Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

Nervos de Ferro

Jogo de Terror

Documentação

Técnicas de Interação Avançadas

Ana Salvado 44299

Inês de Matos 43538

João Vicente 44294

Carlos Duarte 44313

ÍNDICE

INTRODUÇÃO

As interfaces multimodais dão ao utilizador a possibilidade de poder interagir com um sistema de várias formas diferentes, sejam estas formas de entrada ou saída de informação. Isto permite uma comunicação mais natural e flexível. Considerámos que a multimodalidade cria um ambiente mais envolvente, com menos distrações e que requer mais interação por parte do utilizador, oferecendo assim uma melhor experiência para o mesmo.

No âmbito da cadeira de TIA foi-nos proposto o desenvolvimento de um projeto que incorporasse vários dispositivos de interação de forma a aplicar o conceito de multimodalidade. Depois de considerarmos várias opções, decidimo-nos realizar um jogo de terror, isto porque, para além de ser mais divertido de desenvolver, dava-nos a hipótese de combinar as várias ferramentas que queríamos usar.

Para isto, os dispositivos que decidimos usar foram o Kinect, o BITalino, a mesa interativa e o Dive. O jogo foi ainda desenvolvido em na plataforma Unity.

JOGO

A nossa ideia para este jogo de terror consiste em ter dois jogadores a “competir” um contra o outro. Um jogador está envolvido num ambiente imersivo para tentar escapar ao labirinto até recolher todas as chaves que lhe dão acesso à porta de saída; o outro tenta dificultar-lhe esta tarefa, introduzindo os tais obstáculos (por exemplo um fantasma no meio do caminho) de forma a assustar o oponente e a criar stress ao mesmo. Os detalhes do jogo serão desenvolvidos mais à frente.

DECISOES DE DESENHO

A introdução de dois jogadores foi decisiva, pelo que foi a base para a utilização das tecnologias escolhidas. Pensámos em ter um jogador imersivo, com o DIVE, no jogo e um outro jogador que, externamente, controla objetos e materiais do jogo, através da mesa, influenciando as decisões e comportamentos do primeiro.

O primeiro jogador iria enfrentar um labirinto, em que o objetivo seria sair dele, após a recolha de três chaves, que estão espalhadas aleatoriamente ao longo do percurso. O segundo jogador, iniciaria o jogo colocando, então, as tais chaves no mapa em qualquer setor.

Através dos sinais biológicos, nomeadamente, os batimentos cardíacos criaríamos então o desafio. Quanto mais nervoso o primeiro jogador estivesse, mais pontos o jogador dois ganharia para o poder “atacar”.

CONCEITO POR DE TRÁS DO JOGO

Como referido, o jogo baseia-se na interação de dois jogadores. O jogador 1 está inserido num labirinto assustador e tem como objetivo apanhar três chaves para conseguir sair do mesmo. O jogador 2 terá um mapa do labirinto, com os obstáculos e os seus respetivos custos para ativar no jogo, as informações do jogador 1 (batimentos cardíacos e bateria da lanterna) e saberá constantemente a posição do primeiro jogador no mapa. O jogo é iniciado quando o jogador 2 introduzir as chaves em secções de sua escolha.

Este ambiente imersivo e assustador é proporcionado pelo DIVE e pelo jogo que desenvolvemos no Unity. Porém, existem monstros e outro tipo de obstáculos, colocados pelo jogador 2, que serão introduzidos à medida que o jogo avança. Neste sentido, o objetivo do segundo jogador será tentar, então, assustar o mais que possa o jogador 1, deste modo é preciso que este mantenha a calma, porque quanto mais assustado ele estiver, mais difícil o jogo se tornará pois estará a dar pontos ao jogador 2.

Estes pontos são obtidos através dos batimentos cardíacos obtidos pela medição do nível de *stress*, através do sensor ECG, do BITalino. Os valores que este sensor fornece são transformados, por fim, em pontos que serão dados ao jogador 2. A função do jogador 2 é impedir que o jogador 1 saia do labirinto, e consequentemente este tem algumas ferramentas à sua disposição que lhe permitem pôr obstáculos ao jogador 1, tais como, interferir com os efeitos sonoros, luz ambiente, colocar monstros em cena, teletransportar o jogador 1 para uma outra sala, entre outros. Mas estas componentes têm um custo, como já foi referido, que poderão ser pagos pelos pontos obtidos pelo sensor ECG.

Para tentarmos garantir um ambiente mais interativo, o jogador 1 desloca-se pelo mapa ao caminhar no mundo real, usando o Kinect para registar estes movimentos.

TECNOLOGIAS USADAS

Referente à multimodalidade, optámos pelas tecnologias usado pelo Bitalino, Kinect, mesa interativa e o DIVE (mais giroscópio), obtendo as interações relativas às assinaturas biológicas, às movimentações corporais, manipulação de objetos no jogo e à movimentação imersiva (movimento do campo de visão no jogo - 360º).

Este jogo foi concebido no Unity e para haver a comunicação com as outras componentes foi usado o node JS.

Os dispositivos escolhidos foram o DIVE para mostrar o jogo, o Kinect para registar os movimentos, o Bitalino com o sensor ECG (eletrocardiograma) para conseguirmos medir os batimentos cardíacos do utilizador.

Tínhamos intenção de usar a mesa interativa para a manipulação e interação de objetos no jogo, mas apenas conseguimos utilizar no browser.

INSTALAÇÃO

Esta secção vai ser dividida em algumas partes de modo a facilitar a compreensão do que é necessário fazer e/ou instalar:

PROGRAMAS A INSTALAR

Como referido precisámos de instalar o node JS nos nossos computadores (mínimo a v4), isto para ligar todos os dispositivos de forma a conseguir transmitir informação entre eles.

O jogo foi desenvolvido no Unity, portanto, este também é um requisito necessário, para além de uma ou mais plataformas que permitam o desenvolvimento em Java (Eclipse) e para Web – js e html. É também necessário usar jquery.

Para o código do projeto do BITalino funcionar corretamente foi preciso instalar o jar, bluecove-2.1.1-SNAPSHOT.

Opcional: Pode ser instalado o programa OpenSignals, para ver se os sensores estão bem colocados, ou se a leitura está a ser efetuada corretamente.

Para o Kinect é necessário instalar o Visual Studio, pois foi desenvolvido em c#. Para por esta componente a correr, é ligar o respetivo equipamento a uma entrada USB 3.0. Após o aparelho estar ligado, tem de ser posicionado para que apanhe sempre os pés do jogador.

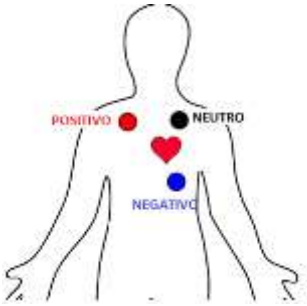
Relativamente à mesa interativa é necessário ter um browser (aconselhado o Chrome), e instalar o python, rabbitMQ, java, eye toy driver, community core vision e erlang. Para iniciar o website, é preciso estar na pasta do projeto e correr o ficheiro www dentro da pasta bin com o comando “node www”.

BITALINO: COLOCAR SENSORES

Sensor positivo – abaixo da clavícula direita.

Sensor neutro – abaixo da clavícula esquerda, paralelamente.

Sensor negativo – abaixo do coração (lado esquerdo), na direção do sensor neutro.



COMO MUDAR O IP DO NODEJS NO JOGO

1. Ir a hierarquia do jogo;
2. Selecionar socketio;
3. Ir ao campo ws://(ip):(porta)/socket.io/?EIO=4&transport=websocket;
4. Substituir IP e porta pela usada pelo nodejs;
5. Gravar.

Nota: quando passar o jogo para o dispositivo o nodejs e o smartphone tem que estar na mesma rede.

COMO CRIAR UM PONTO DE ACESSO

Para conseguirmos fazer a comunicação entre dispositivos usando o node JS precisámos de estar todos na mesma rede e para isso usámos um ponto de acesso:

1. Ir as Definições do telemóvel;
2. Selecionar mais;
3. Partilha de Internet;
4. Ativar Hotspot WI-FI móvel.

COMO INSTALAR JOGO NO SMARTPHONE

1. Ligar smartphone ao pc
2. Ir ao unity
3. Selecionar File > Build & Run ou (ctrl + b)
    a. Caso deseje alterar as definições da build pressionar (ctrl + shift + b)

CÓDIGO

Nesta secção vai ser explicado como foram desenvolvidas as várias componentes, e para simplificar também iremos dividi-las em secções.

UNITY

Scripts

Os scripts estão localizados na pasta scripts e encontram-se divididos em quatro subdiretórios:

Enemy

- Nesta pasta encontram se os scripts de cada monstro. Estes tratam dos movimentos usando o itween Path. (freedyScript, SlenderScript, SpiderScript).

Objects

- Nesta pasta encontram-se os scripts existentes para alguns objetos no mapa neste caso os colecionáveis (baterias e chaves) e alguns objetos com reação, por exemplo portas.

Player

- Nesta pasta encontram-se os scripts que tratam do movimento do utilizador, da gestão da camara e da colisão com os objetos existentes no jogo.

Spawn

- Nesta pasta são encontrados os scripts que recebem a informação proveniente do nodejs e que tratam das áreas de criação do mapa.

Prefabs

Nesta pasta encontram se todos os objetos do jogo e estes estão divididos em 5 pastas.

Collectables

- O conteúdo desta pasta são os objetos colecionáveis, neste caso as baterias e as chaves.

Enemy

- Nesta pasta encontram-se os inimigos.

Jump

- Nesta pasta encontram-se os prefabs dos scareJumps.

Objects

- Objetos variados.

Player

- Um prefab do utilizador.

NotTrash

- Objetos que não foram utilizados.

iTween

Este plugin trata dos caminhos existentes para os monstros, este objeto pode ser localizado na hierarquia.

Fantasmas: Spawners>GhostSpawn>GhostPath.

Aranhas: Spawners>SpiderSpawn>SpiderPath.

Tank/Grunt/Slender: Os caminhos destes objetos estão integrados no prefab correspondente.

Sounds

Nesta pasta encontram se todos os sons do vídeo jogo.

Script Player

As funções a baixo são encontradas na função Start() no player script.

```
socket.On("heartBeat",heartRef);
socket.On("movement",movementPlayer);
socket.On("teleport",teleportPlayer);
socket.On ("keys", startGame);
socket.On ("flashlight", lightScary);
```

Estas tratam da informação que vem do nodejs. O segundo argumento é a função que vai tratar deste tipo de pedidos.

heartRef: Trata dos pedidos que vêm do BITalino.

movementPlayer: Trata dos pedidos vindos do Kinect.

teleportPlayer: Armadinha vinda da mesa iterativa.

startGame: Trata da informação das chaves.

lighScary: Trata da informação da flash light vinda da mesa iterativa.

O script player trata das colisões existentes:

```
void OnTriggerEnter(Collider other) { … }
```

Esta função é ativada quando dois objetos colidem, desta forma é comparada a tag do segundo objeto com uma lista de tags existentes na função. Caso esteja na lista é realizada a ação definida na função do OnTriggerEnter.

BITALINO

Usámos um projeto que foi fornecido na aula e adaptámos para funcionar com o sensor pretendido. O projeto vinha com dois pacotes, um que contém o ficheiro main (test.java) e o pacote BITalino que tem ficheiros de configuração. Foi tudo desenvolvido no ficheiro test.java e foi o único ficheiro a ser modificado.

Basicamente, este ficheiro pede para ser selecionado um sensor (no nosso caso o ECG), mas foram deixadas as outras opções caso fosse preciso adicionar sensores extras. É também pedido o número de amostras que se pretende recolher. De seguida faz-se a conexão ao nosso dispositivo Bitalino, se estiver tudo correto, é comparado se o canal ocupado corresponde ao que nós selecionámos. A partir daí, enquanto a pessoa estiver a jogar os batimentos vão ser registados. Os valores usados são baseados numa média feita das amostras introduzidas. Inicialmente, é feito um registo dos primeiros 15 valores médios e a partir daí é estabelecido uma base para podermos comparar com os valores recolhidos durante o jogo, pois assim é possível saber quando é que o jogador está a ficar mais nervoso. Relativamente aos pontos dados, é sempre enviado 1 ponto quando se envia o batimento cardíaco (de 2 em 2 segundos) e se houver uma subida do batimento cardíaco ligeira, ou seja, se a média dos batimentos atuais tiver uma diferença de 20 para a média base, são enviados 30 pontos, e se houver uma grande subida, diferença de 40, são enviados 70 pontos. Tanto os valores dos pontos como do batimento cardíaco são guardados num ficheiro (valoresbpm.txt) para depois serem enviados. Isto porque não foi possível fazer a comunicação com o socketio neste ficheiro, pois a versão que funcionava no Java era inferior à versão que estava a ser usada pelas restantes componentes (que era superior) e não foi possível alterar.

Para permitir a comunicação com as restantes componentes foi então desenvolvido um ficheiro: teste.js. Este ficheiro tem como principal função ler os valores do ficheiro “valoresbpm.txt” e passa-los para um objeto JSON, enviando depois este objeto para a mesa interativa e para o jogo.

KINET

O Kinect foi desenvolvido com base num exemplo oferecido pelo Kinect.

A base do código passa por detetar um corpo através do sensor. O corpo está previamente classificado, através de uma biblioteca própria, em que cada articulação corresponde a um JOINT (ponto do corpo) e assim é criado um vetor que liga estas junções, por exemplo, temos o tornozelo direito (um JOINT) que se liga ao joelho direito (outro JOINT).

Após a deteção do corpo, é possível então originar comandos a partir dos gestos humanos e da matriz do Kinect nas 3 dimensões. No nosso caso, foi só necessário a deteção em 2 dimensões (X e Y).

O Kinect foi essencialmente usado para detetar os passos do corpo e enviar uma mensagem para o servidor central sempre que este ocorre. Um passo é detetado através de dois thresholds (mínimo e máximo) relativo ao tornozelo. Optámos por este JOINT pois devido à instabilidade que o Kinect às vezes tem nas extremidades, pode induzir em erro a deteção das coordenadas, daí a nossa preferência em comparação ao JOINT dos pés.

Os thresholds são usados para detetar quando é que um pé está levantado e a mensagem só é enviada no fim do movimento, para evitar que as mensagens sejam enviadas, frame a frame, durante este. Ou seja, o problema seria, se enviássemos uma mensagem por cada vez que o pé se levantasse (movimento entre o threshold mínimo e máximo), seriam enviadas N mensagens por cada N frames. Também guardamos a última posição do pé para comparar com o threshold mínimo de forma a confirmar que o movimento terminou.

Esta implementação é feita no método Reader_FrameArrived(), que basicamente é executado por cada frame obtida pelo Kinect. Para a comunicação com o servidor nodeJS é utilizada a biblioteca Quobject.SocketIoClientDotNet.Client e System.IO.

Abaixo é apresentada a parte principal do código, que filtra o movimento e envia para o servidor:

```
int diferenca_yLeft = this.lastPositionYLeft - (int)jointPoints[JointType.FootLeft].Y;
int diferenca_yRight = this.lastPositionYRight - (int)jointPoints[JointType.FootRight].Y;

Console.WriteLine("RIGHT  DIFF: " + diferenca_yRight);
Console.WriteLine("Left   DIFF: " + diferenca_yLeft);
if (this.step && upDown == 0)
{
    Console.WriteLine("RIGHT  DIFF: " + diferenca_yRight);
    Console.WriteLine("Left   DIFF: " + diferenca_yLeft);
    this.lastPositionYLeft = (int)jointPoints[JointType.FootLeft].Y;
    this.lastPositionYRight = (int)jointPoints[JointType.FootRight].Y;
    Console.WriteLine("########################## STEP");
    step = false;
    upDown = -1;
    socket.Emit("movement", "walk");
}
else
   if ((diferenca_yLeft > 10 && diferenca_yLeft < 50) || (diferenca_yRight > 10 && diferenca_yRight < 50))
    {
    Console.WriteLine("RIGHT  DIFF: " + diferenca_yRight);
    Console.WriteLine("Left   DIFF: " + diferenca_yLeft);
    Console.WriteLine("UPPPPP");
    step = true;
    upDown = 1;

}
else
{
    upDown = 0;
}
```

MESA INTERATIVA

Ficheiros

As únicas pastas que interessam são as seguintes:

- Pasta "public" - aqui encontram-se os ficheiros de css e javascript, e as imagens do jogo (style.css, index.js)
- Pasta "views" - aqui encontra-se a página html que é apresentada (layout.hbs, index.hbs)

Scripts

A página "layout.hbs" apresenta o layout da página.

A página "index.hbs" inicialmente apresenta:

- uma caixa de texto para inserir o IP do servidor
- uma caixa de texto para inserir o porto do servidor
- um botão para para estabelecer a ligação ao servidor
- de seguida qualquer alteração na página é feita pelo ficheiro index.js

A comunicação com o servidor é feita pela biblioteca socket.io.js

"index.js"

Métodos	Definição
createMap()	Cria o novo layout da página, cria o mapa do jogo, as divisões dos monstros e das armadilhas, dos sons, das chaves, e da secção d testes
checkPoints()	Verifica se os pontos atuais do jogador são suficientes para ativar alguma das armadilhas ou monstros, fazendo com que os botões respetivos fiquem ativos ou inativos.
startup2()	Apresenta uma mensagem inicial a indicar quais dos dispositivos (Kinect, Dive, BITalino), estão ligados e prontos para começar. Assim que estiver tudo ligado, o jogo começa.
movement()	Função de teste que emite movimento ao jogador 1.
sendKeys()	Envia as posições das chaves para o jogador 1, para estas serem colocadas no jogo.
gameOver()	Finaliza o jogo.
$(document).ready	Associa eventos de mensagens recebidas pelo servidor a partir da função "socket.on":
lvl	recebe informação sobre a posição do jogador 1 e apresenta-a no mapa do jogador 2
kinnect	liga o kinnect ao jogo (testes)
batteryState	recebe informação da bateria da laterna do jogador 1
keyPickup	recebe informação de quando o jogador 1 apanha uma chave
heartBeat	recebe informação da pulsação do jogador 1
finish	recebe informação a indicar se o jogo acabou

CONCLUSÃO

Num modo geral, a realização deste projeto foi bastante gratificante no fim. O seu processo fez com aprendêssemos a implementar um sistema multimodal usando equipamentos com que nunca tínhamos trabalho antes. Apesar do bom feedback das pessoas que testaram o nosso jogo, sabemos que o jogo em si não estava a 100% de acordo com as nossas expectativas. Isto é, sabemos que ao nível da jogabilidade houve algumas falhas, por não ser muito assustador ou não termos concretizados os objetos tangíveis na mesa interativa. Porém, não deixamos de estar orgulhosos com o nosso projeto, pelo que conseguimos implementar o que era essencial e que nos propusemos, alcançando uma aplicação completamente apresentável e jogável.

É ainda importante referir alguns problemas que encontrámos nas diferentes tecnologias. Houve dificuldades na implementação da ligação entre o Unity e o nodejs, bem como com o Java e o nodejs.

A transformação do jogo de 2D para 3d foi um pouco problemática devido a múltiplas formas de resolver este problema. Contudo foi usado o Google cardbox, este resolve grande parte dos problemas de camara e de apresentação, dando ao desenvolvedor um prefab da própria camara existente na aplicação cardbox da Google.

No desenvolvimento da aplicação o objeto player quando deslocava acontecia que player passava as paredes que limitavam a área de jogo. Este problema foi resolvido trocando o tipo de movimento do utilizador de translate para addForce.

Relativamente ao BITalino, a informação disponível não era muita, e a encontrada era muito técnica, pelo que levou algum tempo até perceber a dinâmica do seu funcionamento.

Infelizmente, também não foi possível fazermos a demostração na mesa devido à dificuldade da calibração do toque, pelo que tivemos de usar apenas o browser.