FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Jogos Digitais

LUIZ CARLOS PINHEIRO JUNIOR

DESENVOLVIMENTO DE BACK-END PARA JOGOS MULTIJOGADORES COM SOCKET.IO E NODE.JS

Americana, SP

2014

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Jogos Digitais

LUIZ CARLOS PINHEIRO JUNIOR

frickajr@gmail.com

DESENVOLVIMENTO DE BACK-END PARA JOGOS MULTIJOGADORES COM SOCKET.IO E NODE.JS

Trabalho monográfico, desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Jogos Digitais da Fatec Americana, sob orientação do Prof. Me. Vitor Brandi Junior

Área de concentração: Jogos Digitais.

Americana, SP

2014

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

Pinheiro Junior, Luiz Carlos

P72d Desenvolvimento de back-end para jogos multijogadores com socket.io e node.js. / Luiz Carlos Pinheiro Junior. – Americana: 2014.

43f.

Monografia (Graduação de Tecnologia em Jogos Digitais). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.

Orientador: Prof. Me. Vitor Brandi Junior

1. Jogos digitais I. Brandi Junior, Vitor II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.6

LUIZ CARLOS PINHEIRO JUNIOR
frickajr@gmail.com

DESENVOLVIMENTO DE BACK-END PARA JOGOS MULTIJOGADORES COM SOCKET.IO E NODE.JS

Trabalho de conclusão de curso apresentado à Faculdade de Tecnologia de Americana como parte dos requisitos para obtenção do título de Tecnólogo em Jogos Digitais

Área de concentração: Jogos Digitais

Americana, 30 de Junho de 2014.

Banca Examinadora:

_____
Prof. Vitor Brandi Junior (Presidente)
Mestre em Gerenciamento de Sistemas de Informação
FATEC Americana

_____
Francesco Artur Perrotti (Membro)
Mestre em Engenharia Elétrica
FATEC Americana

_____
Luciene Maria Garbuio Castello Branco (Membro)
Mestre em Linguística Aplicada
FATEC Americana

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer aos colegas de classe e professores que me acompanharam nessa jornada que foi a graduação em Jogos Digitais. Gostaria de agradecer, também, à minha namorada, família e amigos pelo apoio e a todos que contribuíram de algum modo no desenvolvimento deste trabalho.

RESUMO

Este trabalho apresenta um estudo sobre desenvolvimento de jogos para multijogadores, explorando suas definições e seus tipos, buscando assim um melhor entendimento do tema.

Aborda também o estudo das tecnologias: Node.Js, uma ferramenta que possibilita a execução de Javascript no servidor. Socket.io é um módulo do Node.js que facilita a comunicação entre o cliente e o servidor. Também o Construct 2, uma ferramenta que auxilia no desenvolvimento do jogo, com ambiente visual e programação baseado em eventos.

Pretende-se mostrar como funciona a comunicação de um jogo multijogador utilizando essas tecnologias, através do desenvolvimento de um jogo da memória multijogador *online* baseado em turnos, o qual utilizou essas tecnologias para o desenvolvimento do jogo.

Verificando e apresentando o funcionamento do jogo desenvolvido, colocando o mesmo em um servidor na internet e executando partidas entre dispositivos diferentes, abordando seu funcionamento e podendo verificar assim, a eficácia das tecnologias estudadas.

Palavras-chave: Jogos Multijogador. Jogos Online. Back-End para Jogos

ABSTRACT

This paper presents a study on developing multiplayer games, exploring their definitions and their types, thus seeking a better understanding of the topic.

Also addresses the study of a few technologies: Node.js, a tool that enables the execution of JavaScript on the server. Socket.io is a module that facilitates communication between the client and the server. There is also Construct 2, a tool that assists in the development of the game, which provides a visual environment and event-based programming.

It is intended to show how the communication works in a multiplayer game using these technologies, by developing an online multiplayer turn-based memory game, which used these technologies in its development.

Verifying and presenting the operation of the game developed, putting it on a server on the Internet and running matches between different devices, addressing may verify their operation and thus the effectiveness of the studied technologies.

Keywords: Multiplayer Games. Online Games. Back-End Gaming.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AJAX: *Asynchronous Javascript and XML*.

CS: *Counter Strike.*

FPS: *First Person Shooter*.

HTML5: *HyperText Markup Language*.

LAN: *Local Area Network*.

OS: *Operating System*.

PC: *Personal Computer*.

SDK: *Software Development Kit*.

XML: *eXtensible Markup Language*.

SUMÁRIO

1 INTRODUÇÃO

A introdução começa com as considerações iniciais, depois serão expostos os objetivos que se pretende atingir e, por último, as justificativas para o desenvolvimento deste trabalho.

1.1 Considerações Iniciais

Desde o início dos jogos eletrônicos a interação entre multijogadores já estava presente, pois o fato de os jogadores terem de se enfrentar garantia o desafio e a imersão para os participantes. Com base nisso e no fato dos jogos para multijogadores via internet serem preferidos por boa parte dos jogadores, foi desenvolvido este trabalho.

1.2 Objetivos

O objetivo principal deste trabalho foi criar *back-end* para jogos multijogadores utilizando a tecnologia Node.js e seu módulo Socket.io, como complemento os outros objetivos são:

- Estudar jogos Multijogadores.
- Estudar a tecnologia Node.js, com o Socket.io.
- Explorar os recursos existentes no Socket.io.
- Criar uma proposta de trabalho utilizando as tecnologias estudadas.
- Desenvolver um jogo multijogador com as tecnologias citadas.
- Verificar a solução apresentada e seus resultados.

1.3 Justificativas

O desenvolvimento de jogos se tornou algo mais procurado, e jogos para multijogadores *online,* são umas das partes importantes nesse passo, visto que as pessoas ficam cada vez mais tempo conectadas.

Este trabalho pretende contribuir com as pessoas que estão desenvolvendo jogos para multijogadores, mostrando a eficiência das ferramentas utilizadas, Node.js e Socket.io, para facilitar a criação do jogo, otimizando o tempo de produção.

Explorar os protocolos de comunicação via internet, com o objetivo de verificar a questão de desempenho e confiança na troca de informação, buscar uma solução sólida para o desenvolvimento de jogos, para vários jogadores que se comunicam pela rede sem a perda de dados. Com isso gerar a boa experiência de enfrentar outras pessoas conectadas, jogando como se estivessem no mesmo lugar.

2 JOGOS PARA MULTIJOGADORES

Inicialmente, os jogos eram processados apenas pelo mesmo dispositivo, ou seja, jogadores jogavam com controles ligados no mesmo local e acompanhando pela mesma tela. Nos primeiros videogames os jogos já eram multijogadores (jogos para dois ou mais jogadores) e boa parte dos consoles de videogames já vinha de fábrica com dois controles. Outros, porém, podiam ser adquiridos, pois tinham conexão para mais de um controle.

Com a evolução da internet, os jogos para multijogadores começaram a ser desenvolvidos de uma nova forma, na qual cada jogador poderia estar em um local diferente, com um computador diferente e jogar ao mesmo tempo, uma mesma partida. Para isso, precisariam estar com o computador ligado a uma linha telefônica que permitisse a comunicação com a internet, surgindo, assim, os jogos para multijogadores *online*, ou via internet. Outra forma de jogos multijogadores são os via rede, jogados localmente, que são parecidos com multijogadores via internet, porém ocorre apenas em rede local.

Novak (2012) aborda os jogos multijogadores da seguinte forma:

> “[...] Os primeiros consoles continham dois botões, um para jogar com um jogador e outro para o modo de dois jogadores. Os jogadores se revezavam contra o próprio jogo. A experiência de cada jogador era idêntica ao que teria sido no modo para um jogador. A única diferença era que o jogo manteve o controle de pontuação de ambos os jogadores e comparou os dois para determinar o vencedor. Os jogadores não competiam um contra o outro na mesma tela, mas ambos sabiam que o jogo iria declarar apenas um deles como vencedor. [...]”.

Para explicar um pouco mais o conceito de jogos multijogadores, serão abordados separadamente os diferentes modos de multijogador: local, em rede e *online*, assim também como a diferença de intervalos de tempo (jogos por turno e em tempo real).

2.1 Multijogador Local

No modo multijogador local todos os jogadores devem estar no mesmo espaço e jogar na mesma tela usando controles separados. Este é um modo comum nos jogos de console, que possibilita quatro jogadores jogarem no mesmo local. Uma vez que todos os jogadores partilham a mesma tela, cada jogador pode ver o que os outros jogadores estão fazendo. Os jogadores também podem participar em jogos locais em um computador, envolvendo não só o compartilhamento de uma tela, mas os dispositivos de entrada como o teclado e mouse (NOVAK, 2012).

Figura 1 – *Tennis for Two*

Fonte: http://www.bnl.gov/about/history/firstvideo.php

Os jogos para multijogador local existem desde os primeiros jogos inventados. Em 1958, o Dr. William Higinbotham criou o *Tennis for Two* no Brookhaven National Laboratory (Figura 1). Considerado o primeiro jogo eletrônico do mundo, ele foi desenvolvido utilizando um computador analógico (Donner Model 30) e um osciloscópio de cinco polegadas. Foi desenvolvido para curar o tédio de visitantes e tinha uma mecânica que simulava uma partida de tênis entre dois jogadores (NOSOWITZ, 2008).

Em 1972 nascia a Atari, fundada por Nolan Bushnell e com ela veio o *Pong*, projetado em uma máquina de *arcade*, que foi um sucesso desde o primeiro dia. Duas pessoas se enfrentavam em uma partida de *Pong*, usando como controle um potenciômetro para movimentar cada jogador uma barra em lados opostos da tela (SALEN; ZIMMERMAN, 2012).

2.2 Multijogador Rede

Jogos baseados em rede permitem que os jogadores compartilhem o jogo em uma rede local (LAN) sem compartilhar a mesma tela ou dispositivo de entrada. Ao contrário do multijogador local, onde é compartilhada a mesma tela, os jogadores podem ocultar informações uns dos outros, porque não estão compartilhando a mesma tela, só o jogo em si. Em uma partida em rede, os jogadores poderão levar seus computadores pessoais para um local e jogar juntos como um grande grupo. (NOVAK, 2012).

Figura 2 – *Counter-Strike* e *Age of Empires*.

Fonte: Elaborada pelo autor.

O fato de os jogadores não dividirem mais a mesma tela, possibilitou a disputa baseada em estratégias, pois como um jogador podia esconder as ações do outro aumentava a possibilidade de fazer algo para surpreender e vencer o jogador adversário. Exemplo disso são os jogos mostrados na Figura 2.

2.3 Multijogador *Online*

Como os jogos em rede, jogos *online* também representam uma forma onde os jogadores podem conectar seus computadores a uma rede e partilhar o jogo, só que no caso de jogos *online*, a rede é a Internet, ou seja, os jogadores não necessitam ficar no mesmo local, desde que estejam conectados na internet (NOVAK, 2012).

Por volta de 1989 começaram a surgir os primeiros jogos para multijogadores *online*, quando jogadores utilizavam modens ligados a linha telefônica para se comunicarem e desafiarem uns aos outros para uma partida. Com a chegada da banda larga, ficaram acessíveis a um público maior, conexões de alta velocidade e a partir disso os jogos *online* se popularizaram.

Figura 3 – *Half-Life*

Fonte: http://store.steampowered.com/app/70/

Os jogos do tipo FPS conquistaram a preferências dos jogadores. *Half-Life*, lançado em 1998, foi eleito "Melhor Jogo do Ano" e considerado um dos jogos mais revolucionários da história do estilo (VALVE CORPORATION, 2014).

Counter Strike, umas das modificações de *Half-Life*, virou um dos jogos mais jogados de 2013, o CS, que se utilizava de uma mecânica onde era possível escolher ser polícia ou ladrão, com diversos objetivos, como derrotar os oponentes e

outros específicos de cada lado, terroristas plantando bomba e antiterroristas salvando reféns (FREDERICO, 2014).

Os jogos multijogadores *online* já se tornaram algo comum, são jogados por várias pessoas e em diversos dispositivos, tais como smartphones, computadores e consoles. Com o passar dos anos, desde o primeiro jogo para multijogadores, a tecnologia evoluiu e junto com ela a internet, assim com as ferramentas de desenvolvimento, tornando o consumo desses jogos mais produtivos e dinâmicos.

Figura 4 – *League of Legends*

Fonte: http://gamesided.com/2014/02/13/league-of-legends-servers-attacked/

League of Legends (Figura 4) é outro exemplo de jogo *online* para multijogadores. Trata-se de um jogo de batalha em arena que foi reconhecido como esporte nos Estados Unidos, transformando os jogadores em cyber atletas. (MONTEIRO, 2013) São realizados diversos campeonatos ao redor do mundo e agora os jogadores têm a possibilidade de adquirir os vistos como esportistas. Estes são acontecimentos que estão cada vez mais comuns, pois diversos campeonatos de jogos para multijogadores estão se tornando regulamentados, aumentando ainda mais o esporte eletrônico mais conhecido como *e-sports* (FELIX, 2012).

2.4 Jogos Em Turnos

Jogos multijogador por turno são aqueles em que cada jogador executa suas ações durante sua jogada, e cada jogador joga em sua vez. Exemplos desses jogos são os jogos de tabuleiro e jogos de cartas, conforme apresenta a Figura 5.

Figura 5 – Xadrez e Buraco

Fonte: http://esporte.hsw.uol.com.br/

No tabuleiro tradicional e nos jogos de cartas, cada jogador tem a sua vez de mover uma peça ou jogar uma carta. Normalmente o tempo previsto para cada turno é ilimitado, portanto, os jogadores têm o tempo que eles desejam para refletir e planejar suas jogadas (NOVAK, 2012).

No desenvolvimento de um jogo multijogador em rede e *online* é importante observar que na partida por turno cada jogador poderá tomar uma ou mais ações em sua jogada e estas ações terão que ser repetidas para os outros jogadores, que estão na mesma partida, só que os outros jogadores não podem interagir até que chegue a próxima vez de jogar. Por exemplo, em um jogo de xadrez, quando um jogador move uma peça em sua rodada, a peça também deve ser movida na tela do outro jogador e assim que terminar o movimento a vez passará para o outro jogador.

2.5 Jogos Em Tempo Real (*Real Time*)

O oposto de um jogo baseado em turnos é conhecido como um jogo em tempo real. Neste caso, não existe qualquer intervalo de tempo entre as jogadas. Ao contrário do jogo por turno, esse exige ações físicas e rápidas, ao contrário do pensamento reflexivo e planejamento para jogos por turnos. Estes jogos são extremamente populares *online*, devido à capacidade de se comunicar em tempo real com outros jogadores (NOVAK, 2012).

Figura 6 – Fifa 2014 e Pong

Fonte: Elaborada pelo autor.

Jogos para multijogadores em tempo real são aqueles que trocam informações o tempo todo, como por exemplo, uma partida de Pong via rede ou internet, quando é necessário passar diversas informações para os dois jogadores terem a mesma visão de jogo. Por exemplo, as coordenadas da bola, a posição do jogador conforme sua movimentação, ilustrado na Figura 6.

Pensando nesse tráfego de dados é necessário na programação levar essas informações em conta e verificar a perda de tempo na comunicação, os chamados *delays*, pois em um jogo disputado em tempo real. Milissegundos de diferença podem causar uma má experiência, deixando o jogador insatisfeito e causando os travamentos e impossibilitando o funcionamento correto do jogo.

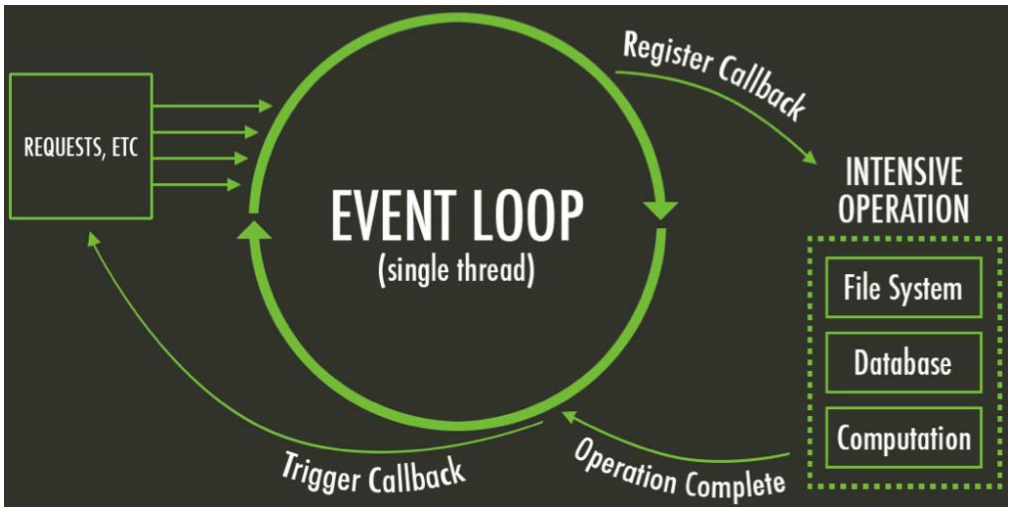
3 TECNOLOGIAS UTILIZADAS

Para o desenvolvimento desse trabalho foram escolhidas as tecnologias Node.js, que será usado no servidor, Socket.io para ajudar no gerenciamento de troca de informações entre o cliente e o servidor e o Construct 2 para desenvolvimento do jogo.

3.1 Node.Js

Node.js é uma plataforma de software usada para facilitar o desenvolvimento de aplicações de rede rápidas e escaláveis. As aplicações são escritas em JavaScript e é possível instalar e executar o Node no Windows, Mac OS e diversas distribuições Linux. Pode-se ver na Figura 7 a ilustração de seu funcionamento.

Figura 7 – Funcionamento do Node.js



Fonte: http://www.packtpub.com/build-network-application-with-node

O Node.js foi criado por Ryan Fahl em 2009, e utiliza a máquina virtual *JavaScript* V8, que foi criada pelo Google. Ela foi escrita em C++, o que garante a possibilidade de integrá-lo em qualquer aplicativo, não sendo necessário um navegador para executá-lo. Com isso, tornou-se possível a programação em

JavaScript também no lado do servidor, unificando a linguagem usada nos dois lados da comunicação (PEREIRA, 2013).

Essa ferramenta utiliza o paradigma de programação orientada a eventos. É altamente escalável e de baixo nível, sendo possível programar diretamente com vários protocolos de rede e internet. Um servidor rodando Node.js pode suportar milhares de conexões simultâneas, pois ao invés de ter que alocar memória para cada conexão, dispara um evento a ser executado dentro de seus processos, evitando gargalos em aplicações web com várias conexões (MOREIRA, 2013).

Figura 8 – *Hello Word* em Node.js

```
1  var http = require('http');
2
3  http.createServer(function (req,res) {
4      res.writeHead(200, {'Content-Type': 'text/plain'});
5      res.end('Olá Node.Js!');
6  }).listen(8080);
7
8  console.log('Servidor rodando em http://localhost:8080/');
```

Fonte: Elaborada pelo autor.

As linhas de códigos presentes na Figura 8 mostram basicamente como o código em Node.js funciona. Na linha 1 é instanciado o módulo HTTP**,** dentro dele será iniciado o servidor, mostrado da linha 3 a 6, que escutará a porta 8080.

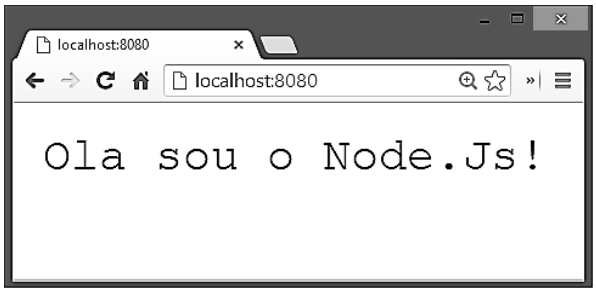
No exemplo, o Node.js aguarda uma notificação do sistema operacional de uma nova conexão, quando receber a conexão ele executará e voltará a dormir, sendo acionado novamente quando aparecer uma nova conexão (JOYENT INC, 2014).

Figura 9 – Executando Node.js

Fonte: Elaborada pelo autor.

Para executar o código e ativar o servidor de Node.Js, basta digitar a seguinte linha comando em seu terminal: `node teste.js`, conforme apresentado na Figura 9.

Figura 10 – Resposta ao Acessar o Node.js



Fonte: Elaborada pelo autor.

Pode-se verificar o servidor node rodando acessando a porta definida no teste.js, como o exemplo da figura 10.

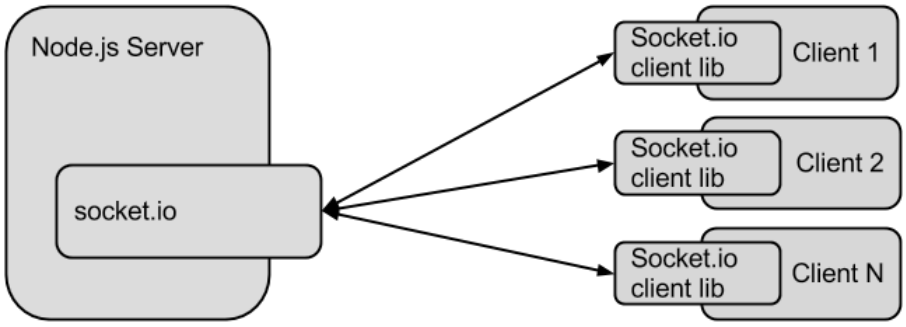
3.2 Socket.Io

Socket.io é uma biblioteca desenvolvida em JavaScript para troca de informações em tempo real, composta de duas partes, uma executada no cliente e outra junto com o Node.js no servidor

Segundo (PEREIRA. 2013) a vantagem dessa biblioteca é que ela verifica a configuração do cliente e, se não for possível fazer um conexão bidirecional com o *WebSocket* [1] , automaticamente emula uma conexão unidirecional, mandando requisições assíncronas de informações, utilizando AJAX para o servidor. Garantindo assim a conexão e a compatibilidade com diversos navegadores e clientes.

[1] WebSocket é uma tecnologia que permite a comunicação bidirecional por canais *full-duplex* sobre um único soquete, foi desenvolvido para ser usado junto com HTML5.

Figura 11 – Comunicação Socket.io



Fonte: http://blog.lightstreamer.com/2013/05/benchmarking-socketio-vs-lightstreamer.html.

O Socket.io funciona através de uma conexão criada entre o cliente e o servidor, ilustrado na Figura 11, possibilitando a troca de mensagens entre eles, não sendo necessário fazer a atualização da página, podendo explorar o conceito de uma aplicação em tempo real, onde é estabelecida uma comunicação entre os integrantes da rede, na qual o cliente envia a mensagem que é processada pelo servidor e logo em seguida reencaminhada para os demais clientes da rede.

Figura 12 – Exemplo Socket.io Servidor

```
1 var io = require('socket.io').listen(8080);
2 io.sockets.on('connection', function (socket) {
3     socket.emit('novo', { Ola: 'mundo' });
4     socket.on('outro evento', function (data) {
5         console.log(data);
6   });
7 });
```

Fonte: Elaborada pelo autor.

A figura 12 mostra um exemplo de código para servidor feito utilizando Socket.io. Na linha 1 o módulo socket.io está sendo instanciado e configurado para escutar a porta 8080, assim como mostrado no código da figura 7. Na linha 2 está sendo iniciado o Socket.io; a linha 3 está emitindo a mensagem “novo” para todos os que se conectam. Na linha 4, se o servidor receber a mensagem “outro evento”, o conteúdo será mostrado no console conforme a linha 5.

Figura 13 – Exemplo Socket.io Cliente

```
1 <script src="/socket.io/socket.io.js"></script>
2 <script>
3     var socket = io.connect('http://localhost:8080');
4     socket.on('novo', function (data) {
5         console.log(data);
6         socket.emit('outro evento', { my: 'data' });
7   });
8 </script>
```

Fonte: Elaborada pelo autor.

A Figura 13 apresenta a outra ponta da comunicação, ou seja, o cliente, possibilitando o entendimento de como o Socket.io funciona. A linha 3 inicia a comunicação, a linha 4 recebe a mensagem, na linha 5 é mostrada a mensagem recebida no console e logo em seguida, na linha 6, é enviada a resposta “outro evento” para o servidor que irá tratar conforme os códigos apresentados na Figura 12 na linha 4.

No Socket.io também é possível utilizar diversas salas de comunicação onde uma não interfere na outra. Isso permite que os eventos sejam emitidos para subconjuntos de lista de clientes conectados na mesma sala, simplificando a gestão das salas de comunicação.

Para participar de uma sala tem que ser chamada a função `join()` em um objeto de *socket* conectado. Por exemplo: `socket.join('sala')`. Para deixar uma sala deve ser chamada a função `leave()` em um objeto de *socket* conectado., por exemplo, `socket.leave('sala')`.

Há duas maneiras para emitir mensagens em uma sala, podendo ser usado `socket.broadcast.to('sala')` ou `io.sockets.in('sala')`. As transmissões são enviadas a partir de um objeto de *socket* e são recebidas por todos os clientes na sala, exceto para o que está emitindo.

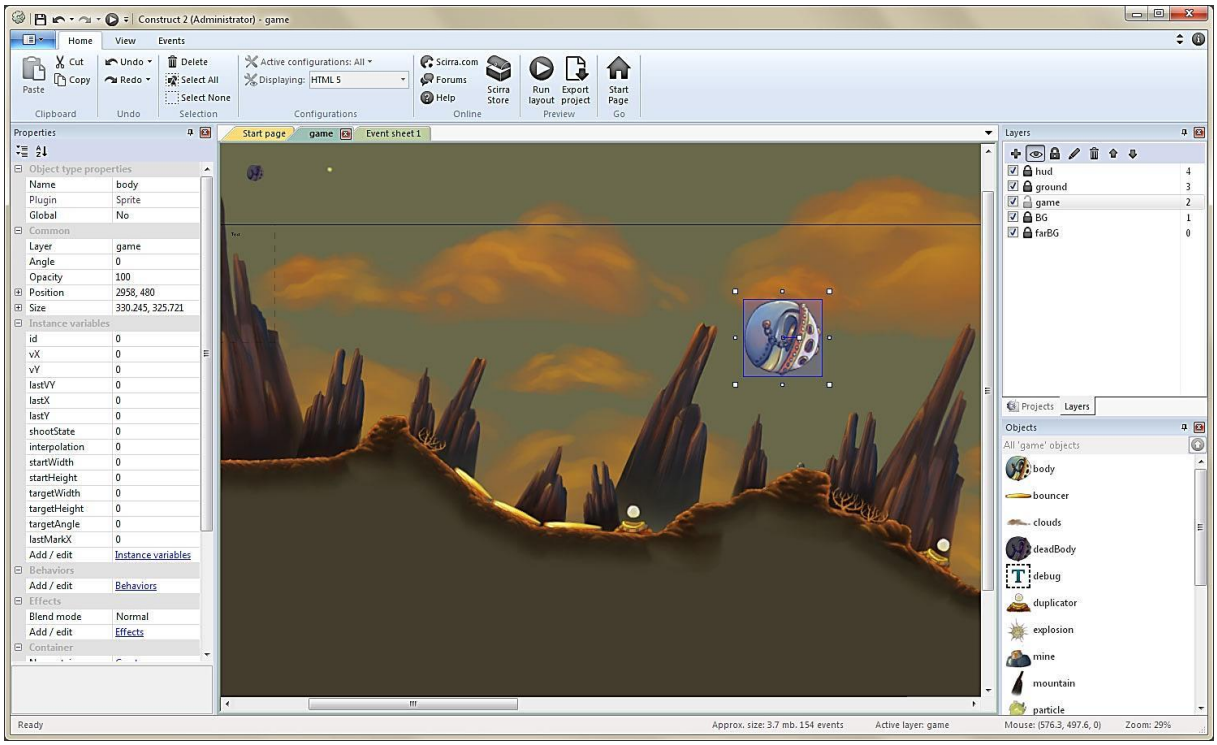
Para enviar mensagem para todos na sala incluindo o *socket* que enviou pode-se utilizar: `io.sockets.in(sala).emit('mensagem', data)`, para enviar uma mensagem para todos o clientes conectados independente das salas, incluindo o emissor, existe o comando `io.sockets.emit('mensagem', data)`.

3.3 Construct 2

Construct 2 é um motor de jogo criado pela Scirra, uma empresa de Londres, fundada em maio de 2011 pelos irmãos Ashley e Thomas Gullen. Essa é a segunda versão do Construct, a primeira era de código aberto escrita em C++ (SCIRRA, 2014).

Essa nova versão é focada no desenvolvimento de jogos em HTML5 e utiliza o paradigma de programação baseado em eventos, o que acaba auxiliando quem está começando na programação de jogos dada a sua facilidade. Sua interface também é baseada em clique e arraste, utilizando assim a familiaridade de outros aplicativos, além de ter vários comportamentos pré-programados, deixando o processo de desenvolvimento mais rápido (SCIRRA, 2014).

Figura 14 – Construct 2 - Editor de Layout

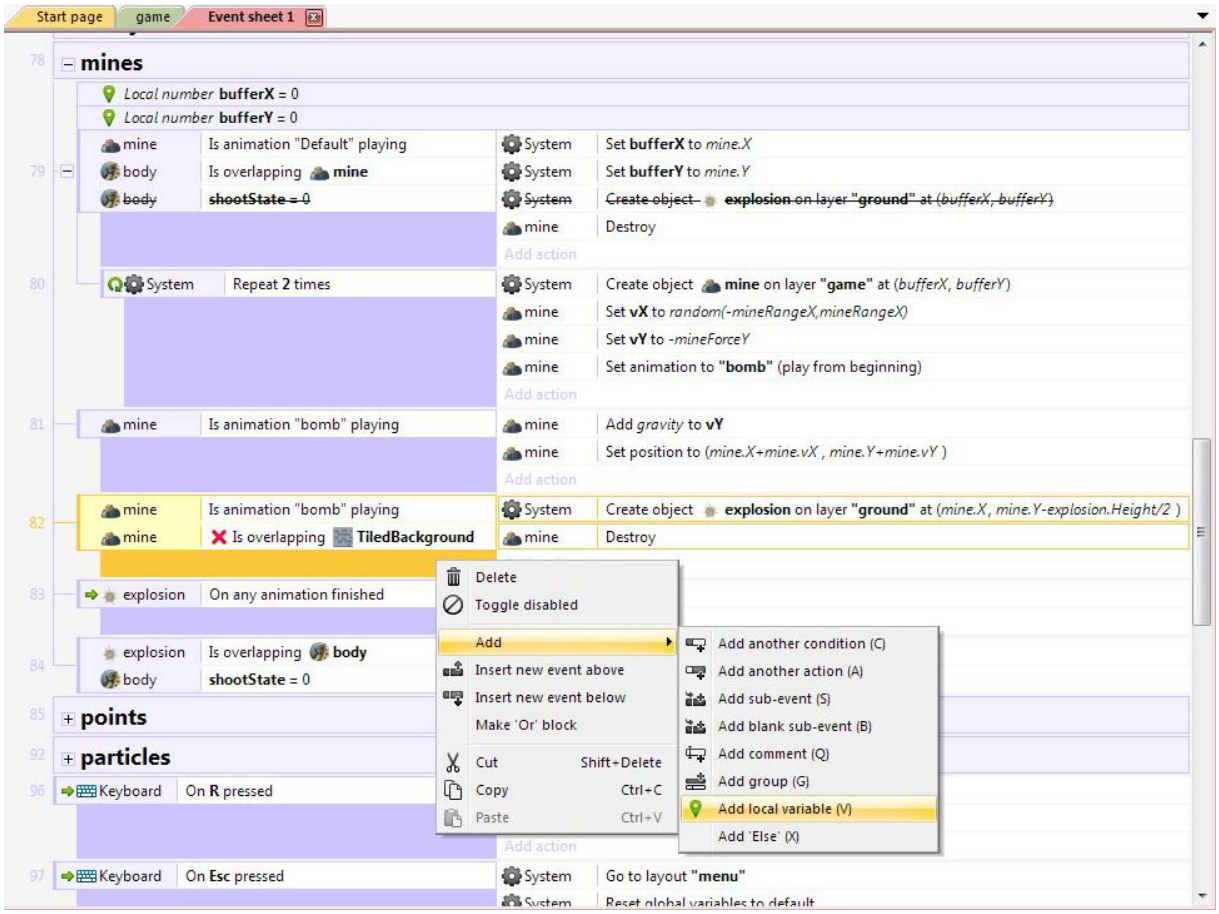


Fonte: https://www.scirra.com/construct2

O Editor de *Layout* fornece uma interface visual para projetar os níveis (Figura 14). É possível arrastar, girar e redimensionar objetos, visualizar efeitos

aplicados e, alterar as configurações na Barra de propriedades. Os objetos podem ser organizados em camadas separadas para uma melhor organização. Permitindo paralaxe e efeitos de mistura entre as camadas, considerando que paralaxe é o efeito visual onde dois objetos se movem na tela com velocidades diferentes, causando a sensação de profundidade na imagem mostrada.

Figura 15 – Construct 2 - Sistema de Eventos



Fonte: https://www.scirra.com/construct2

No *Event Sheet* (folha de eventos), deve-se escolher o objeto, selecionar uma condição ou ação, e adicioná-lo para o evento. Cada folha evento tem uma lista de eventos que contêm instruções condicionais ou gatilhos, conforme apresentado na Figura 15. Uma vez que estas forem verdadeiras, ações ou funções podem ser realizadas. Os grupos podem ser usados para ativar ou desativar vários eventos ao mesmo tempo e para a organização de grandes projetos.

A tecnologia Construct 2 baseada em HTML5 pode ser exportada para diversos dispositivos, o que garante a possibilidade do jogo desenvolvido ser

multiplataforma, ampliando assim o alcance do produto criado. Por exemplo, quando exportado como Web, pode ser distribuído por meio da publicação em um site, ou utilizando outros serviços de distribuição de jogos *online* disponíveis na Web. Outra forma de exportação é como aplicativo para PC desktop, Mac e Linux utilizando o *Node-Webkit*[2]. Além disso, também pode ser exportado para dispositivos móveis Android, Black Berry, iOS e Windows Phone 8.

O Construct 2 permite criar *plug-ins* e comportamentos adicionais utilizando Javascript SDK[3]. Baseado nisso foi instalado o *plug-in* para o gerenciamento do Socket.io na parte do cliente, desenvolvido por membros da comunidade *online* do fórum do site oficial do Construct 2 (SCIRRA, 2014).

[2] *Node-Webkit* permite que aplicativos feitos em HTM5 possam ser executados sem a necessidade de um navegador.

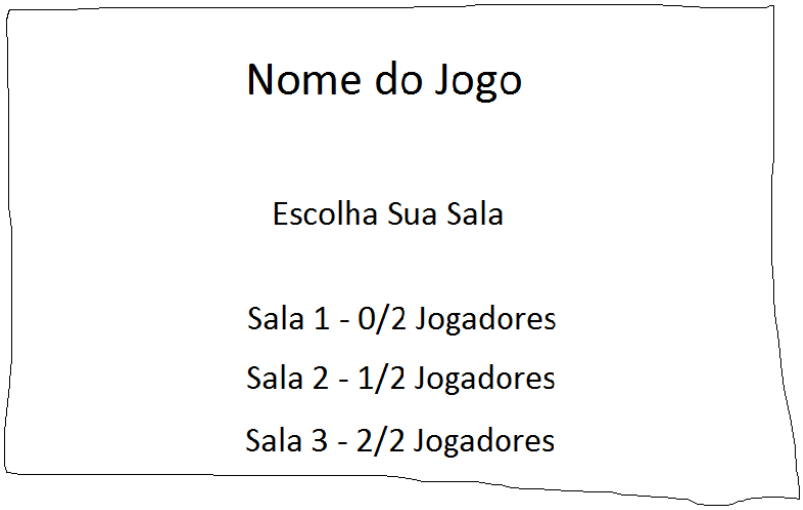
[3] Javascript SDK é um Kit para Desenvolvimento de Software em JavaScript.

4 PROPOSTA DE TRABALHO

Com as ferramentas abordadas anteriormente, foi desenvolvida uma proposta de trabalho para um jogo da memória, *online* e jogado por turno. Utilizando primeiramente o Construct 2 para a criação do jogo e sua jogabilidade. A comunicação entre o servidor e os clientes com o Socket.io e Node.js para gerenciar o processamento do servidor.

Nesta proposta constam algumas características, a saber: deverá conter uma tela que permitirá escolher em qual sala se deseja entrar para disputar a partida, sendo que todas as salas aceitarão até dois jogadores.

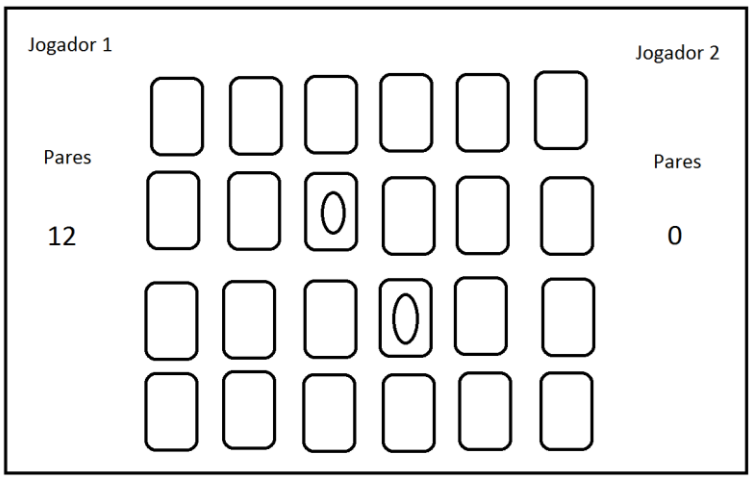
Figura 16 – Rascunho - Escolha de Sala



Fonte: Elaborada pelo autor.

O Jogador pode escolher a sala que deseja jogar, conforme a Figura 16, desde que a mesma não esteja cheia. Se for o primeiro a conectar na sala, o jogador vai para uma tela de espera onde aguardará o próximo se conectar. Caso seja o segundo jogador a entrar na sala os dois jogadores conectados são redirecionados para a tela de jogo.

Figura 17 – Rascunho – Tela de Jogo



Fonte: Elaborada pelo autor.

Na tela de jogo, Figura 17, é sorteado qual jogador iniciará a partida e, após ser definido quem começa, o mesmo escolhe duas cartas a serem viradas. Se o par for acertado o jogador que tem a vez repete o procedimento até errar o par, quando passa a vez para o outro jogador. O jogador que não tem a vez tem seu clique na tela bloqueado até chegar sua vez.

Quando a partida terminar, os dois jogadores são redirecionados para a tela de escolha de sala e limpa a sala para dois novos jogadores se conectarem.

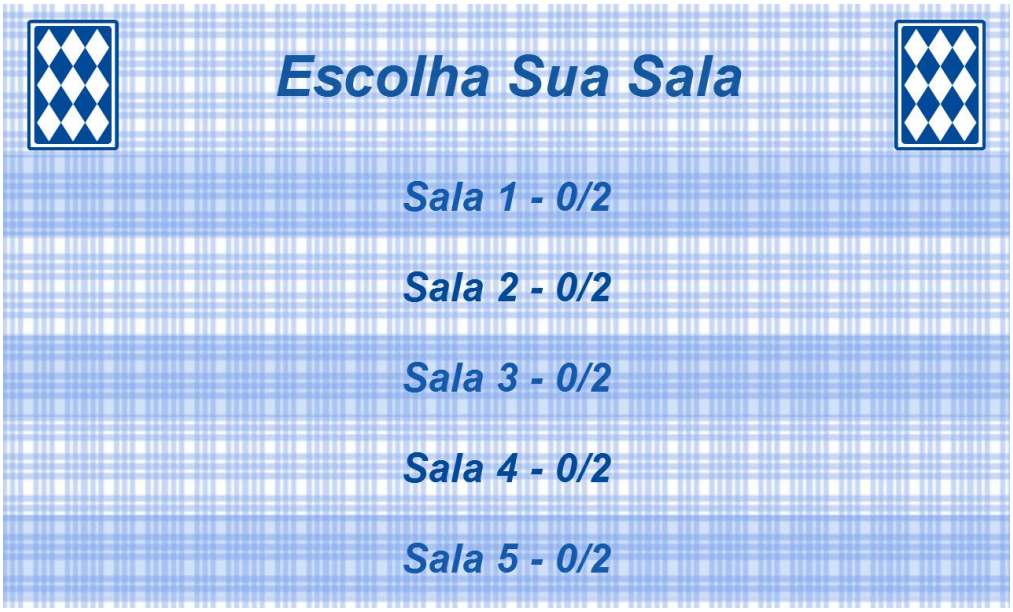
O jogo também necessita algumas características de comunicação entre o cliente e servidor. No servidor roda o Node.Js com o módulo Socket.Io, onde é gerenciada a conexão dos usuários ao servidor e as salas, as quais estão conectados, tratando toda informação recebida pela rede.

No cliente será o Construct2 com o *plug-in* Socket.Io, o jogador pode conectar nas salas e as informações do jogo são enviadas para o servidor.

5 DESENVOLVIMENTO

Baseado no tutorial escrito por Ball (2014) foi iniciado o desenvolvimento do jogo. Porém o tutorial ensina fazer um jogo para apenas um jogador, ou seja, como o objetivo é fazer um jogo para multijogadores o tutorial será apenas para auxiliar na mecânica do jogo e será necessário transformá-lo em multijogador desenvolvendo a parte do *back-end*.

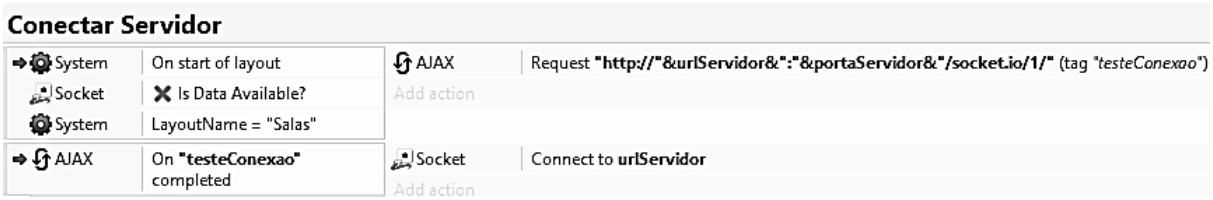
Figura 18 – Tela de Escolha de Salas



Fonte: Elaborada pelo autor.

A Figura 18 é a imagem da primeira tela do jogo. Nela o jogador pode escolher em qual sala deseja se conectar. Assim que a tela é carregada, é feita a conexão com o servidor conforme os comandos mostrados na Figura 19. As informações das salas são atualizadas dinamicamente conforme a entrada e saída dos jogadores. Isso é feito por uma troca de mensagens entre o servidor e os clientes como mostra as linhas de códigos das Figuras 20, 21 e 22.

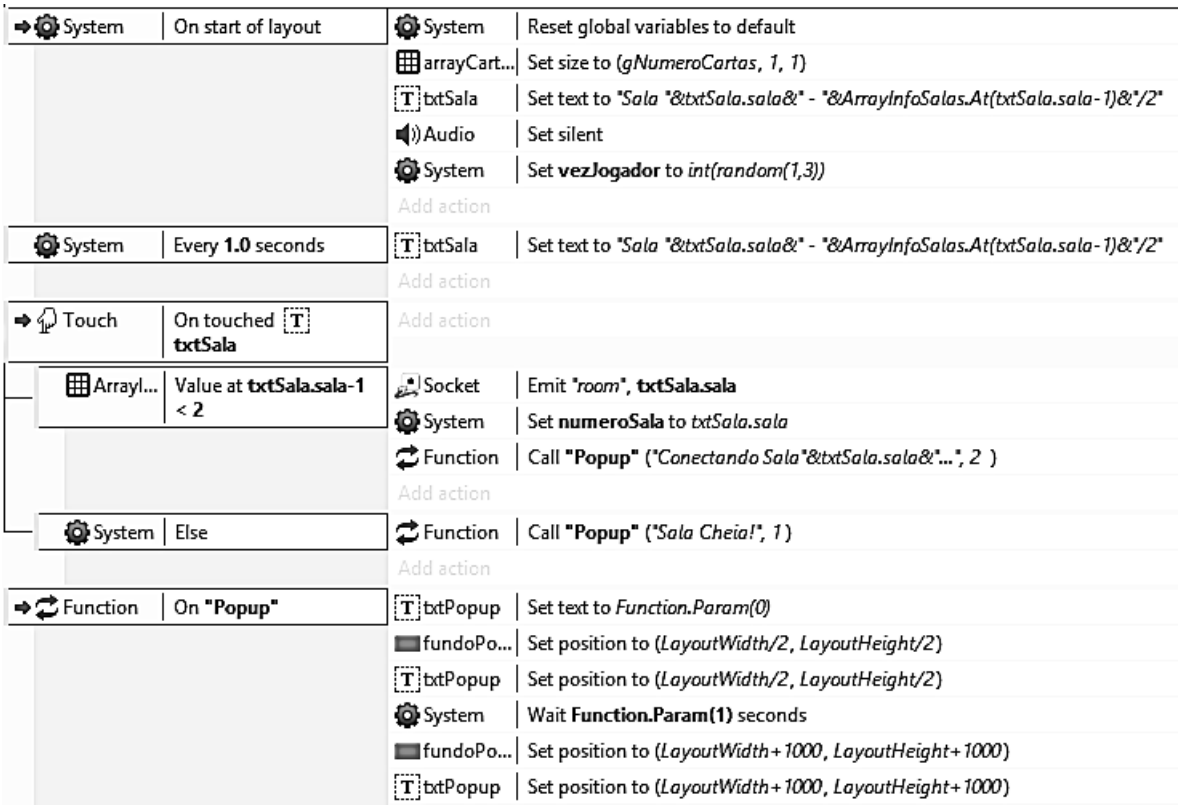
Figura 19 – *Event Sheet* - Conexão ao Servidor



Fonte: Elaborada pelo autor.

A Figura 20 é o *Event Sheet* que controla a primeira tela do jogo. Ao ser iniciado o jogo são zeradas as variáveis, é recebido um *array* [4] vindo do servidor, conforme comandos mostrados na Figura 21, possibilitando a atualização das informações da sala. Quando for escolhida uma sala o cliente envia uma mensagem para o servidor, informando em qual sala o cliente entrou e, caso a sala esteja cheia, é mostrada uma mensagem informando o jogador.

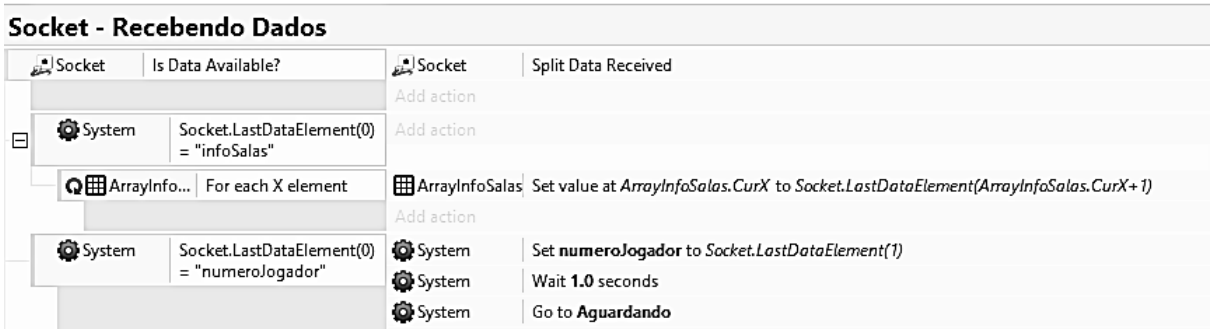
Figura 20 – *Event Sheet* - Escolha de Salas



Fonte: Elaborada pelo autor.

[4] *Array* também chamado de vetor (unidimensionais) ou matriz (bidimensionais) mantém uma série informações indexadas, geralmente do mesmo tamanho e tipo de dados.

Figura 21 – *Event Sheet* – Recebendo os dados do Servidor



Fonte: Elaborada pelo autor.

Na Figura 22 da linha 1 a 3, foram definidas as variáveis que serão usadas para gerenciar as salas. A variável `sala` guardará a informação de qual sala foi conectada, `qdteSalas` guarda a quantidade de salas e `infoSalas` é um *array* de informações de todas as salas. Através desse *array*, os clientes se comunicarão com o servidor para mostrar as informações apresentadas na figura 18.

Figura 22 – *Back-End* - Informações da Sala

```
1  var sala = 0;
2  var qtdeSalas = 5;
3  var infoSalas = "";
4
5  var io = require("socket.io").listen(3002);
6  io.set('log level', 1);
7  io.sockets.on("connection", function (socket)
8  {
9      socket.send(infoSalas);
10     socket.on("room", function(sala) {
11
12         socket.join(sala);
13         numeroJogador[sala] = contador[sala]++;
14         socket.send('numeroJogador,' + numeroJogador[sala]);
15
16         atualizaInfoSalas();
17
18         function atualizaInfoSalas() {
19             infoSalas = "infoSalas";
20             for (i = 1; i <= qtdeSalas; i++)
21             {
22                 infoSalas += ',' + numeroJogador[i];
23             }
24             socket.broadcast.emit('message', infoSalas);
```

Fonte: Elaborada pelo autor.

Pode-se ver na Figura 22 na linha 10 a criação da sala e na linha 12 a conexão do jogador na sala. Assim que entra na sala o jogador recebe o número de conexão naquela sala, conforme a linha 14, esse número é recebido pelo cliente conforme os comandos apresentados na Figura 19. Da linha 16 a 24 do código mostrado na Figura 22, é executada a função que atualiza as informações da sala.

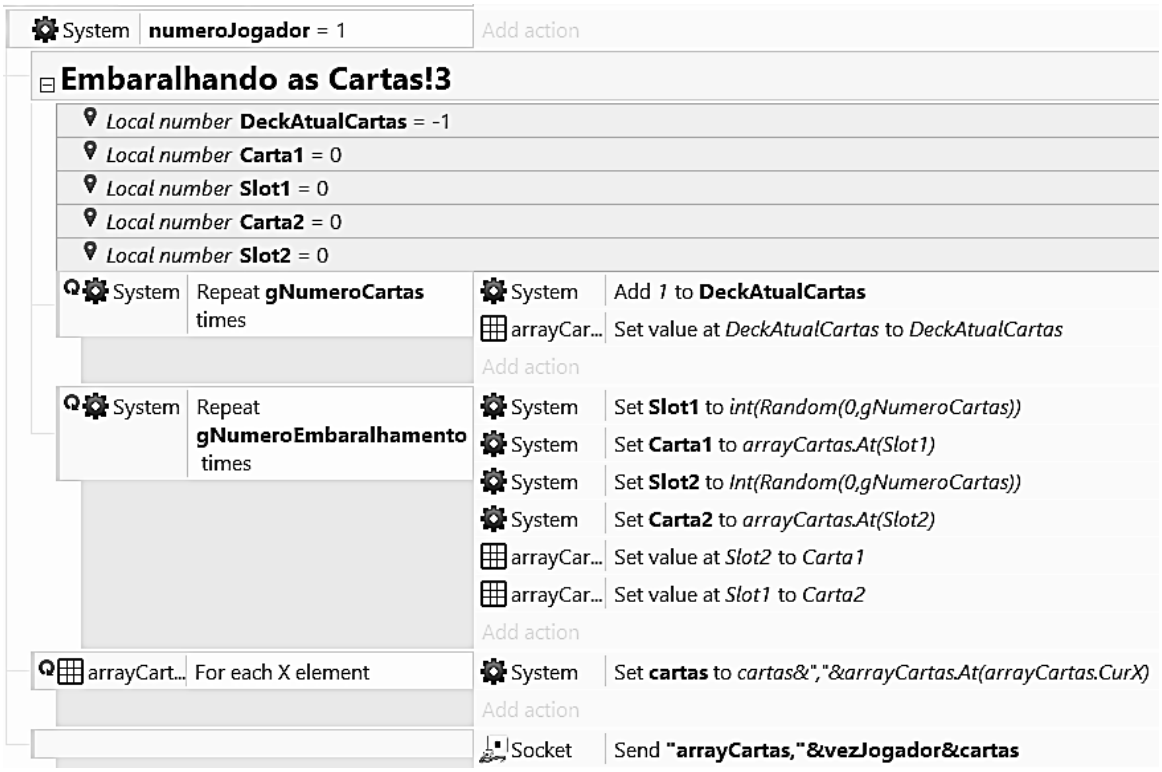
Figura 23 – Tela Aguardando Jogador

Fonte: Elaborada pelo autor.

Após escolher a sala o jogador é redirecionado para a tela da Figura 23, se for o jogador 1, as cartas do jogo são colocadas em um *array* e são embaralhadas pelo o número de vezes informado na variável `gNumeroEmbaralhamento` conforme o *Repeat* mostrado na Figura 24.

No próximo passo é enviado para o servidor o *array* de cartas já embaralhados e o jogador 1 aguarda o outro jogador conectar-se a sala para assim iniciar o jogo.

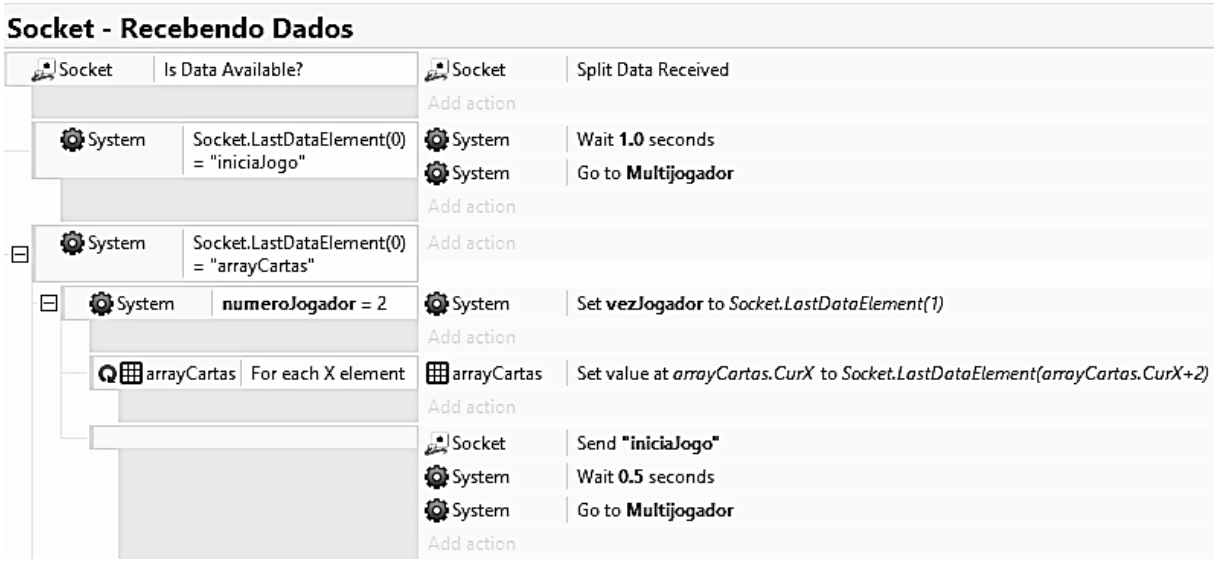
Figura 24 – *Event Sheet* – Embaralhando as Cartas



Fonte: Elaborada pelo autor.

Quando o jogador 2 conectar na sala, recebe as cartas embaralhadas vindas do servidor e junto com o jogador 1 são redirecionados para a tela de jogo (Multijogador), conforme eventos mostrados no *event sheet* da Figura 25.

Figura 25 – *Event Sheet* – Recebimento das Cartas e Inicio de Jogo



Fonte: Elaborada pelo autor.

A Figura 26 mostra os códigos para o tratamento na parte do servidor, recebendo as informações enviadas pelos clientes, como na Figura 24, reencaminhando para os outros clientes e sendo recebidas e tratadas pelos eventos no cliente, apresentadas na Figura 23.

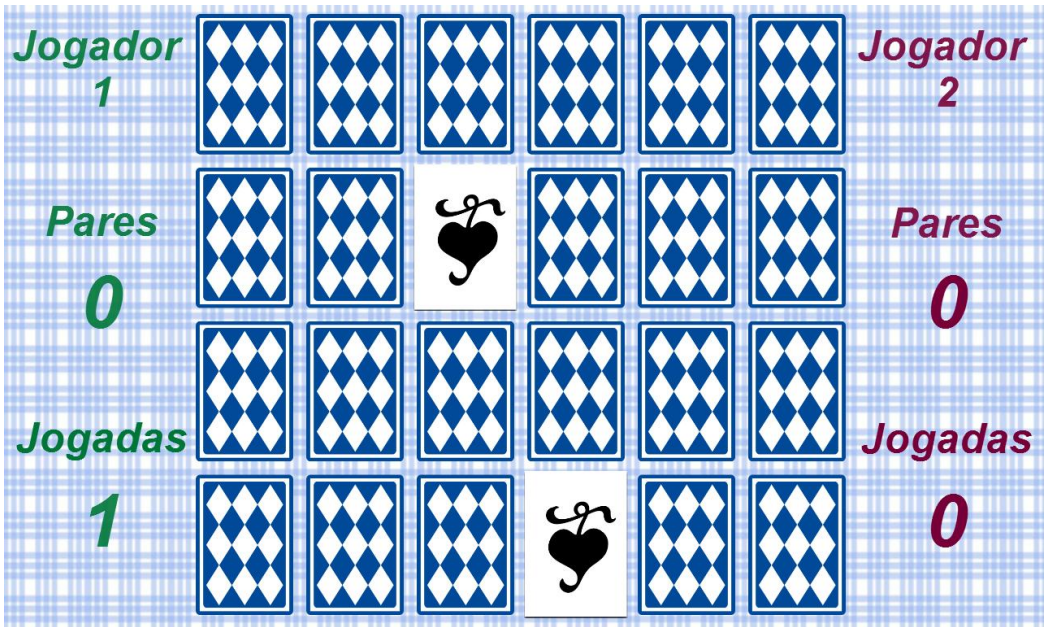
Figura 26 – *Back-End* - Recebimento das Cartas e Inicio de Jogo

```
57  socket.on("message", function (data)
58  {
59      var new_data = data.split(',');
60
61      switch (new_data[0])
62      {
63          case 'iniciaJogo':
64              socket.broadcast.to(sala).emit("message", 'iniciaJogo');
65              break;
66
67          case 'arrayCartas':
68              arrayCartas[sala] = data;
69              break;
```

Fonte: Elaborada pelo autor.

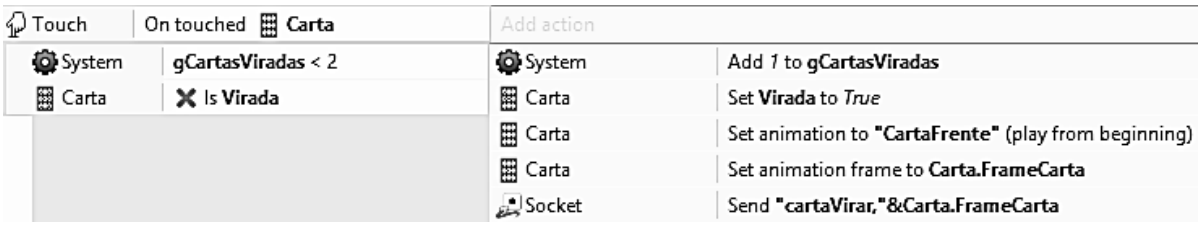
A Figura 27 mostra a tela de jogo para onde os jogadores são redirecionados quando todos estão conectados. Nela são mostrados os pares de jogadores e quantas jogadas foram executadas.

Figura 27 – Tela do Jogo



Fonte: Elaborada pelo autor.

Figura 28 – *Event Sheet* – Enviando Carta Virar



Fonte: Elaborada pelo autor.

Ao entrar na tela de jogo, ilustrada na Figura 27, o jogador com a vez pode escolher as cartas e, a cada carta clicada, é enviada uma mensagem para o servidor com o número da carta, conforme comandos na Figura 28. Essa informação será recebida pelo servidor e será reencaminhada para todos os clientes, conforme as linhas de código da Figura 29.

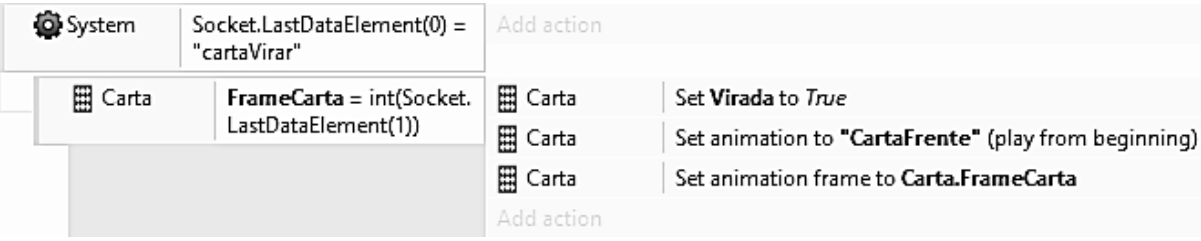
Figura 29 – Servidor Carta Virar

```
case 'cartaVirar':
    socket.broadcast.to(sala).emit("message", 'cartaVirar,' + new_data[1]);
    break;
```

Fonte: Elaborada pelo autor.

Quando a mensagem do servidor, contendo o número das cartas é recebida pelos outros clientes, as cartas são viradas também na tela do outro jogador conforme comandos mostrados na Figura 30.

Figura 30 – *Event Sheet* – Recebendo Carta Virar



Fonte: Elaborada pelo autor.

Figura 31 – *Event Sheet* – Verifica Par e Passa a Vez

Fonte: Elaborada pelo autor.

Quando o jogador vira a segunda carta é verificado se as duas são um par, conforme o e*vent sheet* na Figura 31 e, se verdadeiro, será computado ponto e as cartas são destruídas. Caso contrário, as mesmas são desviradas e a vez passa para o outro jogador. Essa informação é enviada para o servidor, sendo tratada conforme os códigos da Figura 32, que repassará para os outros jogadores da sala, onde são executadas as ações presentes na Figura 33.

Figura 32 – *Back-End* – Desvirar, Destruir Cartas e Passa Vez

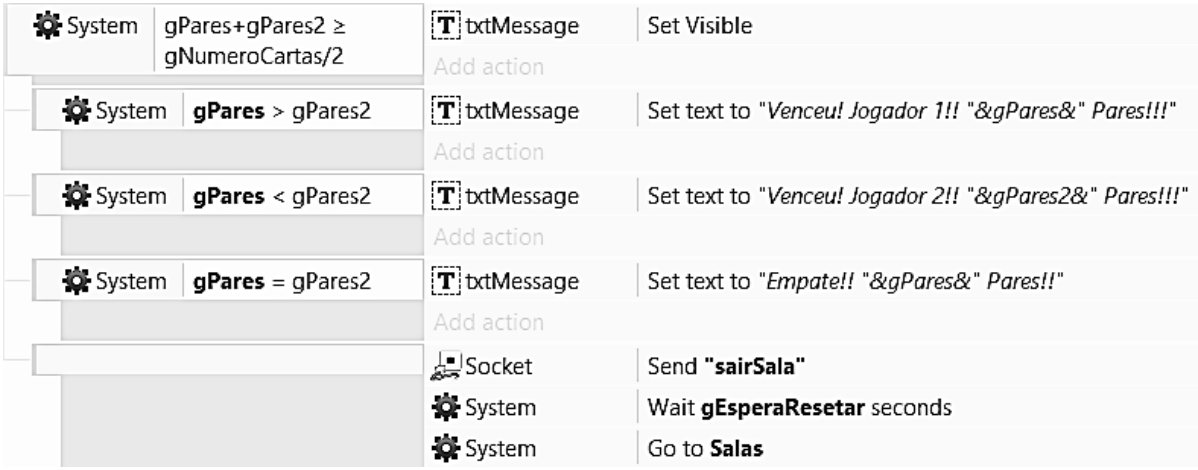
```
case 'cartasDesvirar':
    socket.broadcast.to(sala).emit("message", 'cartasDesvirar,' +
                                    new_data[1] + ',' + new_data[2]);
    break;

case 'destruirCartas':
    arrayDestruirCartas[sala] += ',' + new_data[1] + ',' + new_data[2];
    socket.broadcast.to(sala).emit("message", arrayDestruirCartas[sala]);
    break;

case 'vezJogador':
    vezJogador[sala] = new_data[1];
    socket.broadcast.to(sala).emit("message",'vezJogador,'+ vezJogador[sala]);
    break;
```

Fonte: Elaborada pelo autor.

Figura 33 – *Event Sheet* – Recebendo Desvirar, Destruir Cartas e Passa Vez

Fonte: Elaborada pelo autor.

Figura 34 – *Event Sheet* – Verificando Final de Jogo



Fonte: Elaborada pelo autor.

Quando as cartas acabam é verificado qual jogador possui mais pares, conforme comandos apresentados na Figura 34, logo depois a sala será limpa conforme as linhas de código da Figura 35 e todos os jogadores voltarão para a tela de escolha de salas.

Figura 35 – *Back-End* – Sair e Limpar Sala

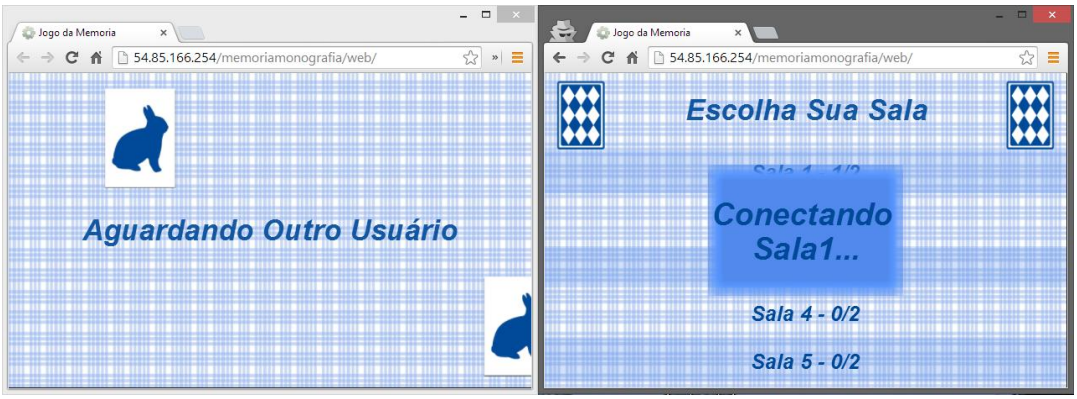
```
        case 'sairSala':
            socket.broadcast.to(sala).emit("message", data);
            socket.leave(sala);
            ResetarSala();
            break;
    }
});

function ResetarSala()
{
    contador[sala]=1;
    numeroJogador[sala]=0;

    atualizaInfoSalas();
}
```

Fonte: Elaborada pelo autor.

5.1 Considerações Finais do Desenvolvimento

Ao final do desenvolvimento foi instalado o servidor em uma instância na internet para testes e verificação dos resultados. Foram realizados testes com vários dispositivos diferentes, tanto computadores como dispositivos móveis disputando a mesma partida.

Figura 36 – Jogadores Conectando na Sala

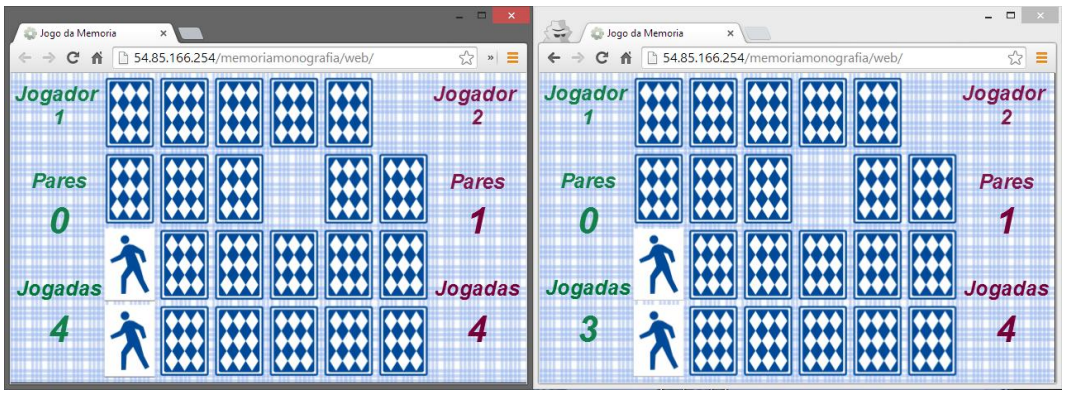


Fonte: Elaborada pelo autor.

A Figura 36 mostra que o jogador um já conectou na sala, e está aguardando o jogador dois que está se conectando.

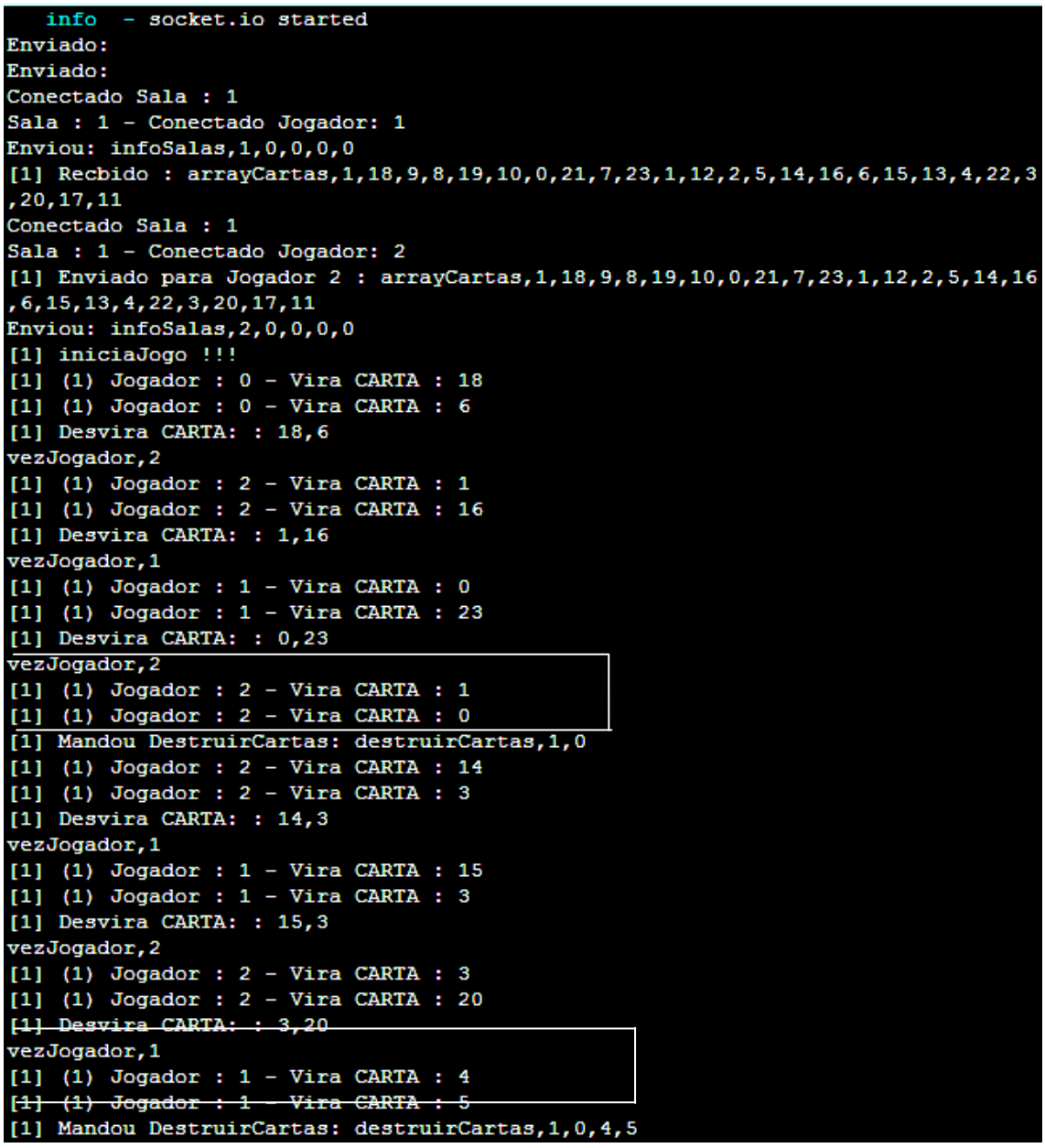
Pode-se verificar na Figura 37 a funcionabilidade da comunicação durante o jogo funcionando, as cartas clicadas pelo jogador um, estão sendo mostradas para o jogador dois, e as cartas acertadas pelo jogador dois tambem foram destruídas na tela do jogador um.

Figura 37 – Resultados na Tela de Jogo



Fonte: Elaborada pelo autor.

Figura 38 – Resultados de *Back-End*



Fonte: Elaborada pelo autor.

Os resultados do servidor podem ser observados no *Log*[5] ilustrado na Figura 38, onde todas as ações importantes são notificadas, possibilitando assim verificar o funcionamento do servidor e corrigir possíveis falhas na comunicação.

[5] Log é uma expressão utilizada para descrever o processo de eventos.

Analisando a imagem, pode-se ver no primeiro quadro da Figura 38 que o Jogador dois virou a carta 0 e 1 que são um par, por isso na linha de baixo, foram enviados os números das cartas a serem destruídos. Igualmente no segundo quadro da Figura 38, o Jogador um também acertou o par, essa ação também está ilustrada na Figura 37.

6 CONCLUSÃO

Durante o desenvolvimento deste trabalho foi possível compreender e classificar os tipos de jogos para multijogadores. Explorar as tecnologias Node.js, Socket.io e o Construct 2, produzindo assim um jogo de memória multijogador *online* baseado em turnos, que pode ser jogado em diversas plataformas e por várias pessoas ao mesmo tempo.

As tecnologias se mostraram eficientes, pois o jogo respondeu como esperado, permitindo a comunicação entre as ações dos dois jogadores. Durante o desenvolvimento foi verificada a importância de se tratar as mensagens trocadas entre o servidor e o cliente e também programar rotinas para garantir que todos os dados chegaram intactos.

Durante o desenvolvimento do trabalho a Scirra, desenvolvedora do Construct 2 lançou uma nova versão, que contém uma ferramenta que auxilia no desenvolvimento de jogos multijogador, portanto pode ser um tema interessante a se abordado em trabalhos futuros.

Outro tema que pode ser abordado em trabalhos futuros é a utilização do Node.js e Socket.io com outras linguagens e ferramentas não abordadas neste trabalho, verificando também a possibilidade de comunicação entre clientes programados com linguagens diferentes.

Por se tratar de uma tecnologia nova foi difícil encontrar referências, apesar disso os objetivos do trabalho foram concluídos e os projetos, incluindo a parte do servidor e o desenvolvimento do jogo, podem ser baixados do repositório GIT: https://bitbucket.org/frickajr/memoriamonografia

REFERÊNCIAS

BALL, Kim. **Creating a Memory Match Game**. Disponível em: <https://www.scirra.com/tutorials/280/creating-a-memory-match-game>. Acesso em: 20 mar. 2014.

COLUMBIA UNIVERSITY COMPUTING HISTORY. **Tennis for Two**. Disponível em: <http://pongmuseum.com/>. Acesso em: 18 abr. 2014.

FELIX, Felipe Santana. **E-Sports! Isso ainda vai ser grande no Brasil.** Disponível em: <http://canaltech.com.br/materia/games/E-SportsIsso-ainda-vai-ser-grande-no-Brasil/>. Acesso em: 22 ago. 2012.

FREDERICO, Carol. **Counter Strike é de longe o jogo mais popular nas LAN Houses**. Disponível em: <http://www1.folha.uol.com.br/folha/informatica/ult124u13182.shtml>. Acesso em: 18 fev. 2014.

JOYENT INC. **Node's goal is to provide an easy way to build scalable network programs**. Disponível em: <http://nodejs.org/about>. Acesso em: 18 abr. 2014.

MONTEIRO, Rafael. **League of Legends é reconhecido como esporte pelos Estados Unidos.** Disponível em: <http://www.techtudo.com.br/jogos/noticia/2013/07/league-of-legends-e-reconhecido-como-esporte-pelos-estados-unidos.html>. Acesso em: 15 jul. 2013.

MOREIRA, Rafael Henrique. **O que é Node.js?** 2013. Disponível em: <http://nodebr.com/o-que-e-node-js/>. Acesso em: 18 abr. 2014.

NOSOWITZ, Dan. Retromodo: **Tennis for Two, the World's First Graphical Videogame**. 2008. Disponível em: <http://gizmodo.com/5080541/retromodo-tennis-for-two-the -worlds-first-graphical-videogame>. Acesso em: 09 abr. 2014.

NOVAK, Jeannie. **Game Development Essentials**. 3. ed. New York: Cengage Learning, 2012.

PEREIRA, Caio Ribeiro. **Node.js**: **Aplicações web real-time com Node.js**. São Paulo: Casa do Código, 2013. 143 p.

SALEN, Katie; ZIMMERMAN, Eric. **Regras do Jogo: Fundamentos do Design de Jogos**. São Paulo: Blucher, 2012.

SCIRRA. **Construct 2**. Disponível em: <https://www.scirra.com/>. Acesso em: 18 abr. 2014.

VALVE CORPORATION. **Awards and Honors**. Disponível em: <http://www.valvesoftware.com/awards.html>. Acesso em: 18 fev. 20114.