INSTITUTO DE MATEMÁTICA E ESTATÍSTICA - USP
Trabalho de Formatura Supervisionado

WordStorming: Jogo de Palavras Multiplayer

Allan Amancio Rocha
Supervisor: Carlos Eduardo Ferreira

São Paulo
2020

A minha irmã mais nova, por ter contribuído, ainda que indiretamente, na inspiração e motivação para este trabalho.

AGRADECIMENTOS

Ao longo deste ano, tive a oportunidade de assimilar melhor a importância do coletivo e acredito que isso mereça ser ressaltado.

Um trabalho individual, como este, dificilmente é feito realmente sozinho. Sempre existem outras pessoas envolvidas de alguma forma, seja através de um ensinamento, de um apoio emocional, de uma mentoria ou qualquer outro tipo de ajuda ou contribuição.

Portanto, gostaria de agradecer a todos que me ajudaram de alguma forma, diretamente ou indiretamente, a construir este trabalho.

RESUMO

A prática vocabular em um novo idioma não costuma ser necessariamente prazerosa. Neste trabalho, propõe-se um jogo de palavras *multiplayer* - intitulado aqui WordStorming - com objetivo tanto de entretenimento quanto de aprendizado. Uma descrição completa do jogo é feita nesta monografia, juntamente com uma exposição do seu protótipo em forma de aplicação *web*. A realização do protótipo permitiu validar que os objetivos almejados do jogo são razoáveis. Portanto, criou-se aqui um jogo de palavras *multiplayer* em formato *web* com relativa validação, tanto no sentido de entretenimento quanto de aprendizado.

Palavras-chave: jogo; vocabulário; site; entretenimento; aprendizado.

ABSTRACT

The vocabulary practice in a new language is not usually pleasant by itself. In this dissertation, it is proposed a multiplayer game of words - entitled here WordStorming - with purpose of entertainment and helping in the learning process. A full description of the game is done in this monograph, alongside with an exposition of a prototype built as a web application. The prototype allows to verify that the purpose of the game is achievable. Therefore, with this work it was possible to create a multiplayer game of words as a web application and validate it as entertainment as a learning game.

Key-words: game; vocabulary; site; entertainment; learning.

SUMÁRIO

1 INTRODUÇÃO **1**
1.1 Objetivos 1
1.2 Motivações 1
1.3 Metodologia 2
1.4 Organização 2

2 JOGOS SEMELHANTES EXISTENTES **3**
2.1 Palavras Cruzadas 3
2.2 Caça-palavras 4
2.3 Boggle 4
2.4 Wordstorm 5
2.5 Scrabble 6

3 WORDSTORMING **7**
3.1 Regras 7
3.1.1 Elementos 7
3.1.1.1 Jogadores 7
3.1.1.2 Partida 8
3.1.1.3 Rodada 8
3.1.1.4 Sequência de Letras 8
3.1.1.5 Palavra 8
3.1.2 Pontuação 8
3.1.2.1 Pontos 9
3.1.2.2 Bônus 9
3.1.3 Objetivos 9
3.2 Funcionalidades 10
3.2.1 Configurações 10
3.2.2 Dicionário 10
3.2.3 Histórico 10
3.2.3.1 Sugestão 11
3.2.3.2 Relatório 11
3.2.3.3 Experiência 11
3.2.3.4 Conquistas 11
3.2.3.5 Habilidades 11
3.2.4 Singleplayer 12

4 BENEFÍCIOS EDUCACIONAIS **13**
4.1 Vocabulário 13
4.2 Aprendizado Contínuo 13
4.2 Desenvolvimento Cognitivo 13

5 CONHECIMENTOS NECESSÁRIOS **14**
5.1 Disciplinas Relevantes 14
5.1.1 Introdução a computação (MAC0110) 14
5.1.2 Algoritmos e estruturas de dados I (MAC0121) 14
5.1.3 Técnica de programação I (MAC0216) 15
5.1.4 Sistemas operacionais (MAC0422) 15
5.1.5 Técnica de programação II (MAC0218) 15
5.1.6 Introdução ao desenvolvimento de software (MAC0350) 15
5.1.7 Princípios de interação humano-computador (MAC0446) 15
5.2 Conceitos de Computação 15
5.2.1 Desenvolvimento web 15
5.2.1.1 Front-End 16
5.2.2.2 Back-End 16
5.2.2 Framework 16
5.3 Tecnologias Utilizadas 17
5.3.1 HTML 17
5.3.2 CSS 18
5.3.3 JavaScript 18
5.3.4 jQuery 19
5.3.5 JSON 19
5.3.6 Ajax 20
5.3.7 Bootstrap 20
5.3.8 Python 21
5.3.9 Flask 22
5.3.10 Trello 23
5.3.11 Visual studio code 24
5.3.12 Git 25
5.3.13 ngrok 26
5.3.14 Heroku 26

6 PROTÓTIPO **28**
6.1 Cronograma 28
6.2 Tarefas Primárias 29
6.2.1 Planejamento do projeto 29
6.2.2 Configuração do ambiente 30
6.2.3 Arquitetura de pastas 31
6.3 Páginas Iniciais 32
6.4 Identificação 34
6.5 Modo Treino 36
6.5.1 Sequência de letras aleatória 37
6.5.2 Temporizador 38
6.5.3 Interface principal 39

6.5.4 Entrada de palavras 40
6.6 Sala de Espera 41
6.7 Modo Partida 42
6.7.1 Sincronização dos jogadores 43
6.7.2 Fluxo do jogo 45
6.7.3 Detalhes técnicos 49
6.8 Hospedagem do Site 49
6.9 Dívidas Técnicas 51
6.9.1 Bugs 51
6.9.2 Refatorações 52
6.9.3 Melhorias 52
7 AVALIAÇÃO 53
7.1 Resultados 53
7.1.1 Perfil 54
7.1.2 Opinião 54
7.1.3 Valor 55
7.1.4 Extra 55
8 CONCLUSÃO 56
8.1 Aprendizados Pessoais 57
REFERÊNCIAS BIBLIOGRÁFICAS 58

1 INTRODUÇÃO

Um dos pontos chaves no aprendizado de um novo idioma é a aquisição e a retenção de vocabulário. Em geral, adquirir o vocabulário é a parte menos difícil, enquanto que retê-lo costuma ser o problema. Um bom método para resolver essa questão é a imersão contínua no idioma, seja através da vivência em um local onde se fale a língua ou através da simulação de um ambiente onde a pessoa esteja constantemente ouvindo, lendo e eventualmente falando o idioma. No entanto, muitas pessoas não possuem essa possibilidade, por conta de motivos geográficos, financeiros ou mesmo pessoais.

Afortunadamente, existem métodos mais acessíveis e específicos para a melhoria de vocabulário, como: a repetição e anotação de palavras, associação de palavras com desenhos ou imagens, *flashcards* (cartões de memorização), entre outras estratégias. Atualmente, com o ambiente digital, soluções computacionais se tornam também uma possibilidade cada vez mais presente na vida das pessoas. Alguns exemplos disso são as plataformas online de cursos de idiomas, programas de *flashcards* com sistema de repetição espaçada, assim como diversas outras ferramentas disponíveis com esse foco de melhorar o aprendizado idiomático e o desenvolvimento vocabular.

Em geral, essas soluções são boas e cumprem satisfatoriamente o papel de auxílio para a retenção de vocabulário. Contudo, elas possuem o ônus de requererem certa disciplina para que os resultados sejam alcançados. Esse fato acaba dificultando para algumas pessoas a aderência a esses métodos em médio e longo prazo, e consequentemente, da continuidade dos seus benefícios. Em outras palavras, as estratégias para o auxílio na retenção vocabular existem, mas comumente dependem de uma sujeição que eventualmente impede as pessoas de se favorecerem apropriadamente delas.

1.1 Objetivos

Pensando nesse cenário, o objetivo primário deste trabalho é a construção de um jogo original *multiplayer*, intitulado aqui WordStorming, baseado na formação de palavras. E por conta desse caráter linguístico do jogo, pretensiosamente almeja-se também, não só entreter o jogador, mas contribuir com o seu aprendizado em um idioma, mais especificamente na sua retenção vocabular. Portanto, de maneira geral, a ideia é que esse jogo sirva como forma de descanso ou complemento aos métodos adotados ao se aprender um idioma, pois além de se divertir, igualmente é possível praticar o vocabulário.

Além disso, um objetivo secundário deste trabalho é a avaliação da eficácia do jogo como tal, ou seja, como entretenimento propriamente dito, e a coleta de possíveis indícios do seu potencial educacional.

1.2 Motivações

A concepção do jogo proposto neste trabalho nasceu há cerca de 1 ano. A ideia foi naturalmente se aprimorando conforme foi sendo testada e jogada manualmente no papel com algumas pessoas (familiares e colegas de curso). Nisso, percebeu-se o potencial funcional do jogo. Em vista disso, juntamente com a constatação do valor linguístico que o jogo poderia ter, pensou-se nele como um bom trabalho de conclusão de curso a ser produzido, principalmente por conta da relevância que possui, se validado como possível solução para o problema de retenção vocabular aqui posto.

Um fator também fundamentalmente importante para a escolha desse tema é a possibilidade do aprendizado de habilidades computacionais orientadas para o desenvolvimento *web*. Conhecimentos técnicos como HTML, CSS e JavaScript, voltados

para o Front-End; Python e Flask, voltados para o Back-End; versionamento de código e arquitetura MVC são alguns exemplos de possibilidades de estudo através deste trabalho.

1.3 Metodologia

O desenvolvimento do jogo foi feito em plataforma *web*, utilizando, principalmente, as linguagens de programação *Python* e *JavaScript*. Os testes para a verificação da eficácia do jogo enquanto entretenimento e ferramenta de aprendizado foi feito através de formulários e entrevistas.

Um ponto relevante aqui é a complexidade do jogo criado. Esse fato, juntamente com a limitação de tempo e de recursos humanos, fora o caráter de incerteza funcional do jogo justificam a escolha da sua implementação no formato de um protótipo.

1.4 Organização

Para melhor exposição e aprofundamento deste trabalho, esta monografia é dividida em três partes principais:

Em sua primeira parte é feita uma exposição mais teórica, apresentando-se os jogos similares ao aqui proposto; descrevendo de maneira completa o jogo original e comentando-se o seu possível caráter como ferramenta de aprendizado.

Na segunda parte, toda a parte computacional é abordada, desde a construção dos esboços e definição do protótipo, até a sua implementação e hospedagem *online*.

Por fim, são mostrados os testes realizados, assim como os seus resultados, além de uma reflexão pessoal de como foi a realização deste trabalho.

2 JOGOS SEMELHANTES EXISTENTES

Apresentam-se aqui quais são os principais jogos existentes similares ao jogo WordStorming proposto nesta monografia. Para melhor compreensão, essas similaridades são evidenciadas através de comparações, com o apontamento das semelhanças ou diferenças observadas. Essa exposição inicial tem como objetivo, tanto a corroboração da originalidade da ideia deste trabalho, quanto servir de uma breve introdução ao jogo WordStorming, pelo entendimento da sua posição no universo dos jogos similares.

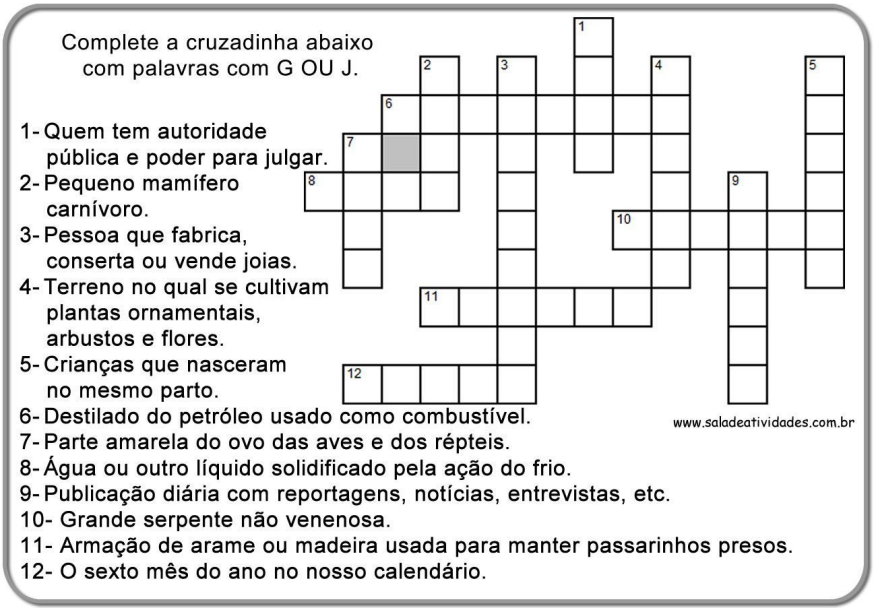
Em geral, os jogos apresentados aqui são tidos como similares, pois, compartilham da mesma base do jogo desta monografia: compõem um jogo de palavras; possuem letras aleatórias; e envolvem a construção de palavras. E WordStorming se diferencia deles, possuindo assim certa originalidade, sobretudo pela sua pretensão educativa que molda os seus aspectos funcionais.

2.1 Palavras Cruzadas

Conhecido também como “cruzadinhas”, esse é um jogo composto de vários quadrados brancos adjacentes, cada um deles representando uma letra a ser descoberta, e portanto, que precisa ser preenchido. Esses quadrados adjacentes são dispostos de modo a formarem linhas, tanto horizontais quanto verticais, representando assim as palavras do jogo. O fato dessas linhas se cruzarem, ou seja, compartilharem um quadrado, explica o nome do jogo ser “palavras cruzadas”. Apesar das palavras por si só já ajudarem no preenchimento dos quadrados (por conta das letras em comuns), também existem as dicas para cada palavra, que sempre acompanham as “cruzadinhas”.

Esse é um dos jogos menos parecido com o jogo proposto nesta monografia, mas ainda assim ele foi trazido aqui por ser um dos jogos de palavras mais famosos e que já exercita a construção, ou mais especificamente, a descoberta de palavras.

Figura 1 - Cruzadinhas



Retirada do site http://www.saladeatividades.com.br/cruzadinhas/

2.2 Caça-palavras

Conhecido também como “sopa de letras”, esse é um jogo composto de uma grade de letras aleatórias, à primeira vista, como é possível ver na figura 2 abaixo. No entanto, isso compõe exatamente o desafio do jogo, que é encontrar e circular nesse amontoado de letras aleatórias, palavras que foram intencionalmente “escondidas” horizontal, vertical ou diagonalmente nessa “sopa de letras”. Muitas vezes, as palavras possuem um tema em comum que fica explicitado no jogo. E às vezes, essa lista de palavras escondidas é explicitada também, facilitando assim o trabalho do jogador de encontrá-las.

Juntamente com as “cruzadinhas”, esse jogo é um dos mais distantes, em termos de semelhança, do jogo aqui proposto. No entanto, ele traz essa ideia de “construção” de palavras a partir de letras aleatórias, que consegue conectá-lo com o WordStorming. Além disso, ele também é um dos jogos de palavras mais conhecidos pelas pessoas.

Figura 2 - Caça-palavras (parcialmente resolvido)

Ara • ~~Aries~~ • Auriga • Boötes • Caelum • Camelopardalis • Cancer • Canes Venatici • Canis Major • Canis Minor • Capricornus • Carina • Cassiopeia • Centaurus • Cepheus • Cetus • Chamaeleon • Circinus • Columba • Coma Berenices • Corona Australis • Corona Borealis • Corvus • Crater • Crux • Cygnus • Delphinus • Dorado • Draco • Equuleus • Eridanus • Fornax • Gemini • Grus • Hercules • Horologium • Hydra • Hydrus • Indus • Lacerta • Leo • Leo Minor • Lepus • Libra • Lupus • Lynx • Lyra • Mensa • Microscopium • Monoceros • Musca • Norma • Octans • Ophiuchus • Orion • ~~Pavo~~ • Pegasus • Perseus • Phoenix • Pictor • Pisces • Piscis Austrinus • Puppis • Pyxis • Reticulum • Sagitta • Sagittarius • Scorpius • Sculptor • Scutum • Serpens • Sextans • Taurus • Telescopium • Triangulum • ~~Triangulum Australe~~ • Tucana • Ursa Major • Ursa Minor • Vela • Virgo • Volans • Vulpecula

Retirada do site https://pt.wikipedia.org/wiki/Ca%C3%A7a-palavras

2.3 Boggle

Conhecido também como “parole”, na versão brasileira, esse é um jogo que já dispõem de elementos físicos mais elaborados para que possa ser jogado, que são os 16 dados e uma ampulheta. Além disso, requer no mínimo 2 jogadores para jogá-lo. Cada um dos dados possui letras ao invés de números e eles devem ser sorteados e organizados de tal maneira a formar uma grade 4 por 4, como mostrado na figura 3 abaixo. Feita essa organização, o objetivo dos jogadores é formar o máximo de palavras utilizando as letras

localizadas no topo dos dados, antes do tempo da ampulheta acabar. As palavras formadas não podem repetir letras e elas devem ser formadas apenas com letras adjacentes na grade, seja horizontal, vertical ou diagonalmente. Como é de se esperar, ganha o jogo o jogador que fizer mais palavras.

Com algumas poucas diferenças, esse jogo já compartilha muito da base do jogo proposto aqui nesta monografia.

Figura 3 - Boggle

Retirada do site https://en.wikipedia.org/wiki/Boggle

2.4 Wordstorm

Esse jogo é um que se localiza nativamente no mundo virtual, em forma de aplicativo de celular. No jogo o usuário é apresentado a 9 letras aleatórias - postas em um octógono dividido em 9 partes - e precisa descobrir as 35 palavras mais comuns previamente listadas pelo jogo. As palavras precisam ter no mínimo 4 letras e quanto mais difícil ela for, mais pontos o jogador ganha. O jogo é relativamente parecido com o jogo “boggle”, portanto, também se assemelha ao jogo proposto nesta monografia.

A título de curiosidade, WordStorming era intitulado de WordStorm inicialmente, mas foi renomeado assim que a existência desse aplicativo de jogo foi constatada.

Figura 4 - WordStorm

Retirada do site https://play.google.com/store/apps/details?id=com.windowsgames.wordstorm

2.5 Scrabble

Esse é um jogo de tabuleiro quadriculado que utiliza peças representando letras aleatórias. Cada jogador inicia com um conjunto aleatório dessas peças. Em toda rodada, cada jogador precisa escolher uma de suas letras aleatórias e colocar no tabuleiro a fim de conseguir completar uma palavra. É possível ir formando uma palavra no tabuleiro usando as letras aleatórias colocadas pelo oponente também. O jogador que colocar a última letra de uma palavra, ganha os pontos associados a ela. Essa pontuação é calculada pela soma dos pontos de cada letra. A depender de onde a palavra foi formada no tabuleiro, é possível receber bônus por isso. Simplificadamente, esse jogo pode ser entendido como o jogo de "palavras cruzadas" sendo jogado entre 2 pessoas (ou mais) de maneira bastante interativa.

Scrabble tem uma considerável relevância no cenário mundial de jogos de palavras, uma vez que está presente em diversos países e, consequentemente, disponível em diversos idiomas. As palavras dos jogos são validadas através de um dicionário oficial do jogo, que é facilmente encontrado na *web* nos dias atuais.

A característica do Scrabble de cada letra possuir uma pontuação própria foi algo que inspirou a melhoria da pontuação no jogo desta monografia. Em termos conceituais, Scrabble provavelmente é o jogo mais similar ao WordStorming, pois, ambos possuem letras aleatórias, construção de palavras, pontuação por letras, pontuações bônus, caráter *multiplayer* e interatividade.

Figura 5 - Scrabble

Retirada do site https://en.wikipedia.org/wiki/Scrabble

3 WORDSTORMING

O propósito principal desta monografia é o desenvolvimento computacional de um jogo original multiplayer de palavras nomeado de WordStorming. Portanto, uma descrição completa do jogo e da sua origem é importante de ser realizada aqui, antes de se partir para o processo de construção do jogo, propriamente dito.

A ideia inicial do jogo nasceu em uma sala de aula de inglês, onde a professora despretensiosamente sugere uma brincadeira a ser feita. Basicamente, grupos de alunos deviam tentar formar palavras no papel com as letras aleatórias que ela havia sorteado. A partir dessa simples premissa, já fora da sala de aula, um jogo com regras e pontuações cada vez mais elaboradas tomava forma. Com a complexidade final que o jogo adquiriu, a sua migração para o meio computacional foi algo inevitável.

Já desde a sua concepção, o jogo possui um viés linguístico e educacional, devido à sua pretensão em auxiliar na retenção vocabular em um idioma. No entanto, apesar disso, a sua intenção primária é a de ser um jogo, ou seja, de entreter a quem joga. Apenas em segunda instância, existe esse objetivo de contribuir, através do divertimento, com a retenção de palavras do jogador. Portanto, esse interesse idiomático, ao menos, a princípio, é desnecessário para se jogar o jogo.

3.1 Regras

O conceito central do WordStorming é a construção de palavras a partir de letras aleatórias, por isso o nome do jogo (tempestade de palavras, em tradução livre). Durante o jogo ocorre uma série de rodadas que compõem uma partida. Em cada rodada, os jogadores compartilham um mesmo conjunto de letras aleatórias e devem tentar pontuar o máximo possível através de formação de palavras. O vencedor da partida é o primeiro que conseguir pontuar 50 pontos à frente dos outros jogadores.

Para um melhor entendimento dos fundamentos do jogo, realiza-se nos próximos parágrafos uma exposição detalhada das regras, organizada em três partes: elementos, pontuação e objetivos.

3.1.1 Elementos

O jogo é composto de cinco elementos imprescindíveis: os jogadores, a partida, as rodadas, as sequências de letras e as palavras.

3.1.1.1 Jogadores

Jogadores são quem formam palavras, acumulam pontos e vencem (ou perdem) uma partida.

Evidentemente, pelo caráter *multiplayer*, no mínimo, 2 jogadores são exigidos para que ocorra uma partida no WordStorming. Aconselha-se um máximo de 4 jogadores, mas esse limite é ditado, sobretudo, pelas limitações técnicas e da possibilidade de entretenimento. Recomenda-se também que os jogadores possuam, pelo menos, a idade de 8 anos. Visto que, a essa altura, a alfabetização na língua materna já estaria sendo efetuada.

3.1.1.2 Partida

Uma partida se define por uma sequência de rodadas - separadas por intervalos de 30 segundos no máximo - compartilhada entre mesmos jogadores. E, apesar de raro, uma partida pode ser composta de apenas uma rodada.

O resultado do jogo acontece neste nível. Ou seja, vence-se uma partida no jogo, não uma rodada.

3.1.1.3 Rodada

Em uma rodada é quando o jogo acontece: cada jogador forma o máximo de palavras que conseguir com as letras da rodada. Uma rodada tem uma duração máxima de 2 minutos e apresenta uma mesma sequência de letras aleatórias para os jogadores da partida.

Os jogadores não veem as palavras formadas pelos seus oponentes até o término da rodada. Ou seja, esse momento é reservado para o intervalo entre as rodadas.

3.1.1.4 Sequência de Letras

Toda rodada possui uma sequência de letras aleatórias compartilhada entre todos os jogadores da partida. O tamanho de uma sequência varia de 3 a 26 letras e as letras podem se repetir. Além disso, uma sequência sempre contém vogais.

3.1.1.5 Palavra

As palavras formadas no WordStorming devem conter apenas as letras aleatórias (incluindo as repetidas) de cada rodada. Ou seja, o tamanho máximo, mas não necessário, de uma palavra em cada rodada é o número de letras na sequência apresentada.

Os vocábulos construídos devem ser no idioma do jogo, a priori, em inglês. E, naturalmente, eles necessitam também ser validados como verbetes em algum dicionário na língua em questão para serem aceitos no jogo.

3.1.2 Pontuação

Em cada rodada, os jogadores - individualmente - possuem a chance de acumular pontos através da formação de palavras. Esse é um aspecto muito importante do jogo, pois é o que permite definir o vencedor de uma partida. Em razão dessa importância e da maior complexidade do assunto, uma seção é dedicada a esse tópico.

Essencialmente, o sistema de pontuação no WordStorming se fundamenta nos seguintes princípios:

1. Letras possuem pontos próprios.
2. A pontuação atribuída a uma palavra é a soma dos pontos de suas letras.
3. Um jogador forma uma palavra nova em uma dada rodada, se ela nunca foi usada nas rodadas anteriores da partida em questão.
4. Um jogador forma uma palavra diferente, se ela é uma palavra nova e que foi usada apenas por ele na rodada em questão.
5. O conceito de palavra diferente só faz sentido no modo estático do jogo (onde não é possível ver as palavras dos outros jogadores durante a rodada).

Dessa maneira, para cada palavra que um jogador constrói, ele recebe a pontuação atribuída a essa palavra. E ao fim de uma rodada, ele pode ainda receber as pontuações bônus referentes à construção de eventuais palavras novas e palavras diferentes na rodada.

3.1.2.1 Pontos

A determinação de pontos para cada letra no WordStorming é baseada na frequência da sua ocorrência no idioma que esteja se jogando o jogo. No caso do inglês, temos a seguinte tabela:

Tabela 1 - Pontos das Letras

Letras	**Pontos (pts)**
A, E, I, O, U, R, N, S	1
T, H, D, L, C, M	2
F, Y, B, G, P	3
W, V, K, X	5
Q, J, Z	7

Cada palavra nova rende a pontuação de 1 pts para o jogador. E cada palavra diferente, a pontuação de 2 pts.

3.1.2.2 Bônus

Determinados tipos de palavras podem render pontuações extras ao jogador no fim de uma rodada. São elas:

1. Palavras que usem todas as vogais (sem as repetições) da sequência de letras da rodada.
2. Palavras que usem todas as consoantes (sem as repetições) da sequência de letras da rodada.
3. Anagramas.
4. Palíndromos.
5. Palavras com encontros consonantais.
6. Palavras com encontros vocálicos.
7. Palavras que usem todas as letras (sem as repetições) da sequência da rodada.

Essas pontuações são inclusivas, ou seja, elas se somam e podem ocorrer simultaneamente com a mesma palavra. Por esse motivo, cada uma delas rendem 1 pts ao jogador.

3.1.3 Objetivos

O objetivo principal do jogo é acumular o máximo de pontos durante uma partida através da formação de palavras, de forma a superar a pontuação dos oponentes. Vence a partida o primeiro jogador a atingir a diferença de 50 pontos na sua pontuação em relação a

dos outros jogadores ao fim de uma rodada. E se o vencedor não é determinado em até 16 minutos, ganha quem tiver mais pontos ou a partida acaba em empate.

Em vista desse objetivo principal, é possível traçar dois objetivos menores do jogo.

O mais básico deles seria a formação de palavras. Para isso, é essencial que o jogador respeite a lista de letras da rodada. Ou seja, as palavras formadas por ele devem conter apenas as letras da lista, mas não necessariamente todas elas. A palavra formada deve usar cada letra da sequência no máximo na quantidade que foi apresentada. E, como já dito, a palavra formada precisa aparecer no dicionário.

O segundo objetivo menor deriva diretamente do primeiro, pois ele consiste em acumular o máximo de pontos possíveis durante as rodadas, o que é feito através da formação de palavras. Isso é importante, porque, cada rodada é potencialmente a última da partida, já que ela sempre pode definir o vencedor do jogo.

3.2 Funcionalidades

As regras expostas até aqui compõem, basicamente, a essência do jogo. Isso quer dizer que, sem muitos empecilhos, somente com o que foi apresentado, ele pode ser jogado manualmente, através de lápis e papel, por exemplo. No entanto, a fim de permitir uma dinâmica muito maior ao jogo, diversas outras funcionalidades também foram elaboradas. E pela complexidade inerente delas, elas foram pensadas naturalmente sendo computacionais, já que seriam difíceis de serem viabilizadas de uma outra forma.

As funcionalidades mais relevantes são descritas aqui, fechando assim a apresentação completa do WordStorming.

3.2.1 Configurações

Por padrão, o jogo inicia-se no modo estático, configurado no idioma inglês e com rodadas e intervalos de tempo máximo de 2 minutos e de 30 segundos, respectivamente. Contudo, tudo isso pode ser alterado através das configurações.

Os jogadores podem mudar o tempo máximo das rodadas e dos intervalos; podem mudar o idioma do jogo para algum outro idioma latino, incluindo o português; e podem substituir o modo estático do jogo pelo modo dinâmico, o que os permitem ver as palavras formadas pelo seus oponentes durante a rodada. Nesse modo, a pontuação por palavra diferente não ocorre, pois os jogadores são impedidos de usar as palavras já sendo usadas pelos seus oponentes.

3.2.2 Dicionário

A qualquer momento da partida, os jogadores podem verificar o significado (no idioma configurado do jogo) de qualquer palavra utilizada na rodada corrente, sejam as próprias ou as formadas pelos oponentes.

3.2.3 Histórico

O jogo guarda todas as palavras (sem repetições) usadas por cada jogador durante todas as suas partidas. Dessa maneira, em momentos pontuais determinado pelo jogo, o jogador pode rever as palavras que já utilizou e principalmente, ter benefícios das seguintes funcionalidades: sugestão, relatório, experiência, conquistas e habilidades.

3.2.3.1 Sugestão

Durante o intervalo, os jogadores podem receber uma sugestão de palavra que se encaixa na sequência de letras aleatórias da rodada em questão, mas que não foi usada durante a partida, até então. Essa sugestão é baseada no histórico de cada jogador.

3.2.3.2 Relatório

Ao fim da partida, com base no histórico, o jogo informa a proporção de palavras novas e palavras repetidas usadas pelo jogador durante a partida, além do tempo total e o número de rodadas na partida.

3.2.3.3 Experiência

Cada jogador ganha experiência em relação ao seu desempenho na partida. Por exemplo, expandir o seu histórico de palavras durante uma partida é um bom desempenho. O acúmulo de experiências coloca cada jogador em um determinado nível, refletindo assim a sua senioridade no WordStorming.

Alguns outros exemplos de bom desempenho na partida:

- número de partidas ganhas;
- uso de palavras com pontuações bônus; e
- não repetir palavras em uma rodada, com exceção da primeira.

3.2.3.4 Conquistas

Os jogadores são recompensados através de medalhas virtuais, por cada conquista realizada no WordStorming. Como, por exemplo:

- histórico com mais de 100 palavras;
- 3 partidas seguidas sem usar palavras repetidas;
- 20 partidas ganhas;
- uso de 50 palavras com letras raras;
- uso de 50 palavras em uma determinada classe gramatical;
- uso de pelo menos uma palavra em todas classes gramaticais; e
- atingir o nível 10 do jogo.

3.2.3.5 Habilidades

A cada novo nível no WordStorming, os jogadores possuem a chance de ganhar uma nova habilidade. Uma habilidade é o poder dos jogadores de causar alguma ação no jogo em benefício próprio durante uma rodada.

O jogador vai acumulando habilidades conforme o passar dos níveis, mas pode apenas utilizar 3 delas durante uma partida e apenas 1 por rodada. Segue alguns exemplos delas:

- Deixar os oponentes sem jogar por 5 segundos;
- Sortear mais uma letra aleatória na rodada (essa letra extra não rende ponto);
- Poder repetir uma das letras aleatórias na formação das novas palavras; e
- Receber uma sugestão de palavra aleatória (pode já ter sido utilizada na rodada).

3.2.4 Singleplayer

Os jogadores conseguem jogar contra o computador, ao invés de uma pessoa. Nessa modalidade, o ganho de experiência é adaptado, não é possível realizar as conquistas e não é permitido o uso das habilidades.

4 BENEFÍCIOS EDUCACIONAIS

Como mencionado anteriormente, o aspecto linguístico e educacional é um ponto relevante do jogo, tanto que na sua concepção ele já possuía esse viés. No entanto, o desenvolvimento significativo e específico desse ponto demanda tempo e atividades extras envolvendo o auxílio de profissionais adequados, tais como linguistas, pedagogos e afins. Por conta dessa razão e por fugir do escopo deste trabalho, esse aspecto se resguarda apenas como um objetivo secundário do WordStorming.

Esse aspecto se manifesta no jogo através de 3 vias: o sistema de pontuação, as funcionalidades nativamente computacionais e a sua semelhança com alguns jogos já apresentados.

4.1 Vocabulário

O jogador acumula pontos através das palavras de acordo com o seu desempenho dentro da rodada. Isso é feito pela pontuação da palavra derivar da pontuação de cada letra e pelo fato das letras mais raras valerem mais pontos. Essa lógica estimula que o jogador use o máximo de palavras possíveis, pois todas elas rendem pontos; que as palavras sejam longas, pois elas usam mais letras, e portanto, em média, rendem mais pontos; e que as palavras sejam menos comuns, pois contêm letras que valem mais pontos.

Além disso, as pontuações bônus também colaboram para o uso de palavras que o jogador ainda não utilizou na partida, que os outros jogadores não utilizaram e que possuam formações complexas, como encontro consonantais ou vocálicos, por exemplo.

4.2 Aprendizado Contínuo

Os jogadores ganham experiência de acordo com o seu desempenho dentro das partidas. A diferença entre pontos e experiência é que o primeiro beneficia os melhores jogos e o segundo, os melhores jogadores. As duas funcionalidades são importantes, pois, enquanto a pontuação estimula o jogador a construir bons jogos no curto prazo, a experiência o estimula a se aprimorar como um bom jogador no longo prazo. Como forma de reconhecimento do seu esforço, o acúmulo de experiências permite ao jogador atingir níveis cada vez maiores, o que lhe garante medalhas e habilidades para usar durante as rodadas.

Outras funcionalidades menores como a sugestão de palavras e relatório da partida também contribuem para o progresso do jogador.

4.2 Desenvolvimento Cognitivo

Como demonstrado, WordStorming apresenta semelhanças de fundamentos com os jogos cruzadinhas e caça-palavras. Devido a essa similaridade, não seria descomedido dizer que WordStorming também compartilharia dos mesmos benefícios que esses jogos oferecem.

Os mais óbvios deles seriam o enriquecimento vocabular e o treino ortográfico. E devido à característica do WordStorming de estimular a busca de palavras em uma lista de letras aleatórias, o benefício de desenvolvimento do raciocínio lógico e da memória.

Em vista disso e de maneira bastante pretensiosa, pode-se dizer também que o jogo poderia servir de opção digital e educacional para crianças, por permitir tanto o entretenimento quando o desenvolvimento intelectual delas**.**

5 CONHECIMENTOS NECESSÁRIOS

Em sua versão inicial, WordStorming pode ser jogado manualmente, ainda que improvisadamente. No entanto, a evolução natural das suas funcionalidades acabou lhe concedendo uma propícia natureza computacional, tornando assim, o conhecimento nessa área fundamental para o desenvolvimento apropriado do jogo.

De maneira geral, WordStorming poderia ser concebido em diversas plataformas digitais, desde que essa plataforma atendesse às necessidades do jogo. Ele poderia ser idealizado, por exemplo, como um aplicativo *mobile*, um jogo eletrônico para consoles ou uma aplicação *desktop*. Todavia, por uma série de razões, escolheu-se apresentá-lo em plataforma *web*. As razões para isso são objetivas e subjetivas.

Objetivamente, escolheu-se o mundo *online*, pois ele permite o caráter *multiplayer* do jogo a baixo custo. Além disso, o mundo *online* facilita o jogo ser expressivamente acessível às pessoas, já que ele é bastante comum nos dias de hoje e é contemplado por diversos dispositivos (celulares, tablets, computadores etc), quase sempre sem requerer uso extra da memória desses aparelhos.

Subjetivamente, escolheu-se esse mundo pela oportunidade abrangente de aprendizado em uma relevante área computacional na contemporaneidade. E essa também seria a plataforma mais viável de se construir o jogo, levando em conta as condições de tempo e recursos para o desenvolvimento deste trabalho.

Nos parágrafos que se seguem, descrevem-se detalhadamente os conhecimentos técnicos necessários, prévios e adquiridos, para o desenvolvimento deste jogo. Comenta-se ainda as disciplinas mais relevantes com influências diretas na produção deste trabalho, além de alguns conceitos importantes.

5.1 Disciplinas Relevantes

Houve algumas disciplinas cursadas ao longo da graduação que merecem ser destacadas aqui, devido aos seus impactos significativos neste trabalho. Seja isso pela contribuição à formação de um adequado pensamento computacional, o entendimento de alguns conceitos ou aquisição de conhecimentos técnicos específicos. A menção a cada uma das disciplinas é feita em ordem de aparição ao longo dos anos do curso.

5.1.1 Introdução a computação (MAC0110)

Uma das primeiras disciplinas cursadas durante o primeiro semestre da graduação. Teve a sua importância ao proporcionar um primeiro contato apropriado com programação, principalmente do ponto de vista lógico. Mas fora a introdução ao pensamento computacional, a familiarização com algumas linguagens de programação e uma certa segurança em lidar com elas também foram competências importantes construídas aqui.

Por fim, o primeiro contato e o aprendizado da linguagem de programação Python - um dos principais conhecimentos técnicos para a elaboração deste trabalho - aconteceu com esta disciplina.

5.1.2 Algoritmos e estruturas de dados I (MAC0121)

Essa matéria foi importante para explicitar a importância de um código elegante e eficiente. Um código não elegante, ou seja, de difícil entendimento e com complicações desnecessárias, afeta a sua manutenção e desempenho. E um código não eficiente gasta tempo e memória descabidamente, sem análises devidas de complexidade do algoritmo.

5.1.3 Técnica de programação I (MAC0216)

De uma certa forma, essa matéria cumpriu o papel de uma disciplina de engenharia de *software*. Aqui trabalhou-se o planejamento de um projeto de *software* e a sua organização em módulos, por exemplo. Ainda nesse sentido, o primeiro contato com um outro paradigma de programação, programação orientada a objetos, que possui esse viés de organização do código, foi realizado por meio desta disciplina.

5.1.4 Sistemas operacionais (MAC0422)

Dois conceitos importantes foram apresentados e devidamente explorados com esta disciplina: programação concorrente e modelo cliente-servidor. Ambos conceitos são relevantes para um bom desenvolvimento *web*, principalmente o segundo.

Uma outra contribuição relevante desta disciplina foi o trabalho de desenvolver alguns programas mais complexos.

5.1.5 Técnica de programação II (MAC0218)

Essa é uma matéria, que no oferecimento em questão, tratou-se basicamente sobre desenvolvimento *web*. Então, nesse sentido, ela teve sua importância clara para a composição deste trabalho. Mas, destaca-se aqui, a exposição aos conceitos de *model-view-controller*, uma arquitetura comum para a criação de *sites*, e de *frameworks*, ferramentas importantes para o desenvolvimento ágil de *softwares*.

A *framework web* Flask, baseado em Python e utilizada neste trabalho, foi conhecida através desta matéria.

5.1.6 Introdução ao desenvolvimento de software (MAC0350)

Essa disciplina foi importante, pois permitiu o contato com o desenvolvimento de software utilizando a *framework web* Django, que é em Python e possui suas semelhanças com a *framework web* Flask. Através desta matéria, também pode-se melhor explorar e entender alguns conceitos de desenvolvimento *web* (formulários e sessões).

5.1.7 Princípios de interação humano-computador (MAC0446)

Fortuitamente, esta matéria também permitiu o contato com o desenvolvimento *web* utilizando o Django e teve a sua contribuição diferenciada por ajudar a compreender como realizar a arquitetura *model-view-controller* deste trabalho.

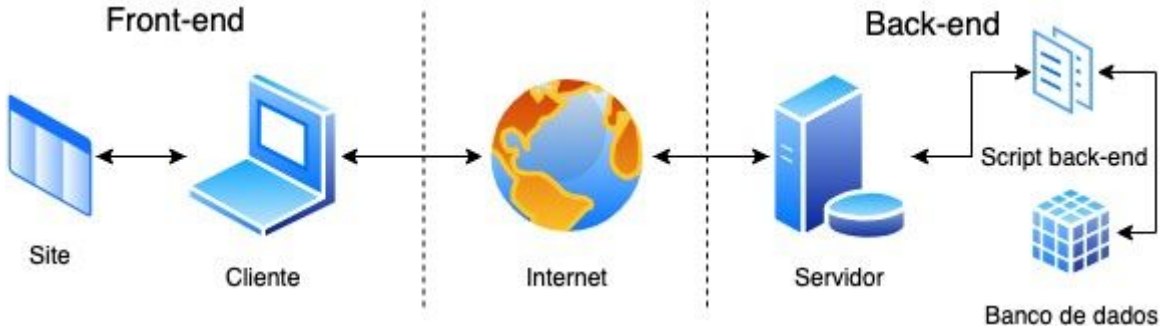
5.2 Conceitos de Computação

Em geral, houve alguns conceitos que foram importantes para o entendimento de aspectos computacionais do jogo ou para a aquisição de algumas ferramentas para auxiliar na sua construção. As principais são citadas aqui.

5.2.1 Desenvolvimento web

Ao passar dos anos, o desenvolvimento de uma aplicação *web* foi ficando cada vez mais complexo, de forma que atualmente já é natural dividi-lo em algumas camadas. Em especial, todo desenvolvimento *web* pode ser dividido em duas partes principais: front-end e back-end, as quais correspondem, essencialmente, ao cliente e ao servidor.

Figura 6 - Front-end e back-end



Retirada do site
https://www.homehost.com.br/blog/wp-content/uploads/2019/09/1557080784763701500.jpg

5.2.1.1 Front-End

O desenvolvimento nessa camada se dá basicamente na esfera do navegador *web*, pois é ele quem faz o papel de cliente em um computador. Portanto, o Front-End de uma aplicação *web* se dá nos dados apresentados visualmente e passíveis de interação pelo usuário em um *site* qualquer.

As principais linguagens usadas nesse desenvolvimento são: JavaScript, HTML e CSS. Todas elas foram usadas neste projeto.

Figura 7 - Principais linguagens usadas no desenvolvimento Front-End de um *site*

Retirada do site https://apexensino.com.br/wp-content/uploads/2017/11/html-css-javascript.jpg

5.2.2.2 Back-End

O back-end de uma aplicação *web* é representado por todo código que roda na parte do servidor, ou seja, a máquina que está sempre recebendo requisições e enviando respostas ao cliente (o navegador *web*). Nesse sentido, a sua tarefa mais básica acaba sendo de entregar as páginas solicitadas pelo usuário. E muitas vezes, devidamente processadas com base nas interações que ele realize. Como o servidor é uma parte onde o cliente dificilmente interage, convenientemente, é onde é feito também a locação da base de dados, a validação de formulários enviados pelo usuário, assim como diversas outras tarefas que requeiram maior segurança.

A priori, qualquer linguagem de programação pode ser utilizada no back-end de um *site*. Neste projeto, especificamente, foi utilizada a linguagem de programação Python.

5.2.2 Framework

Esse é provavelmente um dos conceitos mais complexos apresentados aqui e ao mesmo tempo um dos mais importantes para a construção do jogo nesta monografia.

Resumidamente, *frameworks* podem ser entendidas como provedoras de estruturas ideais para o desenvolvimento de alguns tipos de aplicações (um *site* ou aplicativo *mobile*, por exemplo). Ou seja, elas servem como base sólida no desenvolvimento de alguma aplicação, e portanto, conduzem e ditam a sua estrutura e o seu fluxo em algum grau.

Uma característica essencial nessas ferramentas é a reutilização de código. Para formar a estrutura ideal mencionada, uma *framework* precisa reunir vários códigos comuns na construção de um determinado tipo de aplicação, a fim de criar um ambiente computacional favorável para realizar essa construção. Esses códigos podem ser desde bibliotecas, integrações com APIs, a conexões com base de dados ou determinados códigos prontos que são razoavelmente frequentes no tipo de aplicação especificado.

Assim sendo, com o uso de uma *framework*, o programador não precisa mais se preocupar com tarefas recorrentes e repetitivas que sempre aparecem no desenvolvimento de certas aplicações, o que acelera então o seu progresso e permite que se dedique ao que realmente interessa.

5.3 Tecnologias Utilizadas

Diversas tecnologias foram utilizadas para o desenvolvimento deste trabalho, e em especial, ferramentas voltadas para o front-end e back-end de um desenvolvimento *web*. As primeiras seções a seguir se referem às ferramentas utilizadas no front-end: HTML, CSS, JavaScript, jQuery, JSON, Ajax e Bootstrap. E as duas seções subsequentes a elas se referem às ferramentas utilizadas no back-end: Python e Flask.

Além dessas ferramentas, conhecimentos técnicos não necessariamente computacionais ou diretamente relacionados com o desenvolvimento *web* também foram importantes neste trabalho e são comentados nas últimas seções deste capítulo:

- Trello, utilizado para o gerenciamento deste projeto;
- Visual Studio Code e Git, para a escrita e controle de versão dos códigos-fontes, respectivamente; e
- Ngrok e Heroku para simulação e teste de hospedagem do WordStorming, respectivamente;

5.3.1 HTML

Sem a linguagem HTML, acrônimo para *HyperText Markup Language*, os *sites* na Web praticamente não existem. Pois, ela cumpre duas funções importantes: a criação dos hipertextos (e atualmente, hipermídias), assim como a marcação deles.

Como a Web é formada essencialmente de hipertextos, é possível entender porquê essa linguagem é importante para existência dos *sites*. Basta entender que cada página na Web é um hipertexto gerado por documento HTML. Mas, além de dar existência a essas páginas, a linguagem HTML também organiza a estrutura das informações (textos, imagens, vídeos e sons) que aparecem nelas. Isso é feito, especificamente, através das marcações, que são as chamadas *tags* dentro da linguagem.

Um documento HTML é dividido em várias *tags* que contêm todas as informações da página. E essas *tags* ditam não apenas a ordem das informações, mas também o significado delas na página. Ou seja, se é o título do *site*, se é o conteúdo principal, se é uma imagem e assim por diante.

Figura 8 - Exemplo de código HTML

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="description" content="a descrição do seu site em no máximo 90 caracteres">
        <meta name="keywords" content="escreva palavras-chaves curtas, máximo 150 caracteres">
        <title>Título do Documento</title>
    </head>
    <body>
    <!-- Aqui fica a página que será visível para todos, onde pode-se inserir
    textos, imagens, links para outras páginas, etc, geralmente usa-se: -->

    <div>Tag para criar-se uma 'caixa', um bloco, mais utilizada com "Cascading Style Sheets
         (Folhas de Estilo em Cascata)</div>

    <span>Tag para modificação de uma parte do texto da página</span>

    <img src="endereço_de_uma_imagem.jpg" />

    <a href="http://www.wikipedia.org">Wikipedia, A Enciclopédia Livre</a>
    </body>
</html>
```

Retirada do site https://pt.wikipedia.org/wiki/HTML

5.3.2 CSS

Toda página HTML é praticamente igual. As informações são organizadas em uma ordem linear dentro da página. A linguagem CSS, acrônimo para *Cascading Style Sheets*, é utilizada para conceder um estilo próprio para cada *site*. Ou seja, ela serve para colorir as diferentes partes de uma página HTML; posicionar os elementos em partes específicas da página; mudar o tamanho da fonte dos textos, entre outras questões de estilo.

A especificação de algum estilo qualquer no documento HTML é sempre feita em relação a uma de suas *tags*. Isso pode ser feito diretamente na linha da *tag*, em uma *tag* HTML reservada para o código CSS ou em um arquivo separado, o qual o documento HTML importa.

Por fim, uma característica interessante da linguagem CSS é o seu caráter hereditário, sinalizado pelo "*Cascading*" em seu nome. Isso quer dizer que um estilo conferido a uma *tag* no HTML é sempre reverberado para todas as suas *tags* internas, a menos que seja explicitado algo diferente.

Figura 9 - Exemplo de código CSS em uma tag específica de um documento HTML

```html
<h1 style="color: red;">Chapter 1.</h1>
```

Retirada do site https://en.wikipedia.org/wiki/CSS

5.3.3 JavaScript

E para terminar a tríade das linguagens principais usadas em qualquer desenvolvimento Front-End, temos a linguagem de programação multiparadigma JavaScript, uma das mais populares do mundo. Assim como as linguagens HTML e CSS, ela é executada nativamente em todos os principais navegadores *web* e é a responsável por conferir dinamicidade e interatividade nas páginas.

Hoje em dia, dificilmente um *site* é puramente estático, ou seja, apenas páginas de leituras e apresentação. E para que elas não sejam assim, o JavaScript está presente. Por possuir mecanismos próprios para lidar com o HTML dinamicamente, essa linguagem consegue permitir que o usuário interaja com as páginas *web*. Dessa maneira, possibilitando coisas como o funcionamento de um botão, uma animação na tela ou mesmo eventos específicos sempre que o *mouse* passa em cima de uma certa imagem, por exemplo.

A integração de código JavaScript em um arquivo HTML é feito da mesma maneira que o código CSS: diretamente na linha de uma *tag*, em uma *tag* HTML reservada para o código JavaScript ou em um arquivo separado, o qual o documento HTML importa.

Figura 10 - Exemplo de código JavaScript

```
// Mostra um alerta de Confirmar e Cancelar.
if ( confirm( 'Escolha "Ok" ou "Cancelar" para ver a mensagem correspondente.' ) ) {
 alert( 'Você apertou o botão "OK"' ); // mostra um alerta para resposta "OK"
} else {
 alert( 'Você apertou o botão "Cancelar"' ); // mostra um alerta para resposta "Cancelar"
}
```

Retirada do site https://pt.wikipedia.org/wiki/JavaScript

5.3.4 jQuery

jQuery é uma das bibliotecas JavaScript mais utilizadas no mundo. Ela requer pouquíssima memória do computador e simplifica bastante a sintaxe JavaScript e, portanto, o código, por conseguir deixá-lo mais claro e compacto.

Atividades como a navegação e interação com o HTML, a manipulação de eventos, o desenvolvimento de aplicações Ajax (uma ferramenta que utiliza JavaScript) são todas bastantes facilitadas com essa biblioteca. Além disso, ela também possui importância para a existências de outras grandes ferramentas da *web* que fazem uso dela, como o Bootstrap (*toolkit* para o desenvolvimento front-end de um *site*).

Figura 11 - Comparação entre códigos JavaScript puro e jQuery

Exemplo 1: Um código em Javascript puro, para atribuir o valor "5" em um elemento qualquer.

```
document.getElementById( 'Teste' ).value = 5;
```

O mesmo código em jQuery.

```
$( '#Teste' ).val( 5 )
```

Retirada do site https://pt.wikipedia.org/wiki/JQuery

5.3.5 JSON

JavaScript Object Notation (JSON) é uma estrutura compacta de dados criada para trafegar rapidamente em qualquer protocolo *web*. E apesar de JavaScript no nome, essa estrutura é feita também para ser um formato padrão independente, portanto, comum a diversas linguagens.

Por conta dessas razões, o formato JSON é totalmente textual e segue algumas regras como:

- não pode ter funções;
- não pode ter comentários; e
- todo texto tem aspas duplas.

Figura 12 - Estrutura de dados JSON

```
{
   "id":1,
   "nome":"Alexandre Gama",
   "endereco":"R. Qualquer"
}
```

Retirada do site https://www.devmedia.com.br/o-que-e-json/23166

Por essa simplicidade e flexibilidade, esse formato é utilizado em vários contextos. Seja para retornar dados de certos tipos de API ou para transferir dados do back-end para o front-end, por exemplo.

JSON não é exatamente um objeto front-end (nem back-end), mas foi colocado aqui pela conveniência em seu nome, que contém a principal linguagem de programação do front-end.

5.3.6 Ajax

Ajax é uma técnica que permite a criação de aplicações *web* mais interativas. Ou seja, páginas que carreguem mais rapidamente ou que não precisem recarregar totalmente, sempre que uma informação nova chega do back-end para o front-end. Isso é possível porque o Ajax permite a interação entre o cliente e o servidor de uma maneira assíncrona, que é de onde vem seu nome, acrônimo para *Asynchronous JavaScript and XML*.

Para que a técnica funcione, algumas tecnologias já providas por navegadores *web*, como o próprio JavaScript e o XML, são utilizadas. Atualmente, o JSON é utilizado no lugar do XML, para realizar as transferências de dados entre o back-end e o front-end. Isso porque ele é uma estrutura de dados mais compacta, leve e rápida, como foi mostrado na seção anterior.

Figura 13 - Exemplo de código jQuery para uso do Ajax

```
$.ajax({
  method: "POST",
  url: "some.php",
  data: { name: "John", location: "Boston" }
})
  .done(function( msg ) {
    alert( "Data Saved: " + msg );
  });
```

Retirada do site https://api.jquery.com/jquery.ajax/

5.3.7 Bootstrap

A última tecnologia utilizada no Front-End deste trabalho é o *toolkit* Bootstrap, um projeto *open source* inicialmente criado pela equipe do Twitter (famosa rede social). Basicamente, essa tecnologia consiste em um conjunto de ferramentas, como bibliotecas, reunidas para acelerar o desenvolvimento *web*, utilizando as linguagens HTML, CSS e JavaScript. Além disso, ela também possui um propósito de permitir a construção de *sites* com uma experiência agradável ao usuário e com páginas responsivas, ou seja, que se adaptem às diferentes telas de dispositivos, como celulares, tablets e computadores.

Figura 14 - Página inicial do *site* do Bootstrap

Retirada do site https://getbootstrap.com/

Com o Bootstrap é possível também ter alguns componentes frequentemente utilizados em desenvolvimento *web* à disposição, além de diversos *plugins* construídos com jQuery, por exemplo. Essa é uma das ferramentas front-end mais utilizada do mundo e colabora com muitos desenvolvedores na construção rápida de protótipos *web*.

5.3.8 Python

Figura 15 - Logo do Python

Retirada do site https://www.python.org/

Com a filosofia de ser mais voltada ao ser humano do que ao computador, a linguagem de programação multiparadigma Python possui diversas características que a tornam simples de aprender. Em outras palavras, a leitura e programação de código Python costuma ser mais fácil do que em outras linguagens. Seguem algumas das características do Python que contribuem com essas facilidades:

- Ser de alto nível, ou seja, mais próxima do ser humano. Isso significa uma maior abstração da máquina, o que permite uma sintaxe mais clara da linguagem, por exemplo;
- Tipagem dinâmica, tornando desnecessária a declaração do tipo das variáveis dentro do programa, pois a linguagem consegue deduzi-lo automaticamente; e
- Indentação obrigatória, que naturalmente força uma organização hierárquica do código.

Figura 16 - Soma de 2 números em programa Python

```
# This program adds two numbers

num1 = 1.5
num2 = 6.3

# Add two numbers
sum = num1 + num2

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

Retirada do site https://www.programiz.com/python-programming/examples/add-number

A linguagem Python consegue ser utilizada em diversas áreas da computação como: inteligência artificial, ciência de dados, computação gráfica e até no desenvolvimento *web*. Além disso, ela é uma linguagem que se integra com certa facilidade com outras linguagens como C e C++, por exemplo. Possivelmente tudo isso, aliada à sua simplicidade, contribui para ela ser uma das linguagens mais populares atualmente.

Como dito, essa linguagem pode ser utilizada no desenvolvimento *web*. Mais especificamente no back-end de um site. Ela é considerada uma boa escolha por conta de sua produtividade, embora existam críticas sobre o seu desempenho, quando comparada com outras linguagens. Apesar disso, ainda assim ela foi utilizada em diversos *websites* famosos como: Google, DropBox, Instagram, Spotify e Reddit.

5.3.9 Flask

É natural que o desenvolvimento *web* exija tarefas, algumas vezes complexas, que são muito recorrentes e comuns a qualquer aplicação *web*. Portanto, ter que constantemente refazê-las soa como algo contraproducente. A construção de um *site*, principalmente os que possuem uma camada back-end - praticamente todos atualmente -, pode se tornar desnecessariamente mais complicada por conta disso. Portanto, nesse sentido, surgem as frameworks *web*, tais como Flask e Django, que facilitam e provêm todo o básico que um *site* provavelmente precisa em algum momento.

Figura 17 - Logo do Flask

Retirada do site
https://upload.wikimedia.org/wikipedia/commons/thumb/3/3c/Flask_logo.svg/1920px-Flask_logo.svg.png

Para o desenvolvimento *web* utilizando Python, uma das *frameworks* mais famosas é o Django. Como ele possui essa filosofia de promover um desenvolvimento *web* ágil, muitas configurações já são feitas (apesar de poderem ser alteradas) e um arsenal de ferramentas é disponibilizado, assim como alguns componentes pré-prontos, por exemplo.

No entanto, justamente por conta dessas coisas, o programador não possui muita flexibilidade quanto ao que é pré-definido pela *framework* para o fluxo do desenvolvimento *web*. E isso pode ser um problema para alguns.

Para solucionar esse problema, existem as chamadas *microframeworks* (ou *frameworks* minimalistas), que apesar de não serem tão eficientes em produtividade como o Django, por exemplo, permitem a flexibilidade na construção de uma aplicação *web* por proverem apenas o mínimo necessário para o seu desenvolvimento. O restante fica a cargo do programador. Nesse cenário, a framework *web* em Python mais famosa possivelmente é o Flask.

O Flask é uma *microframework* baseada nas bibliotecas WSGI Werkzeug e Jinja2. Como ela é minimalista, isso significa que ela é formada por apenas um núcleo simples para o desenvolvimento *web*, mas que ao mesmo tempo é extensível. Ou seja, a *framework* consegue possuir várias funcionalidades que o Django já dispõem naturalmente, por exemplo, mas elas precisam ser integradas manualmente na *framework* pelo programador, eventualmente fazendo configurações ou implementações extras.

Figura 18 - Simples página HTML com o texto “Hello, World!” construída em Flask

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Retirada do site https://flask.palletsprojects.com/en/1.1.x/quickstart/

Como mencionado, o Flask se baseia em duas bibliotecas. O WSGI, que lida com as questões de conexão na rede, como receber a requisição do cliente (navegador *web*), assim como lhe retornar uma resposta. Além de também tratar os dados da requisição para que o servidor possa agir sobre eles. E o Jinja2, que basicamente facilita a criação de *templates* (HTML) de uma aplicação *web*. Ou seja, é possível usar parcialmente código Python para facilitar algumas especificações nos *templates*, como a repetição de estruturas, determinação de hierarquia entre os *templates*, além de torná-los reutilizáveis, por exemplo.

Figura 19 - Exemplo de documento HTML utilizando Jinja2

```
<title>{% block title %}{% endblock %}</title>
<ul>
{% for user in users %}
  <li><a href="{{ user.url }}">{{ user.username }}</a></li>
{% endfor %}
</ul>
```

Retirada do site https://jinja.palletsprojects.com/en/2.11.x/

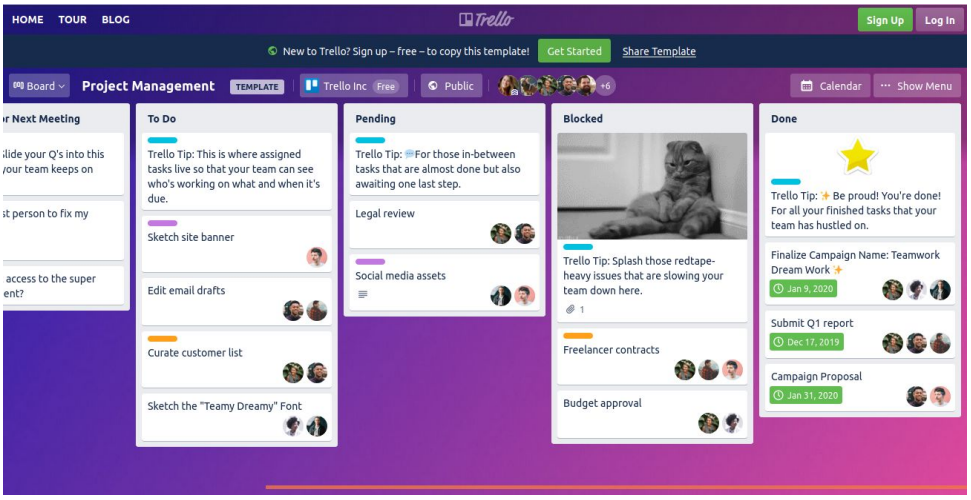
5.3.10 Trello

Trello é uma ferramenta *online* voltada para o gerenciamento colaborativo de projetos. A parte da colaboração não é exatamente um pré-requisito, pois é possível usá-lo individualmente. O funcionamento da ferramenta se baseia basicamente na metodologia de gestão *kanban*. Ou seja, o Trello permite aos seus usuários o controle e acompanhamento das tarefas de um projeto de uma maneira visual e sem muitos recursos.

No Trello, cada projeto é representado por um tabuleiro composto de várias colunas ordenadas na horizontal. As colunas representam as fases evolutivas do projeto. Nessas colunas ficam os quadros, que são as tarefas específicas do projeto. Sempre que uma

tarefa é executada, ela é posicionada em uma coluna mais à direita no tabuleiro, demonstrando a evolução do projeto.

Figura 20 - Exemplo de projeto no Trello

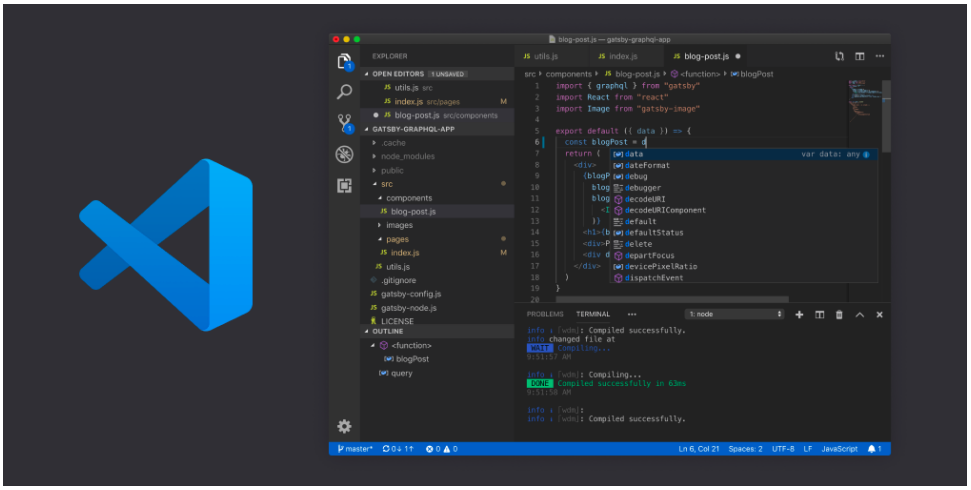


Retirada do site https://trello.com/

5.3.11 Visual studio code

Um dos editores de código-fonte mais populares atualmente, Visual Studio Code (VS Code) é desenvolvido pela Microsoft e pode ser usado nos principais sistemas operacionais do mercado: Linux, Windows e macOS.

Figura 21 - Logo e demonstração da interface do Visual Studio Code



Retirada do site https://code.visualstudio.com/download

É considerada uma boa opção de *Integrated Development Environment* (IDE) para programadores em geral, por sua abrangência de ferramentas, que ajuda tanto na escrita do código, quanto no gerenciamento de um projeto de *software*. Além disso, o editor possui suporte para diversas linguagens de programação e aceita várias extensões disponíveis na *web*, relativamente fáceis de configurar.

E finalmente, o editor é repleto de funcionalidades relevantes para o desenvolvimento de virtualmente qualquer programa mais complexo, como: integração com o git (ferramenta de versionamento de *software*), suporte para *debug*, complementação

inteligente do código e terminal interno (o que permite escrever e rodar o código em um só lugar).

Com tudo isso, provavelmente, a sua desvantagem mais significativa é o desempenho que pode demandar da máquina.

5.3.12 Git

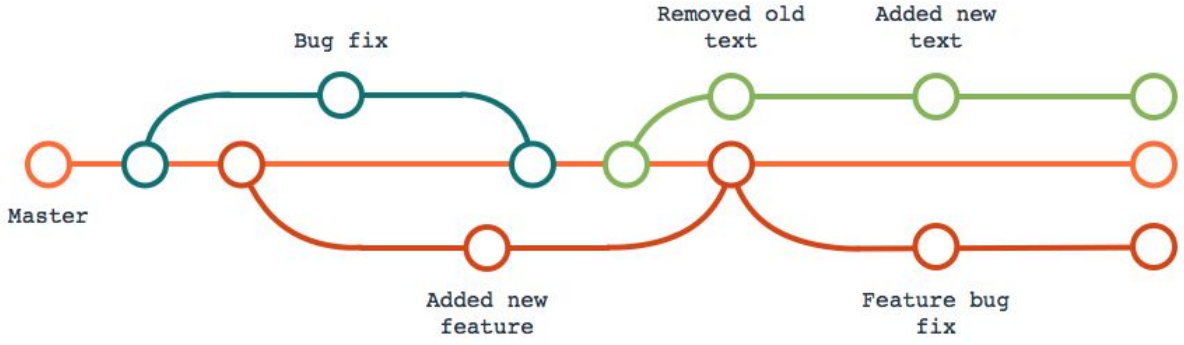
Git é uma ferramenta de controle de versões, principalmente, de *softwares*. Isso porque apesar de poder ser utilizado para gerenciar versões de qualquer tipo de arquivo, o Git é utilizado sobretudo com códigos-fontes. Essa ferramenta foi inicialmente criada para auxiliar no desenvolvimento do *kernel* do Linux. No entanto, o seu sucesso no projeto acabou fazendo ser adotado por muitos outros projetos, de modo que, eventualmente, se tornou central no universo de versionamento de *software*.

Figura 22 - Logo do Git

Retirada do site https://upload.wikimedia.org/wikipedia/commons/e/e0/Git-logo.svg

O Git funciona através de um sistema de diretórios abstraído como um repositório. Esse repositório guarda o estado geral dos diretórios. Dessa forma, se torna possível retornar a uma versão anterior dos arquivos nos diretórios, quando os mesmos estavam diferentes. Ou mesmo criar uma ramificação do repositório, de modo a ter duas versões de um mesmo sistema de diretórios: uma principal (*master*) e uma outra para a realização de mudanças específicas, que posteriormente é fundida novamente com a ramificação principal.

Figura 23 - Exemplo de ramificações e fundições de um repositório Git



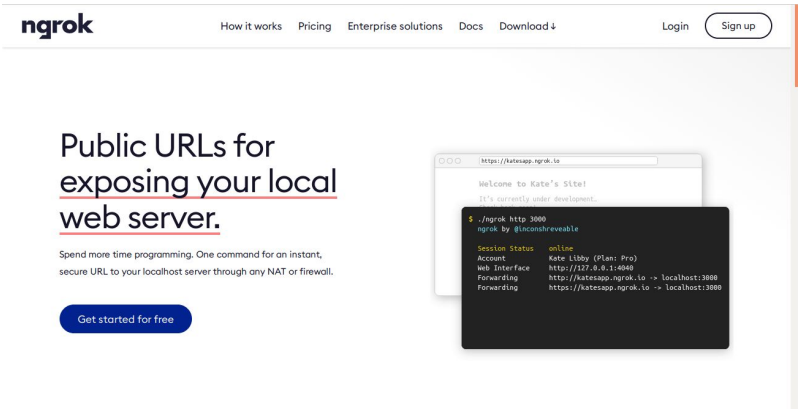
Retirada do site https://blog.cpanel.com/wp-content/uploads/2018/05/image2018-2-8_17-46-1.png

Uma característica importante dessa ferramenta é o seu funcionamento local. Ou seja, todas as diferentes versões de um repositório são mantidas na máquina do usuário que utiliza o Git. No entanto, nos dias de hoje existem soluções que possibilitam que esses repositórios fiquem *onlines* para que os programadores possam desenvolver seus projetos em qualquer máquina ou mesmo desenvolvê-los colaborativamente com outras pessoas, que podem ter acesso a esse repositório. O GitHub é uma dessas soluções *online* e possivelmente a mais famosa.

5.3.13 ngrok

Ngrok é um serviço *online* que ajuda a expor o *localhost* de uma máquina para outras máquinas. Dessa maneira fica possível emular uma hospedagem local de um *site*, por exemplo, e realizar os devidos testes no seu desenvolvimento de *software*. A ferramenta se propõe ainda a fazer esse serviço de uma maneira confiável, ao utilizar os seus chamados "túneis de segurança" para realizar essa exposição do *localhost*.

Figura 24 - Página inicial do ngrok



Retirada do site https://ngrok.com/

O uso dessa solução é bastante simples. Basta instalá-lo, executar o *site* em construção no *localhost* e, por fim, gerar um link através do ngrok para ser compartilhado com as pessoas envolvidas no teste dessa aplicação *web*, por exemplo.

Figura 25 - Exemplo de links de acesso para o *localhost* de uma máquina utilizando o ngrok

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Account                       Allan (Plan: Free)
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://8057395be542.ngrok.io -> http://localhost:5000
Forwarding                    https://8057395be542.ngrok.io -> http://localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

5.3.14 Heroku

Por fim, apresenta-se aqui como último dos principais conhecimentos necessários para o desenvolvimento desta monografia, a plataforma de nuvem Heroku. Como uma das ferramentas mais populares em seu ramo, essa ferramenta possui o foco de facilitar o *deploy* de qualquer aplicação *web* com back-end. Ou seja, o lançamento de um *site* para o público. Tanto que, atualmente, a ferramenta possui suporte para diversas linguagens, incluindo o Python. Com isso, o Heroku consegue agilizar a concretização de uma ideia de *site*, ao tornar a sua hospedagem e disponibilidade na internet algo simples e intuitivo.

Figura 26 - Logo do Heroku

https://upload.wikimedia.org/wikipedia/en/a/a9/Heroku_logo.png

A plataforma funciona com um sistema de *dynos*, que basicamente são ambientes Linux isolados no Heroku, que são onde se assentam as diferentes aplicações hospedadas na plataforma. Portanto, para que o *deploy* seja feito é necessário que o código das aplicações *web* estejam em algum repositório Git ou em algum *container* docker (que também é uma forma de isolar a sua aplicação, basicamente).

6 PROTÓTIPO

Boa parte deste trabalho, realizado de maneira individual, consiste no aprendizado das tecnologias necessárias para a construção do WordStorming. Diante desse fato e da complexidade do projeto, resolveu-se concentrar aqui em desenvolver no tempo disponível, pelo menos, a parte básica do jogo.

Em outras palavras, o objetivo central deste trabalho pode ser traduzido na construção de um protótipo do WordStorming: o desenvolvimento das características mais elementares do jogo, que permitam tanto a sua interação e entretenimento, como também a sua possível avaliação, por exemplo. Na prática, isso significa a descrição completa feita do jogo no capítulo 3 da monografia, com as seguintes ressalvas:

- caráter *multiplayer* limitado a dois jogadores;
- ausência das pontuações bônus;
- ausência do modo dinâmico; e
- ausência das funcionalidades expostas na seção 3.2 da monografia.

Ao longo dos próximos parágrafos, os pontos chaves do desenvolvimento do jogo são descritos. Precedendo-os, expõe-se também o cronograma deste projeto, com suas expectativas iniciais e concretizações finais.

6.1 Cronograma

Sem se ater ao tempo e aos recursos disponíveis, o desenvolvimento completo do jogo proposto neste trabalho pode ser dividido em três fases: protótipo, produto intermediário e produto final. A segunda fase se refere ao protótipo acrescido das pontuações bônus e algumas funcionalidades. A terceira fase se refere ao jogo na sua plenitude. O cronograma para este trabalho incorpora as duas primeiras fases, com o foco na realização e entrega da primeira fase (protótipo).

Figura 27 - Cronograma

Tarefas	Fevereiro	Março	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro
Proposta de Trabalho										
Definição das Tecnologias										
Configuração de Ambiente										
Páginas Iniciais (Front-End)										
Modelagem POO (Protótipo)										
Criação de Partidas e Sessões										
Validação de Letras Aleatórias e Palavras										
Pontuação das Palavras										
Modo Treino (primeira versão do jogo)										
Implementação do Multiplayer										
Finalização do Protótipo e Teste de Eficácia										
Banco de Dados e Conta (Usuários)										
Histórico de Palavras										
Significado e Sugestão de Palavras										
Pontuação Extra das Palavras										
Monografia Final										
Pôster										
Apresentação Oral										

A primeira parte do cronograma acima, representando a primeira fase, se compõe das seguintes tarefas em destaque:

- “Modelagem POO (Protótipo)”, se referindo a criação da base lógica do jogo;
- “Modo Treino (primeira versão do jogo)”, se referindo a um modo “*lonelyplayer*” do jogo, melhor descrito nas próximas seções; e

- “Finalização do Protótipo e Teste de Eficácia”, se referindo a concretização do protótipo do jogo e a avaliação do mesmo.

Já a segunda parte do cronograma, por sua vez, representando a segunda fase, se compõe das seguintes tarefas em destaque:

- “Significado e sugestão de palavras”, se referindo às funcionalidades “Dicionário” e “Sugestão” descritas no capítulo 3 da monografia; e
- "Pontuação Extra das palavras”, se referindo às pontuações bônus do jogo.

Naturalmente, na prática, o cronograma acabou sofrendo vários reajustes, como a inclusão de algumas tarefas, até então, não pensadas; exclusão de outras; e, principalmente, adiamento da maior parte delas. As únicas tarefas que não foram adiadas foram as 4 primeiras. No fim das contas, apenas a primeira parte do cronograma foi concretizada. Evidentemente, as tarefas não previamente pensadas, como, a sala de jogadores - melhor descrita nas próximas seções - também entram nessa concretização.

Em especial, o ano de 2020 foi atípico, por conta da situação de quarentena causada pelo novo coronavírus (COVID-19). Isso acabou trazendo mudanças tanto para o curso que teve que se adaptar temporariamente a um modelo EaD improvisado, quanto para o espaço da faculdade que teve o seu acesso por um bom tempo impedido e, atualmente, restringido. Isso tudo afetou um pouco este trabalho não ter alcançado parte da segunda parte do cronograma, que apesar de considerado um desafio, esperava-se conseguir cumprir, pelo menos, parcialmente.

6.2 Tarefas Primárias

Antes do desenvolvimento do jogo ser iniciado, propriamente dito, algumas tarefas foram feitas previamente, como o breve aprendizado de algumas das tecnologias utilizadas neste trabalho. Em especial, dedicou-se um tempo para entender o básico da *framework* Flask, por exemplo, pois ela sustenta todo o desenvolvimento.

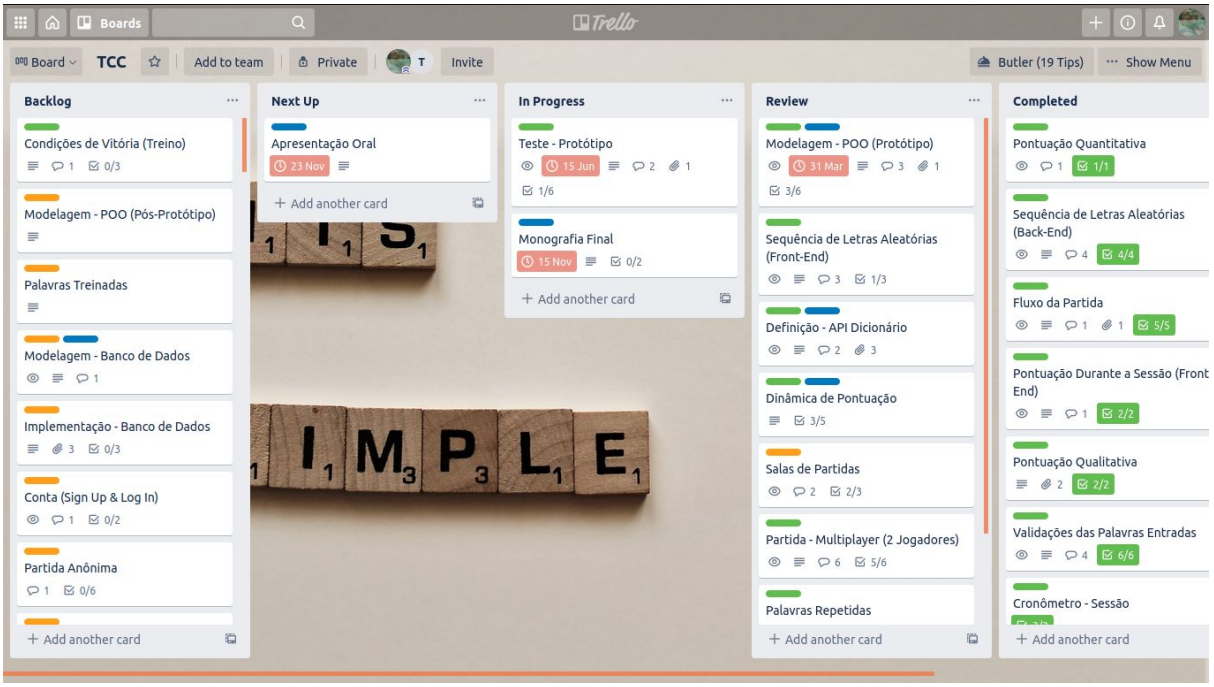
Fora isso, três coisas importantes foram feitas também, antes da criação das primeiras páginas do jogo: o planejamento temporal do projeto; configuração do ambiente de desenvolvimento do *site*; e a definição da arquitetura da aplicação. Elas são melhor comentadas nos próximos parágrafos.

6.2.1 Planejamento do projeto

Uma vez que o WordStorming foi concebido formalmente e definiu-se as tecnologias principais a serem utilizadas na sua construção, o passo natural seguinte foi o da criação de um projeto no Trello. Basicamente, elaborou-se um completo plano de ação - uma versão mais detalhada do cronograma apresentado, por assim dizer - para melhor gerenciamento do desenvolvimento do jogo.

Com o tempo, o projeto foi sofrendo algumas adaptações. No entanto, isso não mudou a importância do Trello, seja para a visibilidade ampla do desenvolvimento, seja para o controle adequado da produção deste trabalho.

Figura 28 - Gerenciamento do WordStorming no Trello



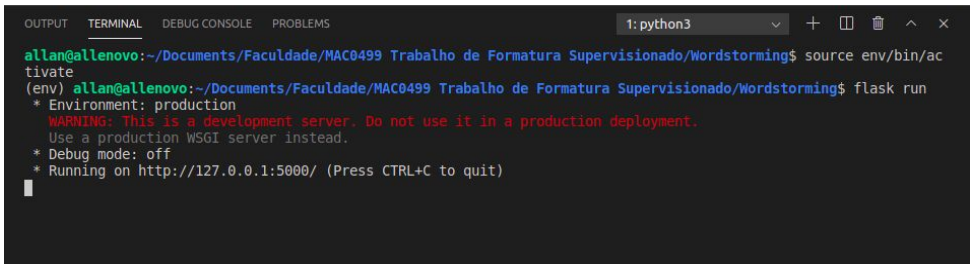
Rascunhou-se ainda, nesta etapa do projeto, os *wireframes* da aplicação *web*. Em outras palavras, criou-se vários rascunhos (no papel) representando os diferentes *layouts* das páginas do *site*. Com isso, aproveitou-se também para esboçar a ligação entre essas diferentes páginas, além da URL de cada uma delas.

6.2.2 Configuração do ambiente

Para um projeto de computação da magnitude deste trabalho, a estruturação básica de um ambiente de programação é sempre muito importante. Para esse fim, utilizou-se aqui o ambiente virtual do Python. Em poucas palavras, um ambiente isolado contendo uma versão específica do Python, juntamente com bibliotecas e pacotes requisitos para o desenvolvimento de um *software* específico. Isso é feito para que não haja confusão no desenvolvimento de diferentes aplicações em uma mesma máquina, pois elas podem possuir dependências conflitantes, como versões diferentes da linguagem Python, por exemplo.

A criação de um ambiente virtual do Python é relativamente simples. Dada as devidas instalações, basta utilizar o comando “python -m venv env” em um terminal, que um ambiente virtual nomeado “env” é criado. Naturalmente, a primeira dependência instalada no ambiente virtual deste projeto foi a *framework* Flask, que como já dito, é baseada na linguagem Python e é onde ocorreu todo o desenvolvimento *web* do WordStorming.

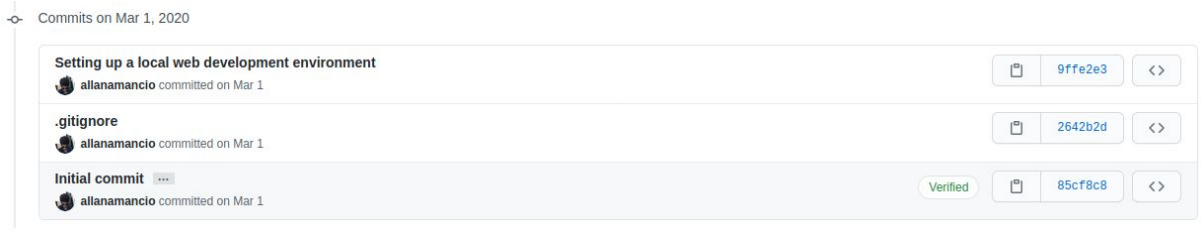
Figura 29 - Ativação de um ambiente virtual Python, seguido da execução do WordStorming em Flask (no terminal do VS Code)



A configuração para o versionamento do código do projeto também foi feita. Basicamente, através do Git, criou-se um repositório local do *site*, que ficou conectado com um repositório *online* no GitHub. Dessa maneira, todo *commit* - efetivação das mudanças do repositório acompanhada de comentário - no projeto pode ser facilmente refletido no repositório *online*.

Por fim, para a edição dos códigos-fontes, utilizou-se o Visual Studio Code.

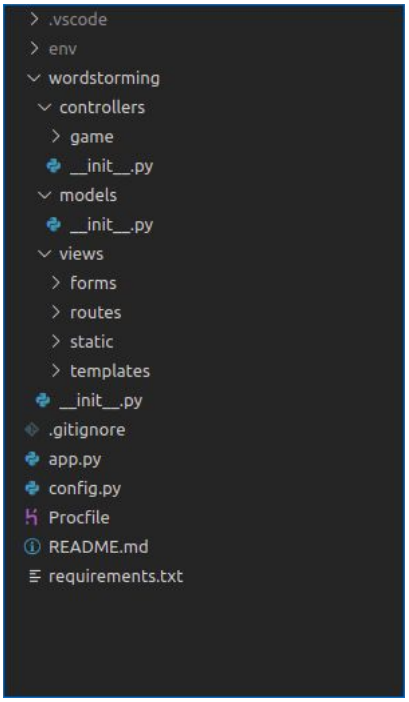
Figura 30 - Primeiros *commits* do projeto no GitHub



6.2.3 Arquitetura de pastas

A última tarefa primária importante aqui, é a hierarquia das pastas e dos arquivos essenciais do projeto. Apesar dessa arquitetura ter sido aprimorada ao longo do seu desenvolvimento, em grande parte, ela foi realizada antes da construção das primeiras páginas do *site*.

Figura 31 - Arquitetura de pastas do WordStorming



Como o Flask não impõe nenhuma arquitetura prévia, isso permitiu a criação de uma arquitetura própria para este projeto, que foi inspirada no padrão de arquitetura *Model-View-Controller* (MVC). Como o nome do padrão sugere, ele divide a arquitetura em 3 camadas interconectadas: *model*, *view* e *controller*. A ideia dessa divisão é tanto a divisão lógica e de conceitos da aplicação, quanto o aproveitamento de código pelo seu reuso.

Dessa maneira, a arquitetura do WordStorming basicamente se dá em três pastas de mesmo nível, nomeadas "views", "models" e "controllers", organizando todos os arquivos da aplicação *web*:

- A pasta "views" é formada de quatro pastas: "templates", "static", "routes" e "forms".
    - A primeira pasta é onde ficam os arquivos HTML, representando as páginas do *site*.
    - A segunda pasta é onde ficam os arquivos CSS e JavaScript, dando cor e dinamicidade ao *site*, basicamente, além de ser onde são guardadas as imagens e documentos do *site*.
    - A terceira pasta é onde ficam os arquivos Python responsáveis por realizar o mapeamento dos arquivos HTML com as suas respectivas URLs.
    - E a quarta pasta é onde ficam os arquivos responsáveis por lidar, no back-end, com os formulários HTML. Esses formulários são melhor explicados nas próximas seções.
- A pasta "models" é onde ficam os eventuais dados da aplicação *web*, como informações sobre o usuário, por exemplo. No fim das contas, para o objetivo deste protótipo foi mais produtivo utilizar as sessões do Flask. Essas sessões são melhor explicadas nas próximas seções.
- Por fim, a pasta "controllers" é formada de uma única pasta "game", que é onde ficam os arquivos responsáveis por lidar com a lógica do jogo, como: a construção de sequência de letras aleatórias e a pontuação de palavras.

Para que essa arquitetura funcionasse no projeto, o entendimento e a utilização das ferramentas de importações do Python foram necessários. A maneira mais simples de um arquivo Python importar funções de um outro arquivo Python é quando ambos estão na mesma pasta e no mesmo nível de hierarquia. Nesse cenário, o arquivo contendo as funções a serem importadas é chamado de módulo.

O problema aqui é quando os módulos estão em outras pastas e não necessariamente no mesmo nível do arquivo requisitando as suas funções. Nesse caso, para que a importação seja possível, é preciso que a pasta contendo o módulo seja um *package*, pois isso a faz agir como um "módulo", por assim dizer. Para que isso aconteça, um arquivo "__init__.py" precisa ser criado na pasta contendo o módulo, para que a linguagem Python entenda que aquela pasta se trata de um *package*, permitindo assim a importação de suas funções.

6.3 Páginas Iniciais

Com os *wireframes* construídos, os esforços nesta etapa puderam ser majoritariamente concentrados em aprender as tecnologias HTML, Bootstrap e Jinja2 (parte da *framework* Flask) e, consequentemente, na implementação das páginas iniciais: "Início", "Sobre" e "Regras".

A linguagem HTML e o *toolkit* Bootstrap tiveram o seu maior uso com essas páginas, principalmente com a página de "Início". Através da primeira tecnologia, conseguiu-se conceber o *template* comum das páginas da aplicação, e através da segunda, a determinação das cores padrões e de componentes básicos, como alguns botões, que seriam utilizados ao longo de todo o *site*.

Como mencionado, a página "Início" (de URL "/" ou "/home") foi uma das que mais se utilizou dos conhecimentos HTML e Bootstrap na sua construção, por, naturalmente, ter sido a primeira página a ser construída. O papel dessa página no *site*, como esperado, é de

receber os jogadores ao entrar no site, anunciando o seu propósito geral: ser um jogo de palavras *multiplayer* com o fim de entreter e estimular a criatividade linguística.

Figura 32 - Página “Início” do WordStorming

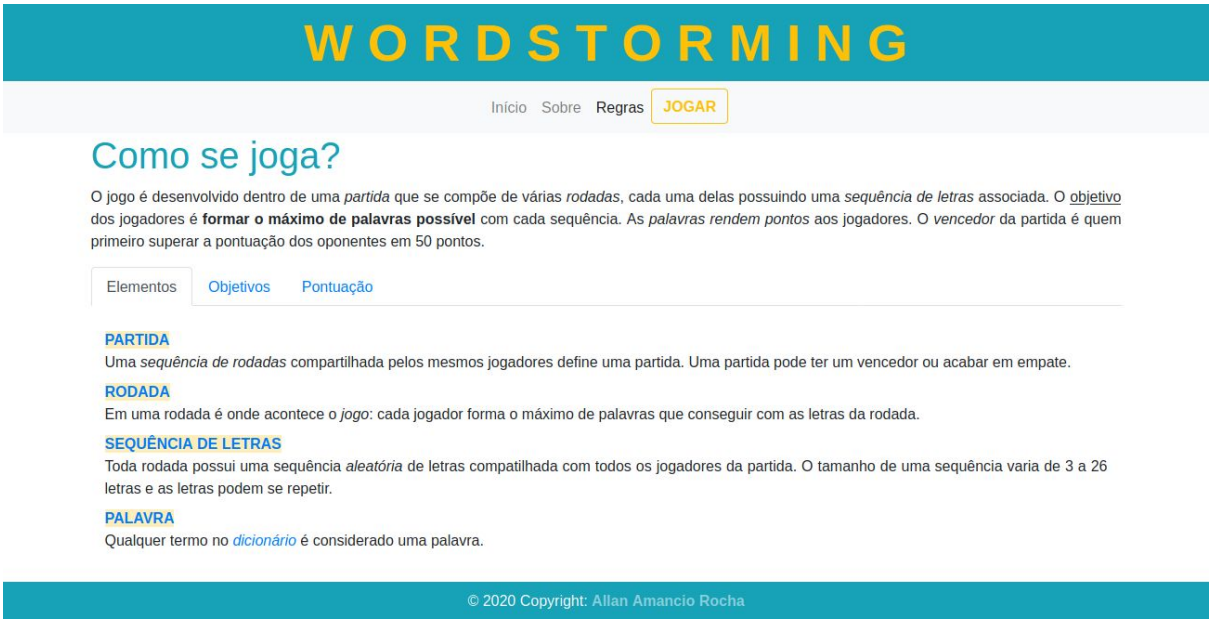
A tecnologia Jinja2, ferramenta para a construção facilitada de *templates* HTML com a linguagem Python, foi utilizada aqui justamente para esse fim. Através dela, um *template* HTML pode herdar ou importar outros *templates* HTML. Dessa maneira, boa parte do trabalho realizado na página “Início” com as tecnologias HTML e Bootstrap não precisa ser repetido nas páginas posteriores do *site*, incluindo as páginas “Sobre” e “Regras”. Ou seja, a tarefa se torna, então, desenvolver as especificidades de cada página.

A página “Sobre” (de URL “/about”) é a parte responsável do *site* em contextualizar os jogadores sobre do que se trata o jogo e de como ele foi realizado. A página também disponibiliza os arquivos produzidos com este projeto, incluindo esta monografia.

Figura 33 - Página “Sobre” do WordStorming

A página “Regras” (de URL “/rules”) explica como funciona o jogo. Basicamente, inicia com uma demonstração resumida de como jogá-lo para, na sequência, destrinchar todos os detalhes das regras, divididas em “Elementos”, “Objetivos” e “Pontuação”.

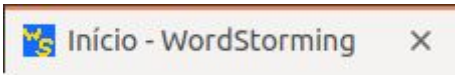
Figura 34 - Página “Regras” do WordStorming



Para acessar cada uma dessas páginas, basta estar em qualquer uma delas e clicar em um dos três primeiros *links* no menu fixo superior do site. Com exceção da página “Início”, todas as páginas iniciais podem ser visualizadas nesse endereço: https://linux.ime.usp.br/~allanrocha/mac0499/.

Em paralelo ao desenvolvimento dessas páginas, criou-se também o favicon do *site*, uma pequena imagem (16 por 16 pixels) que pode ser utilizada nos navegadores *web*. Guardada dentro da pasta “static” da pasta “view” do *site* do WordStorming, ele serve aqui como ícone presente ao lado do título da página, como mostrado na figura 35.

Figura 35 - Favicon do WordStorming na aba de um navegador *web*



6.4 Identificação

A primeira página efetivamente relacionada ao jogo é a página de “Identificação” (de URL “/game”). Para acessá-la, basta estar em uma das páginas iniciais e clicar em “JOGAR” no menu fixo superior do site. Através dessa página, o jogador se identifica com um *username* de sua escolha e seleciona o modo de jogo de sua preferência: treino ou partida. Ambos modos são melhor comentados em seções posteriores deste capítulo.

Para a construção dessa página, um dos requisitos importantes foi o uso de formulário HTML. Composto normalmente de, pelo menos, um campo de texto, ele é a maneira pelo qual as aplicações *web* podem receber dados dos seus usuários, através do front-end do *site*. Para facilitar esse processo, principalmente do envio dos dados do front-end para o back-end, utilizou-se a extensão Flask-WTF do Flask, devidamente instalada no ambiente virtual do projeto. Com ela, torna-se possível criar os formulários HTML no back-end do *site*, por assim dizer, e depois apenas refleti-los no front-end do *site*.

Dessa maneira, não só a criação desses formulários fica facilitada, como também a integração entre ambas partes da aplicação *web*, lidando com os formulários.

Teve-se que se aprender também a passar dados do back-end para o front-end do *site*, pois, embora o primeiro seja responsável pela validação do *username* inserido, o segundo é quem retorna o *feedback* de validação. Essa validação basicamente confere se o *username* não é nulo, repetido ou se possui caracteres inválidos (“/” e espaço em branco, por exemplo).

Figura 36 - Página “Identificação” do WordStorming (com *username* inválido)

Uma vez que o usuário se identifica, a aplicação precisa armazenar essa informação, ou seja, o *username* escolhido. Isso é feito utilizando as sessões do Flask.

O protocolo HTTP utilizado na Web é *stateless*. Ou seja, toda requisição feita a um servidor, ainda que vinda de um mesmo navegador, é tratada como única e isolada. Portanto, o servidor não tem como saber as ações prévias de um navegador *web* específico. Os *cookies*, pacotes de dados guardados no navegador, surgem para resolver esse problema. Eles armazenam as atividades prévias relevantes do cliente e informam o servidor a cada requisição feita.

Sessões baseiam-se em *cookies* e possuem um comportamento parecido, se diferenciando, essencialmente, com os dados do cliente sendo armazenados no próprio servidor. O *cookie* é utilizado apenas para identificar o cliente. As sessões em Flask, no entanto, se comportam como se fossem *cookies* normais, ou seja, os dados do cliente são armazenados em *cookies*, ao invés de no servidor. No entanto, por padrão, sessões em Flask criptografam esses dados e não permitem que eles sejam modificados pelo navegador *web*, a menos que o usuário o utilizando tenha a senha necessária, guardada no servidor.

Caso o jogador já tenha se identificado, a página de “Identificação” mostra o seu *username* escolhido. No entanto, caso ele queira, ele pode trocá-lo clicando no *link* “*Quero mudar de usuário*" logo abaixo do *username* em questão. Quando isso é feito, basicamente, a sessão que tinha sido gerada anteriormente no back-end é excluída e ele volta para a mesma página quando havia escolhido um *username*.

Figura 37 - Página “Identificação” no WordStorming (com jogador identificado)

6.5 Modo Treino

Para que um jogador possa entender a lógica do WordStorming, sem necessariamente depender da presença de outro jogador, o modo treino do jogo foi criado. Mas para além desse fim, essa modalidade permite o jogador também treinar a formação de diversas palavras com diferentes sequências de letras aleatórias, daí o seu nome.

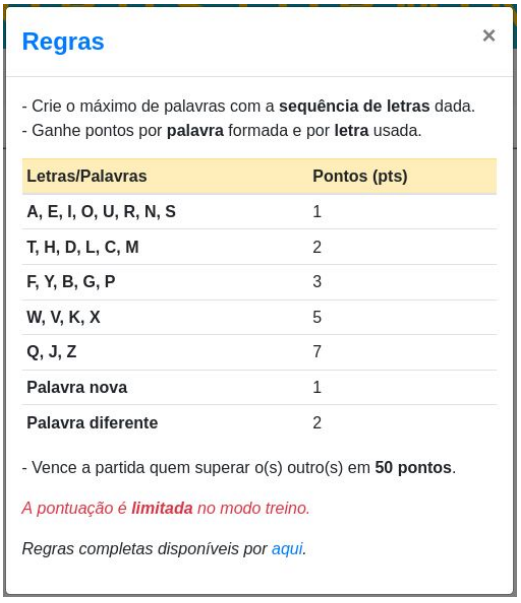
Pode-se dizer que esse modo é uma versão simplificada do jogo aqui desenvolvido, propriamente dito. As diferenças principais para a versão normal são que o jogador joga sozinho aqui (*“lonelyplayer”*) e as partidas se compõem de uma única rodada. Portanto, por conta dessas diferenças, não existe pontuação por palavras novas ou diferentes.

Figura 38 - Página “Treino” do WordStorming

Para acessar essa página de “Treino” (de URL “/game/practice”), basta se identificar (caso já não esteja identificado) na página de “Identificação” e clicar no botão “Treino”. Nessa página, o menu fixo superior muda em relação ao como ele estava nas páginas iniciais. Basicamente, os três *links* iniciais viram dois, “Voltar” e “Regras”, e o botão “JOGAR” se transforma em “COMPETIR”.

O primeiro *link* faz o jogador ir para a página de “Identificação”; o segundo *link* lhe mostra um resumo das regras do jogo através de um *pop-up* (surgimento de janela na mesma página); e o botão “COMPETIR” o leva para o modo partida do jogo.

Figura 39 - Resumo das regras do WordStorming



Como evidenciado, com exceção do caráter *multiplayer*, o modo treino não difere muito da lógica do jogo em modo partida. Sendo assim, as subseções seguintes tratam do jogo construído aqui de maneira a “ignorar” a existência dos modos. Se voltando ao modo treino, tema desta seção, apenas quando necessário.

6.5.1 Sequência de letras aleatória

Uma das primeiras partes do jogo, naturalmente, é a geração de letras aleatórias para o início de uma rodada. Uma partida só é iniciada mediante sinal do jogador. Portanto, até que isso aconteça, no local da página reservado para a sequência de letras aleatórias é mostrado apenas o seu tamanho, sem anunciar ainda as letras que a compõem.

Figura 40 - Prévia da sequência de letras aleatórias na página do jogo

Levando em conta que, essencialmente, toda palavra possui vogais, a sequência de letras aleatórias é formada de acordo com as seguintes regras:

- tamanho mínimo de 3 letras e máximo de 26 letras;
- existência de pelo menos uma vogal;
- cada letra possui 51% de probabilidade de ser vogal e 49% de ser consoante;

As letras da sequência são mostradas na página através do uso do Ajax, com o auxílio da biblioteca Jquery, portanto, utilizando a linguagem JavaScript. Isso é feito para que não seja necessário atualizar a página do jogo a cada rodada, quando uma nova sequência de letras aleatórias é gerada.

Uma API é, basicamente, uma interface construída com regras que permitem a integração entre aplicações via código. E para utilizar o Ajax, é necessária a criação das APIs RESTful - ou seja, baseadas no sistema REST (uma abstração da arquitetura *web*) - no back-end do *site*. Uma maneira simples de entendê-las, é imaginá-las como se fossem *sites* em formato de API, pois agem da mesma maneira. Em outras palavras, a interação com elas é realizada através de requisições utilizando URLs - que é, basicamente, o que o Ajax faz - processadas pelo protocolo HTTP.

Nesses tipos de APIs, comumente utiliza-se o formato JSON para a transferência dos dados (o caso deste jogo).

6.5.2 Temporizador

Como dito, uma partida só começa mediante sinal do jogador. Isso é feito através do botão “Começar”, localizado logo abaixo do local das sequências de letras aleatórias. Quando ele é clicado, um cronômetro regressivo se inicia, anunciando: o tempo da rodada de 2 minutos; uma sequência de letras aleatória gerada; e o desbloqueio do campo de texto da página, para a entrada de palavras formadas pelo jogador.

Figura 41 - Cronômetro das rodadas no WordStorming

O cronômetro foi feito utilizando JavaScript puro, ou seja, sem utilizar a biblioteca jQuery e afins. A linguagem nativamente já possui funções temporizadoras, que, naturalmente, ajudam na construção do cronômetro. Por exemplo, em JavaScript, é possível colocar uma dada função para ser executada a cada intervalo específico de tempo.

Quando faltam 15 segundos para o cronômetro zerar, o tempo fica vermelho, sinalizando que a rodada está chegando ao fim. Caso o jogador queira, a qualquer momento, ele pode encerrar a rodada antes do tempo acabar, clicando no *link* “*Encerrar rodada*”, logo abaixo do tempo do cronômetro. Quando a rodada chega ao fim, a entrada de palavras formadas pelo jogador é travada novamente.

Especificamente no modo treino, quando o tempo acaba, surge um botão “De novo” para que uma nova partida de uma única rodada se inicie novamente. Este botão possui o mesmo efeito do botão “Começar”, ou seja, anuncia o tempo da rodada de 2 minutos, gera uma nova sequência de letras aleatórias e permite novamente a entrada de palavras formadas pelo jogador.

Figura 42 - Botão “De novo” no modo treino do WordStorming

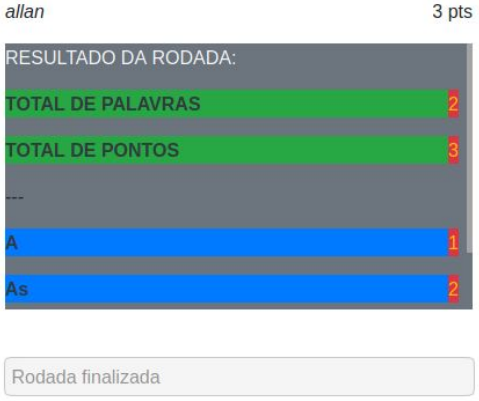
6.5.3 Interface principal

Toda a interação do jogador com o jogo acontece pelo que está se chamando aqui de interface principal. Através dela, além de ver o seu *username*, o jogador também vê informações sobre suas jogadas, como:

- total de pontos acumulados na partida;
- palavras aceitas na rodada, acompanhadas de suas pontuações;
- total de pontos da rodada;
- total de palavras aceitas na rodada;

Em especial, as duas últimas informações listadas acima apenas aparecem ao fim de uma rodada e antes do início de uma próxima rodada, pois elas compõem o “RESULTADO DA RODADA”.

Figura 43 - Interface principal de um jogador específico no WordStorming



Até aqui, o Bootstrap estava sendo usado para realizar, basicamente, todas as tarefas envolvendo o uso da linguagem CSS. No entanto, o núcleo da interface principal - o quadrilátero regular cinza - mudou um pouco essa abordagem, por conta das suas especificidades não resolvíveis por Bootstrap. Uma dessas especificidades, provavelmente a mais complicada para um leigo em desenvolvimento *web*, é a construção de um *scroll* (barra de rolagem) em uma parte específica de uma página HTML.

Em geral, um *scroll* já existe nativamente em uma página *web*, para todo o seu conteúdo. No entanto, quis-se aqui criar um *scroll* específico, apenas para o núcleo da interface principal, o que acabou tendo que ser realizado utilizando diretamente a linguagem CSS.

As atualizações de informações das jogadas do jogador (pontuação acumulada, inserção de palavras etc) foram todas tarefas feitas utilizando a tecnologia Ajax também, associada com o jQuery. Pois, dessa maneira, o jogador não precisa agir ativamente para que essas atualizações aconteçam.

6.5.4 Entrada de palavras

Quando o jogador digita uma palavra no campo de texto da página, para a entrada de palavras formadas por ele, essa palavra é enviada ao back-end do *site* para validação. Validação feita, inclusive, com a ajuda da extensão Flask-WTF do Flask. Caso a palavra seja válida, ela é devidamente pontuada, de acordo com a pontuação de cada letra na palavra, e inserida na interface principal do jogo, como já demonstrado. E o jogador recebe ainda um *feedback* positivo em cima do campo de texto, sinalizando uma jogada certa.

Figura 44 - *Feedback* de palavra válida e a sua pontuação básica no WordStorming

Do contrário, o jogador recebe um *feedback* negativo sobre a invalidez da palavra colocada, se enquadrando em um dos quatro casos abaixo:

1. “Palavra inválida. Use apenas letras sem nenhum espaço.”
2. “Palavra não respeita a sequência de letras aleatórias dada”
3. “Palavra já utilizada na rodada.”
4. “Palavra não existe no dicionário.”

Os dois primeiros casos são diretos. Simplesmente, o jogo recusa a ausência de palavras, formação de frases e palavras que não são possíveis de serem formadas com as letras dadas na rodada ou que possuam caracteres inválidos. Com relação a esse último, como as palavras devem ser inglês, esse processo é facilitado, pois considerou-se aqui que não são utilizados acentos nas palavras, por exemplo.

Quanto ao terceiro caso, para que o jogo saiba quais palavras já foram utilizadas na rodada, elas são armazenadas na sessão do Flask, a mesma utilizada para identificação do jogador. Os navegadores *web*, normalmente, limitam o tamanho dos *cookies* em 4KB. Isso já é mais do que suficiente para o armazenamento das palavras usadas por um jogador durante uma partida. Principalmente em um contexto de prototipagem.

Figura 45 - *Feedback* de “Palavra já utilizada na rodada” do WordStorming

Por fim, para o quarto caso, a biblioteca pyenchant baseada na biblioteca Enchant, em linguagem C, é utilizada. Com o propósito de conferir a ortografia correta das palavras, uma vez devidamente instalada no ambiente virtual do projeto, ela é utilizada aqui para verificar se as palavras inseridas no WordStorming existem semanticamente, e portanto, no dicionário.

Ainda nesse último caso, alguns desafios foram levantados ao longo do desenvolvimento. Testou-se algumas outras bibliotecas, verificou-se alguns dicionários *online* (inclusive o do jogo Scrabble) e percebeu-se que todas elas validam palavras que o WordStorming, a priori, não deveria aceitar, tais como siglas e abreviações. Recorreu-se a alguns arquivos disponibilizados na Web contendo uma lista de palavras válidas, mas eles

possuíam o mesmo problema. Pensou-se em criar uma lista de palavras próprias para o jogo aqui, mas isso, evidentemente, seria inviável.

No fim das contas, permaneceu-se com a biblioteca pyenchant pela sua rapidez e pelo seu "problema" maior ser em cometer falsos positivos, do que falsos negativos. Ou seja, validar palavras que "não são" palavras, em oposição a invalidar palavras que são palavras. Além disso, fez-se uma simples restrição para o WordStorming não aceitar palavras formadas apenas com uma letra, com exceção do "i" ou "a".

6.6 Sala de Espera

Ao clicar no botão "Partida" na página de "Identificação", o jogador é direcionado para a página de "Sala de espera" (de URL "/game/multiplayer"). O menu fixo superior nessa nova página é igual ao menu fixo da página de "Treino", com exceção do botão "COMPETIR", que se torna o botão "TREINAR", e que leva o jogador para a página de "Treino".

Anteriormente, essa era para ser uma sala de jogadores, onde seria possível ver jogadores *online* e convidar um deles para iniciar uma partida. Devido às diversas complicações que isso ocasiona, por conta das diferentes ações que o usuário pode tomar, simplificou-se o funcionamento da página para uma "Sala de espera". Dessa maneira, os jogadores só necessitam aguardar na sala, pois o jogo cria automaticamente os pares de cada partida, com base em quem está *online*. Além disso, como sugerido, o protótipo construído se limita a partidas com apenas 2 jogadores.

Figura 46 - Página "Sala de espera" do WordStorming

Os jogadores *online* são identificados através de um sistema de *heartbeats*, ou seja, sinais periódicos (no caso, de 1 segundo) gerados pelos jogadores quando estão na página de "Sala de espera". Isso é feito com o uso do Ajax, para que o navegador *web* sinalize seu estado ativo para o servidor, sem atualização de página. Já para a formação de pares, realiza-se um processo análogo ao *three-way handshake* (acordo de três vias), comumente utilizado na área de redes de computadores para o estabelecimento de conexões no TCP, um dos protocolos no qual a *internet* se assenta. Em resumo, o servidor, com os jogadores *online* mapeados, simula tentativas de conexões entre eles de forma periódica (no caso, a cada 5 segundos):

1. Um primeiro jogador tenta estabelecer conexão com um segundo jogador.
2. O segundo jogador aceita a conexão com o primeiro jogador.
3. O primeiro jogador confirma a conexão.

Ou seja, apenas quando esse acordo de três vias é realizado que, um par de jogadores, para a realização de uma partida no WordStorming, é formado.

Cada passo de tentativa de conexão é realizado quando a página “Sala de espera” automaticamente se atualiza a cada 5 segundos, com o uso de JavaScript. Isso sinaliza ao servidor que o jogador ainda está *online* e deseja iniciar uma partida. O usuário pode manualmente atualizar a página em um tempo menor, caso deseje.

Os valores de tempos periódicos (1 e 5 segundos) foram definidos empiricamente no protótipo. Percebeu-se com tempos menores, em um cenário com mais de 5 jogadores, a página “Sala de espera” não tem o comportamento esperado. A ocorrência de muitas requisições HTTP pode acabar “escondendo” várias outras, de maneira que o servidor não perceba que um jogador ainda estava *online*, por exemplo.

Em paralelo ao uso de tempos periódicos “bons” para a “Sala de espera”, utilizou-se também algumas técnicas de programação concorrente. Em especial, os semáforos. Basicamente, eles foram usados para tornar parte da simulação *three-way handshake* no servidor o mais sequencial possível, para que a formação de pares de jogadores não seja prejudicada. Além disso, nessa simulação, há a utilização de algumas variáveis globais no servidor, as quais o acesso precisa ser controlado.

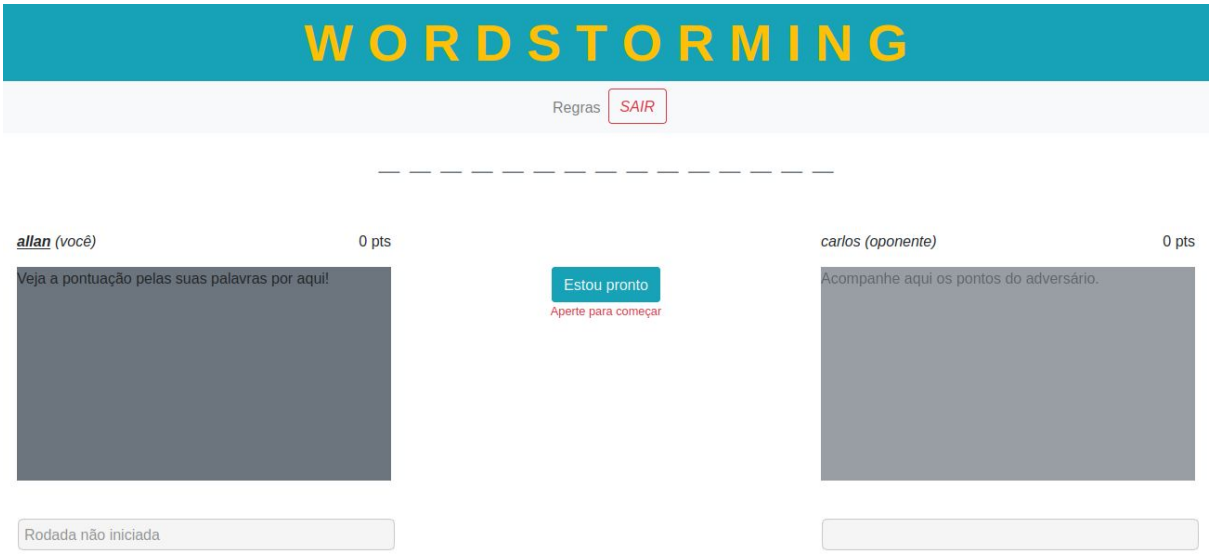
6.7 Modo Partida

Finalmente, tem-se aqui a página de “Partida” (de URL “/game/multiplayer/2/<player1>&<player2>”), que é onde, de fato, acontece o jogo. Para acessá-la, basta estar identificado na página de “Identificação” e clicar no botão “Partida”. Como comentado, o modo partida não difere muito do modo treino do jogo, com exceção do caráter *multiplayer* e a possibilidade de várias rodadas em uma mesma partida. O caráter *multiplayer* traz três diferenças visíveis na página de “Partida”, em relação a página de “Treino”:

1. o menu fixo superior;
2. o botão de início de partida; e
3. a duplicação da interface principal.

Nesse sentido, foca-se aqui, ao longo dos próximos parágrafos, em comentar as peculiaridades do modo partida do WordStorming - em relação ao seu modo treino -, incluindo as três diferenças visíveis citadas. Além de eventuais detalhes de implementação, que mereçam atenção.

Figura 47 - Página “Partida” do WordStorming



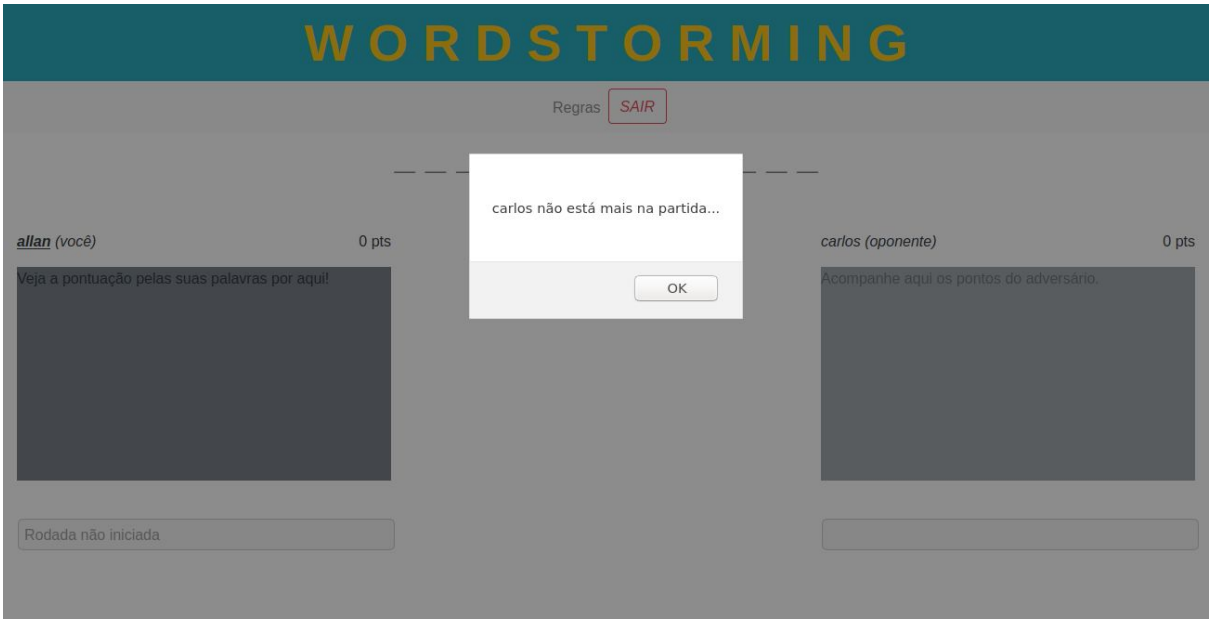
6.7.1 Sincronização dos jogadores

O esforço maior na construção da página de “Partida”, foi a comunicação entre os dois jogadores nela. Ou seja, dois clientes *web* diferentes. Isso é tanto relativamente complicado computacionalmente, quanto constante ao longo da partida. E, naturalmente, essa comunicação foi feita por meio do servidor, com o auxílio da ferramenta Ajax. Portanto, pode-se dizer que esta etapa do jogo foi onde se utilizou mais essa tecnologia e, consequentemente, as duas linguagens de programação do jogo: JavaScript e Python (responsável pela parte do servidor).

A primeira conversa interna entre os jogadores da partida acontece quando ambos estão, pela primeira vez, na página de “Partida” em seus respectivos navegadores *web*. Para que o fluxo da partida não seja afetado, é necessário que os navegadores dos jogadores estejam constantemente avisando um ao outro, através do servidor do WordStorming, que eles ainda estão na partida. Isso é feito pelo mesmo sistema de *heartbeat* citado na página de “Identificação”. Ou seja, cada jogador emite um sinal periódico ao servidor (no caso, a cada 0,5 segundo).

A cada 10 segundos, cada jogador confere com o servidor se o oponente ainda está na partida, por assim dizer. Se durante esse tempo o oponente não emitiu nenhum sinal, o navegador *web* conclui que “não tem mais ninguém” na partida e emite um alerta (uma janela implementada com JavaScript) informando ao único jogador na partida que o oponente não está mais nela. Uma vez que o jogador entende a mensagem, ou seja, clica em “OK” no alerta, ele é redirecionado para a página de “Identificação”.

Figura 48 - Alerta informando que o oponente não está mais na partida no WordStorming



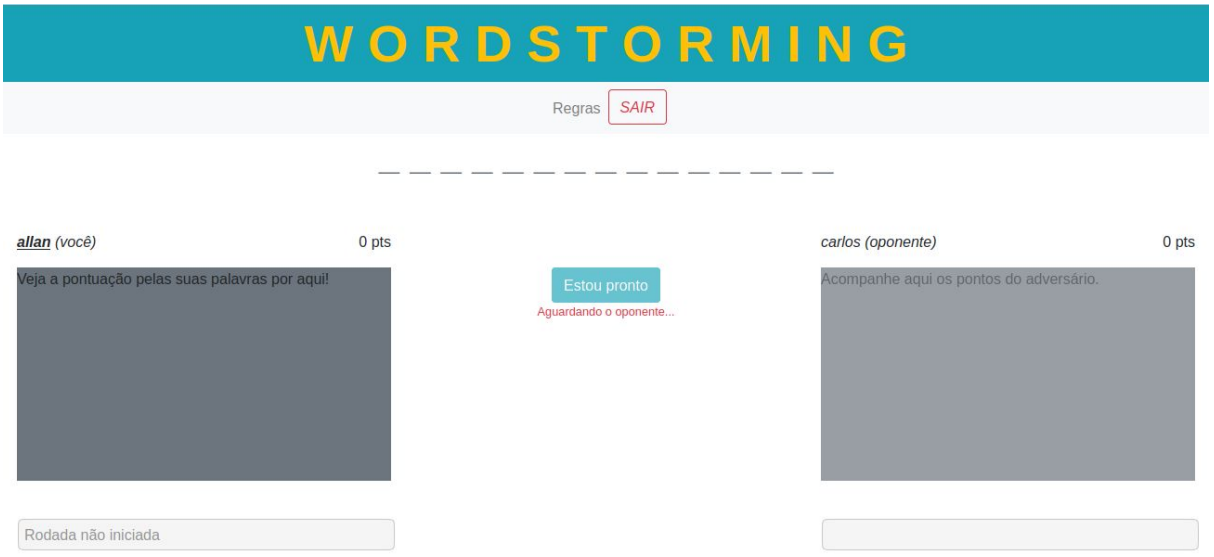
Essencialmente, existem duas maneiras de sair da partida: “sem querer” ou intencionalmente. A primeira opção engloba os casos onde a internet do jogador perdeu conexão, por exemplo, ou mesmo o jogador fechou a janela da partida, acidentalmente ou não. A segunda opção se refere a quando o jogador utiliza o menu fixo superior. Nesse menu, existe o *link* Regras, que é basicamente o mesmo da página “Treino” e existe o botão “SAIR”. Quando o jogador clica nesse botão, ele sai suavemente da partida e é redirecionado para a página de “Identificação”.

Figura 49 - Botão “SAIR” no menu fixo superior da página “Partida” do WordStorming

Uma vez que ambos jogadores da partida estão posicionados na página “Partida”, é necessário que ambos sinalizem que estão prontos para começar. No modo treino do jogo, isso seria o equivalente ao clicar no botão “Começar” logo abaixo do local da sequência de letras aleatória. A diferença aqui é que esse botão é chamado de “Estou pronto” e fica posicionado entre as duas interfaces principais da página. Além disso, clicar no botão não inicia imediatamente a partida, pois talvez ainda seja necessário esperar o oponente clicar no botão “Estou pronto” do seu lado, para que a rodada da partida comece sincronizada para ambos os jogadores.

Essa sincronização é feita através do Ajax e quase da mesma maneira que é feita a formação de pares na página de “Sala de espera”. Ou seja, utiliza-se aqui parcialmente a lógica do *three-way handshake* dos sistemas de redes de computadores. Basicamente um primeiro jogador sinaliza que está pronto de forma periódica e depois o outro jogador sinaliza de forma periódica que está pronto também. Quando o servidor percebe que ambos sinalizam prontidão, ele envia um sinal para que a rodada de ambos comece simultaneamente. Mais do que isso, envia também a mesma sequência de letras aleatória a ambos, já que essa é a regra do jogo em *multiplayer*.

Figura 50 - Jogador aguardando o seu oponente sinalizar prontidão no WordStorming



6.7.2 Fluxo do jogo

Uma vez que a partida é iniciada, ou seja, ambos jogadores estão prontos para começá-la, surge um cronômetro no lugar do botão “Estou pronto”. Tal como acontece no modo treino com o botão “Começar”. O cronômetro possui as mesmíssimas características do modo treino também: duração de 2 minutos; tempo avermelhado a partir dos 15 segundos; e pode ser encerrado antecipadamente a qualquer momento.

Caso um dos jogadores encerre a rodada antecipadamente, como ele não está jogando sozinho, ele é obrigado a esperar o término da rodada do seu oponente. No entanto, o seu oponente também pode encerrar a rodada antecipadamente. Nesse caso, os navegadores *web* de cada jogador, também através do Ajax, conferem constantemente com o servidor se isso ocorreu, para que ambos possam começar a próxima rodada igualmente de forma antecipada.

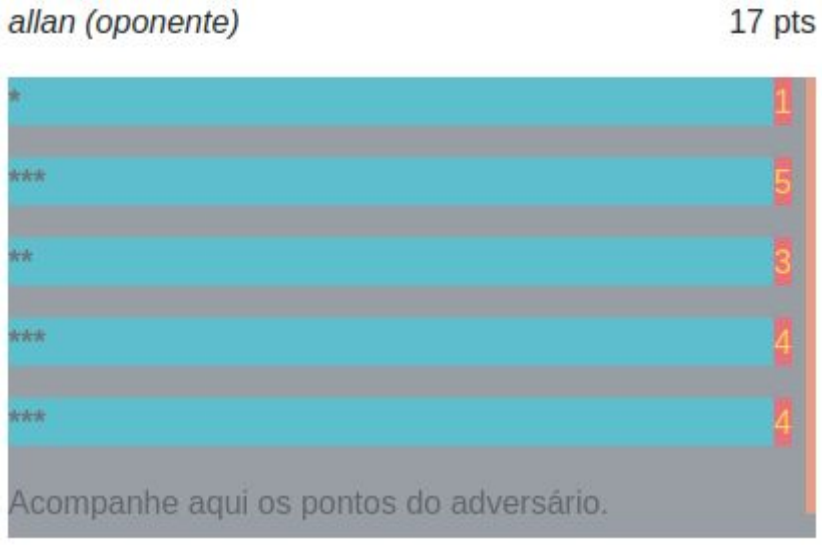
A vitória de um jogador é estabelecida quando ele consegue superar em 50 pontos, a pontuação adquirida pelo seu oponente, ao longo de toda partida. Por conta disso, durante a rodada, é mostrado logo abaixo do cronômetro, a diferença entre os pontos acumulados por ele e os pontos acumulados pelo seu oponente. Dessa maneira, o jogador consegue acompanhar seu progresso durante a partida e o quão próximo está da vitória (ou da derrota). Para deixar isso mais visual, a diferença de pontos é colorida de acordo com os seguintes critérios:

- pontuação negativa é colorida em vermelho;
- pontuação positiva menor que 25 é colorida em laranja;
- pontuação positiva maior ou igual a 25, mas menor que 40 é colorida em amarelo;
- pontuação maior que 40 é colorida em verde;

Figura 51 - Início de uma partida *multiplayer* no WordStorming

A página de “Partida” apresenta duas interfaces principais, uma à esquerda e outra à direita da tela. A da esquerda é a interface do jogador jogando na página e a da direita, a do seu oponente. Para que essa diferenciação fique visível, o *username* na primeira interface é sempre acompanhada de um “(você)” e o da direita, acompanhada de um “(oponente)”. Com a apresentação da segunda interface, um jogador passa a conseguir acompanhar tanto a pontuação total do seu oponente, quanto a pontuação das palavras que o seu oponente está colocando ao longo da rodada. Tudo em tempo real. A única coisa que o jogador fica impedido de ver do oponente, ao longo da rodada, são as palavras em si, pois elas ficam codificadas com asteriscos.

Figura 52 - Interface principal do oponente na página de “Partida” no WordStorming



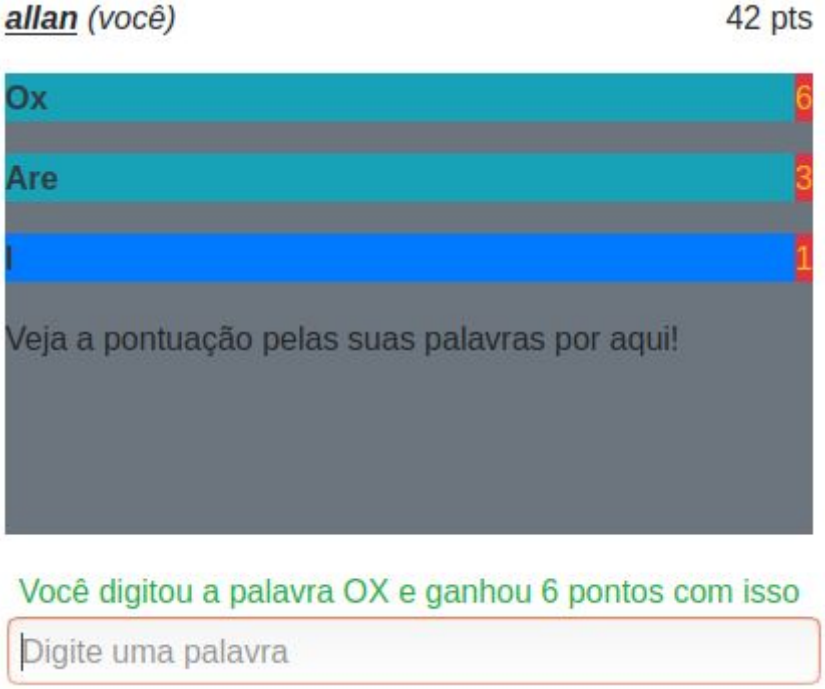
A implementação da interface principal do oponente, certamente foi uma das partes mais complicadas do protótipo. O envolvimento com a tecnologia Ajax e as linguagens JavaScript e Python foram intensos, principalmente com os dois primeiros, já que o conhecimento sobre eles era menor. A complicação nesse cenário se situa no fato de que

não basta apenas se comunicar com o oponente através do servidor. É necessário requisitar as suas palavras e as suas diversas pontuações específicas (total, das palavras e da rodada, por exemplo). E o oponente precisa receber essa requisição e enviar esses dados. E, por fim, a parte mais simples depois de tudo isso: codificar as palavras com asteriscos. Isso tudo o mais tempo real possível.

No modo partida, a inclusão de duas pontuações no WordStorming não existentes no modo treino, se torna possível: palavras novas e palavras diferentes. A primeira, pela possibilidade de múltiplas rodadas em uma mesma partida, e a segunda, pelo caráter *multiplayer*.

Por padrão, os jogadores só recebem essas pontuações específicas ao fim de cada rodada. No entanto, ao longo da rodada, o jogador já consegue identificar quais das suas palavras são novas, por exemplo. Naturalmente, na primeira rodada de uma partida, todas as palavras são novas. Portanto, apenas a partir da segunda rodada existe a possibilidade dos jogadores verem palavras repetidas. Essa identificação é feita pela coloração diferente que se dá na linha de palavra repetida dentro da interface. Basicamente, um azul mais suave. O jogador ganha 1 ponto a mais por cada palavra nova utilizada na rodada.

Figura 53 - Palavras novas (“I”) e repetidas (“Ox” e “Are”) na interface principal do WordStorming



Para que a identificação de palavras novas de um jogador seja possível, é necessário que se guarde o histórico de palavras, não mais apenas de uma rodada, mas de uma sequência delas. Para realizar isso, utilizou-se também as sessões em Flask, já mencionadas em seções anteriores.

Como dito, as pontuações específicas de palavras novas e diferentes só são atribuídas ao usuário no fim de cada rodada. Especificamente, isso é feito durante o intervalo: uma pausa de 30 segundos realizada antes do início de uma próxima rodada. Através dele, o jogador recebe essas pontuações e visualiza o seu desempenho na rodada, através do “resultado da rodada” na interface principal. Além disso, é onde o jogo realiza o acerto de contas. Ou seja, determina o eventual vencedor (ou perdedor) da partida, por exemplo.

As palavras diferentes não conseguem ser identificadas ao longo da rodada, pois como definido no capítulo 3 da monografia, isso depende de saber as palavras do oponente. O que não é possível, até o intervalo. Nesse momento, a visualização de todas as palavras do oponente, assim como o resumo do seu desempenho - pelo “resultado da rodada” -, se torna possível. As linhas de palavras diferentes na interface se tornam amarelas, sinalizando a inclusão dessa pontuação. O jogador ganha 2 pontos por cada palavra diferente usada na partida.

Isso complica um pouco mais a história da implementação da interface principal do oponente. Pois, para identificar as palavras diferentes, é preciso solicitar do oponente mais do que as palavras utilizadas na rodada, mas também as palavras utilizadas nas rodadas anteriores. Em suma, ter acesso ao seu histórico de rodadas.

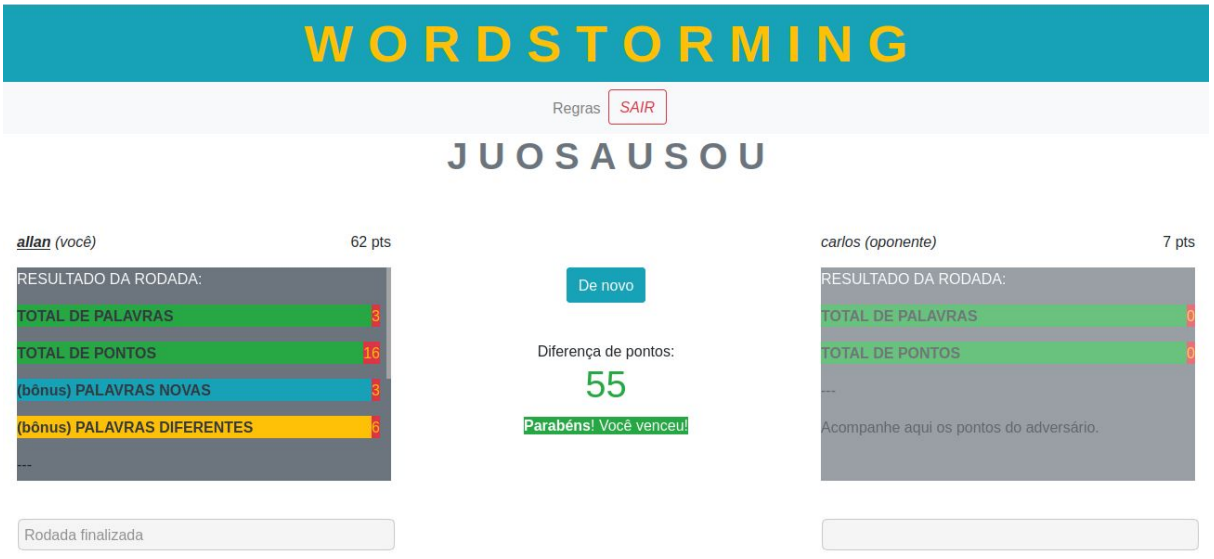
Em relação ao modo treino, o “resultado da rodada” no modo partida inclui mais duas linhas: “(bônus) PALAVRAS NOVAS” e “(bônus) PALAVRAS DIFERENTES”, acompanhadas das pontuações extras. Ambas são coloridas exatamente com a cor das linhas das palavras a quais elas se referem, ou seja, azul não suave e amarelo.

Por fim, um último detalhe interessante que aparece durante o intervalo: um texto abaixo da diferença de pontos, indicando o quão próximo da vitória (ou derrota) um jogador está. No total, são três textos diferentes com cores de fundo cinza, vermelho e verde. O primeiro dizendo ou quanto pontos, pelo menos, o jogador precisa acumular a mais para ganhar do seu oponente ou um caso de empate. O segundo, informando a sua derrota. E o último, a sua vitória.

Assim como na rodada, durante o intervalo, caso o jogador queira, ele pode encerrar o intervalo antecipadamente. No entanto, igualmente como na rodada, ele é obrigado a esperar o término do intervalo do seu oponente. A maneira como o jogo lida com isso é basicamente da mesma maneira em ambas situações.

O único momento em que o jogador não pode encerrar a rodada antecipadamente é quando o jogo decide o resultado da partida. Nesse momento, no lugar do cronômetro, surge um botão nomeado de “De novo”, para o vencedor da partida, e de “Revanche”, para o perdedor. O servidor consegue determinar o resultado do jogo através do fluxo de informações constantemente trocadas pelos jogadores, sobre as suas jogadas, incluindo as suas pontuações acumuladas. Isso permite gerar justamente a diferença de pontos dos jogadores.

Figura 54 - Fim de uma partida vencedora no WordStorming



6.7.3 Detalhes técnicos

De uma certa forma, houve várias linhas de código novas em JavaScript e Ajax, com o modo partida. No entanto, por outro lado, por conta da grande similaridade desse modo com o modo treino, muitas funções JavaScript realizadas especificamente para o modo treino, ou foram duplicadas com as devidas peculiaridades necessárias para o modo partida ou adaptadas para servir aos dois modos do jogo.

O mesmo ocorreu com o código Python, vide as funções de pontuação de palavras e geração de sequência de letras aleatórias. Em especial, utilizou-se a Programação Orientada a Objeto (POO) para realizar a adaptação do código para ambos modos do jogo, transformando-o em classes e objetos. No primeiro momento, tentou-se seguir estritamente um princípio relevante nesse paradigma, o “tell, don’t ask”. Esse princípio diz que, em oposição a um paradigma procedural, por exemplo, deve-se informar ao objeto o que se quer, ao invés de dizer como fazê-lo o que se quer. Como o restante do código da aplicação estava mais procedural do que POO, por assim dizer, não houve muito sucesso nesse objetivo. O tempo disponível para o desenvolvimento do protótipo também afetou um pouco a não implementação adequada desse princípio. De qualquer maneira, transformar essas funções mencionadas em classes e objetos em POO serviu para, pelo menos, modularizar melhor o código (um dos pilares de POO, inclusive), o que por si já facilitou um pouco mais o desenvolvimento do modo partida do WordStorming.

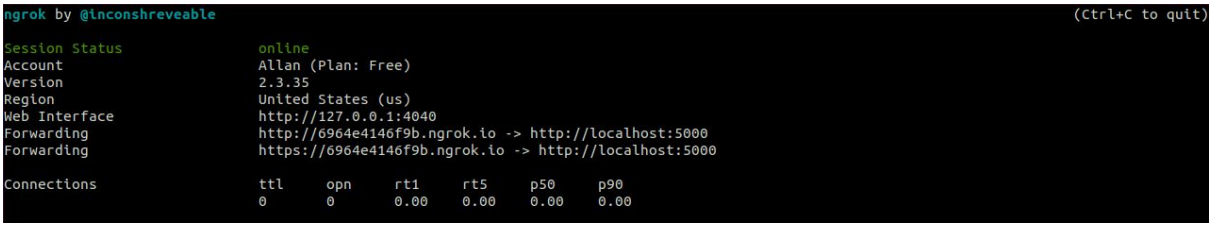
Por fim, na página de “Partida”, houve várias variáveis globais para controlar as diversas e constantes chamadas Ajax dos jogadores, como evidenciado. Portanto, a programação concorrente foi utilizada aqui, novamente na sua forma de semáforo, para assegurar o acesso adequado a essas variáveis.

6.8 Hospedagem do Site

Um objetivo adicional (e natural) do desenvolvimento do protótipo neste trabalho foi a realização do seu *deploy*. Em outras palavras, a sua colocação em produção e, portanto, disponível para uso pelas pessoas: seja para fins de teste ou fins de consumo, propriamente dito. No contexto deste projeto, isso pode ser traduzido em realizar a hospedagem da aplicação *web* em algum servidor dedicado a isso. Para cumprir esse objetivo foram utilizadas as tecnologias Ngrok e Heroku.

A primeira serviu para mostrar e, principalmente, testar o jogo no modo *multiplayer*, pois o uso dela é simples e rápido, como mostrado no capítulo anterior. Basicamente, essa ferramenta consegue tornar o processo mais rápido porque ela mais simula o *deploy*, do que o realiza de fato. Ou seja, ela transforma a máquina de desenvolvimento do jogo em um servidor de hospedagem temporário. Em geral, usar essa tecnologia não trouxe problemas impeditivos ao protótipo, principalmente que não tenham sido eventualmente identificados no desenvolvimento local, por exemplo. Portanto, ela serviu bem a este trabalho.

Figura 55 - Execução do Ngrok no terminal (geração de *link* de acesso para o *localhost*)



O Heroku é feito para permitir o *deploy* de várias aplicações back-end, como é o caso deste protótipo. O processo não é tão rápido quanto a simulação com o Ngrok, mas

também é relativamente simples. Uma vez instalado o Heroku no terminal e criada uma conta em seu *site*, simplificadamente, são necessários os seguintes passos para o *deploy*:

1. isolamento da aplicação *web* com Git, por exemplo;
2. criação de uma aplicação no Heroku; e
3. envio do repositório local, contendo a aplicação *web*, para o repositório remoto do Heroku.

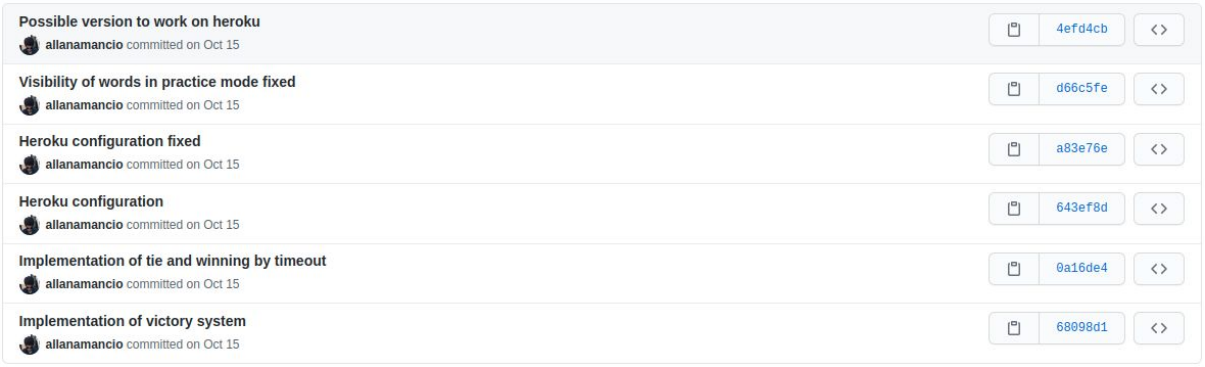
Figura 56 - Exemplo de *deploy* de aplicação *web* para o Heroku através do terminal

```
% heroku login
% git init
% heroku git:remote -a codingx-python
% git add.
% git commit -am "First python app"
% git push heroku master
```

Retirada do site https://dev.to/techparida/how-to-deploy-a-flask-app-on-heroku-heb

Nesse processo, em especial, houve um problema que impediu a hospedagem, em um primeiro momento. A dependência do WordStorming com a biblioteca pyenchant, baseada em uma biblioteca C. Como o Heroku não suporta a linguagem C, essa biblioteca teve que ser substituída na versão para o Heroku. Para esse fim, utilizou-se o módulo Python PyDictionary, que, como o nome sugere, é um dicionário de significados, sinônimos e afins. Em relação à biblioteca pyenchant, esse módulo é mais lento e, pelo menos, algumas palavras válidas não são aceitas por ele. De qualquer forma, isso resolveu o problema da aplicação poder ser hospedada no Heroku.

Figura 57 - Modificação pontual do WordStorming no GitHub para funcionar no Heroku



Por tempo indeterminado, o protótipo hospedado no Heroku pode ser visto através do seguinte link: wordstormingtcc.herokuapp.com. O *site* pode demorar um pouco para carregar pela primeira vez, pois na versão gratuita do Heroku, a aplicação adormece se ficar inativa por 30 minutos.

Em geral, todas as páginas do protótipo estão funcionais na versão do Heroku. No entanto, com a hospedagem, percebeu-se uma possível questão importante: o caráter assíncrono do Ajax. Em um ambiente local de desenvolvimento, essa característica fica menos nítida na aplicação, do que quando ela está em um ambiente de produção, por exemplo. Além disso, o Heroku deixa a aplicação um pouco mais lenta, de modo geral, o

que afeta o seu desempenho. Portanto, por conta desses fatores, o modo partida do jogo no *site* pode apresentar algumas instabilidades não apresentadas com o Ngrok, por exemplo, mas que acontecem com o Heroku.

6.9 Dívidas Técnicas

A escrita de um código elegante é uma prática de programação subjetiva que se sustenta na simplicidade e clareza do código, enquanto mantém a sua corretude e eficiência. Em geral, um código bem organizado, modularizado e confiável é um que foi escrito elegantemente. Tentou-se aqui escrever um código elegante o máximo possível, tanto pelos benefícios levantados aqui, quanto pelo hábito pessoal. No entanto, a partir de um certo momento mais avançado no protótipo, passou-se a se preocupar um pouco menos com a codificação elegante e passar a contrair algumas dívidas técnicas.

Dívidas técnicas, como sugere o nome, podem ser entendidas como “falhas intencionais” no código que, a curto prazo, resultam em uma rapidez de desenvolvimento do *software*. Esse conceito foi uma metáfora criada inicialmente, de maneira simples, por Ward Cunningham, com o objetivo de melhorar a entrega de *software*. Entretanto, ao longo dos anos, essa metáfora acabou se aprimorando e se tornou um objeto de estudo, propriamente dito.

Tentou-se ao máximo não contrair essas dívidas técnicas. Mas a escassez do tempo trouxe necessidades de priorizações e, portanto, aceitar certas coisas como as dívidas técnicas. Mas para que o impacto delas fossem minimizados, elas foram todas anotadas aqui, dessa maneira ficando conhecidas e passíveis de controle e gerenciamento. O desenvolvimento do projeto, apesar de formatado como um protótipo, sempre foi pensado de maneira completa e a longo prazo, portanto, as dívidas técnicas seguem muito nessa linha. No entanto, algumas coisas mais básicas também foram anotadas. Seguem nas próximas subseções, as dívidas técnicas anotadas mais significativas, divididas em *bugs*, refatorações e melhorias.

6.9.1 Bugs

Aqui são expostos os *bugs*, erros de funcionamento, por assim dizer, mais significativos do protótipo:

- Não tratamento de palavras inglesas com acentos (apesar de incomuns);
- Em momentos específicos, o jogo pode aceitar palavras repetidas se elas forem inseridas consecutivamente e com rapidez suficiente;
- Por conta do JavaScript, o tempo do cronômetro anda mais lento se o jogador mudar de aba no navegador *web*;
- Em momentos específicos, o jogo pode recusar palavras que, a priori, seriam válidas;
- Em momentos específicos, durante o intervalo, a pontuação acumulada do oponente pode vir subtraída de alguma pontuação faltante de palavra. Contudo, quando a próxima rodada é iniciada, a pontuação é consertada;
- Instabilidade no modo partida do WordStorming (em *deploy* no Heroku);

6.9.2 Refatorações

Aqui são expostas necessidades de refatorações, ou seja, melhorias da estrutura interna do código do WordStorming. Isso é importante para que, a longo prazo, seja possível a inclusão das outras funcionalidades apresentadas no jogo.

- Melhoria da arquitetura MVC adotada pelo jogo, principalmente com as pastas “views” e “controllers”. Há várias redundâncias de arquivos “__init__.py”, complicando desnecessariamente a lógica de importação dos módulos em Python.
- Simplificação e melhor organização do código, em termos de: nomes das variáveis e funções, modularização, nomeação dos arquivos e afins. Em especial, o arquivo “practice.js”, responsável por ambos modos do jogo no front-end, é um dos que mais precisam dessa refatoração.
- Padronizar a entrada de palavras no back-end do *site*. Por exemplo, convencionar em enviá-las sempre em letras minúsculas.

6.9.3 Melhorias

Por fim, aqui são expostas necessidades que não afetam o funcionamento do jogo e que provavelmente não provocam problemas a médio prazo. No entanto, concede benefícios relevantes, se resolvidas:

- Impedir o acesso manual a URL da partida do jogo (“'/multiplayer/2<player1>&<player2>”), pois apenas o jogo pode gerá-la adequadamente.
- Melhorar a visibilidade do site em sua versão *mobile*. Em especial, o *footer* (a parte mais inferior do *site*) fica fixo na página, impedindo assim a visualização de certos conteúdos.
- Melhorar o entendimento das pontuações na interface principal do WordStorming.
- Permitir que o jogador pause e retorne a uma partida já iniciada. Isso resolveria o problema do jogador sair sem querer da página e quando voltar, não ter que iniciar uma partida do zero.

7 AVALIAÇÃO

Este é um capítulo extra. Idealmente, planejava-se testar o jogo proposto nesta monografia na sua forma de produto intermediário e com o máximo de pessoas possível. Inclusive, a hospedagem do WordStorming em um servidor *online* tinha como objetivo fundamental justamente a realização de testes "em larga escala". Contudo, como mostrado ao longo do capítulo anterior, o desenvolvimento deste trabalho ficou limitado ao protótipo e a hospedagem apresentou algumas instabilidades. Tudo isso aliado a escassez de tempo, dificultou o teste do jogo com muitas pessoas.

No entanto, mesmo não sendo o foco principal deste trabalho, tentou-se improvisadamente angariar algumas avaliações sobre o protótipo aqui desenvolvido, a fim de pelo menos testar a hipótese implicitamente levantada nesta monografia: WordStorming ser um jogo de palavras que permita tanto o entretenimento, quanto o desenvolvimento vocabular em algum idioma (em especial, o inglês).

Isso foi feito, essencialmente, através de um formulário *online* composto de 11 perguntas. O formulário foi dividido em 4 partes: o perfil do avaliado; sua opinião sobre o protótipo; sua percepção de valor sobre o jogo; e eventuais comentários finais. As perguntas foram as seguintes:

1. Perfil do avaliado
    a. "Qual a sua idade?"
    b. "Possui interesse em idiomas?"
    c. "Sabe ou está aprendendo inglês?"
    d. "O que acha de jogos?"
2. Opinião sobre o protótipo
    a. "O que achou do site como um todo? (apresentação, informações, layout, usabilidade etc)"
    b. "Qual nota você daria para o jogo (protótipo)?"
    c. "Qual o motivo da sua nota? O que a faria ser diferente?"
    d. "Conhece algum amigo/a que jogaria esse jogo? Recomendaria a ele/a?"
3. Percepção de valor sobre o jogo
    a. "Você jogaria o jogo no seu tempo livre?"
    b. "Você enxerga algum valor no jogo? Se sim, qual?"
4. Comentários finais
    a. "Caso tenha mais algum comentário, escreva abaixo:"

O contato com o protótipo podia ser feito através do próprio formulário, que apresentava *links* para o *site* no Heroku e para um vídeo demonstrativo de 2 minutos, mostrando uma simulação do jogo no modo partida. No entanto, para aqueles que podiam, o jogo foi apresentado através de uma videoconferência, em substituição aos *links*. E dentre essas pessoas, algumas experimentaram uma partida rápida do jogo, através de um *link* gerado pelo Ngrok.

7.1 Resultados

Por conta da situação relatada, o teste do jogo só pôde ser feito com 9 pessoas. Em geral, pessoas próximas, como familiares, amigos e conhecidos desses amigos. Testou-se o jogo com o supervisor deste trabalho também. Os resultados obtidos foram organizados de acordo com as 4 partes mencionadas do formulário e são resumidos nas subseções seguintes.

7.1.1 Perfil

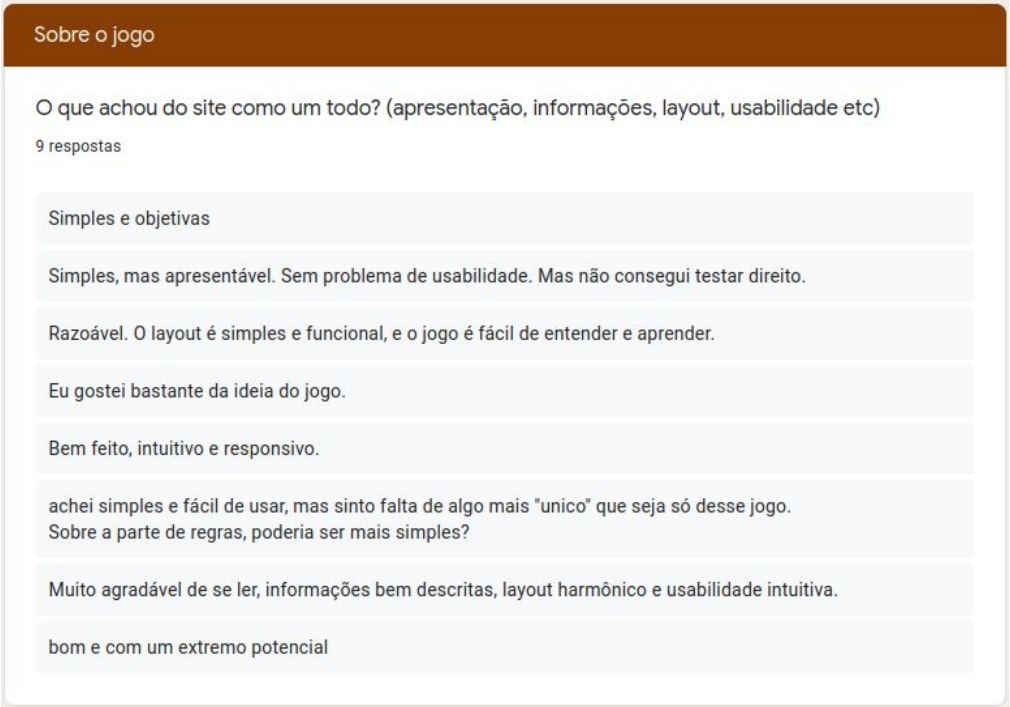
Em geral, as pessoas que deram *feedbacks* são pessoas jovens, que sabem ou estão aprendendo inglês, com interesse em idiomas e jogos. Especificando os números, tem-se o seguinte quadro:

- 44,4% são adultos (possuem de 25 a 65 anos) e 55,6% são jovens (possuem de 15 a 25 anos);
- 88,9% possuem interesse em idiomas, e 11,1%, mais ou menos;
- 88,89% sabem ou estão aprendendo inglês, e 11,1%, mais ou menos; e
- 66,7% gostam bastante de jogos; 22,2% gostam um pouco e 11,1% não costumam jogar.

7.1.2 Opinião

Em geral, as pessoas acharam o *site* do jogo simples e fácil de usar, como é possível verificar pela figura 58.

Figura 58 - Opinião dos 9 avaliandos sobre o *site* do jogo WordStorming.



E de acordo com as notas dos avaliandos, o jogo, em sua forma de protótipo, receberia uma nota média de 8,8, em uma escala de 0 a 10. Sendo que a menor nota foi 5 e a maior, 10. Em geral, as pessoas pareceram gostar do jogo, mas pontuaram que dariam uma nota diferente se o jogo fosse mais dinâmico e visual, por assim dizer. Seguem alguns dos trechos dos comentários feitos nesse sentido:

- “poderia haver algum tipo de animação durante a ‘batalha’”
- “incluir artes e sons no jogo para ficar mais com cara de ‘jogo’”
- “Eu faria com que o jogo fosse mais atrativo visualmente e tivesse uma identidade visual mais forte”
- “Gostaria de uma interface mais convidativa e lúdica.”

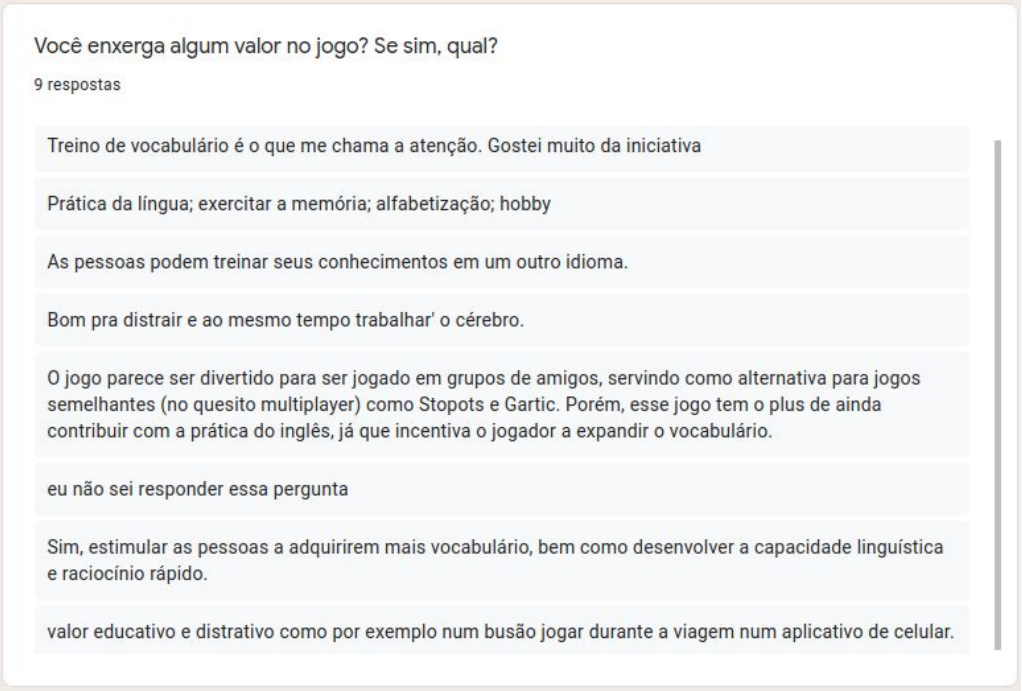
De todo modo, unanimemente, todos os avaliandos disseram que recomendariam o jogo para algum amigo ou conhecido.

7.1.3 Valor

A maioria dos avaliandos disseram que jogariam o jogo no seu tempo livre. Apenas 22,2%, ou não jogariam, ou jogariam apenas como um jogo rápido em uma “fila de banco”.

E em geral, as pessoas parecem enxergar valor no jogo, principalmente pelo seu caráter idiomático e distrativo.

Figura 59 - Percepção de valor dos 9 avaliandos sobre o jogo WordStorming



7.1.4 Extra

Encerrando este breve capítulo de avaliação, relata-se aqui os comentários adicionais feitos por alguns avaliandos, em geral, por via do formulário *online* ou pelas videoconferências.

Uma parte deles expressaram que gostaram do conceito, das regras e de alguns detalhes do jogo. E, com isso, enxergaram potencial nele, juntamente com o seu potencial educativo, principalmente a longo prazo.

Alguns espontaneamente sugeriram algumas funcionalidades, como:

- mecanismos para lidar com jogadores que ficam chutando palavras durante a rodada;
- visibilidade de qual é a quantidade máxima de palavras possíveis de serem formadas com uma dada sequência de letras aleatória; e
- “um mascote”, pois “ficaria legal”.

Uma pessoa disse que lembrou de um jogo que ela jogava no passado: Alphabear. Basicamente, um jogo ao estilo Scrabble.

Por fim, uma última pessoa disse que o ideal seria se o jogo fosse um aplicativo.

8 CONCLUSÃO

Todos os objetivos traçados nesta monografia, tanto os explícitos, quanto os pessoais, foram concretizados ao longo do desenvolvimento deste trabalho. Naturalmente, alguns com mais êxito do que outros. Em resumo, conseguiu-se implementar o básico do jogo proposto nesta monografia, juntamente com sua breve avaliação, e através disso, a aquisição de um amplo conhecimento computacional em desenvolvimento *web*. A competência em gerenciamento de projeto também foi um conhecimento adquirido aqui, em virtude da complexidade do jogo, principalmente em vista do conhecimento inicial que se tinha. Todos esses objetivos alcançados são melhor comentados ao longo dos próximos parágrafos, acompanhado de seus detalhes que o tornaram desafiantes e/ou que trouxeram aprendizados.

Para que pudesse ser desenvolvido computacionalmente, o jogo teve que ser apropriadamente formalizado nesta monografia. Em conjunto com a pesquisa dos jogos semelhantes a ele, isso permitiu tanto a verificação do nível de originalidade do jogo, quanto a composição das suas tarefas prioritárias para o desenvolvimento neste projeto.

Uma vez feito isso, seguiu-se uma sequência constante de aprendizados em todos os aspectos na construção do jogo, ou seja, desde a sua arquitetura e lógica interna à sua interface visual. Principalmente nesse último. Como o back-end do *site* foi feito em linguagem Python com o uso da framework Flask, ele foi comparativamente mais fácil de se desenvolver do que o front-end. A linguagem Python era uma linguagem previamente conhecida e já se tinha tido um contato, ainda que superficial, com as frameworks Flask e Django, por exemplo.

Portanto, o desenvolvimento front-end foi certamente uma das partes onde mais se aprendeu com esta monografia. Basicamente, teve-se que se fazer um curso básico completo de HTML, CSS e JavaScript para a implementação visual das páginas do site, além da dinamicidade, como a interação com o back-end, feita por Ajax, um dos maiores desafios aqui. Além disso, existe uma etapa anterior à aplicação dessas tecnologias, que é a definição do *design* das páginas, o que foi feito aqui através da elaboração dos *wireframes*.

Contudo, o desenvolvimento back-end também teve seus muitos desafios. Uma das razões de escolha do Flask, em detrimento de outras frameworks Python, foi o seu caráter minimalista, que de certa forma, exige mais do conhecimento do programador para ser utilizado. Consequentemente, entendeu-se melhor, por exemplo, sobre *cookies*, pelo uso das sessões em Flask; entendeu-se melhor o protocolo HTTP, uma vez que se lidava com ele constantemente através da interface do Flask, sempre que o *site* estava rodando em *localhost*; e entendeu-se melhor a arquitetura MVC, pois para que fosse usada aqui, ela teve que ser implementada manualmente.

Em paralelo à aquisição desses conhecimentos técnicos, diversos desafios enfrentados neste trabalho produziram outros conhecimentos subjacentes. Por exemplo, ao construir a sala de espera do WordStorming, percebeu-se, despretensiosamente, o porquê, provavelmente, serviços *online* de videoconferência possuem limite de usuários. Para além do uso excessivo de memória (o que provavelmente seria o “menor dos problemas”, em tempos atuais), existe a questão da velocidade de processamento das chamadas HTTP, como foi mostrado.

Na determinação de uma biblioteca ou dicionário de palavras para a validação das mesmas no WordStorming, percebeu-se que nenhuma parecia se encaixar adequadamente. Depois de algumas buscas, verificou-se que o Scrabble possuía um dicionário próprio. Apesar da sua semelhança com o WordStorming, não pareceu ser uma solução melhor do que a biblioteca pyenchant utilizada aqui. Com isso, percebeu-se que, provavelmente, a solução ideal é a construção de um dicionário próprio para o jogo.

E mais ao final do desenvolvimento do projeto, percebeu-se também que existe uma tecnologia provavelmente mais adequada para ser usada no lugar do Ajax em um contexto de jogo: WebSocket. Infelizmente, ela acabou não sendo usada aqui pela questão óbvia do tempo, mas também pela reestruturação completa do projeto que teria que ser feita. De qualquer forma, o seu uso, idealmente, traria uma melhor sincronização entre o front-end e o back-end do *site*, além de uma adequada refatoração do código.

O tempo cada vez mais escasso ao final do trabalho, tornou necessário o acúmulo de algumas dívidas técnicas, como foi mencionado, para que a entrega mínima deste projeto pudesse ser feita. Nesse sentido, uma correlação que percebeu-se aqui, sem um estudo aprofundado, é de que a negligência maior com a codificação elegante potencializa as dívidas técnicas. Portanto, entende-se aqui que ter escrito elegantemente a base do código-fonte do *site* WordStorming foi crucial para que não houvesse uma cadeia de *bugs* catastróficos, por exemplo.

Ao realizar uma retrospectiva deste trabalho, é possível constatar dois “erros” que foram cometidos. O primeiro é ter considerado a hospedagem algo final e extra neste trabalho, quando deveria ser inicial e primordial. Por mais que a hospedagem tenha sido algo adicional neste trabalho, seria melhor se ela tivesse sido feita desde o começo. O segundo é que este trabalho podia ter sido feito em dupla, o que permitiria mais facilmente o desenvolvimento da segunda parte do cronograma, por exemplo. Naturalmente, não se sabia essas coisas no começo, pela falta de conhecimento adequado em desenvolvimento *web* e por conta da noção real de complexidade deste trabalho ter sido compreendida ao longo dele. De todo modo, fica o aprendizado aqui.

Felizmente, no fim, conseguiu-se realizar o objetivo mínimo deste trabalho: o protótipo do WordStorming. E pelos poucos testes realizados com este protótipo, também conseguiu-se dados indicadores de uma possível potencialidade do jogo como entretenimento e prática idiomática, como almejado na introdução desta monografia.

8.1 Aprendizados Pessoais

Para além de conhecimentos técnicos, também foi importante a aquisição de algumas competências mais subjetivas para que o desenvolvimento deste trabalho se tornasse possível.

Ao longo deste ano, diversas emoções se fizeram presentes, tanto por conta do ano atípico, obrigando adaptações, quanto pelo período acadêmico em si. Saber se adaptar e gerir essas emoções foi um processo importante para que se tivesse em condições mínimas para realizar este trabalho.

Algumas vezes ao longo deste trabalho, principalmente em seu começo, teve-se um cuidado preciosista com a perfeição do mesmo. Apesar de entender isso ainda como algo importante, entendeu-se aqui também que o progresso é igualmente relevante e a perfeição é algo subjetivo. Portanto, assimilou-se melhor que o “feito é melhor do que perfeito”, nas suas devidas proporções.

E finalmente, o gerenciamento de tempo e a tomada de decisões ágeis foram as últimas competências necessárias e adquiridas, em algum grau, ao longo deste projeto. Sem elas, possivelmente o projeto não andaria pela falta de flexibilidade, o que, consequentemente, provocaria uma estagnação do jogo.

REFERÊNCIAS BIBLIOGRÁFICAS[1]

__init__.py! Para que serve este arquivo? Caderno de Laboratório. Disponível em: <https://cadernodelaboratorio.com.br/2018/10/01/__init__-py-para-que-serve-este-arquivo/>. Acesso em: 23 dez. 2020.

2-hour landing workshop. Le Wagon. Disponível em: <https://lewagon.github.io/landing/>. Acesso em: 23 dez. 2020.

25 of the Most Popular Python and Django Websites. Shuup, 2020. Disponível em: <https://www.shuup.com/articles/25-of-the-most-popular-python-and-django-websites/>. Acesso em: 23 dez. 2020.

Ajax (programação). Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Ajax_(programa%C3%A7%C3%A3o)>. Acesso em: 23 dez. 2020.

AJAX Introduction. W3Schools. Disponível em: <https://www.w3schools.com/js/js_ajax_intro.asp>. Acesso em: 23 dez. 2020.

ARMÉS, Diego. **A história do Scrabble, das regras às curiosidades.** GQ. Disponível em: <https://www.gqportugal.pt/scrabble-historia>. Acesso em: 23 dez. 2020.

Arquitetura REST: Saiba o que é e seus diferenciais. TOTVS, 2020. Disponível em: <https://www.totvs.com/blog/developers/rest/>. Acesso em: 23 dez. 2020.

BASSI, Giovanni. **Tell, don't ask (ou… fique longe das minhas propriedades).** Lambda3, 2009. Disponível em: <https://www.lambda3.com.br/2009/07/tell-dont-ask-ou-fique-longe-das-minhas-propriedades/>. Acesso em: 23 dez. 2020.

Bootstrap (framework front-end). Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Bootstrap_(framework_front-end)>. Acesso em: 23 dez. 2020.

Boggle. Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Boggle>. Acesso em: 23 dez. 2020.

Building a website with Python Flask. PythonHow. Disponível em: <https://pythonhow.com/building-a-website-with-python-flask/>. Acesso em: 23 dez. 2020.

Caça-palavras. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Ca%C3%A7a-palavras>. Acesso em: 23 dez. 2020.

Cairocoders. **jQuery AJAX Forms Submit with Python Flask.** Youtube, 2020. Disponível em: <https://www.youtube.com/watch?v=ttPuuw5V6JM>. Acesso em: 23 dez. 2020.

[1] Referências organizadas em ordem alfabética

CEZAR ROCHA, Bruno. **What the Flask? Pt-1 Introdução ao desenvolvimento web com Python.** PythonClub, 2014. Disponível em: <http://pythonclub.com.br/what-the-flask-pt-1-introducao-ao-desenvolvimento-web-com-python.html>. Acesso em: 23 dez. 2020.

Classes. documentação Python. Disponível em: <https://docs.python.org/pt-br/3/tutorial/classes.html>. Acesso em: 23 dez. 2020.

Codemy.com. **Push Flask Apps To Heroku for Webhosting - Python and Flask #11.** Youtube, 2020. Disponível em: <https://www.youtube.com/watch?v=Li0Abz-KT78>. Acesso em: 23 dez. 2020.

Código espaguete. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/C%C3%B3digo_espaguete>. Acesso em: 23 dez. 2020.

DE FÁTIMA SANTOS OLIVEIRA, KAMILA. **Conhecendo o Jinja2: um mecanismo para templates no Flask.** iMasters, 2019. Disponível em: <https://imasters.com.br/desenvolvimento/conhecendo-o-jinja2-um-mecanismo-para-templates-no-flask>. Acesso em: 23 dez. 2020.

DE JESUS PINA, Diogo. **Proposta - Dívida Técnica.** Disponível em: <https://bcc.ime.usp.br/tccs/2013/diogo/proposta.html>. Acesso em: 23 dez. 2020.

DE SOUZA, Ivan. **HTTP: entenda o que é, para que serve e como funciona.** Stage - Rock Content, 2019. Disponível em: <https://rockcontent.com/br/blog/http/>. Acesso em: 23 dez. 2020.

DIAS, Emílio. **4 Conceitos sobre REST que Qualquer Desenvolvedor Precisa Conhecer.** AlgaWorks, 2016. Disponível em: <https://blog.algaworks.com/4-conceitos-sobre-rest-que-qualquer-desenvolvedor-precisa-conhecer/>. Acesso em: 23 dez. 2020.

DIAS, Estevão. **Principais verbos HTTP utilizados em um serviço REST.** DevMedia, 2016. Disponível em: <https://www.devmedia.com.br/servicos-restful-verbos-http/37103>. Acesso em: 23 dez. 2020.

Dívida Técnica: A ameaça fantasma. X Semana da Computação. Disponível em: <https://bcc.ime.usp.br/semana-old/2018//palestra/divida-tecnica>. Acesso em: 23 dez. 2020.

Django (framework web). Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Django_(framework_web)>. Acesso em: 23 dez. 2020.

Entendendo o Three-way Handshake (Handshake de Três Vias). Blog do Smith, 2011. Disponível em: <https://andreysmith.wordpress.com/2011/01/02/three-way-handshake/>. Acesso em: 23 dez. 2020.

Favicon. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Favicon>. Acesso em: 23 dez. 2020.

FELIPPE, Gabriel. **Deploy de uma Aplicação Flask com Heroku.** DEV, 2019. Disponível em: <https://dev.to/theakira/deploy-de-uma-aplicacao-flask-com-heroku-5c0e>. Acesso em: 23 dez. 2020.
Frequency Table. Cornell University. Disponível em: <http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html>. Acesso em: 23 dez. 2020.

FINZI, Eduardo. **API, bibliotecas e Frameworks: entenda a diferença entre eles.** Cedro Technologies, 2016. Disponível em: <https://blog.cedrotech.com/api-bibliotecas-e-frameworks-entenda-diferenca-entre-eles/>. Acesso em: 23 dez. 2020.

FIRMINO, Júlia. **O que é kanban e como ele pode ajudar na organização do trabalho.** Runrun.it. Disponível em: <https://blog.runrun.it/o-que-e-kanban/>. Acesso em: 23 dez. 2020.

Flask (framework web). Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Flask_(framework_web)>. Acesso em: 23 dez. 2020.

Framework. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Framework>. Acesso em: 23 dez. 2020.

Front-end e back-end. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Front-end_e_back-end>. Acesso em: 23 dez. 2020.

Front-end web development. Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Front-end_web_development>. Acesso em: 23 dez. 2020.

Getting Started on Heroku with Python. Heroku Dev Center. Disponível em: <https://devcenter.heroku.com/articles/getting-started-with-python#set-up>. Acesso em: 23 dez. 2020.

GRINBERG, Miguel. **The Flask Mega-Tutorial Part I: Hello, World!** miguelgrinberg.com, 2017. Disponível em: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>. Acesso em: 23 dez. 2020.

HENRIQUE, João. **POO: o que é programação orientada a objetos?** Alura. Disponível em: <https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos>. Acesso em: 23 dez. 2020.

Here's What You Need To "Learn JavaScript Deeply". HTML.com. Disponível em: <https://html.com/javascript/>. Acesso em: 23 dez. 2020.

Heroku. Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Heroku>. Acesso em: 23 dez. 2020.

How to create a Python Package with __init__.py. Timothy Bramlett, 2019. Disponível em: <https://timothybramlett.com/How_to_create_a_Python_Package_with___init__py.html>. Acesso em: 23 dez. 2020.

How to play Web Boggle. WEBoggle. Disponível em: <https://weboggle.info/boggle/howToPlay.htm>. Acesso em: 23 dez. 2020.

HTML Anchors: Here's How To Create Links For Fast Navigation. HTML.com. Disponível em: <https://html.com/anchors-links/>. Acesso em: 23 dez. 2020.

HTML Web Forms Tutorial For Coding Beginners. HTML.com. Disponível em: <https://html.com/forms/>. Acesso em: 23 dez. 2020.

Interface de programação de aplicações. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Interface_de_programa%C3%A7%C3%A3o_de_aplica%C3%A7%C3%B5es>. Acesso em: 23 dez. 2020.

Intimidated By CSS? The Definitive Guide To Make Your Fear Disappear. HTML.com. Disponível em: <https://html.com/css/>. Acesso em: 23 dez. 2020.

JavaScript. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/JavaScript>. Acesso em: 23 dez. 2020.

Jinja (template engine). Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Jinja_(template_engine)>. Acesso em: 23 dez. 2020.

jQuery Introduction. W3Schools. Disponível em: <https://www.w3schools.com/jquery/jquery_intro.asp>. Acesso em: 23 dez. 2020.

LEITE, Gabriel. **O que é JSON?** Alura, 2020. Disponível em: <https://www.alura.com.br/artigos/o-que-e-json>. Acesso em: 23 dez. 2020.

Leonardo. **API, Framework ou Biblioteca.** Gigasystems Soluções Inteligentes, 2015. Disponível em: <https://www.gigasystems.com.br/artigo/94/api-framework-ou-biblioteca>. Acesso em: 23 dez. 2020.

Letter Frequencies in the English Language. University of Notre Dame. Disponível em: <https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html>. Último acesso em: 23 dez. 2020.

Letter frequency. Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Letter_frequency>. Acesso em: 23 dez. 2020.

Lidando com arquivos - Aprendendo desenvolvimento web. MDN. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Aprender/Getting_started_with_the_web/lidando_com_arquivos>. Acesso em: 23 dez. 2020.

Model-View-Controller (MVC) Explained - With Legos. Real Python. Disponível em: <https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos/>. Acesso em: 23 dez. 2020.

MORAES DE CARVALHO, Rogério. **O que é o AJAX.** DevMedia, 2007. Disponível em: <https://www.devmedia.com.br/o-que-e-o-ajax/6702>. Acesso em: 23 dez. 2020.

MORAES, Frank. **HTML For Beginners The Easy Way: Start Learning HTML & CSS Today.** HTML.com. Disponível em: <https://html.com/>. Acesso em: 23 dez. 2020.

MVC as a subset of client-server architecture? Stack Exchange, 2016. Disponível em: <https://softwareengineering.stackexchange.com/questions/314341/mvc-as-a-subset-of-client-server-architecture>. Acesso em: 23 dez. 2020.

O que é back-end e qual seu papel na programação? TOTVS, 2020. Disponível em: <https://www.totvs.com/blog/developers/back-end/>. Acesso em: 23 dez. 2020.

O que é codificação elegante? Stack Overflow, 2020. Disponível em: <https://pt.stackoverflow.com/questions/173216/o-que-%C3%A9-codifica%C3%A7%C3%A3o-elegante>. Acesso em: 23 dez. 2020.

O que é dívida técnica? Stack Overflow, 2019. Disponível em: <https://pt.stackoverflow.com/questions/32420/o-que-%C3%A9-d%C3%ADvida-t%C3%A9cnica>. Acesso em: 23 dez. 2020.

O que é um Framework? Tableless. Disponível em: <https://tableless.github.io/iniciantes/manual/js/o-que-framework.html>. Acesso em: 23 dez. 2020.

Organizing your project. Explore Flask. Disponível em: <https://exploreflask.com/en/latest/organizing.html>. Acesso em: 23 dez. 2020.

OROZCO, Franccesco. **Change Flask Root Folder for Templates and Static Files.** coding(dose), 2018. Disponível em: <https://codingdose.info/2018/05/12/change-flask-root-folder/>. Acesso em: 23 dez. 2020.

Os métodos HTTP: quais são e para que servem? Gabs Ferreira. Disponível em: <http://gabsferreira.com/os-metodos-http-e-a-diferenca-entre-eles/>. Acesso em: 23 dez. 2020.

Palavras Cruzadas. Ludijogos. Disponível em: <https://www.ludijogos.com/multiplayer/palavras-cruzadas/>. Acesso em: 23 dez. 2020.

Palavras cruzadas. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Palavras_cruzadas>. Acesso em: 23 dez. 2020.

Parole (jogo). Wikipedia. Disponível em: <https://pt.wikipedia.org/wiki/Parole_(jogo)>. Acesso em: 23 dez. 2020.

PETER LAUBE, Klaus. **Entendendo os Cookies e Sessões.** Klaus Laube, 2012. Disponível em: <https://klauslaube.com.br/2012/04/05/entendendo-os-cookies-e-sessoes.html>. Acesso em: 23 dez. 2020.

Pretty Printed. **Submit AJAX Forms with jQuery and Flask.** Youtube, 2016. Disponível em: <https://www.youtube.com/watch?v=IZWtHsM3Y5A>. Acesso em: 23 dez. 2020.

Programação concorrente. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_concorrente>. Acesso em: 23 dez. 2020.

Python and Flask Tutorial in Visual Studio Code. Visual Studio Code, 2019. Disponível em: <https://code.visualstudio.com/docs/python/tutorial-flask>. Acesso em: 23 dez. 2020.

Python: conheça mais sobre a linguagem que vem influenciando o mundo. Tecnisys, 2018. Disponível em: <http://www.tecnisys.com.br/noticias/2018/python-conheca-mais-sobre-a-linguagem-que-vem-influenciando-o-mundo>. Acesso em: 23 dez. 2020.

Python. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Python>. Acesso em: 23 dez. 2020.

Qual a idade certa para alfabetização? Instituto NeuroSaber, 2019. Disponível em: <https://institutoneurosaber.com.br/qual-a-idade-certa-para-alfabetizacao/>. Acesso em: 23 dez. 2002.

Refatoração. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Refatora%C3%A7%C3%A3o>. Acesso em: 23 dez. 2020.

REST. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/REST>. Acesso em: 23 dez. 2020.

REST e HTTP são a mesma coisa? Stack Overflow, 2018. Disponível em: <https://pt.stackoverflow.com/questions/95836/rest-e-http-s%C3%A3o-a-mesma-coisa>. Acesso em: 23 dez. 2020.

RICARDO TEIXEIRA, José. **jQuery Tutorial: Introdução à biblioteca JavaScript jQuery.** DevMedia, 2013. Disponível em: <https://www.devmedia.com.br/jquery-tutorial/27299>. Acesos em: 23 dez. 2020.

Scrabble. Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Scrabble>. Acesso em: 23 dez. 2020.

SeattleDataGuy. **Building Your First Website With Flask - Part 1**. Medium, 2019. Disponível em: <https://medium.com/better-programming/building-your-first-website-with-flask-part-1-903a8b44e806>. Acesso em: 23 dez. 2020.

SeattleDataGuy. **Building Your First Website With Flask - Part 1**. Medium, 2019. Disponível em: <https://medium.com/better-programming/building-your-first-website-with-flask-part-2-6324721be2ae>. Acesso em: 23 dez. 2020.

Sopa de Letras. Racha Cuca. Disponível em: <https://rachacuca.com.br/palavras/sopa/>. Acesso em: 23 dez. 2020.

Tips and Tricks - Highlighting Active Menu Items. Jinja Documentation. Disponível em: <https://jinja.palletsprojects.com/en/2.10.x/tricks/#highlighting-active-menu-items>. Acesso em: 23 dez. 2020.

Visual Studio Code. Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Visual_Studio_Code>. Acesso em: 23 dez. 2020.

WEISSMANN, Henrique Lobo. **Receita - expondo seu localhost ao mundo com Ngrok.** itexto, 2020. Disponível em: <https://www.itexto.com.br/site/receita-expondo-seu-localhost-ao-mundo-com-ngrok/>. Acesso em: 23 dez. 2020.

What exactly is Werkzeug? Stack Overflow, 2019. Disponível em: <https://stackoverflow.com/questions/37004983/what-exactly-is-werkzeug>. Acesso em: 23 dez. 2020.

What is an API? MuleSoft. Disponível em: <https://www.mulesoft.com/resources/api/what-is-an-api>. Acesso em: 23 dez. 2020.

What is JSON? W3Schools. Disponível em: <https://www.w3schools.com/whatis/whatis_json.asp>. Acesso em: 23 dez. 2020.

ZANETTE, Alysson. **Framework x Biblioteca x API. Entenda as diferenças!** Becode, 2016. Disponível em: <https://becode.com.br/framework-biblioteca-api-entenda-as-diferencas/>. Acesso em: 23 dez. 2020.

Zynga. **Palavras com Amigos**. Google Play. Disponível em: <https://play.google.com/store/apps/details?id=com.zynga.wwf2.free&hl=pt_BR>. Acesso em: 23 dez. 2020.