Faculdade de Tecnologia de Americana

Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais

DESENVOLVIMENTO DE ARQUITETURAS CSS APLICADO EM JOGOS WEB

FELIPE WALFLAN DE SOUZA

Americana, SP

2018

Faculdade de Tecnologia de Americana

Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais

DESENVOLVIMENTO DE ARQUITETURAS CSS APLICADO EM JOGOS WEB

FELIPE WALFLAN DE SOUZA

felipewalflan@gmail.com

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais, sob a orientação do Prof. Jonas Bodê

Área: Jogos Digitais

Americana, SP

2018

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

S715d SOUZA, Felipe Walflan de

Desenvolvimento de arquitetura CSS aplicado em jogos web. / Felipe Walflan de Souza. – Americana, 2018.

61f.

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

Orientador: Prof. Esp. Jonas Bodê

1 WEB – rede de computadores 2. Desenvolvimento de software I. BODÊ, Jonas II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU 681.519

Felipe Walflan de Souza

Desenvolvimento de Arquitetura CSS Aplicado em Jogos WEB

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Jogos Digitais, pelo CEETEPS/Faculdade de Tecnologia – Fatec/ Americana.
Área de concentração: Desenvolvimento WEB

Americana, 27 de junho de 2018.

Banca Examinadora:

Jonas Bodê (Presidente)
Especialista
Fatec Americana

Mariana Godoy Vazquez (Membro)
Doutora
Fatec Americana

Benedito Aparecido Cruz (Membro)
Mestre
Fatec Americana

BANCA EXAMINADORA

Prof. Jonas Bodê

Prof. Dr. Mariana Godoy Vazquez

Prof. Me. Benedito Aparecido Cruz

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer minha família, meu pai Walflan, minha mãe Rosângela e as minhas irmãs, Jessica e Heloá, que com muito carinho me apoiaram nessa jornada. Por sempre me incentivarem e fazer acreditar que tudo seria possível.

Também agradeço ao coordenador do meu curso Cleberson Forte por toda atenção, dedicação e colaboração em todas as etapas que tive nessa graduação tanto dentro como fora da sala de aula. Agradeço também muito ao meu orientador Jonas Bode, por todas as conversas e incentivo no desenrolar do trabalho. A tudo sou muito grato.

Aos meus amigos e colegas de trabalho, meus agradecimentos pelo apoio e companheirismo no andamento dessa jornada.

A todos, meu obrigado!

DEDICATÓRIA

Dedico este trabalho a minha família, amigos e colegas de trabalho, que foram minha fonte de motivação.

RESUMO

O texto traz inicialmente uma contextualização da importância de arquiteturas no mundo do desenvolvimento junto com uma apresentação do mundo dos jogos digitais se aprofundando no mundo do desenvolvimento web, consequentemente nos *games* desenvolvidos para a Internet, explicando então suas características e dando foco totalmente no estilo em cascata, o CSS destes jogos. Também propõe a utilização de técnicas de arquiteturas CSS visando uma melhoria geral da implementação, organização e manutenção do seu jogo. Com uma abordagem ampla, citando não só uma arquitetura, mas várias, e ao mesmo tempo simplista, o estudo contido neste trabalho traz três casos de uso que exemplificam a utilização dessas arquiteturas, aplicando, portanto, de diversas maneiras o estilo em cascata no seu desenvolvimento. Este documento também especifica cada característica de cada arquitetura CSS utilizada, abordando seus objetivos e benefícios.

Palavras Chave: CSS, *games*, jogos *web,* arquiteturas CSS, desenvolvimento *web*.

ABSTRACT

The text initially brings a contextualization of the importance of architectures in the world of development along with a presentation of the world of digital games deepening in the world of the web development, consequently in the games developed for the Internet, explaining then their characteristics and giving totally focus in the cascading style, the CSS of these games. It also proposes the use of CSS architectures techniques aiming at an overall improvement in the implementation, organization and maintenance of your game. With a broad approach, citing not only an architecture, but several, and at the same time simplistic, the study contained in this work brings three use cases that exemplify the use of these architectures, thus applying in various ways the cascading style in its development. This document also specifies each feature of each CSS architecture used, addressing its goals and benefits.

Keywords: CSS, games, web games, CSS architectures, web development.

SUMÁRIO

GLOSSÁRIO

***Software**:* São sistemas ou aplicativos de um computador que possibilita o processamento de dados e informações.

***Hardware**:* São equipamentos físicos que possibilitam o uso de um computador.

E-Sports: É um termo criado e utilizado para as competições organizadas de *games* eletrônicos, o esporte eletrônico.

Tecnologia da Informação (TI): A Tecnologia da Informação é o estudo da junção de tudo que se refere à serviços, soluções e atividades providas por um procedimento computacional que necessita de algum uso de informação.

Gameboard: Tabuleiro ou local onde algum jogo será realizado.

MVC: É uma sigla utilizada para representar uma dinâmica de desenvolvimento onde o código é dividido em 3 camadas muito simples, onde cada camada tem sua responsabilidade. Todas as regras de negócio ficam na camada de controle, chamada de '***Controller'***, já os dados que estão sendo utilizados na camada de controle ficam na camada de modelo, a '***Model'***, e por último, a '***View'***, camada responsável pela visualização.

***Inline**:* É um termo utilizado para uma das possibilidades de aplicação do CSS no HTML, este termo significa que o CSS estará na dentro do conteúdo HTML que está utilizando aquele CSS.

Tag* (HTML):* São todos os elementos que compõe uma página web, portanto sempre estarão dentro de um arquivo HTML entre as tag's principais <html> e </html>, início e fim do arquivo. E como pode se notar, por convenção a abertura de uma tag se dá como mostra o exemplo 'tag_exemplo_titulo', assim <tag_exemplo_titulo> e o fechamento da mesma forma, mas com uma barra antes do nome, assim </tag_exemplo_titulo>.

Atributos (HTML): São dados que se configuram dentro de uma tag para que a mesma aja de uma certa maneira.

Game Design Document: Um documento de design de games, também chamado de GDD, é criado por toda a equipe de desenvolvimento de um jogo para organizar em um só lugar tudo que será realizado dentro do jogo. Assim organizando todo o processo de desenvolvimento.

Role-playing game: Também chamado pela sigla RPG, é um estilo de onde os jogadores interpretam papéis de personagens e desenvolvem com os mesmos, diferentes narrativas onde todos têm a possibilidade de muda-la.

Massively multiplayer online: São games eletrônicos preparado para de suportar gigantescas quantidades de jogadores simultaneamente e em um mesmo ambiente, todos online através da internet.

Puzzle: É um estilo de game onde a proposta, o desafio do jogo se baseia na resolução de enigmas, e ou quebra-cabeças.

Tower Defense: É um estilo de game onde o objetivo está em defender, algo ou alguém, utilizando estratégias contra o adversário em questão.

Runner: É um estilo de game de plataforma, portanto faz com que seu game seja jogado fixo em uma tela, onde o jogador tende a se locomover de uma direção para a outra da tela, escapando dos desafios propostos no game com certas tomadas de decisão.

Turn Based Strategy: É um estilo de game de estratégia onde a ordem de jogar é baseada em turnos assim como os jogos de tabuleiros.

Servidores de hospedagem*:* Local onde os arquivos de uma página web ficam para que seu conteúdo seja acessado na Internet.

Browsers*:* Navegadores de Internet, softwares utilizados para o acesso à Internet.

Orientado a objetivo: O termo utilizado em linguagens de desenvolvimento orientação a objetos significa arquitetar seu código de modo onde qualquer estrutura do mundo real possa ser representada por um objeto, que incorporará a si um conjunto de operações que possibilita alterações de seus próprios dados.

Frameworks: Um software desenvolvido que serve como ferramenta que facilita o desenvolvimento de uma linguagem específica, desde a melhora de performance, economia de código e concatenação de regras. Sendo assim, quando se diz em utilização de frameworks há uma necessária uma adaptação do próprio para o código do seu projeto.

Bootstrap: Um framework de desenvolvimento web, desenvolvido em HTML, CSS e Javascript, para o desenvolvimento de interfaces para sites, aplicações e serviços web de forma muito mais fácil e intuitivo.

GitHub: É uma plataforma de serviço de hospedagem de códigos-fonte de projetos.

LISTA DE FIGURAS E DE TABELAS

1. INTRODUÇÃO

O desenvolvimento de um software, que neste documento será um sinônimo de desenvolvimento de jogos digitais, pode ser muito custoso e demorado, e apresentando em cenários caóticos por diversos motivos como solicitações divergentes, equipe de desenvolvimento (funcionários), tempo de entrega, entre outros. Mas uma das maiores causas desses cenários conturbados é a falta de organização e de uma arquitetura pré-definida pensada para aquele *software*, *game* ou sistema. Não importa o que a equipe de análise e desenvolvimento pretende fazer, se não houver uma organização prévia será praticamente impossível entregar um sistema de qualidade aos clientes e, ou usuários. Ao desenvolver um software, a equipe envolvida precisa fazer uma série de escolhas que delimitam a forma do produto final. Nesse processo, a falta de definição de critérios para tomadas de decisão pode impactar profundamente na qualidade do software e na sua aceitação dentro do contexto para o qual foi proposto.

A maneira em que os códigos são arquitetados e escritos colaboram e estão estreitamente ligados aos fatores de qualidade, tanto na organização citada acima quanto na manutenção após uma entrega, afinal desenvolvedores do mercado de trabalho raramente desenvolvem projetos individuais, e mesmo se ocorresse tal fato, ainda assim seria necessário fazer a manutenção quando necessária, sendo assim demonstrando a importância deste estudo em qualquer caso que utilize as ferramentas abordadas neste documento, principalmente o CSS.

No decorrer do tempo, mesmo sendo em um curto de espaço tempo, as comunidades de desenvolvimento foram moldando as suas maneiras de desenvolvimento para que todos esses problemas citados fossem sendo minimizados e com isso as maneiras de desenvolvimento de códigos, organizações de projetos, até mesmo as plataformas (Conhecidas como IDE’s, do inglês *Integrated Development Environment*) foram se polindo, justapondo e integrando com as práticas anteriores, criando assim um termo que hoje já é globalmente utilizado e difundido na comunidade de desenvolvimento pelo mundo, que são as técnicas, ou arquiteturas de desenvolvimento.

Contudo essas técnicas de desenvolvimento podem ter diversas definições e interpretações, até mesmo do modo em que são nomeadas. Neste documento se

resumirá como um conjunto de regras estipuladas pelo mercado de trabalho junto com uma convenção de desenvolvedores visando uma melhor *performance* em qualquer ótica do projeto. Óticas de fugacidade, rendimento, funcional e operacional.

Óticas de fugacidade do projeto: A visão da velocidade em que andou o projeto, o quanto algum impedimento prejudicou o prazo, ou os prazos, e se o projeto foi entregue de acordo com o previsto.

Ótica de rendimento do projeto: O lucro real do projeto, normalmente a conta é feita com o valor cobrado do cliente pelo projeto menos a quantidade das horas pagas aos desenvolvedores, já em jogos digitais se o jogo for para a própria empresa fica um pouco mais difícil pois o valor que a empresa receberá pelo jogo entrar na empresa de forma mais lenta, fica mais difícil calcular o lucro de um projeto a longo prazo.

Ótica funcional do projeto: A maneira que o cliente aceita aquele serviço ou produto.

Ótica operacional do projeto: A maneira que toda equipe de trabalho se adapta ao ambiente de desenvolvimento.

Essas óticas aqui definidas se baseiam na definição abaixo de Abel (2003) onde determina que escopos, prazos e riscos são fatores importantíssimos quando se diz sobre gestão de projetos, portanto, este estudo procura apresentar técnicas que refletiram na mensuração do escopo e prazo, também minimizando os riscos como se refere a própria citação.

> “[...] práticas bem-definidas e sistemáticas de gestão de projetos visam a garantir que objetivos claramente estabelecidos entre contratante e contratado sejam atingidos em prazos [...], tempo e dinheiro são recursos escassos e como tal devem ser administrados. Neste sentido, escopo, prazo, recursos, riscos e comunicação são fatores críticos cuja administração ao longo de um projeto, por práticas rigorosas e consistentes, permitem garantir o alcance de objetivos, minimizando riscos, e antecipando problemas.”. Abel, 2003.

2. JOGOS DIGITAIS

2.1. MERCADO

O mercado de *games* já faz um tempo que tem receitas gigantescas, consumidores fiéis e muitos telespectadores, citando E-Sports.

Este mercado já é sólido, lucrativo e até mesmo essencial para a indústria de outros segmentos. Hoje já se consolidou como uma ramificação da tecnologia da informação e surpreendentemente continua em ascensão diante dos dias atuais.

> “O desenvolvimento de jogos digitais movimenta US$ 50 bilhões e cresce, em média, 20% ao ano, chegando a superar a indústria cinematográfica. O que antes era tratado como uma brincadeira de criança, hoje é motivo de disputa pelas grandes indústrias do entretenimento. O Brasil mostra-se no caminho certo e é apontado como um dos grandes mercados do século XXI“. Ávila, 2009.

Esse crescimento e essa importância dos *games* dentro da área tecnológica faz com que tudo o aprendido e utilizado nos outros mercados de TI, mercados mais antigos ou maiores, vão se adaptando para o mundo dos jogos digitais agregando exponencialmente ao mercado, as suas tecnologias, enfim, a todos os ramos da ciência dos *games* (*Game Design, Game Concept e Game Development*).

> "O jogo é uma atividade ou ocupação voluntária, exercida dentro de certos e determinados limites de tempo e de espaço, segundo regras livremente consentidas, mas absolutamente obrigatórias, dotado de um fim em si mesmo, acompanhado de um sentimento de tensão e de alegria e de uma consciência de ser diferente da 'vida cotidiana'. " . Huizinga, 1938.

Esta definição é um dos termos mais aceitas e estudadas quando se pensa em uma definição de “jogo”. Existem obviamente pela subjetividade da palavra diferentes outras definições, e que justamente por essa subjetividade tornam várias destas corretas, mesmo está citada que é de um livro de 1938, antes mesmo da criação de uma tecnologia computacional que permitiu a criação dos jogos somente digitais. Ela aqui foi utilizada, além de dar uma explicação de um conceito, para ilustrar principalmente com o início de sua definição, ***“O jogo é uma atividade ou***

***ocupação voluntária*”** à necessidade humana em se entreter com jogos, independentemente do modo ou época, contudo nos dias de hoje onde a tecnologia está mais do que presente, se reflete no quanto este mercado de jogos digitais cresce. Quanto mais tecnologia mais acesso aos jogos desenvolvidos, fazendo com que o incrível e poderoso mercado de desenvolvimento de jogos se desenvolva mais rápido do que qualquer outro mercado industrial no mundo. Por isso mesmo existindo desde à década de 70 e 80, com apenas cinco ou quatro décadas movimenta tanto dinheiro como citado em Ávila (2009), “O desenvolvimento de jogos digitais movimenta US$ 50 bilhões e cresce, em média, 20% ao ano”.

2.2. ESTILOS E TIPOS DE *GAMES*

E por esse cenário colossal onde o mercado de jogos se encontra hoje, com o passar dos anos o desenvolvimento de *games* teve que se adaptar às demandas de jogadores, consumidores e também às oportunidades de negócio como qualquer outro setor industrial. Toda a história dos games fez com que os jogos digitais tenham hoje a divisão em diversos gêneros e subgêneros, que apesar de definidos estão em constante mudança e que podem sim se misturar dependendo do jogo. Exemplo dessa mudança e mistura é que conforme alguns autores como **Battaiola (2000)** os gêneros podem ser divididos em: ação, esportes, estratégia, simuladores, aventura, infantil, passatempo, RPG, luta e *war games*. Já outros como, **Lindsay Grace (2005)** dividiria em ação, aventura, *puzzle* (enigma ou quebra-cabeça), *role playing* (interpretação), simulação e estratégia. Portanto, abaixo estará alguns desses gêneros com suas supostas definições, já que sempre pode-se discutir por uma ou outra parte de sua definição, e, junto consigo uma divisão em subgêneros.

> “Agrupar os jogos em caixas com rótulos não é uma tarefa simples. É necessário algum tipo de subjetividade, uma vez que é notório que jogos podem navegar entres seus gêneros. ”. Silva, 2009.

2.2.1. GÊNERO AÇÃO

O game de ação maravilha seus consumidores por trazer consigo histórias complexas, com tramas e imersões incríveis, portanto é de fato um dos gêneros que mais cativa e desafia seus usuários. Possui os seguintes subgêneros:

- Aventura: Esse subgênero se define pela desenvoltura na resolução de quebra-cabeças e objetivos do(s) personagem(s) normalmente baseados em uma longa trama onde o usuário avança de acordo com sua performance no que lhe é proposto pelo jogo. Exemplo de jogo: Tomb Raider.
- Arcade: Esse subgênero se caracteriza pelo tempo curto dos jogos e por não conter histórias complexas, a dinâmica é bem simples, o jogador progride de fase após chegar a uma certa pontuação ou após um determinado tempo, e com isso a cada fase o jogo vai ficando mais difícil. Exemplo de jogo: Pacman.
- Luta: Esse subgênero talvez seja o mais simples de se explicar, os personagens se enfrentam em arenas com o objetivo simples de um derrotar todos os outros e sair vencedor. Exemplo de jogo: Street Fighter.
- Furtiva: Esse subgênero é aquele que ao contrário de propor combate entre o personagem controlado e NPC (*non-player character*), ele propõe a esquivar-se dos desafios como a maneira de chegar nos objetivos do jogo.

Como já dito, esses gêneros e subgêneros podem sim se misturar tendo diferentes, mas ainda assim, corretas definições. Também, não existe nenhuma regra indestrutível que extingue a junção um certo subgênero a outro, tanto no gênero descrito, o de ação, como nos próximos que será citado.

2.2.2. GÊNERO SHOOTER

O *game shooter* é o mais simples em sua definição, consiste em atirar projéteis em outros personagens em busca do seu objetivo dentro do jogo. Dentro desta clara regra o gênero é dividido em 2 subgêneros:

- Primeira pessoa: Esse subgênero o jogador possui exatamente o mesmo ponto de vista do personagem. Exemplo de jogo: Counter Strike.
- Terceira pessoa: Esse subgênero se diferencia do de primeira pessoa somente por propiciar ao jogador um cenário mais abrangente e não a

visão exatamente do personagem controlado. Exemplo de jogo: *Grand Theft Auto.*

2.2.3. GÊNERO AVENTURA

Os jogos de aventura são caracterizados pela necessidade de uma coleta de itens no cenário proposto, junto à uma destreza para ganhar desafios e também à resolução de quebra-cabeças, podendo ou não conter uma melhora gradativa das habilidades do personagem:

- RPG (*Role-playing game*): Nesse subgênero o jogador interpreta um personagem à sua escolha e a partir de então, de acordo com as tomadas de decisões do jogador a narrativa do personagem vai se desenhando. Podem ser do tipo colaborativo, social e competitivo entre os jogadores, um tipo não excluindo o outro. Exemplo de jogo: *Star Wars: Knights of the Old Republic.*
- MMORPG (*Massively multiplayer online role-playing game*): Esse gênero segue o mesmo princípio de jogo que o RPG com a grande diferença na quantidade de jogadores em um mesmo mundo, ou cenário. As suas proporções são bem maiores que a de um RPG convencional fazendo com que seus jogos estejam em um outro subgênero. Exemplo de jogo: *World WarCraft*.
- Sobrevivência/Terror: Esse gênero faz com que o personagem ou tenha de trabalhar com o gerenciamento de recursos, ou resolva desafios em cenários de terror ambos focando na sobrevivência do personagem. Exemplo de jogo: *Resident Evil*.

2.2.4. GÊNERO CONSTRUÇÃO E GERENCIAMENTO

Os jogos de construção e gerenciamento são tão simples de explicar que acabam não tendo a sua divisão em subgêneros. Os games desse gênero consiste em construir de acordo com os recursos que o jogo possibilita ter e expandir de forma que ganhe mais recursos. Exemplo de jogo: *Zoo Tycoon.*

2.2.5. GÊNERO DE SIMULAÇÃO

Os jogos de simulação são facilmente o maior gênero quando se fala em quantidade de subgêneros. Não por ter um estudo quantificando esses subgêneros um por um, já que não teria como fazer isso pensando que um jogo pode se classificar com diferentes gêneros ou subgêneros, mas sim pela possibilidade de se colocar dentro desse grupo de games, os de simulação, subgêneros que poderiam facilmente ser considerado um gênero por si só e consigo ter outros subgêneros como:

- Música e Dança: Esse subgênero permite que o jogador simula por meio do modo de controle instrumentos e sons, ou movimentos de dança. Exemplo de jogo: *Guitar Hero*.
- Esportes: Esse facilmente entra nos seletos subgêneros que poderia ser considerado um gênero, independente da preferência do autor pois essa definição de fato é apenas uma questão de preferência, os jogos de esportes simulam os milhares de esportes existentes no mundo tendo então o desafio de simular com exatidão as minúcias que cada esporte contém. Exemplo de jogo: NBA 2K18.
- Quebra-cabeças: Esse subgênero simula digitalmente jogos chamados de analógicos, que são jogados sem a necessidade de um *software* e *hardware*. Exemplo de jogos *War* e *Xadrez*.

2.2.6. GÊNERO ESTRATÉGIA.

Os jogos de estratégia são também um gênero muito abrangente pois qualquer game que tenha uma etapa onde se tem que pensar e planejar poderia ser classificado nele. Entretanto existem alguns tipos de jogos de estratégia muito conhecidos. São eles:

- Defesa de torre (*Tower Defense*): Esse subgênero tem em seus jogos o conceito simples de guardar uma “torre”, como o próprio nome sugere. Esta “torre” pode ser qualquer lugar, dependendo do tema do jogo em disputa, e até mesmo uma personagem, o objetivo é simples, é traçar uma estratégia de proteção. Exemplo de jogo: Bloons Tower Defense.

- Jogo de turno (*Turn Based Strategy*): Esse subgênero dos games de estratégia são os mais lentos pois cada jogador joga por rodadas. Outra característica é ele não necessitar de destreza e habilidades de controle fazendo com que este estilo foque totalmente na parte cognitiva do jogador, quem melhor montar sua estratégia e também melhor ler as estratégias dos adversários potencialmente tem grande chance de ganhar.
- Jogo em tempo real (Real Time Strategy): Esse subgênero é famosamente conhecido como o estilo dos “4 x’s”, pois os objetivos dos jogos desse subgênero são inteiramente ligados à extração, exploração, expansão e extermínio. Exemplo do jogo: *Warcraft III*.

2.3. DESENVOLVIMENTO DE *GAMES*

Os games de fato possuem diversas possibilidades de gêneros e subgêneros conforme já citado, mas já para o desenvolvimento desse grande leque de tipos dentro dos *games* utilizam-se ferramentas e plataformas que já não se diversificam tanto no mercado. Neste capítulo, portanto terá seu foco em descrever as 3 plataformas mais famosas (*console*, *mobile* e *web*) empregues no desenvolvimento de um jogo digital completo, contextualizando um pouco do mercado da mesma e linguagens de programação que possibilitam o seu desenvolvimento e deixando de lado outras plataformas como *tablets* e *gameboys*.

2.3.1. DESENVOLVIMENTO *MOBILE*.

É a mais nova plataforma presente no mercado. Dizer que esta plataforma, como qualquer outra, está em ascensão seria um pleonasmo, já que no capítulo referente ao mercado de games digitais citou-se que toda a indústria cresce incrível e rapidamente. Mas utilizar o termo ascensão, que se define por se elevar, comparando ao nível de crescimento das outras plataformas (*console, web, tablets, gameboys*) pode se tornar coerente pelo fato da plataforma *mobile* ser a que mais cresce. Alguns fatores são extremamente determinantes para tal elevação. O primeiro se dá pelo fato da quantidade de aparelhos *smartphones* que, em média por pessoa não param de crescer no Brasil e no mundo. Quando se refere a jogos

mobile automaticamente se designa à jogos de smartphones, esta indústria em constante crescimento influência e alavanca diretamente a propagação dos jogos mobiles.

Outro fator de influência no crescimento *mobile* é o aparecimento do termo “Jogos *Indie*”, que significa “Jogos Independentes” e são chamados assim por não serem inicialmente vinculados à uma empresa ou incubadora, esses jogos são desenvolvidos por pequenas equipes e até mesmo individualmente não contando portanto com um grande investimento financeiro. Então, como um aumento significativo de smartphones juntos com a possibilidade de desenvolvimento sem um investimento alto a oferta e procura cresceram juntas, um lado ajudando e influenciando o outro.

Para o seu desenvolvimento, assim como para os demais tipos de plataformas de desenvolvimento de games, possuem diversas linguagens de programação, e algumas das mais conhecidas são Java, Object-C, React e HTML5.

2.3.2. DESENVOLVIMENTO PARA CONSOLE.

Para o desenvolvimento de jogos para console comparando com o desenvolvimento de outra plataforma, não é, em uma questão de arquitetura e linguagens de programação mais difíceis de se desenvolver. Mas pelo fato dos jogos de console serem desenvolvidos por grandes empresas de *games,* com isso, um investimento muito maior, um desenvolvedor “Indie”, por exemplo, não conseguiria disputar o mercado com uma dessas empresas tornando impossível uma competição justa do mercado.

Hoje quem tem um console investe um valor tanto no próprio console tanto no jogo que se utiliza naquele hardware, isso faz com que o desenvolvimento para ele tenha um investimento de pessoal e de tecnologia, portanto a qualidade do software (game) tem que ser bem maior, sendo assim as empresas gigantescas de games no mundo todo conseguem centralizar o desenvolvimento de jogos para console e desenvolvedores de games individuais e até empresas de pequeno e médio porte não conseguem medir força, entrar e se estabelecer neste mercado, pois do que será mais cobrado, nunca chegará à games que tiveram um investimento milionário.

O desenvolvimento para console pode se utilizar diversas linguagens de programação, mas duas das mais famosas são C# e Javascript. Exemplificando, a Unity 3D que é uma IDE de desenvolvimento de games muito famosa no mundo utiliza essas duas linguagens como principais de seus softwares. Importante ressaltar que para jogos de console existe uma cobrança natural a mais em qualidade de imagem acabam sendo bastantes usado alguns sistemas 3D e de efeitos gráficos, tonificando as modelagens e efeitos. Exemplo de softwares são, Autodesk Maya, Blender e a própria Unity 3D que também é utilizada para programação.

2.3.3. DESENVOLVIMENTO PARA BROWSERS

O mercado é considerado recente, mas hoje já é a plataforma mais utilizada por usuários. Isso ocorre em todo mundo não só no Brasil devido às transformações que o acesso à Internet trouxe para nossa sociedade.

> “Todos já sabemos das enormes transformações que a Internet vem causando nas comunicações, no trabalho, no comércio, no entretenimento. Essa rede de computadores descentralizada, quase anárquica, é um verdadeiro fenômeno mundial. O Brasil não está alheio a essa "revolução". Pelo contrário, estamos entre os dez países que mais utilizam a Internet.“. Sepin, 2000.

Além da disseminação da Internet os jogos web são muito fáceis e seguros para seus usuários, neles não há uma necessidade de instalação e os próprios *servidores de hospedagem* e *browsers* fazem a segurança do tráfego pela rede diminuindo o risco ao usuário na sua ação de jogar.

Para o seu desenvolvimento, diferente das outras plataformas, há uma necessidade de utilizar algumas linguagens fixas para a sua programação. Para o desenvolvimento *mobile* e o desenvolvimento para console existem linguagens onde se pode iniciar seu projeto sem problemas, mas para a criação de um jogo web o desenvolvedor necessita ter um conhecimento de HTML para aí sim desenvolver um jogo. Junto com o HTML vem o CSS e Javascript para a estilização do jogo e criação de eventos, como o HTML é uma linguagem de marcação é impossível criar um jogo só com o seu conhecimento, afinal a linguagem de marcação não cria ações que são necessárias para o desenvolvimento de um jogo digital. Contudo o

CSS e Javascript de fato não são obrigatórios, sendo que um necessita do outro para a programação. No decorrer deste documento será mostrado e exemplificado como isso pode acontecer.

3. DESENVOLVIMENTO WEB

Os famosos jogos para Internet como são popularmente conhecidos, são aqueles que em sua execução utilizam as plataformas de navegadores, tema que já foi abordado no capítulo **Desenvolvimento de Games**. Consequentemente, faz com que sua grande diferença com os executados nas plataformas console seja que seus códigos, imagens e áudios não ficam armazenados em um *hardware*, sua arquitetura de complicação é cliente-servidor, com isso conforme a necessidade do jogo de acessar uma imagem, um arquivo de estilo ou outro arquivo que seja, ele “aguarda” o envio dos mesmos pela rede do servidor. Uma outra característica muito positiva dentro dos games web é a possibilidade de se acessar até mesmo de outras plataformas, como *smartphones* e consoles. Se tais plataformas possibilitam a utilização de navegadores e esses navegadores tiverem acesso à Internet, será possível acessar um jogo. Importante lembrar que por ser um desenvolvimento em camadas e ter a necessidade do tráfego pela rede, utilizando outras plataformas e aumentando o caminho do pacote que está em tráfego, o jogo pode sim perder *performance,* mas também existe casos que o jogo após um tráfego se instala localmente no próprio navegador, possibilitando então, após seu carregamento o uso sem necessidade da internet.

3.1. BACKEND E FRONTEND

O processo de desenvolvimento web pode se dividir em duas ou três categorias: *backend* e *frontend*, com o *design* vindo como um terceiro tópico dependendo da interpretação do desenvolvedor que está se referindo ao assunto. No caso das técnicas apresentadas no decorrer do documento não implicará diretamente na escolha ou mudança do design em si. Sendo assim, não será abordado como um tópico diferente. Para ajudar na ilustração da divisão de *frontend* para *backend* a imagem abaixo.

Figura 1 - Divisão ilustrativa do frontend e backend no desenvolvimento web.

Fonte: Pereira (2017).

Como pode se ver na imagem, o *backend* e *frontend* se dividem pelo fato do que se pode ver ou não ver, simples assim. Os navegadores que seria o mar nessa metáfora, faz a divisão, navegadores que são softwares que todos os usuários utilizam para acessar conteúdo na internet (Google Chrome, Firefox, Safari estão entre os mais famosos e utilizados). Mas, voltando para a explicação da divisão entre o *backend* e *frontend*, a camada *frontend* é a que só vai interpretar os códigos de HTML, Javascript e CSS, já todo o código que executa atrás do browser independente da funcionalidade é considerado código *backend*.

Como o desenvolvedor *frontend* é responsável por “dar vida” à interface, ele trabalha com a parte da aplicação que interage com o usuário e é necessário que ele tenha um nível de preocupação com a experiência do usuário. Em outras palavras, o desenvolvedor responsável pelo *frontend* trabalha com foco total na camada de visualização do usuário, e pensando nisso e aplicando a jogos é imprescindível que tenha que ser feito um excelente trabalho.

Figura 2 - Divisão das linguagens interpretadas pelo browser.

Fonte: Grimmett (2017).

Agora subdividindo o *frontend* para um estudo mais aprofundado, os profissionais do meio têm que dominar basicamente, como já dito anteriormente linguagens interpretadas pelo *browser*, que é onde vai executar o game. São elas, HTML (linguagem de marcação), CSS (linguagem de estilo) e o JavaScript, também chamado de JS (linguagem de script/programação) como podem ver na imagem acima.

Como o Javascript é uma linguagem de programação e orientado à objeto, portanto muito mais complexa que o HTML e o CSS tanto no seu entendimento, quanto na sua escrita, no decorrer do documento não vamos nos aprofundar e explicar nos casos de uso deste documento. Este estudo empenha-se para melhorar a *performance* e arquitetura do mais simples dentro do seu projeto, o CSS junto ao HTML, e que por ser simples, a maioria dos desenvolvedores não lhe dão a devida importância e consequentemente conseguem se complicar no desenvolvimento de seu projeto.

Nos tópicos a seguir serão demonstradas as já listadas principais tecnologias para o desenvolvimento de jogos web, tendo seu foco totalmente na linguagem e na arquitetura de estilos, o CSS.

3.2. A LINGUAGEM CSS

CSS é a abreviação para os termos em inglês *Cascading Style Sheet*, traduzido para o português como folhas de estilo em cascata. Uma das definições existentes mais precisa e simples para folha de estilo encontra-se em Silva (2007), e diz: “Folha de estilo em cascata é um mecanismo simples para adicionar estilos (por exemplo: fontes, cores, espaçamentos) aos documentos web”.

Esses estilos em cascata para qualquer desenvolvimento web é vital. Permite personalizar tudo contido na exibição da página web e vincular por meio de suas regras de utilização a manutenção em lote desses arquivos HTML. Suporta, portanto, com uma incrível facilidade o controle de recursos de conteúdo. Parafraseando para uma diferente perspectiva, os estilos CSS podem personalizar *tag’s*, componente HTML presente no **<Body>** (corpo) do Arquivo, e que sejam facilmente replicadas por todo o projeto.

As CSS têm por finalidades desenvolver à (X)HTML o propósito inicial da linguagem. A HTML foi criada para ser uma linguagem exclusivamente de marcação e estruturação de conteúdo. Isto significa que, segundo seus idealizadores, não cabe à HTML fornecer informação ao agente usuário sobre a apresentação dos elementos. Por exemplo: cores de fontes, tamanhos de textos, posicionamentos e todos o aspecto visual de um documento não devem ser função do HTML. “Cabem à CSS todas as funções de apresentação de um documento, e esta é sua finalidade maior”, Silva, 2007.

Concluímos que CSS foi desenvolvido, portanto para libertar o HTML para sua função de criação, e de fato cumpriu, e cumpre até hoje o seu papel, mas, como a tecnologia incessável hoje que fez não só os estilos em cascatas, mas como todas as tecnologias se aperfeiçoarem, existe hoje muito mais funcionalidades e utilização para o CSS além só da limitação de cores, textos e espaçamentos.

3.2.1. APLICANDO CSS

Na prática, a sua aplicação é tão simples quanto o seu significado. O CSS funciona em ordem de leitura, aplicando os estilos configurados de acordo com a sequência que se vai lendo o arquivo, ou arquivos. E no arquivo HTML é onde se aplica o CSS, e seu modo de chamada se dá por três formas: *inline*, internas e externas.

A ordem de leitura do CSS pelo HTML dá-se primeiramente pela leitura do CSS externo, depois o interno e depois o *inline*, sobrescrevendo as regras criadas para os seus elementos de acordo com sua leitura, portanto se uma mesma regra está definida em um CSS externo e em um CSS *inline* ele automaticamente utilizará à do CSS *inline* desprezando as que ele leu primeiro. Exemplificando, no código representado na Figura 3 está sendo definida as cores *red* (vermelho) com o CSS *inline* e *green* com o CSS interno ambos para a *tag* <h1>, neste exemplo a cor da *tag* quando for exibida será *red* (vermelho) justamente de acordo as regras de sua leitura. É relevante saber que quando no mesmo modo de chamado do CSS existir duas definições para a mesma *tag*, o último a ser lido prevalecerá sobre o outro.

Figura 3 - Ilustração dos 3 modos de chamada de um CSS na página HTML.

```
<html>
<head>

<link rel="stylesheet" href="/diretorio/css/animate.css">
<style>
    h1 {
        color:green;
    }
</style>

</head>
<body>

    <h1 style="color:red;">Cabeçalho teste</h1>

</body>
</html>
```

Fonte: Autor.

3.2.2. SINTAXE E REGRAS DE APLICAÇÃO

O código CSS utiliza se de uma simples regra na hora de sua escrita. A regra consiste em uma forma própria e muito simples que se define por seletor, propriedade e valor escritos da seguinte forma: **seletor {propriedade: valor;}**.

> “Seletor é o alvo da regra CSS. Genericamente, é a tag do elemento da marcação ou uma entidade capaz de definir com precisão em qual lugar da marcação será aplicada a regra CSS. [...] Propriedade define qual será a característica do elemento, alvo do seletor, a ser estilizada. [...] Valor é a quantificação ou a qualificação da propriedade.”. SILVA, 2007.

Como pode-se notar de acordo com Silva (2007), o **seletor** define quem será formatado, a **propriedade** define o que será formatado e o **valor** define como será formatado aquela **propriedade**. Como mostra a Figura 4 (Ilustração de um estilo CSS interno com duas declarações de propriedades com seus respectivos valores para somente um seletor, o de um cabeçalho [h1]) e Figura 5 (Ilustração do exemplo acima da Figura 4 com a nomenclatura na divisão de cada elemento do CSS, o seletor, as propriedades, os valores e a declaração), a cada declaração é necessário a utilização de “;” (ponto e vírgula) para separar e diferenciar as definições de **propriedades** do mesmo **seletor**.

Figura 4 - Ilustração de um estilo CSS.

```
<style>
    h1  {   /*declaração1*/
            color:green;

            /*declaração 2*/
            font-size: 10px;
        }
</style>

</head>
```

Fonte: Autor.

Figura 5 - Ilustração do exemplo acima da Figura 4.

Fonte: W3Schools.

Abaixo uma tabela com o CSS e a exata definição de acordo com os seletores, propriedades e valores atribuídos.

CSS do projeto	Definição do acontecimento

div {
color: #000;
font-size: 12px;
background-color: #fff;
background-image:url(bgdicas.jpg);
}
	O texto aparecerá em cor preta, em tamanho de 12 pixels, com a cor do fundo será branca e a imagem bgdicas.jpg será usada como fundo.

h1 {
color: #33ff33;
font-family: arial; verdana;
font-size: 11px;
}
	Os parágrafos definidos como títulos (cabeçalhos) terão a cor verde limão e aparecerão em Arial, ou, se essa fonte não existir, em Verdana, e em tamanho de 11 pixels.

h2 {
color: #ffffff;
font-family: arial; verdana;
font-size: 10pt;
}
	Os parágrafos definidos como títulos (cabeçalhos) terão a cor branca e aparecerão em Arial, ou, se essa fonte não existir, em Verdana, em negrito e em tamanho de 10 pixels.

a:link { color:#00ffff; text-decoration: underline } a:hover { color:#77ff99; text-decoration: none } a:visited { color:#ccffff; text-decoration: underline }	Os links terão a cor #00ffff e serão sublinhados, quando clicados exibirão a cor #77ff99 e não serão sublinhados. Já os links que já foram visitados terão a cor #ccffff e serão sublinhados.

Fonte: Autor.

Entretanto no CSS, a configuração de estilos não se dá somente pela ordem de leitura, além dela existe a “pontuação de especificidade” e a “herança das propriedades”, onde realmente todo o efeito cascata do CSS é controlado.

3.2.3. HERANÇA DAS PROPRIEDADES

Existe dentro da programação, principalmente na programação orientada a objeto, um conceito chamado herança onde consiste em um objeto, chamado de filho, herdar propriedades de outro objeto, chamado de pai. No CSS o conceito é o mesmo onde o objeto filho sempre vai ser a *tag* html dentro de uma outra *tag*, em outras palavras se uma *tag* tiver outra *tag* dentro dela, ela será pai daquela ou daquelas que estão dentro de si. A herança de propriedades então se dá quando a *tag* filho possuir as mesmas propriedades da *tag* pai herdando assim os valores definidos no pai.

Figura 6 - Ilustração de CSS interno para exemplificar o conceito de herança de propriedades.

```
34 <style>
35     body {color: green;}
36     h1 {color: blue;}
37     p {}
38 </style>
39 </head>
40 <body>
41     <h1>Cabeçalho Teste</h1>
42     <p>Parágrafo Teste</p>
43 </body>
```

Fonte: Autor.

No exemplo da Figura 6 acima, na *tag* <body> que é a *tag* que engloba todas as outras, sendo assim o pai de todas as outras tags do código HTML está definindo uma cor verde para seus elementos, consequentemente as tags <h1> e <p> herdam está definição por conterem as mesmas propriedades (*"color"*) que está definida no pai. Novamente em outras palavras, mesmo não tendo uma definição em seus seletores da propriedade color as tags filhas acabam assumindo a cor de sua tag pai e em sua exibição na página web ficaria com a cor verde.

3.2.4. PONTUAÇÃO DE ESPECIFICIDADE

Por último, quando a ordem de leitura e, ou a herança não conseguem definir um conflito de exibição na página web é usada a especificidade. Um divisor definitivo de qual estilo aquele seletor vai se configurar de acordo com seu peso atribuído. Peso de especificidade, onde o peso, ou número mais alto é usado para sobrescrever todos os outros estilos no final.

Tabela 1 – Ilustração de pontuação de especificidade.

	Inline	ID	Classe	Elemento	Especificidade
<p style ...>	1	0	0	0	1000
h1, h2	0	0	0	2	2
h1.primeiro	0	0	1	1	11
#contentor	0	1	0	0	100
#contentor h2	0	1	0	1	101
#contentor #lateral p	0	2	0	1	201
h1 p	0	0	0	2	2
.imagem	0	0	1	0	10

Fonte: Autor.

Como mostra a tabela acima de acordo com a configuração do elemento vai se alterando o número de especificidade, determinando assim, a última regra de aplicação do CSS. Quanto maior a especificidade maior vai ser o poder do elemento em sobrescrever outras regras CSS.

3.3. IMPORTÂNCIA DO CSS PARA JOGOS WEB

A criação de aplicações para web, assim como os outros tipos de aplicações existentes na TI, consiste em uma tarefa dividida em diversos processos flexíveis, podendo ser independentes entre si, mas que, uma hora ou outra, se compõem em alguma fase do projeto. Para os games, e não softwares desktops e sistemas web convencionais, podem ter diversas ramificações além desses processos flexíveis como o game design, a arte gráfica, a sonoplastia, a interação, a inteligência artificial, a física, o *level* design, os motores de jogos, portanto, como citado anteriormente que os jogos web são dependentes das 3 linguagens, várias dessas ramificações ficam vinculadas à uma dessas linguagens, sendo divididas ou não entre elas tornando cada linguagem (HTML, CSS, Javascript) para o desenvolvimento de um jogo mais sobrecarregada. O CSS então se torna automaticamente responsável por várias dessas ramificações, umas parcialmente, como a interação e game *design*, e outras integralmente como as de artes gráficas.

4. ARQUITETURAS CSS

No desenvolvimento de um game, ou na criação de qualquer outro sistema, uma das tarefas onde não se tem para onde fugir é o desenvolvimento da arquitetura deste projeto. Pensar no objetivo, na concepção do game, nos atributos de qualidade, nos requisitos, restrições e tecnologias a ser utilizadas nem sempre é um trabalho fácil por isso há uma necessidade em se arquitetar o projeto antes obviamente sair desenvolvimento. E de acordo com SEI (2005) é nesta etapa onde parâmetros essenciais e totalmente ligados a qualidade e manutenibilidade do nosso projeto/game serão construídos. “Os atributos de qualidade dos sistemas de software são estreitamente relacionados às suas arquiteturas”.

“A arquitetura de software é dita como tendo principal papel na determinação da qualidade e da manutenibilidade do software”. Esses dois fatores citados por Wasserman (1996), a “qualidade” e a “manutenibilidade”, estão totalmente ligados ao custo deste projeto consequentemente refletindo diretamente no lucro do mesmo. Tendo em vista que o desenvolvimento não seja feito para um lazer, e sim como uma fonte de renda ou elevação comercial seja da marca do jogo ou da marca da empresa, o desenvolvimento de uma arquitetura para o game ou sistema é imprescindível.

Como abordado nos capítulos anteriores, o CSS tem sim um valor substancial e repleto de responsabilidades nos projetos que o detém, e por não ser uma linguagem de difícil entendimento (por não ser algo lógico) ou não conter regras de negócio justamente por não ser uma linguagem de desenvolvimento acaba não sendo arquitetado como deveria ter sido, assim quando se junta os assuntos, de fato o CSS não é arquitetado de modo como deveria ser. Por isso a seguir serão apresentados alguns modelos de arquitetura CSS (as mais famosas, não todas) e depois seus casos de uso, tanto na escrita de um GDD (*Game Design Document*) quanto em aplicações práticas em jogos digitais, exemplos que se aplicados e pensados de maneira correta refletiram muito positivamente produto final que foi desenvolvido e subsequentemente trará uma manutenção mas descomplicada.

> “Muito se fala em padrões de projeto para Javascript, mas esquecem que o CSS também é uma das partes fundamentais do projeto e uma

> das mais complexas. Diversos navegadores e versões e também dispositivos.”. Simões, 2015.

4.1. SMACSS

O SMACSS é uma das arquiteturas mais utilizadas tanto por usuários comuns em seus projetos pequenos e individuais, quanto para projetos maiores que contém já algum tipo de *framework* CSS. Por seu modelo ser dividido em 5 partes muito claras e intuitivas, ***base, layout, module, state*** **e** ***theme*** a manutenção do código organizado em cima dessas 5 divisões se torna algo fácil.

- **Base**

A base tem como regra aplicar estilos em seletores de elementos sem o uso de Classes ou ID’s. Ela também contém pseudo-classes e filhos usando apenas seletores de elementos. Tudo isso irá definir um estilo comum para o projeto web.

Uma maneira de configuração da base dentro da arquitetura SMACSS é a utilização dos chamados “CSS *Reset*” do famoso escritor sobre desenvolvimento web, Eric A. Meyer. Ele escreveu conceituados livros sobre o CSS, como “CSS: *The Definitive Guide*: Eric A. Meyer, *“More Eric Meyer on CSS: Home*” e “*CSS Pocket Reference: Visual Presentation for the Web*” e criou o “CSS ***Reset”*** para já auxiliar no início de um projeto web as configurações básicas do desenvolvimento.

Figura 7. Exemplo de CSS Reset.

```css
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
        margin: 0;
        padding: 0;
        border: 0;
        font-size: 100%;
        font: inherit;
        vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
        display: block;
}
body {
        line-height: 1;
}
ol, ul {
        list-style: none;
}
blockquote, q {
        quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
        content: '';
        content: none;
}
table {
        border-collapse: collapse;
        border-spacing: 0;
}
```

Fonte: Meyer.

Outro modelo famoso de “CSS Reset” utilizado para ajudar a implementar a base da sua arquitetura é o “normalize.css”, este modelo é utilizado por gigantescas empresas tanto do ramo de desenvolvimento como de diversos outros ramos, por exemplo: Twitter, Soundcloud, Guardian, e até o próprio Github e Bootstrap.

- **Layout**

No layout definimos os estilos de tudo que não se repete dentro de uma página. O header, footer e sidebar são exemplos claros de elementos que serão “estilizados” nesta parte da arquitetura por não serem reutilizáveis dentro de um mesmo arquivo.

Exemplo de configuração de layout:

```
#header {  background: #fff;  border-bottom: 1px solid #ccc; }.

#footer {  background: #999;  border-top: 1px solid #ccc; }.
```

- **Module**

Já o *module*, o módulo, é incumbido de configurar todas as partes de código que podem ser reutilizados, em outras palavras é responsável por todos os componentes da página como botões, menus, imagens e blocos de texto consequentemente, nesta etapa é onde a maior parte do seu código se localizará.

```
.button {        display: inline-block;
                 box-shadow:none;
                 background:none;
                 border: none;
                 text-transform: uppercase;
                 padding: 10px;
background: #CFCFCF;
font-size: 14px; }
h1, h2, h3 {  border-bottom:2px solid green; color:#CFCFCF;  }
```

- **State**

O *State*, o estado em português, obviamente determina o estilo do estado em que o elemento se encontra, o tipos são “is-active” se o elemento está ativo, “is-selected” se o elemento está selecionado, “is-visible” se o elemento está visível, “is-hidden” se o elemento está oculto e “is-collapsed” se o elemento está selecionado. Caso o desenvolvedor desejar criar um novo estado para seus elementos, nesta camada onde ele escreverá seu código css.

- **Theme**

O tema é a parte menos utilizada na arquitetura, e em muitas aplicações não utilizadas. Seu objetivo é definir uma regra específica dentro de todo o projeto. Um tema de natal que obriga todos os botões padrões que era da cor “gray”, cinza à ficarem “red”, vermelhos é um exemplo do caso.

Exemplo:

/*Botão comum da página*/

.button {background:gray; color:white;}

/*Botão tema de natal*/

.button-natal {background:red; color:white;}

Por fim, para sua arquitetura SMACSS estar funcionando de maneira ideal, há também uma necessidade além das 5 divisões. A chamada dessas divisões, como foi explicado no capítulo CSS, também é determinante no resultado final do seus CSS. Esta arquitetura, como qualquer outra, além do cálculo de especificidade, determina seus estilos por sequência de leitura, sendo assim, para o SMACSS estar de perfeita organização em seu projeto há a necessidade de estar sendo chamada no HTML de forma sequencial. Primeiro a base, depois o *layout*, módulo, estado e por fim o tema, que nem sempre será presente.

Figura 8. Ilustração das chamadas CSS SMACSS.

```
<head>
<link rel="stylesheet" href="/diretorio/css/base.css">
<link rel="stylesheet" href="/diretorio/css/layout.css">
<link rel="stylesheet" href="/diretorio/css/module.css">
<link rel="stylesheet" href="/diretorio/css/state.css">
<!--<link rel="stylesheet" href="/diretorio/css/theme-natal.css">-->
<!--<link rel="stylesheet" href="/diretorio/css/theme-1.css">-->
</head>
```

Fonte: Autor.

Como pode ver na Figura 8, o “theme-natal.css” não está sendo utilizado atualmente mesmo já estando implementado, no exemplo ele está comentado e será utilizado apenas na época de natal.

Outro fator que colabora para o SMACSS ser uma das arquiteturas favoritas dos usuários é pelo fato dela pertencer na implementação de uns dos *frameworks* CSS mais conhecidos e utilizados em todo o mundo, especificamente o Bootstrap, o mais conhecido e utilizado em todo o mundo de fato segundo site *Github*, portal onde profissionais do meio de desenvolvimento utilizam para salvar projetos particulares e, ou compartilhar códigos de todos os tipos com a própria comunidade de desenvolvimento.

> “Hoje estima-se que cerca de 7 milhões de sites utilizem o Bootstrap como framework front-end. Entre suas vantagens, podemos citar a documentação farta e a comunidade muito ativa, a infinidade de componentes que podem ser facilmente chamados em suas aplicações, além da boa base de padrões estéticos, que permitem criar páginas belas e funcionais.”. SCUDERO, 2016.

4.2. OOCSS

A arquitetura OOCSS, CSS Orientado à Objeto consiste em utilizar conceitos das linguagens orientadas a objetos como o princípio da responsabilidade única e a separação de preocupações, para deixar o CSS mais gerenciável.

O princípio de responsabilidade única, que é baseado no primeiro princípio do SOLID criado por Robert C. Martin, nada mais é que uma regra (ou uma boa prática) onde salienta a preocupação de que uma “classe” na orientação à objetos tenha a responsabilidade de desempenhar somente o papel dela e de forma muito eficiente. Já a separação de preocupações do termo (*Separation of Concerns*, com a sigla SoC) consiste em determinar para uma regra específica uma preocupação específica, dividindo assim, a responsabilidade do controle do sistema que utiliza aquela regra. Exemplificando a separação de preocupações, podemos vê-lo seus conceitos em diversos sistemas e arquiteturas, o MVC mesmo, onde separa a arquitetura do código em 3 camadas, camada *Controller* faz o intermédio entre ambas, a *Model* trata o processamento de dados (regras de negócio) e a *View* exibe os dados para o usuário, cada uma com sua responsabilidade específica.

Voltando para o CSS esses princípios do OOCSS fazem com que qualquer objeto que pode se repetir dentro de uma página web possa ter variações de visuais gerando assim código separados para o mesmo objeto e com definições diferentes.

Exemplos:

```
.botao { width: 50px; padding:15px; }

.botao-normal { color: white; background: gray; border: 1px solid gray; }

.botao-perigo { color: white; background: red; border: 1px solid red; }
```

Em outras palavras, a classe “botao” fica responsável pela estrutura, aquilo que o usuário não percebe (altura, largura e margens), já as classes “botao-normal” e “botao-perigo” ficam responsáveis pela pele, por definir aquilo que o usuário percebe (fontes, cores e sombras). Definindo se, portanto o princípio da OOCSS, conforme Sullivan (2013).

> “Two Main Principles of OOCSS. Separate structure and skin: This means to define repeating visual features (like background and border styles) as separate “skins” that you can mix-and-match with your various objects to achieve a large amount of visual variety without much code. [...] SEPARATE container and content: Essentially, this means “rarely use location-dependent styles”. An object should look the same no matter where you put it. So instead of styling a specific <h2> with .myObject h2 {...}, create and apply a class that describes the <h2> in question, like <h2 class="category">.” Sullivan, 2013.[1]

[1] Dois Princípios Principais do OOCSS. Estrutura e aparência separadas: isso significa definir recursos visuais repetitivos (como estilos de plano de fundo e borda) como “capas” separadas que você pode misturar e combinar com seus vários objetos para obter uma grande variedade visual sem muito código. Veja o objeto do módulo e suas skins. [...] Contexto e conteúdo separados: Essencialmente, isso significa "raramente usar estilos dependentes de localização". Um objeto deve parecer o mesmo, não importa onde você o coloque. Então, ao invés de estilizar um <h2> específico com ‘.myObject’ h2 {...}, crie e aplique uma classe que descreva o <h2> em questão, como <h2 class = "category">.

Como pode perceber o segundo princípio da OOCSS também é bem simples. Basta que na escrita do CSS o estilo da classe referenciado não seja alterado de acordo com sua localização.

Exemplo de como não utilizar:

```
#sidebar .list { margin: 3px; }
#sidebar .list .list-header { font-size: 16px; color: red; }
#sidebar .list .list-body { font-size: 12px; color: #FFF; background: red; }
```

Exemplo de como utilizar:

```
.list { margin: 3px; }
.list-header { font-size: 16px; color: red; }
.list-body { font-size: 12px; color: #FFF; background: red; }
```

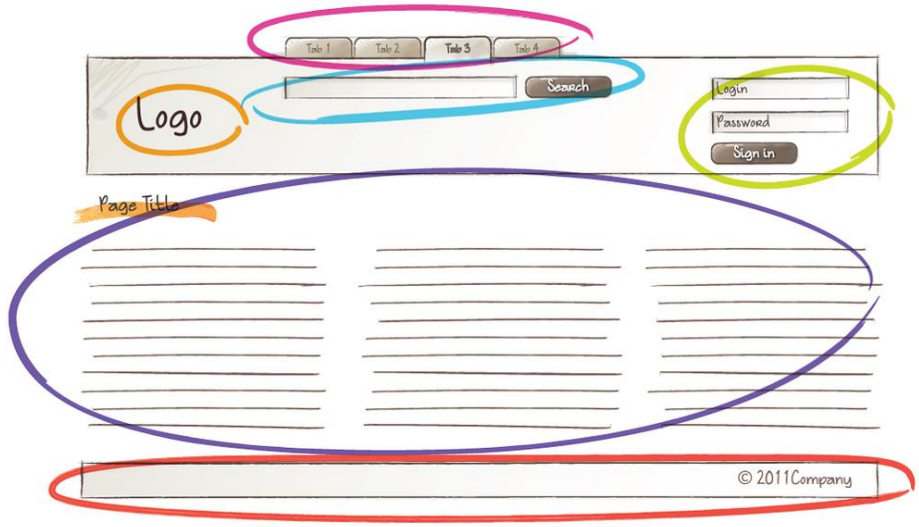
Escrevendo o código da segunda maneira possibilita que em alguma outra parte do game, não só na de ID *sidebar* (#sidebar), utilize-se a classe “list-header” e consiga, portanto, configurar seus atributos na classe definidos.

4.3. BEM

A arquitetura **BEM**, assim como a OOCSS possui regras muito claras. Sua sigla vem da forma que sua arquitetura trabalha, dividindo todo o site, em **blocos**, onde dentro dos blocos, que sempre serão únicos, terão os **elementos** que são o conteúdo do bloco que se repete e os **modificadores,** que são responsáveis pela alteração de aparência ou comportamento de um dos dois itens anteriores.

Quando desenvolvemos um game ou um site institucional, fica fácil saber os elementos da página HTML que não se repete e os que se repetem, assim, em cima desta análise simples e rápida escreve-se o código CSS, como pode ver na imagem abaixo (Figura 9) onde cada círculo colorido definiria na nossa arquitetura um bloco.

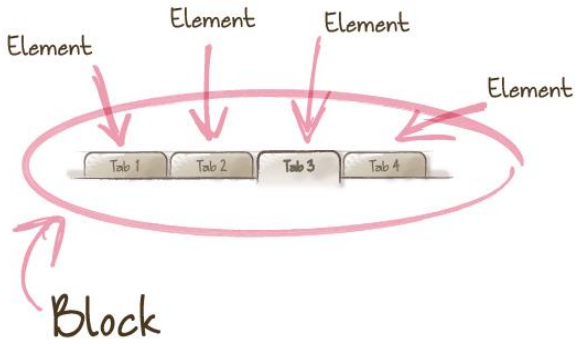
Figura 9 - Ilustração de uma divisão de blocos em uma visão da arquitetura BEM em uma página web.



Fonte: IMASTER, 2015.

Já na Figura 10 mostra a divisão de elementos dentro de um dos blocos do exemplo anterior, no caso o menu.

Figura 10 - Divisão de elementos dentro de um bloco (menu) em um site.



Fonte: IMASTER, 2015.

Outra regra simples, não menos importante, e de fácil entendimento da arquitetura BEM, está na forma de escrita. Por padrão a nomenclatura das classes criadas no BEM devem também estar alinhadas com a divisão de blocos, elementos e modificadores pré-definidos, provocando assim um fácil entendido do código escrito, e por este fator é considerada uma arquitetura de fácil manutenção de acordo com (EMER, 2014): “Este sistema permite escrever sites de maneira rápida,

autoexplicativa e com manutenção descomplicada.”. Na Figura 12 pode-se perceber a maneira muito intuitiva da escrita de classes BEM, exemplo de código que serviria para a definição de bloco, elemento e modificador da Figura 11 apresentada anteriormente.

- HTML

Figura 11 - HTML exemplo de uma escrita de classes na arquitetura BEM.

```
<ul class="menu">
  <li class="menu__item">Tab 1</li>
  <li class="menu__item">Tab 2</li>
  <li class="menu__item menu__item--active">Tab 3</li>
  <li class="menu__item">Tab 3</li>
</ul>
```

Fonte: SANTOS, 2016.

- CSS

Figura 12 - CSS exemplo de uma escrita de classes na arquitetura BEM.

```
.menu {}
.menu__item {}
.menu__item--active {}
```

Fonte: SANTOS, 2016.

Como pode-se notar a classe “menu” no exemplo é o nosso bloco (parte única), o “item” nosso elemento (parte que se repete) e o “*active*” o modificador de estado. Sua escrita, portanto, se dá da seguinte forma: **“bloco__elemento--modificador”**.

Para finalizar esta arquitetura, o BEM possui assim como qualquer arquitetura seus objetivos na sua seleção, e de acordo com WES (2016) são elas as principais: a baixa especificidade, nomenclatura e CSS fáceis de entender e flexível.

> “Low specificity: If specificity battles start between selectors, the code quality starts to nosedive. Having a low specificity will help maintain the integrity of a large project’s CSS for a lot longer. [...] Easy-to-understand naming: Ideally it won’t take mental effort to understand what a class does, or what it should apply to. I also want it to be easy to learn for

> people who are new to the code base [...] Easy to pick up: New developers need to know how and where to write features, and how to work in the project. A good developer experience on a project can help prevent a mangled code base and decrease the stress level. [...] Flexible & sturdy: There's a lot of things we don't control when we build a site. Our code needs to work with different viewports, user settings, when too much content gets added in a space, and plenty of other non-ideal circumstances.". WES, 2016.[2]

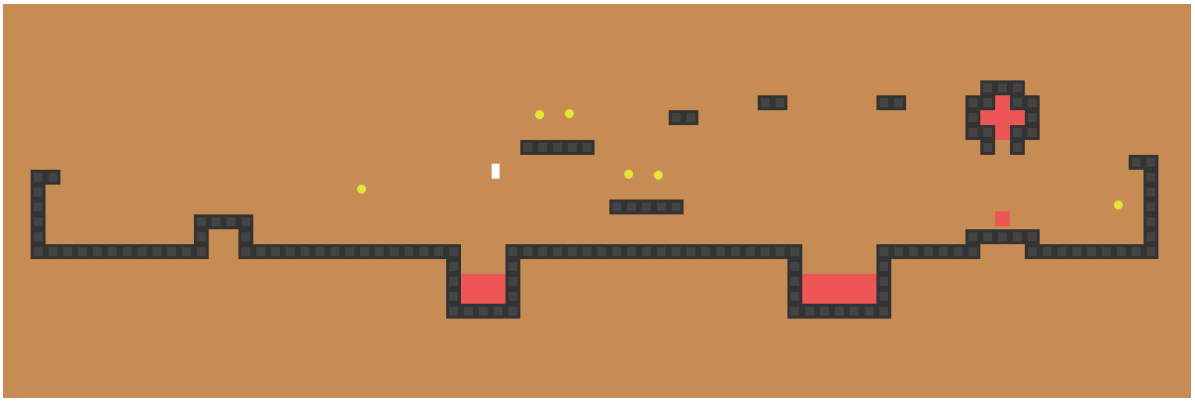
[2] Baixa especificidade: Se batalhas de especificidade começarem entre seletores, a qualidade do código começa a diminuir. Ter uma baixa especificidade ajudará a manter a integridade do CSS de um projeto grande por muito mais tempo. [...] Nome fácil de entender: O ideal é que não seja necessário um esforço mental para entender o que uma classe faz ou a que ela deve se aplicar. Também quero que seja fácil aprender para pessoas que são novas na base de código. [...] Fácil de entender: novos desenvolvedores precisam saber como e onde escrever recursos e como trabalhar no projeto. Uma boa experiência de desenvolvedor em um projeto pode ajudar a evitar uma base de código desfigurada e diminuir o nível de estresse. [...]. Flexível e robusto: há muitas coisas que não controlamos quando criamos um site. Nosso código precisa funcionar com diferentes *viewports*, configurações de usuário, quando muito conteúdo é adicionado em um espaço e muitas outras circunstâncias não ideais.

5. APLICAÇÕES E ANÁLISES DE CSS EM CASOS PRÁTICOS

5.1. CASO 1 - ARQUITETURA SMACSS.

Neste primeiro *game,* 2D do tipo *runner* onde o objetivo do jogo é recolher todas as moedas presentes no cenário sem que o personagem (ponto branco, na Figura 13) morra pela lava (quadrados vermelhos), vamos analisar os arquivos de HTML (Figura 14) e CSS (Figura 15), e aplicar a arquitetura SMACSS com fim de reorganizar e melhorar o projeto do game.

Figura 13 - Interface do game no caso de uso 4.1, game 1.



Fonte: Captain.

Figura 14 - O HTML da interface do game da Figura 13.

```
1   <!DOCTYPE html>
2   <html lang="pt" >
3
4   <head>
5     <meta charset="UTF-8">
6     <title>Game - 01</title>
7       <link rel="stylesheet" href="css/style.css">
8   </head>
9
10  <body>
11      <script  src="js/index.js"></script>
12  </body>
13  </html>
```

Fonte: Captain.

Figura 15 - O CSS da interface do game da Figura 13.

```
4
5   .background { table-layout: fixed; border-spacing: 0; }
6
7   .background td {
8     padding: 0;
9   }
10
11  .lava, .actor {
12    background: #e55;
13  }
14
15  .wall {
16    background: #444;
17    border: solid 3px #333;
18    box-sizing: content-box;
19  }
20
21  .actor {
22    position: absolute;
23  }
24
25  .coin {
26    background: #e2e838;
27    border-radius: 50%;
28  }
29
30  .player {
31    background: #fff;
32    box-shadow: none;
33  }
34
35  .lost .player {
36    background: #a04040;
37  }
38
39  .won .player {
40    background: green;
41  }
42
43  .game {
44    position: relative;
45    overflow: hidden;
46  }
```

Fonte: Captain.

Como pode-se ver o jogo acima possui, ambas as partes tanto HTML quanto o CSS, simples. Mas, aplicando para uma arquitetura mesmo um código simples terá melhoras significativas para o projeto.

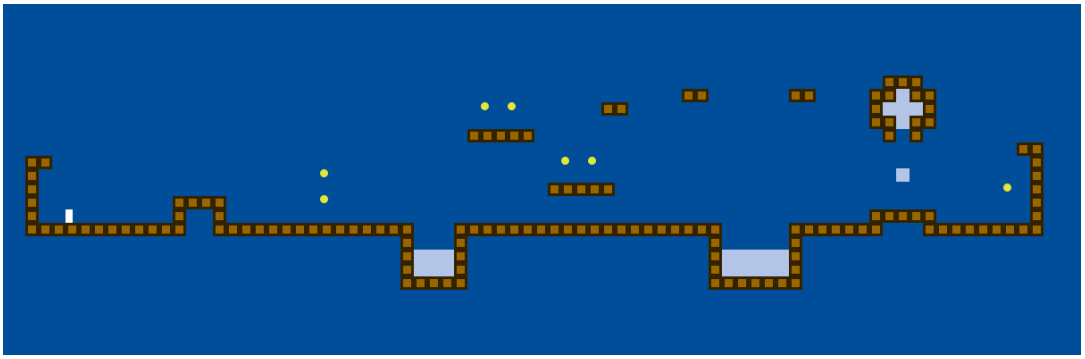
Supondo que há uma necessidade de mudar o tema do game sem que haja uma substituição de código, portanto, temos que adicionar este código CSS a possibilidade de mudança do tema do jogo, “a lava” para um outro, por exemplo um tema “água”. Com isso, pode-se notar na Figura 16 que de fato não seria uma tarefa difícil, mas aplicando arquitetado o resultado final é muito melhor.

Figura 16. O CSS da interface da Figura 13 modificado para que mude o tema do game.

```css
body { background: #c68c53; }

/*Tema Água*/
body { background: #004d99; }

h2 { color: #666;  font-family: monospace; text-align: center; }

.background { table-layout: fixed; border-spacing: 0; }

.background td {
  padding: 0;
}

.lava, .actor {
  background: #e55;
}

/*Tema Água*/
.lava, .actor {
  background: #b3c4e6;
}


.wall {
  background: #444;
  border: solid 3px #333;
  box-sizing: content-box;
}

/*Tema Água*/
.wall {
  background: #996600;
  border: solid 3px #332200;
  box-sizing: content-box;
}

.actor {
  position: absolute;
}
```

Fonte: Autor.

Figura 17 - A interface do game da Figura 13 com o CSS para modificar o tema (Figura 16) para um de estilo “água”.



Fonte: Autor.

Nas próximas Figuras (18, 19 e 20) vamos aplicar neste projeto uma arquitetura SMACSS e analisar então suas vantagens e benefícios não só ao *game,* mas a equipe de desenvolvedores.

Figura 18 - HTML com arquivos CSS no modelo da arquitetura SMACSS.

```html
<head>
  <meta charset="UTF-8">
  <title>Game - 01</title>
    <link rel="stylesheet" href="css/base.css">
    <link rel="stylesheet" href="css/layout.css">
    <link rel="stylesheet" href="css/module.css">
    <link rel="stylesheet" href="css/state.css">
    <link rel="stylesheet" href="css/theme.css">
</head>

<body>
    <script  src="js/index.js"></script>
</body>
```

Fonte: Autor.

Figura 19 - Arquivos CSS do *game* separados de acordo com a arquitetura SMACSS.

base.css

```css
body { background: #c68c53; }

h2 { color: #666;  font-family: monospace; text-align: center; }
```

layout.css

```css
.background { table-layout: fixed; border-spacing: 0; }

.background td {
  padding: 0;
}
```

module.css

```css
.lava, .actor {
  background: #e55;
}

.wall {
  background: #444;
  border: solid 3px #333;
  box-sizing: content-box;
}

.actor {
  position: absolute;
}

.coin {
  background: #e2e838;
  border-radius: 50%;
}

.player {
  background: #335699;
  box-shadow: none;
}
```

state.css

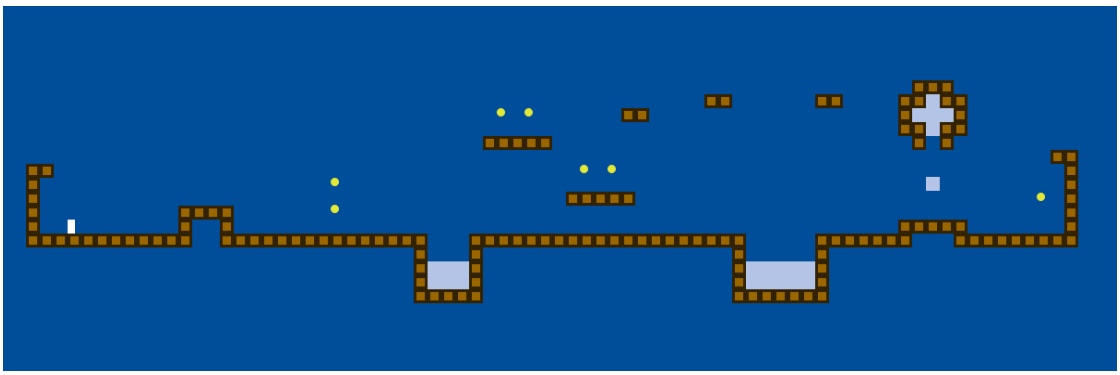
```css
.lost .player {
  background: #a04040;
}

.won .player {
  background: green;
}
```

theme.css

```css
/* Tema Água */
body { background: #004d99; }

.lava, .actor {
  background: #b3c4e6;
}

.wall {
  background: #996600;
  border: solid 3px #332200;
  box-sizing: content-box;
}
```

Fonte: Autor

Figura 20 - *Game* com a nova arquitetura SMACSS.



Fonte: Autor.

Em primeiro lugar podemos ver que o objetivo, a mudança do tema não foi alterada, e nem deveria já que é o mesmo CSS. Mas podemos notar que o CSS foi todo dividido de acordo com as suas respectivas responsabilidades e o tema foi adicionado no arquivo “theme.css” e abaixo será analisado o porquê que a aplicação da arquitetura é mais benéfica ao projeto.

- O CSS antigo “style.css”, como possuía todo o CSS do jogo trazia consigo alguns problemas:
  - Muito mais difícil a localização dos códigos para manutenção.
  - Ordem de leitura quebrada, a ordem de leitura estava layout, módulos, temas, módulos, e consequentemente quanto mais o arquivo vai crescendo pior vai ficando esta leitura.
  - Comentários chamando atenção do menos importante “Tema Água”.

- As vantagens da Arquitetura SMACSS.
  - Muito mais intuitiva a leitura, análise e percepção do que é base, layout, módulos, estados e tema dentro do jogo.
  - Manutenção otimizada do game, pela fácil interpretação e divisão do código.
  - Muito mais fácil a inserção de outros temas no mesmo game. Como pode ver na Figura 21 o “Tema Gelo” comentado. Caso haja uma necessidade de mudança, muito mais fácil a execução da mesma, como pode-se ver na Figura 21.

Figura 21 - Mudança de tema “água” para “gelo”.

```
/* Tema Água */
/*body { background: #004d99; }

.lava, .actor {
  background: #b3c4e6;
}

.wall {
  background: #996600;
  border: solid 3px #332200;
  box-sizing: content-box;
}*/


/* Tema Gelo */
body { background: #ccdbea; }

.lava, .actor {
  background: #eff3fa;
}

.wall {
  background: #ccdbea;
  border: solid 3px #5B4E32;
  box-sizing: content-box;
}
```

Fonte: Autor.

Importante salientar que essa análise está sendo feita em um game simples de CSS com trinta a cinquenta linhas e se escalado para uma proporção maior, todos os tópicos listados acima, as dificuldades proporcionalmente aumentariam com o código.

5.2. CASO 2 - ARQUITETURA BEM.

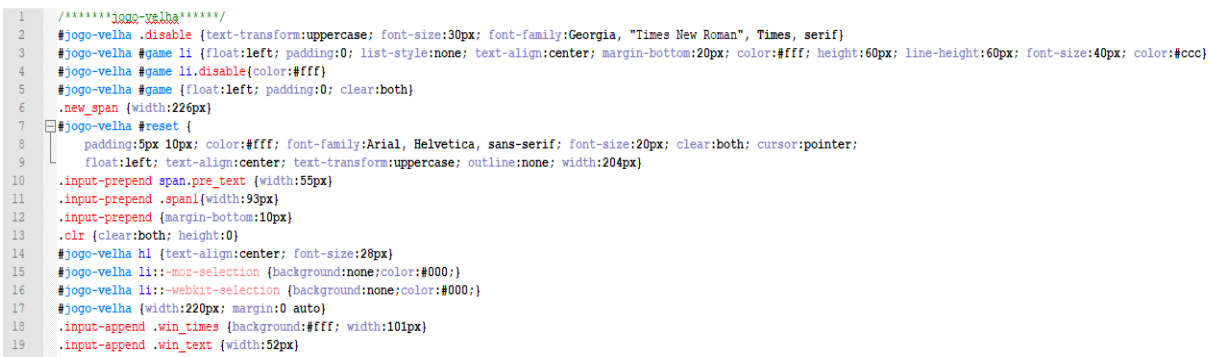
Neste próximo caso de uso, a arquitetura BEM será inserida em cima de um jogo da velha já desenvolvido como podemos observar nas Figuras 22, 23 e 24, seu respectivo código HTML (a *tag* <body>), o seu CSS e a usa interface.

Figura 22 - T*ag <body>* do HTML do jogo da velha estudado no segundo caso de uso deste documento.

```
<div id="jogo-velha">
    <div class="span3 new_span">

    <div class="row">
        <h1 class="span3">JOGO DA VELHA</h1>
        <div class="span3">
            <div class="input-prepend input-append">
                <span class="add-on win_text">O ganhou </span>
                <strong id="o_win" class="win_times add-on">0</strong><span class="add-on">vezes</span>
            </div>
            <div class="input-prepend input-append">
                <span class="add-on win_text">X ganhou </span>
                <strong id="x_win" class="win_times add-on">0</strong><span class="add-on">vezes</span>
            </div>
        </div>
    </div>

    <ul class="row" id="game">
        <li id="one" class="btn span1" >+</li>
        <li id="two" class="btn span1">+</li>
        <li id="three" class="btn span1">+</li>
        <li id="four" class="btn span1">+</li>
        <li id="five" class="btn span1">+</li>
        <li id="six" class="btn span1">+</li>
        <li id="seven" class="btn span1">+</li>
        <li id="eight" class="btn span1">+</li>
        <li id="nine" class="btn span1">+</li>
    </ul>

    <div class="clr"> </div>
        <div class="row"><a href="#" id="reset" class="btn-success btn span3" >Nova disputa</a></div>
    </div>
</div>

<script src='https://code.jquery.com/jquery-1.7.2.min.js'></script>
<script  src="js/index.js"></script>
```

Fonte: Bandal.

Figura 23 - CSS completo do jogo da velha estudado no segundo caso de uso deste documento.



```
/*******jogo-velha******/
#jogo-velha .disable {text-transform:uppercase; font-size:30px; font-family:Georgia, "Times New Roman", Times, serif}
#jogo-velha #game li {float:left; padding:0; list-style:none; text-align:center; margin-bottom:20px; color:#fff; height:60px; line-height:60px; font-size:40px; color:#ccc}
#jogo-velha #game li.disable{color:#fff}
#jogo-velha #game {float:left; padding:0; clear:both}
.new_span {width:226px}
#jogo-velha #reset {
    padding:5px 10px; color:#fff; font-family:Arial, Helvetica, sans-serif; font-size:20px; clear:both; cursor:pointer;
    float:left; text-align:center; text-transform:uppercase; outline:none; width:204px}
.input-prepend span.pre_text {width:55px}
.input-prepend .spanl{width:93px}
.input-prepend {margin-bottom:10px}
.clr {clear:both; height:0}
#jogo-velha h1 {text-align:center; font-size:28px}
#jogo-velha li::-moz-selection {background:none;color:#000;}
#jogo-velha li::-webkit-selection {background:none;color:#000;}
#jogo-velha {width:220px; margin:0 auto}
.input-append .win_times {background:#fff; width:101px}
.input-append .win_text {width:52px}
```

Fonte: Bandal.

Figura 24 - Interface do jogo da velha estudado no segundo caso de uso deste documento.

Fonte: Bandal.

Utilizando-se deste cenário, nota-se que a divisão imposta pelas regras do BEM podem serem feitas, partindo do princípio que todo **bloco** tem que ser único, como mostra a Figura 25, uma divisão da interface se dá de modo simples.

Figura 25 - Divisão da interface em blocos, de acordo com a arquitetura BEM.

Fonte: Autor.

Subsequente, o HTML do *game* terá sua estrutura toda modificada. Serão criadas então três div's (" ") com três classes diferentes e partindo desta classe do **bloco** criada, cria-se as classes dos **elementos** (Figuras 26, 27 e 28).

Figura 26 - "Div Painel", bloco um do *game*.

```
<div class="painel">
    <h1 class="painel__titulo">JOGO DA VELHA</h1>
    <div>
        <div class="painel__input-prepend painel__input-append">
            <span class="painel__addon painel__addon--win">O ganhou </span>
            <strong id="o_win" class="painel__win painel__addon">0</strong>
            <span class="painel__addon">vezes</span>
        </div>
        <div class="painel__input-prepend painel__input-append">
            <span class="painel__addon painel__addon--win">X ganhou </span>
            <strong id="x_win" class="painel__win painel__addon">0</strong>
            <span class="painel__addon">vezes</span>
        </div>
    </div>
</div>
```

Fonte: Autor.

Figura 27 - "Div *Game*", bloco dois do *game*.

```
<ul class="game" id="game">
    <li id="one" class="game__btn" >+</li>
    <li id="two" class="game__btn">+</li>
    <li id="three" class="game__btn">+</li>
    <li id="four" class="game__btn">+</li>
    <li id="five" class="game__btn">+</li>
    <li id="six" class="game__btn">+</li>
    <li id="seven" class="game__btn">+</li>
    <li id="eight" class="game__btn">+</li>
    <li id="nine" class="game__btn">+</li>
</ul>
```

Fonte: Autor.

Figura 28 - "Div Zera", terceiro bloco do *game*.

```
<div class="zera">
    <div class="zera__clr"> </div>
    <a href="#" id="reset" class="zera__btn">Nova disputa</a>
</div>
</div>
```

Fonte: Autor.

Com isso e para concluir a modificação do game para a arquitetura BEM, modifica-se então o CSS e como pode-se notar na Figura 29, sua organização melhora consideravelmente e ainda atinge os principais objetivos da arquitetura

BEM, mantendo um CSS com baixa especificidade, muito fácil de se entender e flexível.

Figura 29 - Estrutura do CSS com a arquitetura BEM aplicada.

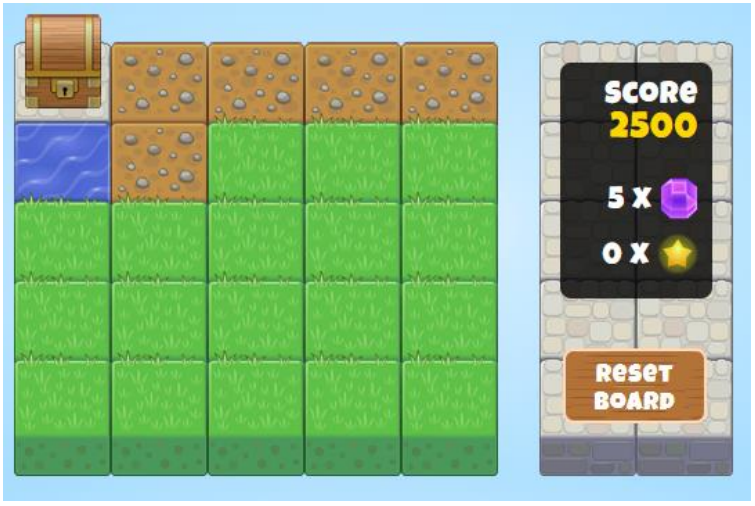
```
/** Bloco Painel **/
.painel__titulo { }
.painel__input-prepend { }
.painel__addon { }
.painel__addon--win { }
.painel__win { }

/** Bloco Game **/
.game { }
.game__btn { }
.game__btn--disable{ }

/** Bloco Zera **/
.zera { }
.zera__btn { }
.zera_clr { }

```

Fonte: Autor.

5.3. CASO 3 - ARQUITETURA OOCSS.

O jogo analisado como mostra a Figura 30 possui uma *gameboard* onde no jogo é necessário ir explorando e coletando o maior número possível de diamantes e estrelas, assim que se dá a pontuação no *game.* A cada clique no quadrado de grama para a exploração, faz com que tenha a possibilidade de se “achar” terra com pedras ou somente água. E, é nesse cenário onde serão aplicadas e analisadas as técnicas da arquitetura.

Figura 30 - Game analisado (*gameboard* e pontuação).



Fonte: Payne.

Neste último caso de uso, utilizando como modelo de arquitetura à OOCSS, será analisado dentro do game de exemplo os dois princípios desta arquitetura, que são a separação entre estrutura e visual baseados no conceito de responsabilidade única, e também a separação entre contexto e conteúdo fazendo com que os estilos no desenvolvimento do jogo não fiquem ligados a localização que estão no HTML.

Para começar a análise do primeiro princípio. Pode-se notar que no desenvolvimento deste jogo há duas melhorias que devem ser realizadas para o padrão OOCSS estar correto, conforme enumerado abaixo. A primeira é que em uma visão de estrutura, seria considerado um erro, como pode ver na Figura 31, a junção de estrutura e visual dentro da mesma classe (“scoreBoard”), o ideal seria a separação da estrutura em uma classe “board”, por exemplo, e somente o visual manter na classe “scoreBoard” sendo chamada no HTML da seguinte forma, ‘<div class=”board scoreBoard”>’ como mostra a Figura X.X o seu CSS correto.

Figura 31 - Configuração de estilo da classe scoreBoard dentro do game.

```
56  .scoreBoard{
57    width:110px;
58    height:170px;
59    margin-top:-34px;
60    padding:10px 10px 0;
61    text-align:right;
62    position:relative;
63    top:21px;
64    left:15px;
65    border-radius:8px;
66    background: rgba(0,0,0,0.8);
67  }
```

Fonte: Payne.

Figura 32 - Ideal configurações de estilos das classes, para exibir informações dentro do game.

```
73  .board{
74    width:110px;
75    height:170px;
76    margin-top:-34px;
77    padding:10px 10px 0;
78    position:relative;
79    top:21px;
80    left:15px;
81  }
82
83  .scoreBoard{
84    text-align:right;
85    border-radius:8px;
86    background: rgba(0,0,0,0.8);
87  }
```

Fonte: Autor.

E a segunda melhoria seria uma mudança na nomenclatura das classes, com o objetivo de mostrar a divisão de responsabilidade e deixar o “objeto” em questão, no caso o botão da *gameboard*, em destaque. O CSS até está dividido de modo correto, mas com a nomenclatura divergente do que a OOCSS instrui. No desenvolvimento deste game como pode-se ver na Figura 33 o nome das classes “*gametile*”, “*water*” e “*dirty*” fazem corretamente a separação de estrutura e visual, com o *gametile* tendo todo o CSS da estrutura e as outras classes com seus respectivos estilos visuais, mas, a alteração de melhoria que devesse fazer para remeter ao princípio da OOCSS seria como mostra a Figura 34.

Figura 33 - Nomenclatura das classes dos quadrados da *gameboard.*

```
<div class="gametile dirt off"></div>
<div class="gametile dirt off"></div>
<div class="gametile dirt off"></div>
<div class="gametile dirt off"></div>
<div class="gametile water off"></div>
<div class="gametile dirt off"></div>
```

Fonte: Payne.

Figura 34 - Correção na nomenclatura das classes dos quadrados da *gameboard* seguindo padrão OOCSS.

```
<div class="quadrao quadrado-dirt off"></div>
<div class="quadrao quadrado-dirt off"></div>
<div class="quadrao quadrado-dirt off"></div>
<div class="quadrao quadrado-dirt off"></div>
<div class="quadrao quadrado-water off"></div>
<div class="quadrao quadrado-dirt off"></div>
```

Fonte: Autor.

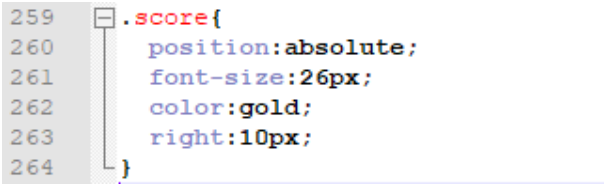
Agora partindo para o segundo princípio da arquitetura OOCSS, notamos na Figura 35 a configuração da div com o atributo “id” = “sidebar”, com uma tag span de classe “score” destacado dentro dela. Nota-se também o estilo desta classe “score” à sua direita na imagem e o que está div de id=sidebar representa dentro da interface do game.

Figura 35 - Configuração da div de id = sidebar, o estilo da classe score e sua representação dentro da interface do game.

Fonte: Autor.

De acordo com Sullivan (2013), como já citado, “Contexto e conteúdo separados: Essencialmente, isso significa ‘raramente usar estilos dependentes de localização’”, e está div dentro do nosso jogo exemplo neste terceiro caso de uso está totalmente fora do que seria adequado, assim uma alteração simples que solucionaria, conforme pode-se notar na Figura 36 é a exclusão do “#sidebar” antes da chamada da classe “score” no CSS.

Figura 36 - Correção da chamada do seletor da classe “score”.



Fonte: Autor.

Observação: Caso o ID (#sidebar) tenha sido adicionado no arquivo com o intuito de sobrescrever um outro estilo pelo nível de especificidade, neste caso pode-se resolver da mesma maneira sem a utilização de ID’s, por exemplo os dois trechos de código CSS abaixo:

- **.score span** { position:absolute; font-size:26px; color:gold; right:10px; }
- **.score .score-right** { position:absolute; font-size:26px; color:gold; right:10px; }

Nestes dois códigos aumentamos a especificidade da chamada do seletor e não a uma necessidade de utilizar um ID.

6. CONSIDERAÇÕES FINAIS

De fato, o CSS tem uma importância indiscutível quando se trata de desenvolvimento web, quando se mergulha no mundo dos *games* ele acaba agregando a si mais reponsabilidades ao seu objetivo e utilização. Mesmo sendo configurado de modo simples, portanto de fácil entendimento, o CSS possibilita uma gama de arquiteturas que soma aos códigos quando utilizado, e que só potencializa o seu *game*. Além disso a implementação de uma arquitetura no desenvolver agrega, e muito na qualidade de um projeto, tanto na qualidade de entrega, quanto na manutenção do mesmo quando necessário.

As arquiteturas SMACSS, BEM e OOCSS neste documento abordadas, são de fato três dos mais famosos modelos de se arquitetar CSS, mas são só uma parte quando se aprofunda mais ainda nestes estudos de arquiteturas CSS. Cada arquitetura possui suas características, facilidades e objetivos, e por isso consequentemente, uma irá se prevalecer sobre a outra em determinados cenários, mas, importante ressaltar que com isso, não se pode qualificar uma delas melhor do que as outras.

7. REFERÊNCIAS BIBLIOGRÁFICAS

ÁVILA. Ricardo Lima Feitosa de Ávila. Jogos Digitais: **Uma brincadeira levada a sério**. 2009. Disponível em: <http://www.infobrasil.inf.br/userfiles/27-05-S4-2-67969-Jogos%20Digitais(1).pdf>, último acesso em 06 de maio de 2018.

BANDAL. Tushar Bandal. **Open Project CodePen**. Disponível em: <https://codepen.io/tusharbandal/pen/mdujc>, último acesso em 30 de maio de 2018.

BATTAIOLA, André Luiz. **Jogos por computador: Histórico, Relevância Tecnológica, Tendências e Técnicas de Implementação**. Anais da SBC 2000. Volume 2. Curitiba, **2000**.

CAPTAIN**.** Captain, **Open Project CodePen**. Anonymous Example. Disponível em <https://codepen.io/anon/pen/MGVPxO>, último acesso em 29 de maio de 2018.

EMER. Jean Carlo Emer. **OOCSS, SMACSS, BEM, DRY CSS: afinal, como escrever CSS?**. Disponível em: <https://tableless.com.br/oocss-smacss-bem-dry-css-afinal-como-escrever-css/>. 2014.

GRACE, Lindsay. **Game Type and Game Genre**. 2005.

GRIMMETT, Chuck. **HTML and CSS Basics for WordPress.** 2017. Figura. Disponível em: <http://www.cagrimmett.com/development/2017/04/24/tools-for-learning-css.html>, último acesso em 05 de junho de 2018.

HUIZINGA, Johan. **Homo Ludes - vom Ursprung der Kultur im Spie**. 1938.

IMASTER. **Arquitetura CSS usando BEM parte 02**. 2015. Figura. Disponível em: <https://imasters.com.br/front-end/css/arquitetura-css-usando-bem-parte-02/?trace=1519021197&source=single>, último acesso em 29 de maio de 2018.

MEYER**.** Eric A. Meyer. **CSS Tools: Reset CSS.** Disponível em <https://meyerweb.com/eric/tools/css/reset/>, último acesso em 04 de junho de 2018.

PAYNE. Adrian Payne. **Open Project Tile Game**. CodePen. Disponível em <https://codepen.io/dazulu/pen/tcwDo>, último acesso em 29 de maio de 2018.

PEREIRA, Fernanda. **Front-End, Back-end e Fullstack – Você sabe o que faz cada um destes profissionais?.** Apex Ensino. 2017. Disponível em

<http://apexensino.com.br/front-end-back-end-e-fullstack-voce-sabe-o-que-faz-cada-um-destes-profissionais/>, último acesso em 04 de Junho de 2018.

REIS, Abel**. Pirâmides, sistemas e gestão de projetos**. Revista *Business Standard*. 2003. Disponível em: <http://noema.typepad.com/dizeres/2006/03/pirmides_sistem.html>, último acesso em 28 de maio de 2018.

SANTOS. Patrícia Gomes dos Santos Silva. **Arquitetura CSS - Parte 2 (BEM)**. 2016. Disponível em <https://pt.linkedin.com/pulse/arquitetura-css-parte-2-bem-patr%C3%ADcia-silva>, último acesso em 04 de junho de 2018.

SCUDERO, Erick. **Os 6 frameworks front-end mais amados no mundo (segundo o GitHub)**. becode.com. Disponível em: <https://becode.com.br/frameworks-front-end-mais-amados-segundo-github/>.

SEPIN. **Evolução da Internet no Brasil e no Mundo**. 2000. Assessoria SEPIN. Coordenação: Luzia Maria Mazzeo Pesquisadoras: Sônia Pantoja Rosângela Ferreira.

SILVA, Prado Rocha. Maycon Silva, Paula Dornhofer, Paro Costa Paulo Sérgio Prampero, Vera Aparecida de Figueiredo. **Jogos Digitais: definições, classificações e avaliação. Tópicos em Engenharia de Computação VI Introdução aos Jogos Digitais**. 2009.

SILVA, Maurício Samy. **Construindo sites com CSS e (X)HTML**. NOVATEC. 2007**.**

SIMÕES, Ney. **Arquitetura CSS**. 2015. Disponível em: <https://medium.com/tableless/arquitetura-css-d344fb01dd18>, Acesso em: 29 de maio de 2018.

SULLIVAN, Nicole. **OOCSS Wiki**. 2013. Disponível em: <https://github.com/stubbornella/oocss/wiki>. Acesso em 29 de maio de 2018.

W3Schools. W3Schools, **CSS Syntax**. Disponível em <https://www.w3schools.com/css/css_syntax.asp>, último acesso em 04 de Junho de 2018.

WES**,** Ruvalcaba. **BEM & Atomic Design: A CSS Architecture Worth Loving**. 2017. Disponível em: <https://www.lullabot.com/articles/bem-atomic-design-a-css-architecture-worth-loving>, último acesso em 28 Mai 2018.