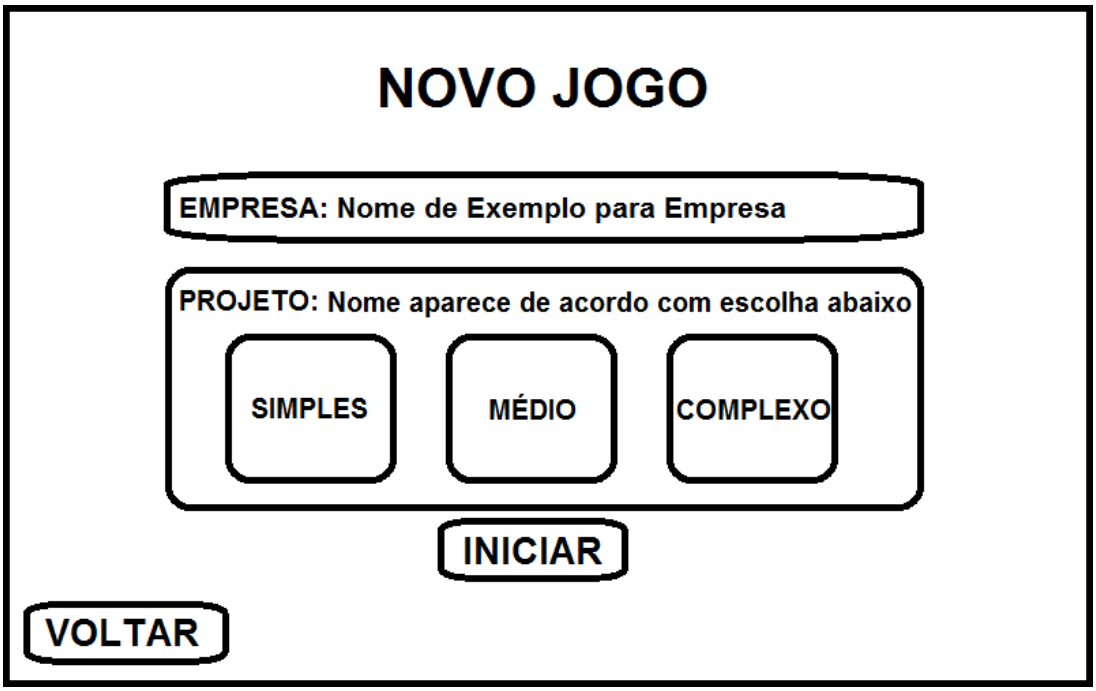


Figura 5. Menu de novo jogo.

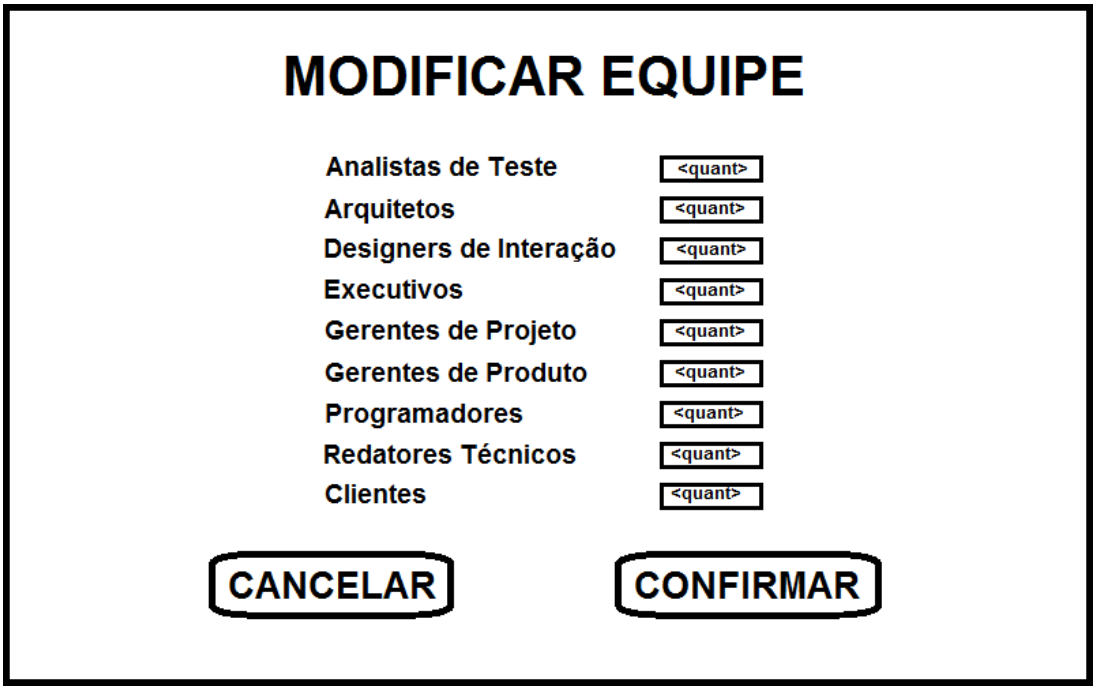


Figura 6. Menu para modificação de quantidade de membros da equipe.

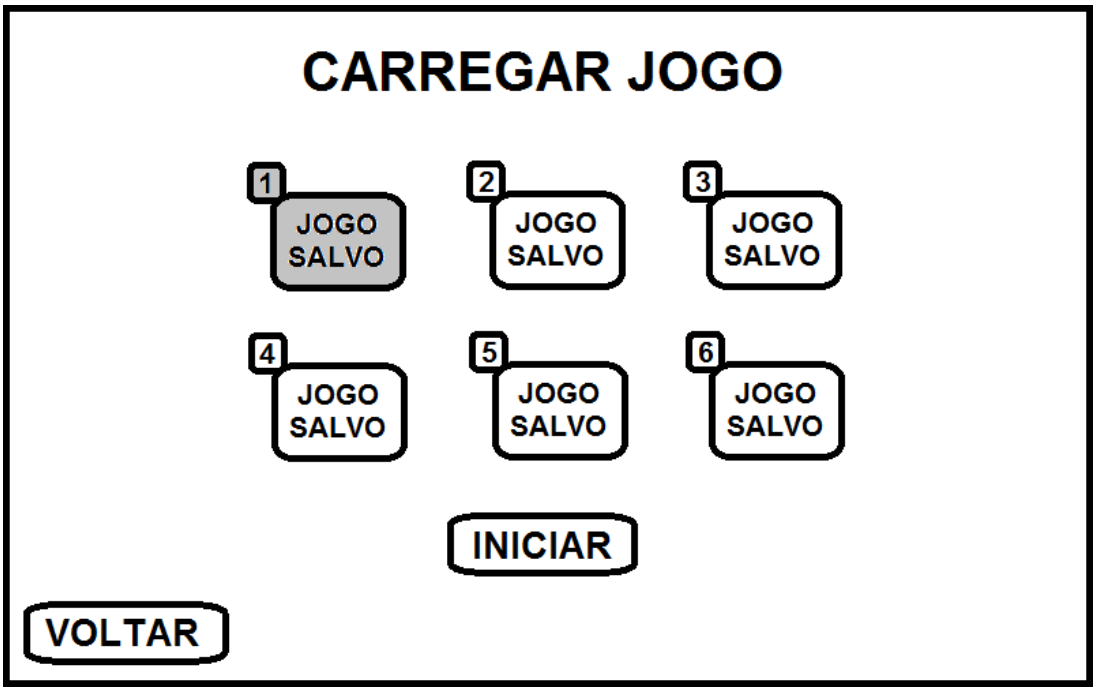


Figura 7. Menu de carregamento de jogo salvo.

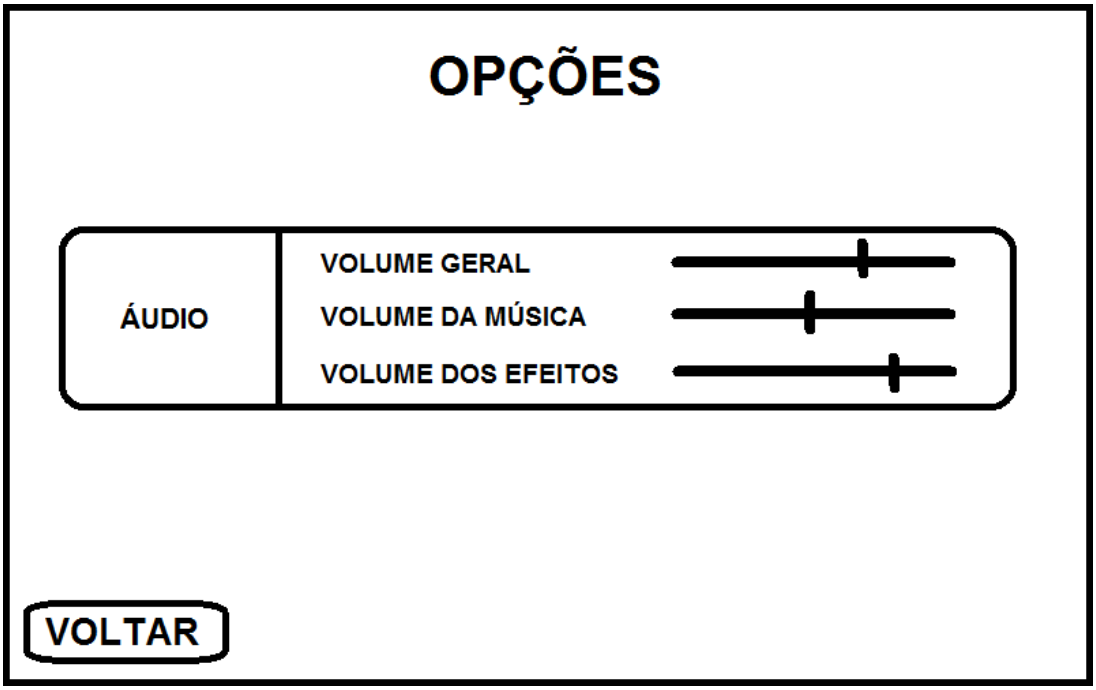


Figura 8. Menu de opções.

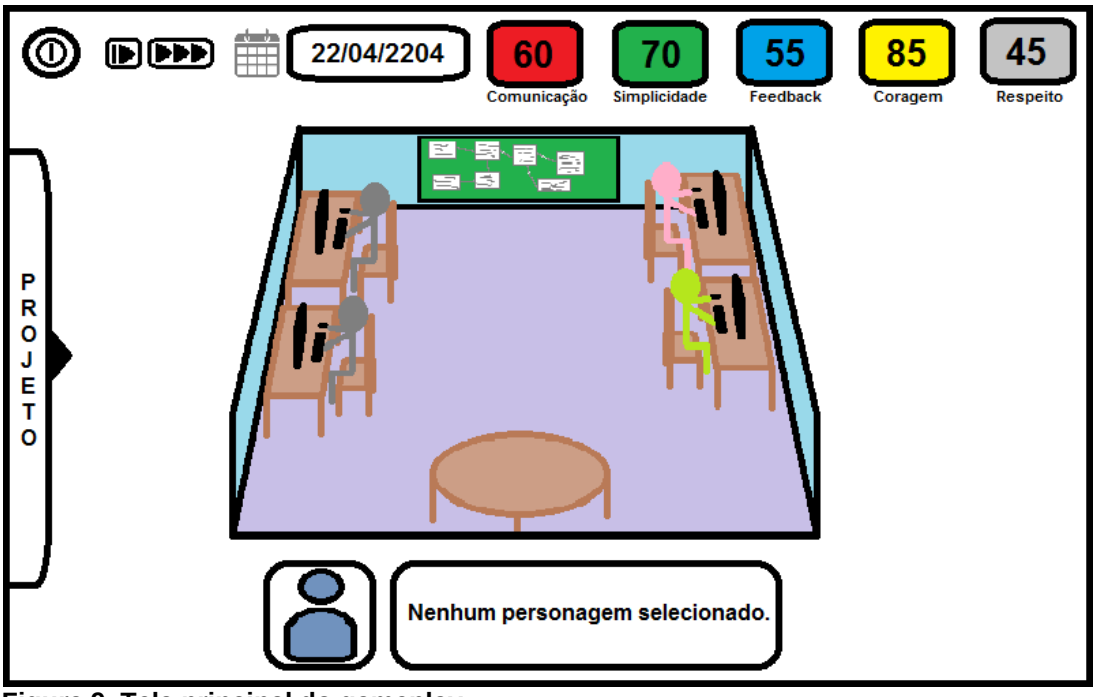


Figura 9. Tela principal do gameplay.

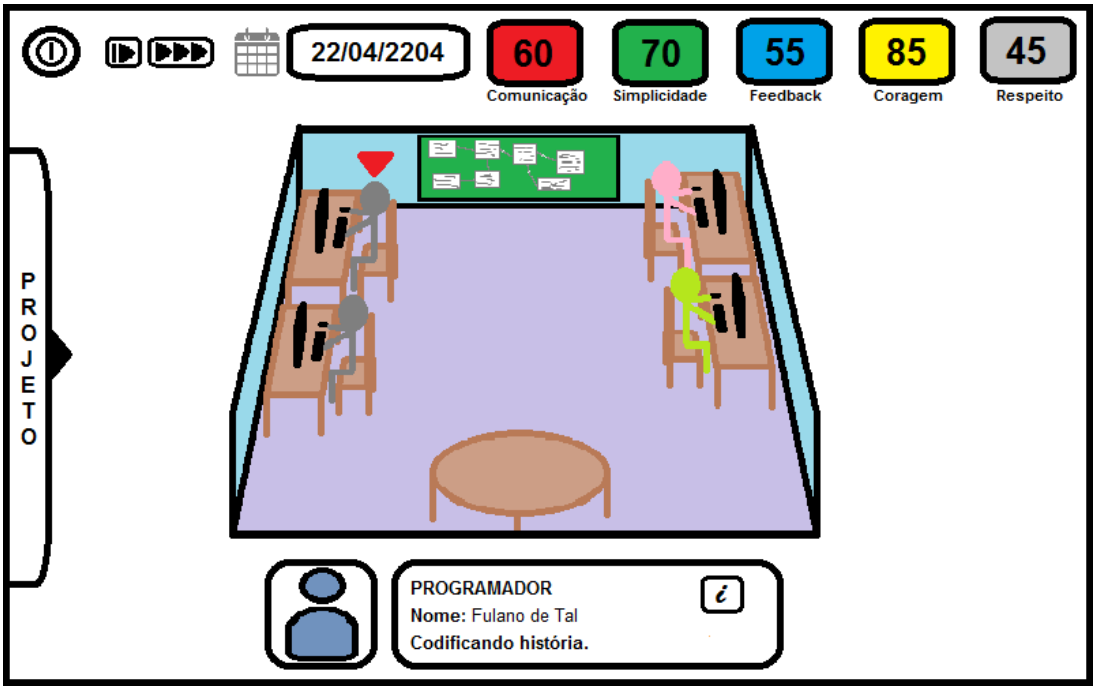


Figura 10. Tela de gameplay após seleção de um personagem.

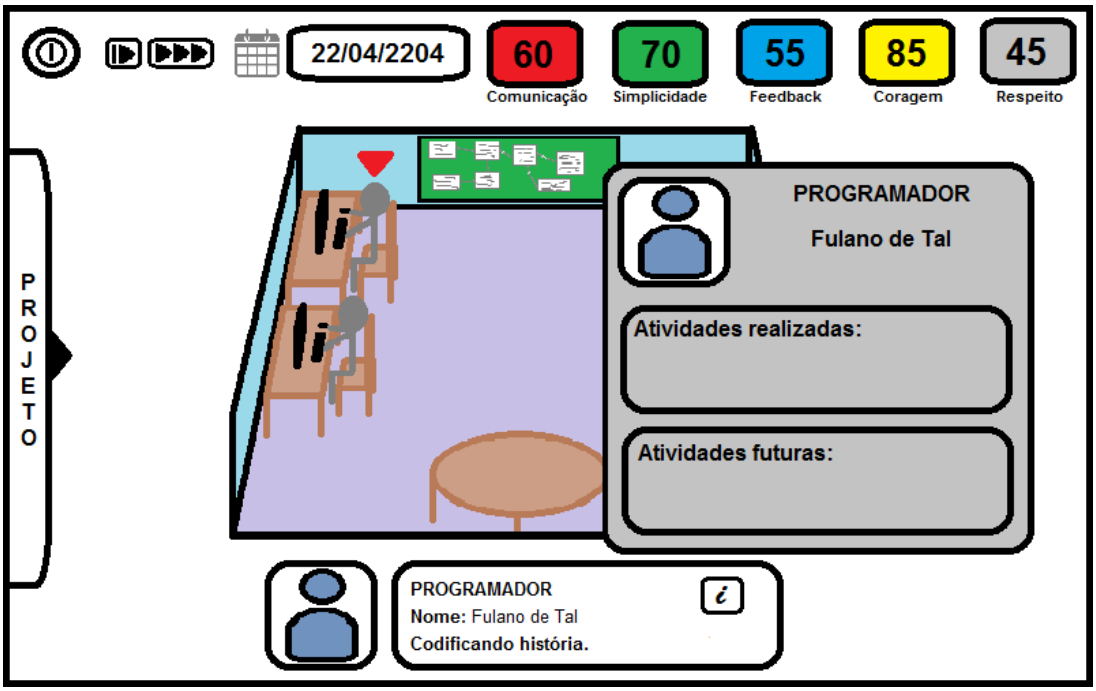


Figura 11. Tela de gameplay de informações do personagem selecionado.

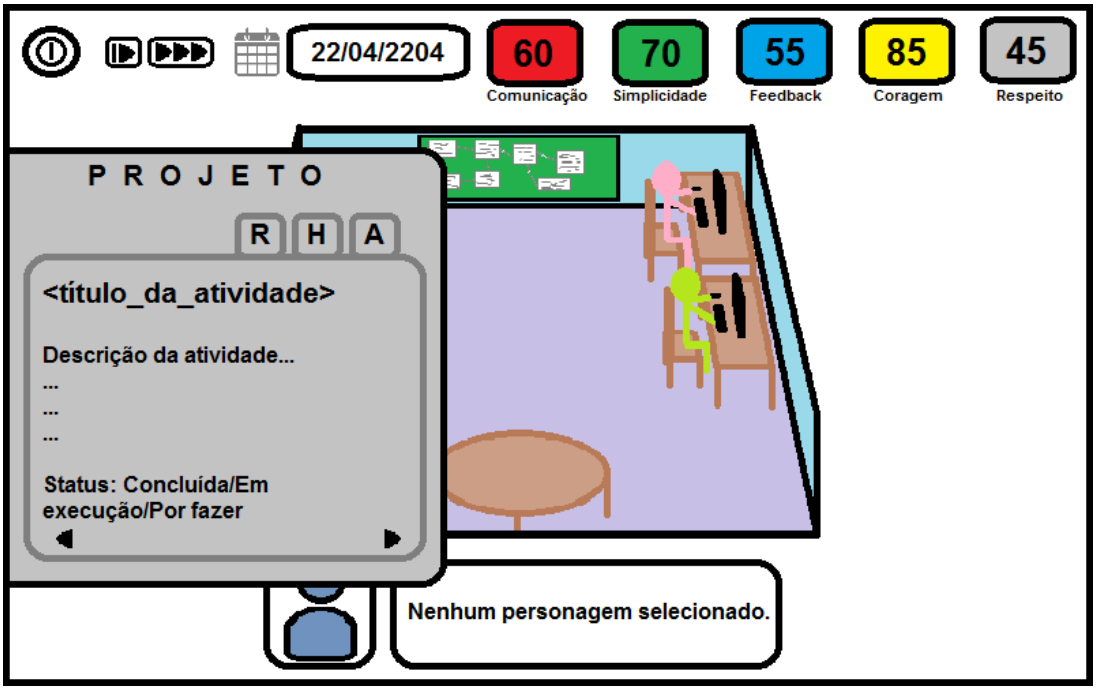


Figura 12. Menu lateral da tela de gameplay.

2.2.2 MENUS

A descrição de todos os menus apresentados na seção anterior é encontrada a seguir:

- **Menu principal**: o jogador poderá selecionar um dos seguintes itens:
  - Novo jogo: Redireciona o jogador para a tela de novo jogo, a fim de iniciar um jogo do começo;
  - Carregar jogo: Redireciona o jogador para a tela de carregar jogo, com o intuito de selecionar um jogo previamente salvo;
  - Opções: Abre um menu para alterar opções em relação aos sons do jogo.
  - Sobre: Mostra informações diversas a respeito do jogo e o desenvolvedor, além de um botão para ajuda, o qual mostra uma tela explicitando todo o jogo;
  - Sair: Sai do jogo.
- **Menu novo jogo**: O jogador insere um nome de empresa de sua escolha e seleciona um tipo de projeto, listados de acordo suas complexidades. O jogador pode voltar para o menu principal ou avançar para a tela de modificar equipe, após ter preenchido todas informações.

- **Menu carregar jogo**: O jogador escolhe um dos jogos salvos listados, podendo voltar para o menu principal ou avançar, carregando o jogo salvo selecionado.

3 MECÂNICA DO JOGO

Esse capítulo discorre a respeito da mecânica do jogo, como os personagens do jogo interagem entre si, com o cenário e com o usuário.

3.1 MECÂNICA BÁSICA

A mecânica básica do jogo é a realização do projeto escolhido pelo jogador, seguindo as atividades metodológicas (etapas) pré-definidas e que correspondem às fases da metodologia XP:

- Planejamento:
  - Colher requisitos
  - Escrita das histórias em cartões:
    - Lista de histórias de acordo com o projeto.
- Projeto + Codificação + Teste:
  - Implementar as histórias na ordem, testando sempre uma a uma após ter implementado.
  - Fazer o teste de integração a cada história adicionada às já implementadas.
- Teste de Aceitação:
  - Testar incrementos com as histórias do cliente.

A realização desse projeto ocorrerá num cenário simulando um ambiente de trabalho voltado à área de desenvolvimento de software. Os personagens, que simulam os membros de uma equipe de desenvolvimento, realizarão as diversas etapas do projeto, podendo o jogador intervir nos momentos permitidos para realizar alguma ação solicitada ou de seu desejo.

As ações que o jogador optar por fazer impactarão na contagem de pontos dos cinco valores da XP: comunicação, simplicidade, feedback, coragem e respeito. Cada opção, bem como o impacto causado está descrito na seção 3.4 desse documento.

3.1.1 AÇÕES DOS PERSONAGENS

Cada personagem (membro da equipe) tem um conjunto de funções específicas durante o processo de desenvolvimento:

- **Analista de teste:**
  1. Ajudar o cliente e os programadores a criar testes para as histórias no início da interação.
  2. Automatizar os testes durante a interação.

- **Arquiteto:**
  1. Ajudar os programadores na programação em par.
  2. Ajudar na refatoração em larga escala.

- **Designer de interação:**
  1. Ajudar o cliente a escrever as histórias.
  2. Criar e refinar interfaces.
  3. Avaliar o uso das funcionalidades pelos clientes.

- **Executivo:**
  1. Ajudar na definição do escopo do projeto.
  2. Comunicar objetivos, alinhando as histórias a eles.

- **Gerente de projeto:**
  1. Facilitar a comunicação entre a equipe e cliente.
  2. Monitorar o progresso da equipe.

- **Gerente de produto:**
  1. Definir histórias para tornar o produto coerente.
  2. Reduzir o escopo, se houver atraso.
  3. Verificar a prontidão das funcionalidades.

- **Programador:**
  1. Implementar histórias.
  2. Automatizar tarefas e testes.

3. Refatorar o sistema.

- **Redator técnico:**
  1. Ajudar a equipe criar e manter a documentação do projeto.

- **Cliente:**
  1. Descrever requisitos.
  2. Escrever histórias.
  3. Tomar decisões durante o processo de desenvolvimento.

Como descrito no capítulo Fluxo do jogo, as atividades metodológicas que os projetos do jogo irão realizar são: Planejamento, Projeto + Codificação + Teste e Teste de aceitação. Cada uma dessas fases tem os personagens, com sua quantidade mínima ou estrita, que necessitam fazer as suas tarefas para que tal fase se complete. Para cada fase, os personagens necessitam realizar as seguintes tarefas (ações):

- **Planejamento:**
  - **Coleta de requisitos:** Gerente de projeto (1,2); Cliente (1); Executivo (1); Redator técnico (1);

  - **Escrita de histórias:** Designer de interação (1); Gerente de projeto (1,2); Executivo (2); Gerente de produto (1); Cliente (2); Redator técnico (1);

- **Projeto + Codificação + Teste:**
  - **Implementar história e testá-la:** Analista de teste (1,2); Arquiteto (1); Designer de interação (2); Gerente de projeto (2); Programador (1,2); Redator técnico (1);
    Ocasionais: Arquiteto (2); Gerente de produto (2); Cliente (3); Programador (3);

  - **Teste de integração:** Analista de teste (2); Gerente de projeto (2); Programador (2); Redator técnico (1);

Ocasionais: Arquiteto (2); Gerente de produto (2); Programador (3); Cliente (3);

- **Teste de aceitação:**
  - ❖ **Testar artefatos dos incrementos com o cliente:** Designer de interação (3); Gerente de projeto (1,2); Gerente de produto (3); Redator técnico (1); Cliente (3);

3.1.2 PENSAMENTOS DOS PERSONAGENS

Durante o decorrer do gameplay, para o jogo ficar mais interativo e humanizado, os personagens irão se comunicar com o jogador através de pensamentos que expressam a tarefa (ação) que esse personagem está fazendo naquele momento. Esses pensamentos serão mostrados em uma janela específica na tela, não produzindo qualquer som de fala correspondente. Os seguintes pensamentos foram escolhidos para cada personagem:

- **Analista de teste:**
  1. Ajudar cliente e desenvolvedor a criar testes para histórias no início da interação:
     - “Estou criando testes para histórias. ”
     - “Estou trabalhando em testes para as histórias. ”
     - “As histórias precisam de testes... ”
  2. Automatizar testes durante interação:
     - “Trabalhando na automatização de testes. ”
     - “Reduzindo trabalho com a automatização. ”

- **Arquiteto:**
  1. Ajudar desenvolvedores na programação em par:
     - “Escolhendo pares para programação. ”
     - “Definindo programadores em par. ”
     - “Alguém precisa de ajuda? ”
  2. Ajudar na refatoração em larga escala:
     - “Ufa! Estou refatorando, em grande escala, o sistema. ”

- “Quantas melhorias precisam aqui? ”

- **Designer de interação:**
  1. Ajudar cliente a escrever história:
     - “Auxiliando o cliente a escrever uma história. ”
     - “Precisa de ajuda? ”
  2. Criar e refinar interfaces:
     - “Trabalhando com interfaces. ”
     - “Amo desenhar! ”
  3. Avaliar o uso das funcionalidades pelos clientes:
     - “Como as funcionalidades estão sendo usadas? ”
     - “Avaliando funcionalidades. ”

- **Executivo:**
  1. Ajudar na definição do escopo do projeto:
     - “Definindo o escopo do projeto. ”
  2. Comunicar objetivos, alinhando as histórias a eles:
     - “Preciso informar os objetivos à equipe! ”

- **Gerente de projeto:**
  1. Facilitar a comunicação entre a equipe e cliente:
     - “Sou uma ponte de comunicação? ”
     - “Auxiliando comunicação entre equipe e cliente. ”
  2. Monitorar o progresso da equipe:
     - “Monitorando a equipe. ”
     - “Hmm... Todos estão trabalhando. ”
     - “Ei, você! Feche esse campo minado... ”

- **Gerente de produto:**
  1. Definir histórias para tornar o produto coerente:
     - “Definindo histórias. ”
     - “Como definir essa história? ”
  2. Reduzir o escopo, se houver atraso:

- “Será que assim não atrasa? ”
- “Reduzindo escopo. ”
- “Puff... Sempre atrasam. ”

3. Verificar a prontidão das funcionalidades:
   - “Verificando funcionalidade. ”
   - “Parece que está funcionando. ”

- **Programador:**

1. Implementar histórias:
   - “Codificando história. ”
   - “Preciso de café! ”
   - “1001110110110”
2. Automatizar tarefas e testes:
   - “Automatizando. ”
   - “Reduzindo trabalho. ”
3. Refatorar o sistema:
   - “Refatorando o sistema. ”
   - “Isso precisa de melhorias... ”

- **Redator técnico:**

1. Ajudar equipe a criar e manter documentação do projeto:
   - “Documentando projeto. ”
   - “Preciso detalhar mais esse aspecto. ”
   - “É necessário documentar isso? ”

- **Cliente:**

1. Descrever requisitos:
   - “Esse programa vai controlar meu estabelecimento. ”
   - “Quero essa cor! ”
   - “Tem que fazer um quadrado, mas num formato triangular. ”
   - “Descrevendo requisitos. ”
2. Escrever histórias:
   - “Era uma vez... ”

- “Escrevendo histórias. ”

3. Tomar decisões durante o processo de desenvolvimento:
   - “Quero adicionar mais isso. ”
   - “Não gostei, faltou isso. ”
   - “É, dá para aceitar. ”
   - “Ok. ”

3.2 PROJETOS

Os projetos que o usuário poderá escolher possui um dentre três níveis de dificuldade: simples, médio ou complexo. Cada projeto contém um conjunto pré-definido de atividades, bem como seu cronograma.

3.2.1 DESCRIÇÃO

Baseado nos níveis de dificuldades, os projetos disponíveis, com suas respectivas atividades são:

- **SIMPLES:**
  - Site para cadastro de alunos em evento:
    - Cadastrar alunos
    - Listar atividades do evento, bem como vagas disponíveis

- **MÉDIO:**
  - Jogo para celular (Asteroid simples):
    - Iniciar novo jogo
    - Salvar progresso em nuvem
    - Comprar itens de melhoria de nave, armas, etc...

- **COMPLEXO:**
  - Sistema para biblioteca universitária:
    - Cadastrar livros, artigos, publicações em geral
    - Empréstimo e devolução de livros
    - Integrar com sistema de cadastro de alunos já existente na universidade
    - Lista de livros mais ou menos utilizados, entre outras

3.2.2 HISTÓRIAS

A partir da descrição abstrata das atividades, foram desenvolvidas as histórias que de fato serão desenvolvidas dentro do jogo para cada projeto. Para todos os projetos, as fases serão idênticas, com exceção que cada projeto realizará as atividades sobre suas histórias. Então, as histórias para projeto são:

- **SIMPLES:**
    - Site para cadastro de alunos em evento:
        - Inserir um aluno
            - Nome, e-mail
        - Buscar uma atividade do evento pelo nome parcial ou total.
        - Listar todas as atividades do evento, mostrando quantidade de vagas disponíveis
        - Efetivar cadastro do aluno em atividade do evento.

- **MÉDIO:**
    - Jogo para celular (Asteroid simples):
        - Modelar nave e obstáculos.
        - Interagir (desviar/destruir) nave com os obstáculos.
        - Implementar vários tipos de naves e armas.
        - Implementar opção para salvar progresso na nuvem.
        - Função para comprar itens de melhoria para nave e armas.

- **COMPLEXO:**
    - Sistema para biblioteca universitária:
        - Inserir um livro, artigo, entre outros (título, autor, editora, ano, data de inserção)
        - Buscar um livro, artigo, entre outros pelo título, autor, editora ou ano.
        - Remover um livro dos registros
        - Cadastrar empréstimo e devolução de livros para estudantes
        - Emitir alerta ao estudante sobre prazos

- Integrar o sistema com o sistema de alunos da universidade, facilitando a verificação de quem é aluno ou não
- Listar livros, artigo e publicações
- Listar os mais e/ou menos utilizados
- Listar quantidade disponível para empréstimo

3.2.3 CRONOGRAMA E PROGRESSÃO DO JOGO

No início de cada interação é feito o "jogo do planejamento", isto é, o cliente determina quais histórias são prioritárias, quantas serão implementadas em cada interação e qual o tempo da interação. Devido ao intuito didático, os projetos do jogo possuem poucas histórias, então a prioridade é definida de acordo com a ordem em que estão colocadas, sendo realizadas/implementadas também nessa ordem. Dessa forma, para cada projeto será definido (informado ao jogador, pois é uma opção pré-definida) uma única vez (na primeira reunião) a quantidade de histórias por interação.

Estando isso definido, o jogador saberá até quando irá o projeto. Mas como em cada interação tem o teste de aceitação, pode ocorrer de no último incremento o software não estar de acordo com as histórias, sendo necessária uma ou mais interações, aumentando a duração do projeto.

Dito isso, o cronograma poderá ser definido da seguinte forma:

- **SIMPLES:**
  - Site para cadastro de alunos em evento:
    - **Duração de cada interação:** 1 semana
    - **Quantidade de histórias:** 4 histórias
    - **Quantidade de histórias por interação:** 2 histórias
    - **Quantidade de interações prováveis:** 2 interações
    - **Tempo provável de conclusão de interações:** 2 semanas
    - **Tempo para teste de aceitação:** 1 semana
    - **Tempo provável total:** 4 semanas

- **MÉDIO:**
  - Jogo para celular (Asteroid simples):
    - **Duração de cada interação:** 1,5 semanas

        - **Quantidade de histórias:** 5 histórias
        - **Quantidade de histórias por interação:** 1 histórias
        - **Quantidade de interações prováveis:** 5 interações
        - **Tempo provável de conclusão de interações:** 7,5 semanas
        - **Tempo para teste de aceitação:** 1 semana
        - **Tempo provável total:** 16 semanas

- **COMPLEXO:**
    - Sistema para biblioteca universitária:
        - **Duração de cada interação:** 3 semanas
        - **Quantidade de histórias:** 9 histórias
        - **Quantidade de histórias por interação:** 2 histórias
        - **Quantidade de interações prováveis:** 5 interações
        - **Tempo provável de conclusão de interações:** 15 semanas
        - **Tempo para teste de aceitação:** 1 semana
        - **Tempo provável total:** 30 semanas

3.3 DIFICULDADE

Ao selecionar um dos projetos, o jogador estará selecionando o nível de dificuldade desejado. O nível de dificuldade implica somente no aumento das atividades a serem executadas, mantendo sempre o intuito de ensinar a metodologia.

3.4 CONDIÇÃO DE VITÓRIA

O jogador será considerado vitorioso se a realização do projeto escolhido não falhar. Para que o projeto falhe, o jogador deve cometer uma série de escolhas erradas durante o processo de desenvolvimento. As possíveis escolhas que ele pode fazer, são:

- Ao final de cada semana, será informado ao jogador a necessidade de realizar uma reunião, para atendimento à prática de ciclo semanal. Caso o jogador negue, será decrementado:
    - 15 pontos de comunicação, por não informar à equipe e ao cliente;

    - 10 pontos de simplicidade, por ser provável que a falta de comunicação implique no surgimento de problemas, que consequentemente aumentarão a complexidade do software;
    - 20 pontos de feedback, por não informar à equipe e ao cliente.
- Seguindo o mesmo princípio, ao final de cada trimestre o jogador também será indagado sobre sua vontade de fazer a reunião trimestral, para seguir à prática de ciclo trimestral. Caso o jogador negue, será decrementado:
    - 25 pontos de comunicação, por não informar à equipe e ao cliente;
    - 20 pontos de simplicidade, por ser provável que a falta de comunicação implique no surgimento de problemas, que consequentemente aumentarão a complexidade do software;
    - 40 pontos de feedback, por não informar à equipe e ao cliente.
- No final de cada dia, o jogador escolherá fazer ou não a integração, em atendimento à prática de integração contínua. Caso ele negue, será decrementado:
    - 2 pontos de comunicação, por não informar à equipe;
    - 5 pontos de simplicidade, por ser provável que a falta de comunicação implique no surgimento de problemas, que consequentemente aumentarão a complexidade do software;
    - 2 pontos de feedback, por não informar à equipe.
- Após eventuais intervenções do cliente solicitando alguma mudança no software, caso o jogador não acate, será decrementado:
    - 10 pontos de comunicação, por não ter chegado a alguma conclusão;
    - 10 pontos de feedback, por não ter chegado a alguma conclusão;
    - 25 pontos de coragem, por não acatar a solicitação;
    - 20 pontos de respeito, por não acatar a solicitação.

Os princípios e práticas restantes aparecerão durante o decorrer do gameplay implicitamente, sem necessidade de intervenção do jogador, mas mantendo-o informado sobre quais princípios e práticas estão sendo realizadas em determinado momento.

4 DETALHAMENTO TÉCNICO

Esse capítulo descreve as ferramentas que poderão ser utilizadas no desenvolvimento do jogo.

4.1 HARDWARE

Considerando a necessidade de utilização dos softwares descritos nas seções seguintes, o hardware recomendado para um bom funcionamento é:

- Computador:
  - ❖ CPU quad core 64-bit;
  - ❖ 8 GB RAM;
  - ❖ Placa Gráfica compatível com OpenGL 3.2 com 2 GB RAM.
- Android:
  - ❖ OS 2.3.1 ou posterior;
  - ❖ ARMv7 CPU (Cortex) com o suporte NEON ou Atom CPU;
  - ❖ OpenGL ES 2.0 ou posterior.

4.2 SOFTWARE

Além da engine, a ferramenta **Blender** será utilizada para construir os modelos dos personagens.

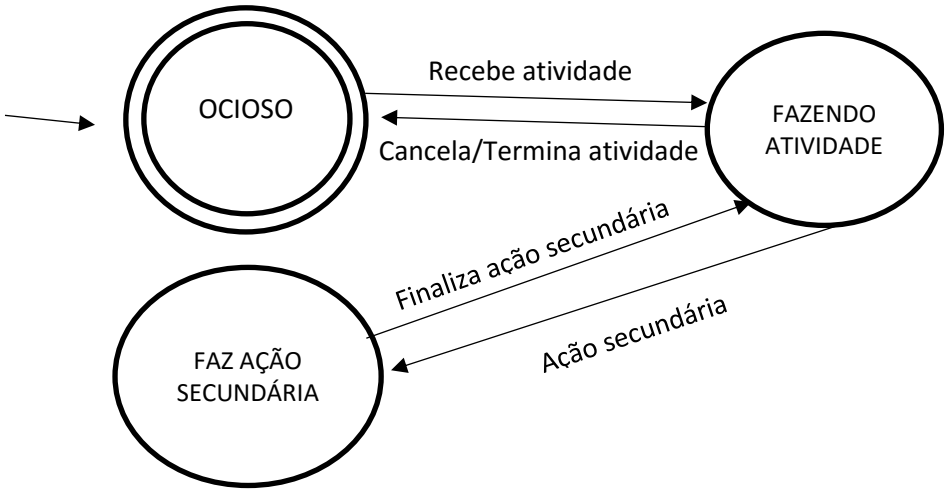
4.3 ENGINE

Considerando que o objetivo é a construção do jogo, a escolha da engine foi baseada na facilidade de aprendizado das ferramentas da engine, disponibilidade de documentação sobre a mesma e a usabilidade das ferramentas. Sendo assim, a que se enquadrou nesses requisitos e se apresentou muito amigável foi a engine Unity 3D.

Além das ferramentas dessa engine ser de fácil entendimento, muitas funcionalidades já estão implementadas, facilitando o trabalho da construção do jogo. Logo, o resultado final poderá ser alcançado com mais celeridade, visto que o tempo poderá ser dispendido na utilização das ferramentas, e não tentando implementá-las.

4.4 I.A.

Cada personagem realizará uma função específica dentro do jogo, como já explicado anteriormente. Contudo, a forma que essas funções serão executadas seguirão o mesmo princípio, obedecendo a seguinte máquina de estado:



5 ARTE

Esse capítulo descreve como o jogo se apresentará graficamente ao jogador, apresentando modelos de personagens, bem como características visuais diversas de objetos que compõe o cenário.

5.1 RESUMO DE ESTILO

O estilo usado no jogo não será foto realista, representando fielmente um ser humano, mas também não será totalmente em forma de desenho animado. O modelo de cada personagem tenta representar alguns estereótipos da função do mesmo.

5.2 CONCEPT ARTS

Essa seção apresentará os conceitos gráficos de como os personagens poderão ser modelados, como é o cenário e os objetos que o compõe.

5.2.1 PERSONAGENS

Devido à natureza do jogo, os personagens representam seres humanos, logo os modelos retratam essa característica, porém de um estilo intermediário ao realista e ao desenho animado. A seguir são mostrados os modelos:

- **Analista de teste**:

Figura 13. Modelo de analista de teste.

- **Arquiteto**:

Figura 14. Modelo de arquiteto.

- **Designer de interação**:

Figura 15. Modelo de design de interação.

- **Executivo**:

Figura 16. Modelo de executivo.

- **Gerente de projeto**:

Figura 17. Modelo de gerente de projeto.

- **Gerente de produto**:

Figura 18. Modelo de gerente de produto.

- **Programador**:

Figura 19. Modelo de programador.

- **Redator técnico**:

Figura 20. Modelo de redator técnico.

- **Cliente**:

Figura 21. Modelo de cliente.

5.2.2 CENÁRIO

O ambiente em que o jogo será executado é uma sala, representando um escritório, com mesas dispostas nela que acomodarão os personagens. Todos eles ficarão nessa única sala, para atender, inclusive, o valor de comunicação. Um protótipo dessa sala pode ser visto na Figura 22.

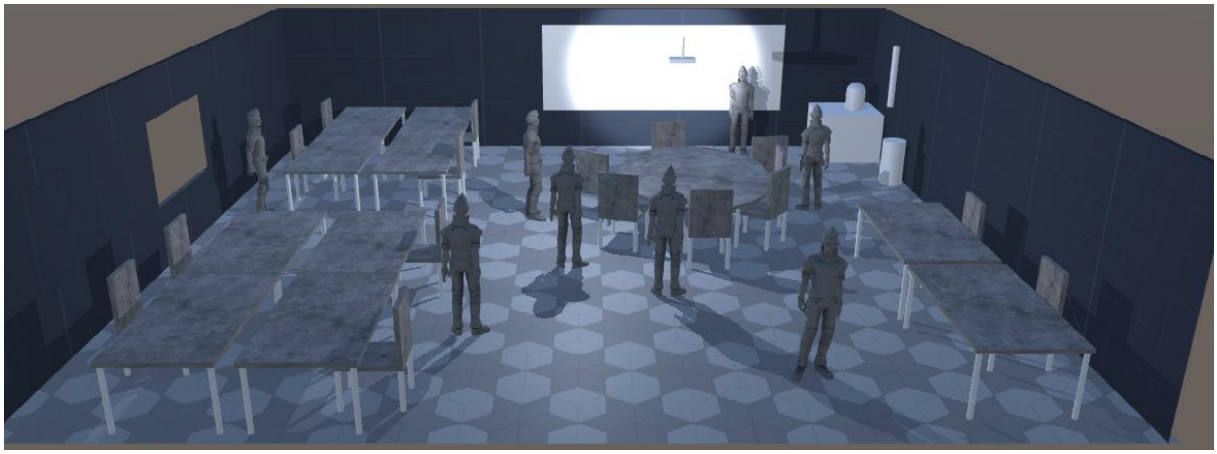


Figura 22. Protótipo do ambiente.

5.2.3 OBJETOS

O ambiente conterá os seguintes objetos:

- Mesa de trabalho com cadeiras;
- Mesa de reunião com cadeiras;
- Computadores;
- Bebedouro;
- Lixeiras;
- Quadro informativo;
- Projetor;
- Objetos relacionados especificamente ao personagem, como por exemplo: xícaras de café, pilhas de papel, entre outros.

5.3 H.U.D. (Head Up Display)

A Figura 23 mostra, de forma enumerada, os itens que compõe o HUD do jogo:

Figura 23. HUD do jogo.

1. Botão para sair do jogo;
2. Botões para controle de tempo;
3. Calendário para mostrar as datas de atividades;
4. Indicadores de pontos de valores;
5. Menu que mostra informações dobre o projeto (requisitos, histórias e atividades);
6. Menu de informações do personagem selecionado.

5.4 ANIMAÇÕES

A animação dos personagens constituirá de movimentos de andar, sentar na cadeira, se levantar, conversar com outros personagens e realizar suas atividades (sumariamente usar o computador). As animações que não estiverem prontas na engine serão realizadas dentro da própria, utilizando as ferramentas que a mesma disponibiliza.

6 SOM

O jogo terá trilhas sonoras e efeitos sonoros lentos e de baixa intensidade para não atrapalhar o objetivo educativo do jogo. Todos os sons terão o formato MP3.

6.1 EFEITOS SONOROS

Durante as ações dos personagens haverão efeitos sonoros nos seguintes itens:

- Movimentação de cadeiras;
- Atividade no computador;
- Caminhada no ambiente;
- Conversas (som indicando uma conversa, porém sem ser possível entendê-la).

6.2 TRILHAS SONORAS

Haverá trilhas sonoras diferentes para os menus e para o gameplay. As trilhas sonoras dos menus terão mais intensidade e serão mais rápidas que as trilhas sonoras durante o gameplay.

7 REFERÊNCIAS

Novak, Jeannie. **Game Development Essentials**. 3ed. 2012.

Google Play Store. Game Studio Tycoon 2. Disponível em: <https://play.google.com/store/apps/details?id=com.msherwin.gamestudiotycoon2&hl=pt_BR>. Acesso em: 31 de jul. 2016.