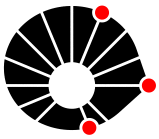

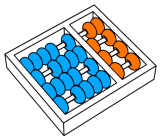

Desenvolvimento de jogo de primeiros socorros baseado em linguagem narrativa

C. Fernandes *A. Santanché*

Relatório Técnico - IC-PFG-18-26
Projeto Final de Graduação
2018 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Desenvolvimento de Jogo de Primeiros Socorros baseado em Linguagem Narrativa

Caio Fernandes André Santanché*

Resumo

Este é um relatório descritivo sobre o processo do desenvolvimento de um jogo digital para ensinar procedimentos de primeiros socorros a leigos. Com base na pesquisa do professor coordenador do projeto e em estudos dedicados ao desenvolvimento de jogos, foi desenvolvido um jogo com enfoque na facilidade de manutenção, com sistemas modularizados de forma a serem facilmente reutilizados em outros projetos. Por ser um jogo que envolve a resolução de casos clínicos, foi usada uma linguagem para desenvolver a estrutura narrativa do mesmo.

1 Introdução

Este trabalho visou o desenvolvimento de um jogo digital como parte de uma pesquisa cujo objetivo é mensurar o impacto do aprendizado através dos jogos digitais na área de saúde. Neste projeto, demos enfoque nos procedimentos de primeiros socorros para jovens na faixa de 10 a 15 anos.

2 Justificativa

O conhecimento na área de saúde é muitas vezes limitado aos praticantes desta área. Porém, expandir o conhecimento sobre procedimentos de primeiros socorros para leigos pode se mostrar extremamente benéfico para a sociedade, de forma a aumentar as chances de que o acidentado sobreviva quando presta algum tipo de socorro.

Ademais, o conhecimento sobre desenvolvimento na área de jogos digitais se mostra muito esparso, principalmente no Brasil, podendo este projeto oferecer uma visão sobre o desenvolvimento de jogos digitais de uma forma geral, em função da organização e problemas comuns encontrados.

3 Objetivos

Neste projeto, tivemos como objetivo principal: O desenvolvimento de um jogo digital para treinamento em saúde fazendo uma de uma linguagem para a escrita de narrativas.

Além disso, os objetivos específicos foram:

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

- Criação de um sistema com partes reutilizáveis e modularizadas.
- Validação do jogo desenvolvido à partir de testes em usuários reais.
- Transmitir a educação de procedimentos de primeiros socorros a partir do jogo.

4 Desenvolvimento do Trabalho

O desenvolvimento do jogo se deu em quatro etapas: concepção, prototipação, teste e desenvolvimento ou retorno à concepção, dependendo dos resultados do teste. No entanto, devido à problemas de calendário e eventos para realização de testes, este trabalho contou apenas com a concepção e desenvolvimento.

Um projeto de jogo digital normalmente conta com a atuação de diversas frentes interdisciplinares, tendo times compostos por artistas, animadores, programadores, escritores, etc. Projetos com grandes orçamentos, conhecidos informalmente na indústria como AAA, tendem a ter diversos times com frentes de trabalho que se comunicam. Porém, jogos de baixo orçamento, compostos por times pequenos, normalmente tendem a lutar com as dificuldades nos aspectos que requerem interdisciplinaridade, sendo jogos de sucesso conhecidos pela forma como transformaram uma fraqueza em uma qualidade.

Ademais, um jogo digital é constituído principalmente por um estilo de arte e uma mecânica principal. Define-se como mecânica, no caso do jogo digital, o fator central que rege as interações do jogador com o mundo do jogo e à partir de uma mecânica principal se tiram as demais interações. Por exemplo, no famoso jogo "Super Mario Bros"temos como principal mecânica o fato de que um jogador pode se mover para os lados e pular. À partir disso derivamos outras mecânicas como aquela de pular em cima da cabeça de inimigos para matá-los; se mover para a direita para avançar o jogo, etc.

4.1 Concepção

Para iniciar este projeto foram levado em consideração os seguintes fatores:

- Limitações do projeto
- Público alvo
- Objetivos a alcançar

Consideramos as seguintes limitações:

- tempo disponível para o desenvolvimento do projeto;
- número de desenvolvedores;
- recursos disponíveis;

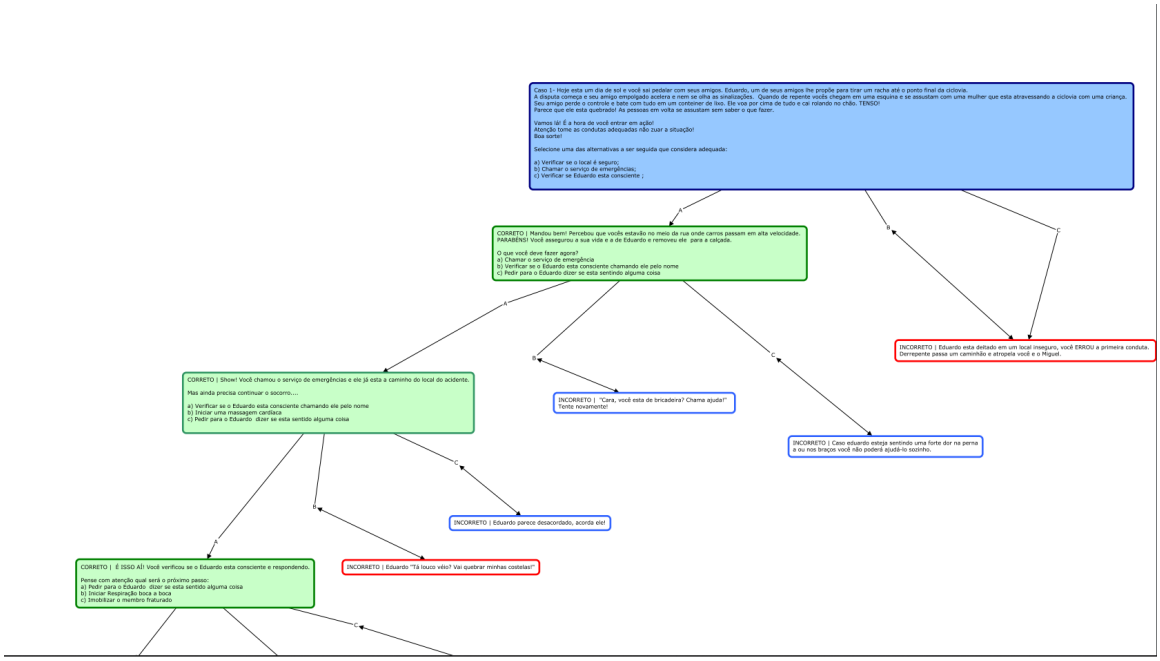

Figura 1: Roteiro do primeiro caso

Considerando as limitações acima pudemos concluir que não devemos iniciar um projeto muito ambicioso, pois o tempo de desenvolvimento é curto para um jogo digital realizado apenas por uma pessoa. Além disso, foi necessário o desenvolvimento de novos recursos para o projeto – e.g., imagens – para se ter uma base de execução.

Ademais, foi feita uma análise do roteiro do primeiro caso, produzido por uma pesquisadora de projeto associado. Esse roteiro é apresentado Figura 1 na forma de um grafo de estados, em que cada nó é um estado da ação e as arestas são decisões. Essa análise nos levou a concluir que o projeto deveria seguir os moldes de casos médicos de cenas comuns, cujo uso rápido de primeiros socorros pode se mostrar crucial.

Seguimos para uma análise do público alvo. Cada tipo de público alvo pode afetar a forma como o jogo será percebido, ou seja, uma mesma experiência pode ser melhor compreendida por um usuário de maior idade do que outro.

Existem algumas diferentes maneiras que podemos utilizar o público alvo para o design de uma aplicação. Para este projeto foi utilizada a criação de personas, ou seja, geramos um ou mais usuários fictícios que representam o nosso público alvo com base em pesquisas com usuários reais. Estas personas são então utilizadas como um ponto de suporte à iterações do desenvolvimento do projeto. Sempre que uma dúvida surge sobre como uma decisão de design pode afetar a forma como o público alvo interpreta a aplicação, seja ela uma nova característica ou apenas uma mudança, as personas criadas são utilizadas para facilitar o entendimento de como essa mudança será recebida.

Para criarmos nossas personas, realizamos então uma pesquisa durante um evento no Museu Exploratório de Ciências na Universidade Estadual de Campinas. Tendo dentre os

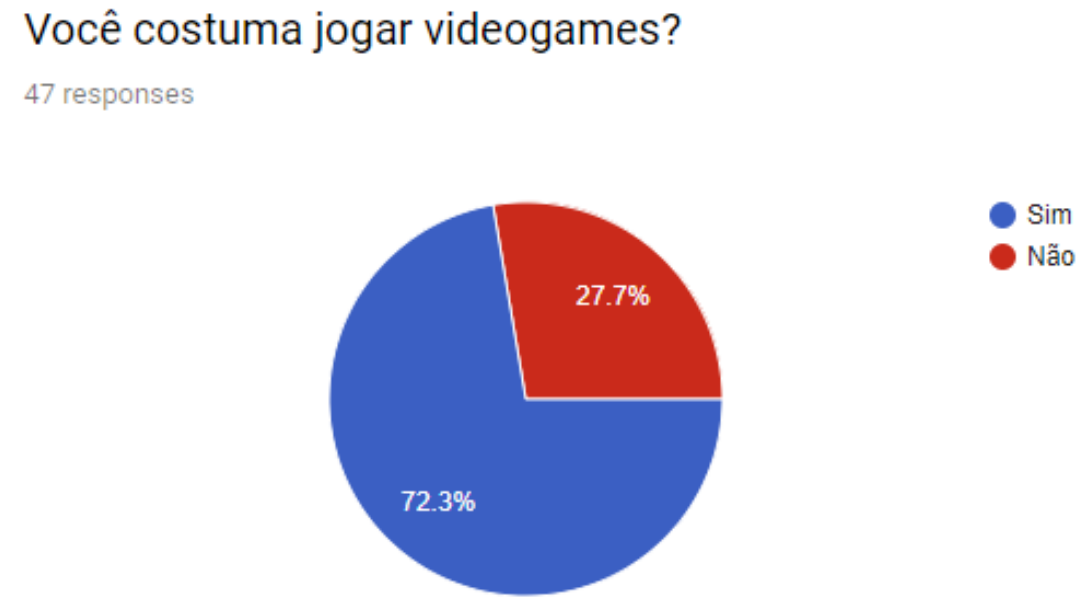

Figura 2: Usuários que costumam jogar videogames

Figura 3: Gêneros favoritos

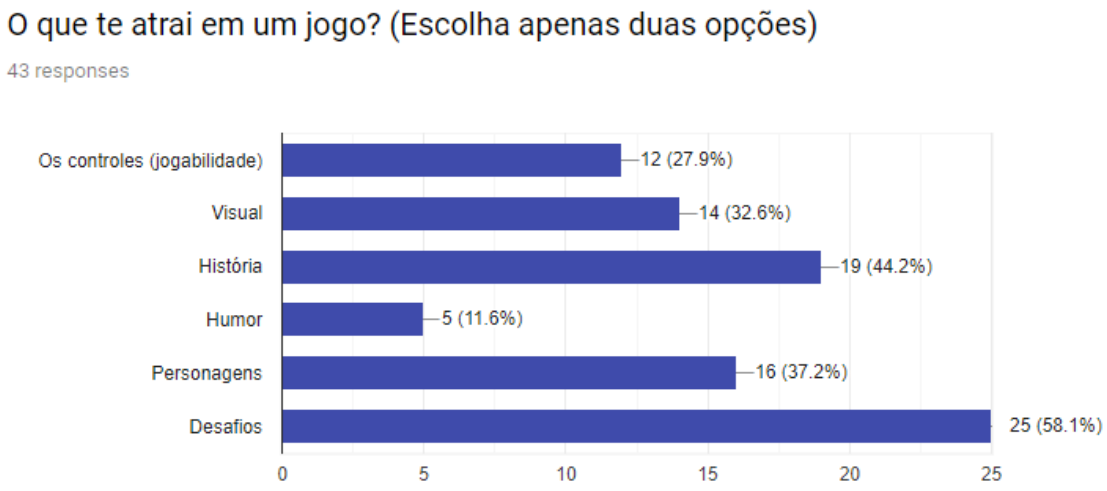

Figura 4: Interesses em um jogo

resultados, as Figuras de 2 a 5, das quais podemos absorver informações como os gêneros favoritos, que o público costuma jogar mais jogos em seus celulares e que gostam de ser desafiados.

Tendo os pontos acima em mente, foi concebida a ideia inicial para o jogo: Um jogo de batalha no estilo de turnos em que o jogador precisa manter a vítima de um acidente viva até a ambulância chegar, escolhendo as ações certas em cada turno.

4.2 Engine

Um dos principais recursos na criação de um jogo digital hoje em dia é a ferramenta de desenvolvimento utilizada. Um jogo digital é composto por uma imensa variedade de sistemas como: renderização de imagem, cálculo de polígonos, simulação de física, I/O , etc. A engine é a ferramenta que possui esses sistemas pré desenvolvidos, de forma que um desenvolvedor não precise se preocupar em programar a física ou renderização em seu jogo pois a engine faz isso por ele.

Para o jogo desenvolvido neste projeto decidimos utilizar a engine Unity3D [3] devido à familiaridade do desenvolvedor e a facilidade de gerar executáveis do jogo para diferentes sistemas operacionais.

Em uma explicação superficial do funcionamento da engine, um jogo é pode ser composto por cenas que representam uma fase ou level, nas quais objetos são criados e renderizados. A Figura 6 apresenta o editor Unity, em que na *Scene View* manipula-se os objetos contidos na *Hierarchy Window*, que apresenta todos os objetos na cena do jogo. Essa view permite posicioná-los ou modificar seu tamanho e outras propriedades de forma visual. Além disso, há também a *Inspector Window*, na qual pode-se ver todas as propriedades de qualquer objeto e adicionar mais propriedades ou scripts a ele.

Um script de objeto herda, normalmente, métodos de operação da classe *Monobehavior*.

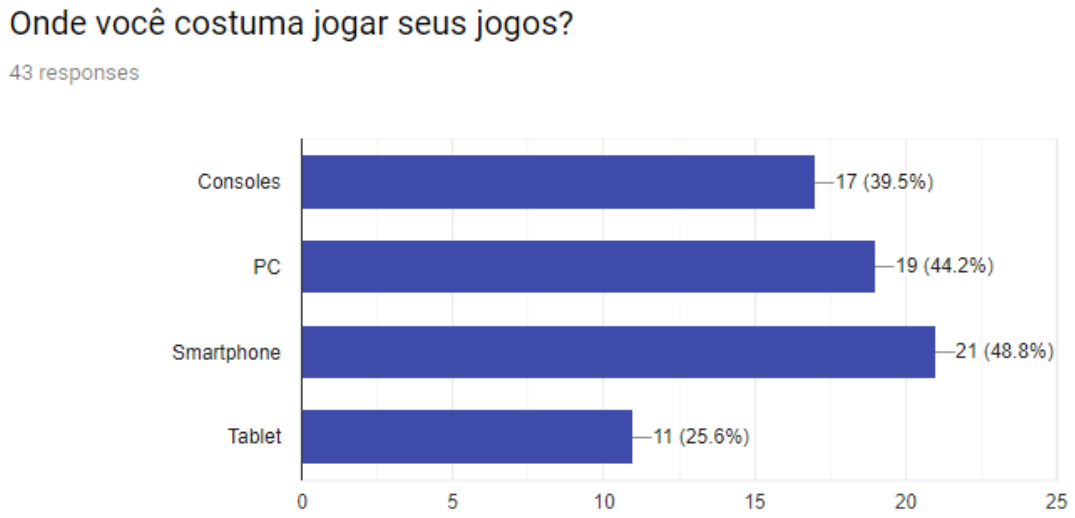

Figura 5: Plataformas jogadas

Dentre eles estão os métodos *Awake e Start* utilizados para inicialização, e *Update*, o qual é chamado uma vez a cada atualização de frame do jogo.

Utilizando estes recursos e outros encontrados na documentação da ferramenta [3], foi possível desenvolver um jogo funcional em um período curto para o número de desenvolvedores envolvidos, no caso apenas um.

4.3 Sistemas

Um dos objetivos do projeto foi o desenvolvimento de um jogo arquitetado de forma a ser duradouro e facilmente manipulável, com a possibilidade de ser expandido ou integrado em outros projetos. Para isso foi preciso desenvolver sistemas modulares capazes de se comunicar entre si de forma a minimizar a dependência entre eles, além de evitar concentrar toda a lógica em um único componente.

4.3.1 Controladores

A arquitetura do projeto foi dividida com base nos controladores, ou seja, scripts responsáveis por controlar certas partes do jogo. Cada controlador se torna responsável por uma área do jogo. Áreas recorrentes em diversos jogos são: Áudio, Input, Cenas e Interfaces de Usuário (UI). Esses controladores são modularizados e realizam suas operações sem depender dos demais.

Por exemplo, o controlador de Input é responsável por ler o input do jogador e transmitir qual tipo de ação foi realizada por ele. O Unity3D possui uma classe de Input própria, que repassa eventos, por exemplo: se uma tecla ou botão foi apertado. Essa atividade se torna mais complexa no contexto de games, em que se leva em consideração o uso de diversos periféricos como controles de diferentes marcas, teclados, etc.

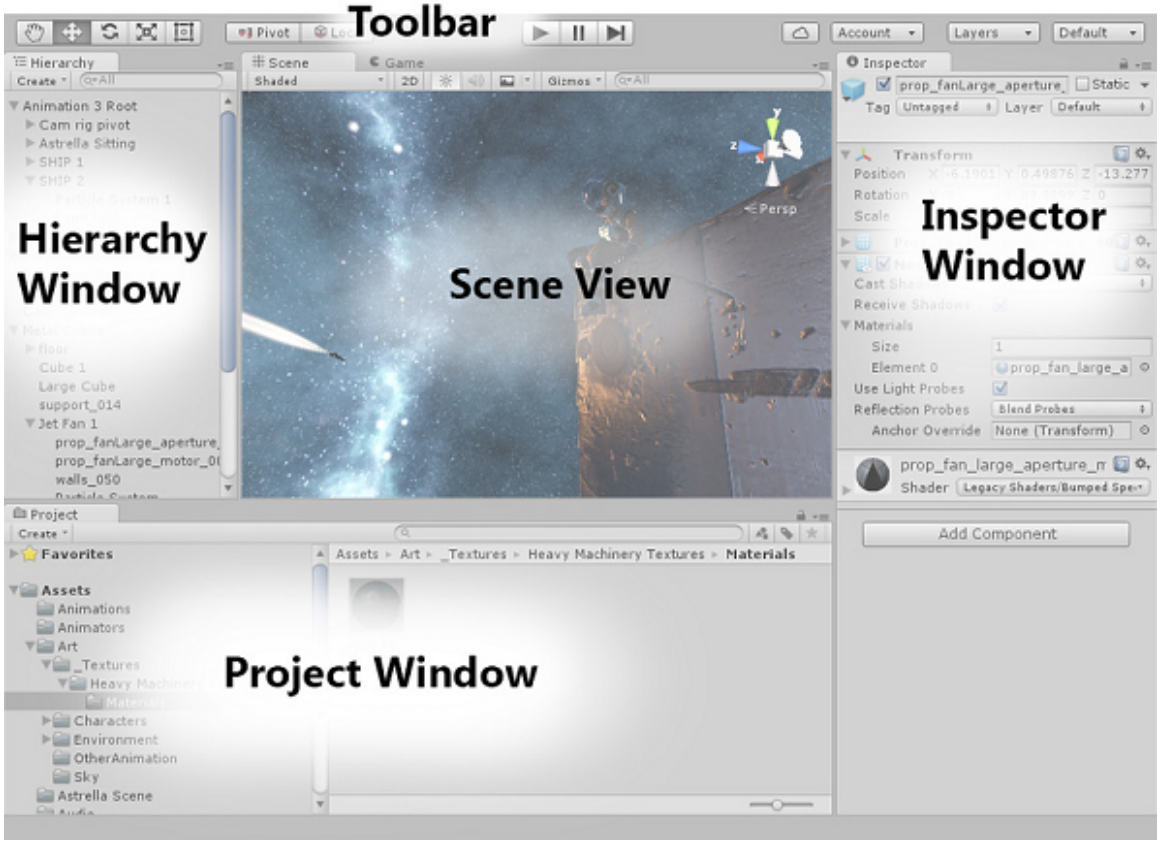

Figura 6: Editor Unity

Assim, o controlador desenvolvido utiliza do sistema de Input da engine para mapear botões virtuais, ou seja, gera uma camada intermediária entre o input do jogador, recebido diretamente pela classe Input, e o programa desenvolvido, que precisa utilizar da entrada do jogador para realizar alguma ação. Essa camada intermediária interpreta a entrada de acordo com o tipo de periférico e então retorna ações pré interpretadas. Por exemplo, o botão 'A' em um controle pode significar "selecionar"no jogo, da mesma forma que o botão "Enter"no teclado, então a camada intermediária retornaria a ação "selecionar"caso uma dessas teclas fosse apertada.

Da mesma forma, os demais controladores gerem suas respectivas áreas de modo a encapsulá-las e melhor interpretá-las para facilmente se adaptar em uma mudança. Por exemplo, o controlador de input desenvolvido precisa apenas adicionar um novo interpretador para um novo periférico existente, em vez de buscar e modificar todos os arquivos que utilizam inputs do jogador.

Com isso, o controlador de Áudio gerencia volume, música de background e efeitos sonoros. Já o controlador de UI gerencia eventos na interface de usuário, como texto, transições e escala de tamanho, dependendo da tela do jogador.

Por fim, o controlador de cenas gere qual cena deve ser carregada em qual momento. Uma boa prática no desenvolvimento de jogos em Unity3D é construir uma cena persistente, a qual irá se manter durante todo o jogo, tendo objetos que não devem ser destruídos ou resetados em uma mudança de cena como, por exemplo, os controladores acima e objetos como a música de fundo, que sofreria cortes se destruída e recriada em uma mudança de cena.

4.3.2 Localização

Algumas decisões de projeto precisam ser tomadas desde o seu início e preparadas desde então, senão será cada vez mais difícil incluí-las. Para jogos digitais, já consideramos algumas plataformas que o jogo iria comportar e da mesma forma os controles para os quais ele daria suporte. Outra decisão muito importante foi a que o jogo possuiria seus textos localizados em outras línguas.

Existem diferentes formas de lidar com a localização. Neste projeto, foi usada a forma chave/valor, em que cada texto no jogo será representado por uma chave e sempre que o texto precisar ser revelado, um controlador de localização criado, é chamado para buscar o seu valor em um arquivo comportando todas as chaves e valores da língua em questão. Assim, cada língua possui seu próprio arquivo de chaves e valores, em que as chaves serão iguais para todos os arquivos, mas o valor será o texto propriamente traduzido na linguagem selecionada.

4.3.3 Linguagem Narrativa Ink

Um dos pontos centrais do jogo girou em torno de encontrar um meio de converter os casos médicos em um jogo digital. Parte da pesquisa procurava por um meio de converter os dados de casos inseridos por médicos facilmente em algo que possa ser utilizado como um banco de dados pelo jogo.

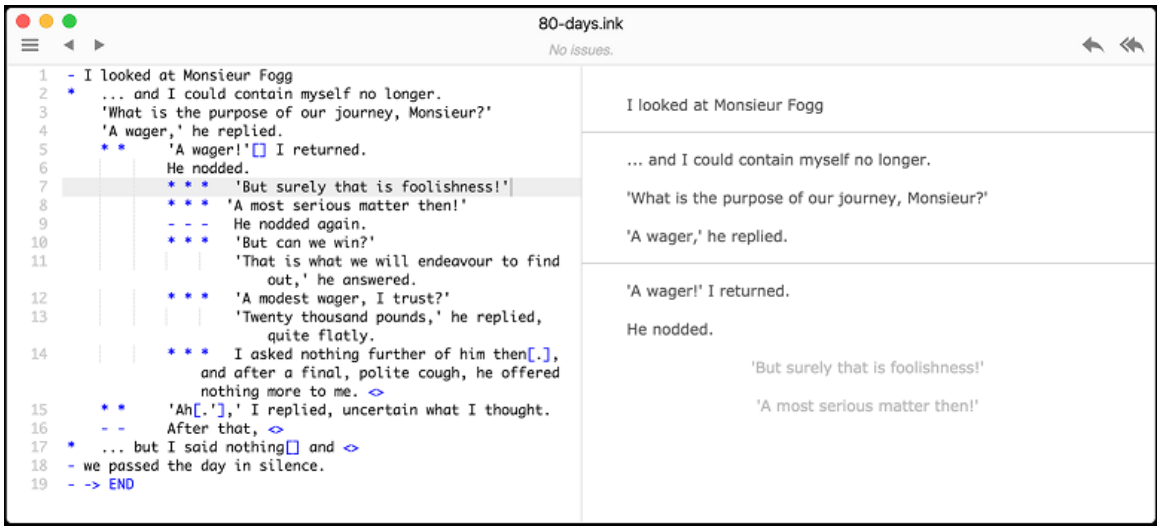

Figura 7: Exemplo Ink

Dessa forma, para este projeto foi decidido utilizar a ferramenta Ink [4]. Esta ferramenta é uma linguagem com foco em gerar textos narrativos com possibilidades de escolhas e caminhos resultantes dessas escolhas. Ela também dispõe de recursos avançados equivalentes a uma linguagem de programação.

Um elemento importante para a interpretação de um arquivo *.ink* pela engine Unity foi o uso de um plugin que acompanha a instalação do pacote Ink no Unity3D. Esse plugin que age como interpretador dos arquivos *.ink* e se comunica com uma API interna do Unity através de diferentes métodos e variáveis.

Essa API, no entanto, possui falhas de funcionamento e falta de alguns métodos que teriam sido úteis durante a interpretação do arquivo. Por exemplo, os métodos de *get* para o texto da fala atual e o método de *get* para o nome do Nó atual não eram consistentes para a primeira fala de um nó, enquanto a fala retornava seu valor correto (a fala em si), o Nó tinha como retorno o nome do último Nó pelo qual o fluxo passou, ao invés do nó em que está, sendo apenas atualizado a partir da próxima fala.

A linguagem utilizada em Ink possui grande similaridade com a linguagem markdown. Uma documentação completa de suas funcionalidades pode ser encontrada em seu site [4], no entanto iremos ressaltar algumas das principais funcionalidades utilizadas:

1. Falas

   Cada fala pode ser representada em um arquivo .ink por uma linha de texto sem nada a sua frente ou com um hífen.

2. Opções

   O ink possibilita o uso de escolhas narrativas pelo jogador. Cada escolha pode ser representada por uma fala, antecedida por um ou mais asteriscos. Falas consequentes

da escolha podem ser escritas logo abaixo dela, assim como novas escolhas subordinadas, que devem possuir um asterisco a mais do que a escolha em nível superior, como pode ser visto na Figura 7.

Existem também alguns elementos adicionais para enriquecer o texto. Por exemplo, cada vez que uma opção é escolhida durante a execução, seu texto é incorporado na narrativa apresentada ao usuário, exceto quando são usados colchetes "[]"com texto dentro. Nesse caso, o texto será apenas apresentado quando a opção da escolha for revelada, quando ela é escolhida só é apresentada a parte que está fora dos colchetes.

3. Nós e desvios

   Os Nós funcionam como funções em linguagens de programação. Indicados por um texto representando o nome do Nó precedido por "==", eles encapsulam sequências de falas e escolhas dentro de si. Assim, a partir de qualquer ponto do arquivo é possível desviar o fluxo de falas para um Nó, indicando o nome do mesmo precedido por -¿". Mas, diferentemente de uma função em linguagens de programação, o fluxo não retorna para o local onde o Nó foi chamado quando este termina seu fluxo de falas.

   Caso um Nó termine, ele deve sempre desviar para um novo Nó ou desviar para o Nó END, indicando que o arquivo chegou ao seu fim.

   Além do Nó podemos também ter sub Nós, demarcando estes como textos precedidos de "= ", os quais agem da mesma forma que os Nós, mas sendo referenciados como uma propriedade do Nó pai. Por exemplo, se temos o Nó pai "Pai"e dentro dele um Nó filho "Filho", poderíamos divergir para o Nó filho através da chamada -¿Pai.Filho".

4. Tags

   Um outro recurso utilizado foi o uso das *Tags*. Elas funcionam como comentários atrelados a uma linha de fala ou escolha, são textos antecedidos por um "#".

   As tags, apesar de não parecer muito importantes, podem ser utilizadas para a comunicação de mudanças de estado (como animações), uma vez que o mecanismo Ink as transmite para a engine Unity junto com a fala e é possível interpretá-las no contexto da Unity. Por exemplo, considere um jogo que está reproduzindo um texto com um personagem fazendo diferentes expressões faciais, dependendo da fala em que está. Para se interpretar uma fala em que ele está zangado podemos utilizar uma tag de "# angry".

5. Lógica

   No exemplo em que o jogo que está lendo o arquivo de falas precise interpretar suas emoções, as Tags funcionam como uma boa ponte para interpretações simples. Porém, dependendo do tamanho do projeto, e número de animações, por exemplo, esse método pode se demonstrar uma má prática na programação do jogo, já que transfere para o Unity uma tarefa de interpretação de narrativa que poderia ser mais facilmente trabalhada pelo mecanismo Ink. Para isso, o Ink possui também lógica própria.

Dentro de um arquivo Ink podemos também incluir estruturas de programação e controle básicas. Podemos criar variáveis, designar valores, realizar operações condicionais e lógicas. O intuito de haver esse nível de interpretação na linguagem Ink é o de minimizar a troca de informações entre a engine Unity e o mecanismo Ink, além de dar mais autonomia ao mecanismo Ink.

No entanto, dado que o foco deste projeto busca facilitar a criação de um caso textual por um médico, o uso de lógica pode dificultar muito esta relação. Dessa forma procurou-se minimizar o uso de lógica no código.

4.3.4 Construindo uma Narrativa em Saúde

Tendo essas funcionalidades em mente, foi escrito um protótipo do primeiro caso, mostrando as principais funcionalidades exploradas, como pode ser visto na Figura 8.

A construção do caso se deu a partir de um roteiro elaborado por uma pesquisadora de um projeto associado com este em conjunto com um médico especialista em traumatologia. Trata-se de um caso de primeiros socorros de uma vítima de acidente de bicicleta.

Nele podemos observar nas seguintes decisões:

1. Divisão de passos em Nós

   Uma qualidade importante do roteiro recebido era a sequência de ações e reações. Observando a Figura 1, vemos que o caso foi dividido em uma sequência de passos até termos a vítima salva com a chegada da ambulância. Assim foram divididos estes passos em diferentes Nós com diferentes opções e cada opção com a sua resposta subsequente.

2. Nós genéricos

   Uma mecânica do jogo gira em torno da vítima perder pontos relacionados à sua vida quando o jogador não toma a decisão correta. Deste modo foi criado um Nó genérico chamado *Damage Time* que cuida dessa perda de pontos.

3. Nós específicos

   Uma outra mecânica presente foi o menu de escolhas do jogador. Antes de tomar qualquer ação para ajudar a vítima no jogo, o jogador pode utilizar o seu celular, ou tentar correr, por exemplo. Essas sequências são sub-estórias que possuem falas e escolhas próprias mas, apesar de parecerem ser uma história separada, interagem com a história principal diretamente.

   Por exemplo, normalmente durante os casos o jogador precisaria prestar socorros imediatos à vítima, para então ligar para a ambulância e por fim manter a vítima viva até a ambulância chegar. Mas dado o caráter mais livre do jogo, em que o jogador seria capaz de tentar ligar para a ambulância a qualquer momento, essa sub-estória do celular precisaria saber em que ponto da história principal o jogador está, para permitir a avaliação do resultado da ação de ligar para a ambulância – i.e., ação correta ou a decisão errada do jogador a depender do ponto do processo.

```
VAR current_step = "START"
VAR current_step_ink = ->START

-> START
===START===
    # intro
    - It's a sunny day, you are biking with a couple of friends around Taquaral Lagoon.
    - When you arrive at the first gate, you meet with the other guys.
    - Eduardo, one of your friends, proposes to race to the end of the bicycle path.
    - And so it begins! Eduardo accelerates without looking at any signs!
    - When suddenly you get to a corner and see a woman holding a child crossing Eduardo's way.
    - Eduardo loses control and hits a trash bin.
    - He flies over it and hits the ground rolling. OUCH!
    - He looks barely alive!
    - People around freak out and don't know what to do.
    - It's time to get to action!
    - Atention take the right paths in order to keep him alive. Good luck!
    - ->Step1


===Step1===
    ~ current_step = "Step1"
    ~ current_step_ink = ->Step1
    *   [Verify the locale's safety] # canCallAmbulance
        ->Step1.Success
    *   [Verify if Eduado is conscious]
        You get closer to Eduado in order to verify his consciouness.
        ->Damage1

    = Success
        Well done! You noticed that you were in the middle of the street where a car could hit you anytime!
        You secured your life and Eduardo's by removing him from the street.

    - ->Step2

===Step2===
    ~ current_step = "Step2"
    ~ current_step_ink = ->Step2
    +   [Verify if Eduado is conscious by calling his name]
        Are you kidding me?! Call for help! ->DamageTime
    +   [Ask Eduardo to tell if he's feeling anything]
        Pain.
        But what can you do about it?
        ->DamageTime
    - ->END

===DamageTime===
    The victim needs help! # damage 10
    ->current_step_ink

===Damage1===
    But before you could realize, another bycicle goes through you and injures Eduardo. # damage 70
    ->Step2


===Items===
    +   [Call the ambulance]
        { Step1.Success:You call asking for help|You've already called}
        { Step1.Success:->Items.Success}
        { not Step1.Success:->Damage1}
    *   [Call mom]
        The phone rings but no one takes it
        { Step1.Success:->DamageTime}
        { not Step1.Success:->Damage1}

    = Success
        - 3 turns until the ambulance arrives
        - ->current_step_ink
```

Figura 8: Caso 1

Para solucionar esse problema foi pensado em duas possíveis soluções. Uma delas seria criar uma Tag que permitisse checar se o jogador já pode chamar a ambulância de forma segura. Mas esse caminho levaria para a má prática discutida anteriormente sobre muitas checagens diferentes por tags. Então, a solução utilizada foi a de criar um Nó específico para essas sub-estórias contendo sequências de falas e sub Nós em partes do código representando que um passo tomado foi um sucesso ou não. Isso foi feito pois o Ink permite lógica condicional em seu código, assim pudemos criar um Nó representando as falas de sucesso e de erro precedidas por uma checagem lógica em que serão apenas retornadas se certo passo foi um sucesso, como podemos ver no Nó *Items* na Figura 8.

4. Variáveis

   A solução acima causou um problema de continuidade. Sendo possível o jogador desviar do passo que está para o Nó de *Items*, nós teríamos que poder retornar para o exato Nó em que ele estava anteriormente. Como apresentado anteriormente, a engine que lê o arquivo *.ink* possui algumas falhas em sua API que impossibilitaram o uso preciso de métodos para armazenar o Nó atual e para desviar o fluxo para ele, uma vez que o Nó específico teve o seu fluxo concluído.

   Para contornar esse problema, foi criada uma variável responsável por armazenar o passo atual, dentro do arquivo *.ink*. Ela é sempre modificada ao entrar em um novo Nó e ao fim do fluxo de um Nó específico, este desvia para o último passo armazenado.

Por fim, uma boa prática para o uso de recursos de terceiros como *plugins* e APIs é criar um encapsulador próprio para os métodos existentes nele. Fazendo isso, o desenvolvedor é capaz de facilmente se ajustar à mudanças no recurso. Por exemplo, se o recurso utilizado em uma atualização remove um método ou unifica duas propriedades em uma, o desenvolvedor pode simplesmente atualizar o seu arquivo encapsulador para seu código continuar funcionando como esperado. Isso mantém inalterados os métodos que utilizam o encapsulador. De outra forma, o desenvolvedor seria obrigado a procurar todas as chamadas da API em seu código e modificá-las sempre que precisar atualizar a versão do recurso utilizado.

5 Resultados

Foi produzido o protótipo de um jogo com sistemas modularizados, facilmente escaláveis e reutilizáveis. Ele faz uso da ferramenta Ink para suporte a construção de um banco de casos baseado em narrativas

Podemos ver algumas telas do jogo nas Figuras de 9 à 11.

6 Conclusão

Podemos ver que para construir um simples jogo digital é preciso antecipar muitas decisões assim como possíveis movimentos do jogador. Além disso, para se construir um

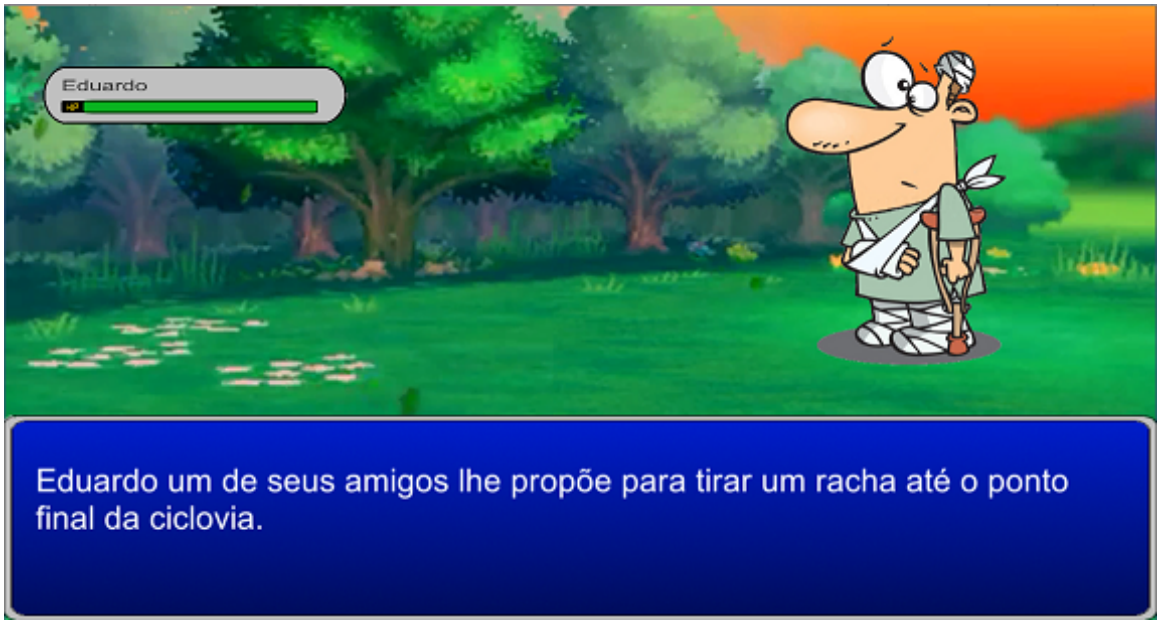

Figura 9: Jogo fala

Figura 10: Jogo ações

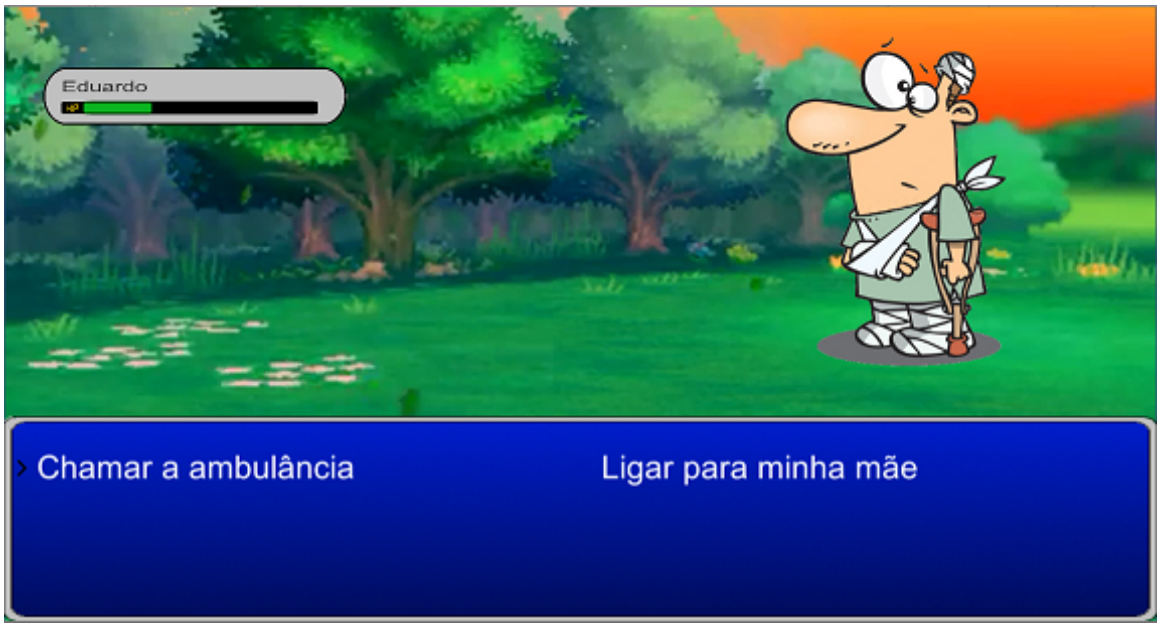

Figura 11: Jogo escolhas

projeto passível de expansão ou transformação é preciso se atentar às boas práticas e prever como possíveis mudanças poderiam afetar o código regendo o jogo.

Um dos principais focos da pesquisa foi a integração com uma linguagem narrativa (implementada pelo Ink) como uma forma de permitir aos médicos para facilmente criar casos e transformá-los em um jogo digital. No entanto, a proposta de dar autonomia ao médico na construção do jogo sem depender do desenvolvedor, usando o Ink, se mostrou difícil. Parte do texto do arquivo *.ink* precisou seguir alguns padrões de programação que não seriam facilmente ensinados para um leigo, como variáveis, organização de código e boas práticas no uso de Nós para serem reutilizados.

Porém, a maior causa disso foi a necessidade de o jogo obter diferentes informações do texto, além de navegar nele para definir diferentes estados. Uma solução para facilitar o uso do Ink seria limitar que tipo de informações que o jogo irá tirar do arquivo *.ink*, ou seja, quanto menos conteúdo do texto recebido ou da escolha feita para a máquina de estados do jogo tiver que expressar ações complexas de programação, mais facilmente será possível separar o texto escrito pelo médico do código feito pelo desenvolvedor.

Outra possível solução gira em torno de não centralizar o conteúdo do jogo ao redor dos casos. Uma vez que é possível ter diferentes arquivos *.ink* gerenciando diferentes árvores de decisão, podemos ter um arquivo responsável pela narrativa e outro responsável por cada caso. Essa pré estruturação pode ser feita pelo desenvolvedor e o médico pode preencher o caso seguindo as heurísticas definidas pelo desenvolvedor. Assim, os casos não deveriam ser um elemento do jogo que cause mudanças de estado durante o decorrer do mesmo, focando talvez apenas nos textos apresentados e no sucesso ou não de etapas do caso para próximas etapas na narrativa do jogo.

Por fim, uma opção é também desenvolver uma nova ferramenta com comportamento similar ao Ink porém focada no uso por médicos. Nessa ferramenta, poderíamos ter o comportamento similar ao do Ink porém, utilizando termos médicos para informar mudanças de estado ou condições do paciente onde o interpretador da linguagem pode reconhecer esses termos, talvez utilizados em *tags* ou até interpretando o conteúdo do texto, para informar ao jogo uma mudança de estado. Algumas informações que podem ser recorrentes em casos, como por exemplo condições do paciente, podem ser interpretadas e enviadas para o jogo. Algumas formas possíveis de passar uma informação específica como o estado do paciente de uma forma neutra pode ser usando diferentes tipos de escalas. Por exemplo, uma escala de 0 a 100 de saúde do jogador ou, para casos mais complexos, como um ponto cartesiano em um plano multidimensional que usa, por exemplo, a distância do ponto à origem, seu quadrante, ângulo, dentre outras informações, para passar o estado para o jogo.

Enquanto que as primeiras soluções focam em modificar o jogo para comportar o texto de casos médicos, a última foca em modificar o Ink para se especializar em casos médicos. Porém o maior trabalho estaria na ponte entre o arquivo de texto médico e o jogo, o interpretador receberia o trabalho de traduzir informações e termos médicos recebidos além do texto, possivelmente até pré-acordados, em um formato que o desenvolvedor consiga entender e utilizar da forma que quiser em seu jogo.

Referências

[1] COSTER, Raph (2013).*A Theory of fun in game design*. Second Edition. O'Reilly Media.

[2] SCHELL, Jesse (2008). *The Art of Game Design*. 1 Edition. CRC Press.

[3] MANUAL UNITY, 2018 Disponível em: https://docs.unity3d.com/Manual/index.html

[4] INK, 2011 Disponível em: https://www.inklestudios.com/ink/