Faculdade de Tecnologia de Mococa – Mário Robertson de Sylos Filho

DESENVOLVIMENTO DE UM JOGO ELETRÔNICO 2D EM JAVA

Ezequiel Aparecido Borges

Faculdade de Tecnologia de Mococa

Discentes do curso de Análise e Desenvolvimento de Sistemas

Profa. Dra. Sandra Cristina Costa Prado

Faculdade de Tecnologia de Mococa

Docente do curso de Análise e Desenvolvimento de Sistemas

RESUMO

O presente trabalho descreve o processo de desenvolvimento de um jogo digital num âmbito geral com teorias e prática, desde o conceito como concepção da ideia do jogo, sua documentação que aborda aspectos gerais do jogo, como gênero, público alvo, ambientação, tecnologias usadas, seu funcionamento e regras, objetivo e motivação, jogabilidade e tudo que um jogador poderá encontrar no jogo, até o desenvolvimento do software que será que controlará as ações dentro jogo. Embora atualmente haja Game Engines para desenvolvimento de jogos como Unreal Engine e Unity, que utilizam linguagens como C++ e C# respectivamente, a linguagem Java se mostra muito eficiente para o desenvolvimento de jogos como no passado era feito para os primeiros smartphones e também pelo grande jogo de sucesso Minecraft.

Palavras-chave: Jogos, *Games*, Java, Desenvolvimento.

INTRODUÇÃO

A indústria de jogos sempre foi uma crescente e rica parcela do mercado de entretenimento, tornando-se a partir dos anos 2000, a mais rica delas.

O desenvolvimento de jogos, hoje, compõe-se de muitos e distintos profissionais, como programadores, escritores, atores, músicos e artistas em geral, e tornou-se multibilionária produzindo muitos outros produtos além de jogos, como filmes, roupas e artigos colecionáveis.

Porém a indústria teve um início simples e humilde, mas sempre, muito promissor

O processo de desenvolvimento de um jogo possuí muitas características, sendo suas principais: o tipo de jogo, e a empresa desenvolvedora. Há também a tecnologia disponível, que pode limitar, ou ampliar o desenvolvimento do jogo. Mas esse processo não consiste somente em códigos, mas também em teorias e análises.

Jeannie Novack (2011) descreve, em seu livro Game Devolopment Essentials (Desenvolvimento de Games), oito etapas de criação de jogos, que são:

1. Conceito – A ideia de um jogo. Pode vir de um *insight*, *brainstorm*, ou mesmo de outras mídias como livros, filmes, revistas, momentos históricos da humanidade etc.
2. Pré-produção – O planejamento do desenvolvimento. É nessa fase que o *Game Design Document* – GDD é feito. Contendo uma descrição do jogo e sua ambientação, jogabilidade, o estilo de arte, o objetivo e estilo do jogo.
3. Protótipo - O divisor de águas no desenvolvimento. É na prototipação que a ideia é concretizada, e se é viável continuar com o projeto. Nele deve conter a parte central do jogo e sua mecânica básica. Deve-se ter em mente que o resultado final do jogo será muito diferente do protótipo, isso porque ao começar a prototipação, muitas outras ideias podem surgir para compor o game final.
4. Produção – Com êxito no protótipo, a produção é iniciada. Muitas das características do jogo final podem ainda não existir ou estar definidas. Mas sua jogabilidade, física, estilo de jogo, e objetivo devem estar formados, pois serão a base para o resto. Um exemplo pode ser um jogo simples de corrida de carros, onde o jogador guiará um veículo. Os carros a princípio serão pré-definidos, mas uma

funcionalidade de customização poderá ser incluída mais tarde como simplesmente mudar a cor do carro.

5. Alfa – Aqui o jogo pode ser jogado do começo ao fim, mas isso não significa que ele já está pronto. Nesse ponto deve ser observado os problemas do game, como arte, sons e bugs da própria programação. Em grandes empresas, testadores externos à equipe de desenvolvimento jogam para encontrar esses problemas.

6. Beta – Nesta etapa, o processo de produção está concluído e o foco torna-se a correção dos problemas encontrados na fase Alfa.

7. Ouro – Fase da produção da mídia física do jogo após sua aprovação.

8. Pós-produção – Após o lançamento do jogo, versões de correções de bugs menos perceptíveis ou inclusões de cosméticos e até mesmo novas opções de jogo como: modos online, DLC's (Dowloadable Content, ou, conteúdo "baixável"), novas dificuldades, podem ser lançadas para manter e dar maior longevidade ao game.

METODOLOGIA

O primeiro passo depois da decisão de criar um jogo como trabalho de conclusão de curso, foi a sua concepção, o que é, justamente, a primeira etapa descrita na introdução. Assim, basicamente, a metodologia usada neste trabalho acompanhou as demais etapas.

NOVAK (2017) foi a bibliografia básica para a compreensão do processo formal de criação de um jogo. Para o desenvolvimento, em nível de programação, foi utilizado o curso de desenvolvimento de jogos da Danki Code (DANKI, 2020). Nele, cria-se e estrutura-se uma *game engine* (motor gráfico), própria para o desenvolvimento de qualquer jogo 2D. Conforme indicado nestas duas referências, deve-se criar o GDD, o que foi feito, ainda que de maneira simplificada.

Como já mencionado, foi utilizada a linguagem Java (2021). O ambiente de desenvolvimento integrado (IDE – *Integrated Development Environment*).) foi o Eclipse.

DISCUSSÃO E RESULTADOS

Conceito

A inspiração para o jogo desenvolvido neste projeto foi Flicky, da versão de Mega Drive, lançado em 1991. Flicky consiste em quatro personagens: Flicky, o personagem-título e herói, que é um pardal que não voa, os Chirps que são pintinhos, o gato Tiger e o lagarto/iguana Iggy, mostrado na figura 1.

O objetivo do jogo é controlar o Flicky e coletar todos os Chirps espalhados pelo cenário e levá-los para a saída, evitando todos os gatos e lagartos.

Em Kittens and Bunnies, o jogador tem como objetivo encontrar todos os gatinhos perdidos num labirinto florestal e levá-los de volta para suas.

Figura 1-Capa e tela inicial do jogo Flicky no qual Kittens and Bunnies foi inspirado.

Fonte: https://www.retroplace.com/en/games/36731--flicky

Pré-Produção

Embora a documentação do jogo seja trabalhada em várias partes do processo, no caso de um desenvolvimento independente, ele pode surgir com o projeto já em andamento ou estágio avançado. Para Kittens and Bunnies foi criado um documento como exemplo.

Documentação de um Jogo

A documentação de um game é criada com base nas fases de conceito, pré-produção e produção. Não existem regras ou normas fixas para sua implementação, mas ela tem a missão de orientar os desenvolvedores, para que identifiquem suas funções, organizando assim o seu trabalho. Tem, também, missão de esclarecer editoras, licenciadoras e fabricantes sobre as funções e objetivos do jogo. Em projetos independentes esse processo pode ser mais simples, dependendo do número de pessoas trabalhando no jogo. Nas seções seguintes, as fases de Kittens and Bunnies serão descritas.

Kittens and Bunnies

Identidade do jogo

O jogador vive na vizinhança da Vila dos Gatinhos, habitada por gatos antropomorfizados, e tem que ajudá-los a voltar para casa pois se perderam num labirinto florestal à procura de seus coelhos que fugiram para a floresta.

Descrição da mecânica

O jogo possuí uma mecânica simples, com controles de personagens em oito direções.

Características

O mundo do jogo se passa num cenário fictício e fantasioso, com cenário natural cheio de árvores e casas de madeiras.

Arte

Jogo em 2D utilizando pixel art, com Tilesets e Sprites Sheets. Fonte das artes utilizadas neste jogo:
https://assetstore.unity.com/packages/2d/characters/tiny-rpg-forest-114685
https://pipoya.itch.io/pipoya-free-rpg-character-sprites-nekonin
https://bakudas.itch.io/generic-rpg-pack

Música/Trilha Sonora

A trilha é composta inicialmente por duas músicas, uma para o menu do jogo e outra para o tema da Vila dos Gatinhos, The Marmots do Filme Heide de 2015, e Concerning of Hobbits de O Senhor dos Anéis A Sociedade do Anel de 2001, respectivamente. Ambas abordam bem a atmosfera do jogo que tem em passar algo feliz e reconfortante.

Interface/Controles

Os controles serão pelo teclado do computador.
Teclas W/A/S/D para movimentar o personagem pelo cenário
Tecla F para interação com personagens e itens.

Dificuldade

Mínima. O jogo tem por objetivo simples apenas de divertir todos os níveis de jogador. Os puzzles do jogo serão inicialmente encontrar os gatos no labirinto, e há a ideia futura de implementar inimigos com um sistema de combate.

Personagem

Nosso personagem será o único humano no jogo, e a sprite sheet do Link do jogo The Legend of Zelda: Link's Awakening, funcionou bem para esse projeto.

Cronograma

O projeto iniciou em 16/01/2020 e previsão para término até 12/2021. Com todas as funcionalidades básicas implementadas e podendo ser jogado do começo ao fim.

Definições gerais

O jogo tem por gênero aventura e apenas um nível com um mapa, com um labirinto como puzzle e dificuldade baixa para todos os públicos.

Produção

Esta fase consiste na escrita do código, com a arte pré-definida, alguns efeitos sonoros e músicas e o enredo principal somente. O trabalho de *Level Design* é colocado em prática e lapidado.

Inicialmente, deve-se criar a tela onde tudo será renderizado utilizando a classe JFrame do Java. A figura 2 mostra o trecho de código que realiza esta tarefa. No método main, é criada e ativada uma instância da classe Game. Na classe Game foi usado o método setPreferredSize que recebe num new Dimension com largura, altura. Foi usada uma escala (variável Scale) para multiplicar esses valores e escalonar o tamanho das imagens a serem renderizadas. Isso aumenta o pixel da imagem e da um visual “retrô” ao jogo. Em seguida, o método initFrame cria uma janela gráfica. Depois, todos os objetos, como o player, as entidades, os NPCs - No Player Characters, o cenário do jogo (mapa), onde todas as interações serão feitas, são renderizados na tela.

A figura 4 mostra o resultado, conforme as opções mostradas na figura 3. Para obter esse resultado, foi utilizada a técnica de Tileset e Tilemap.

Um Tileset é uma imagem com um conjunto de figuras geralmente pixel art, que são reunidas para formar o Tilemap. Um mesmo Tileset pode formar vários diferentes Tilemaps. Essa técnica foi amplamente usada nos primórdios da indústria de desenvolvimento, pois somente uma imagem ocupa o espaço na memória do jogo.

Figura 2 - Exemplo de Tileset utilizado em Kittens and Bunnies.

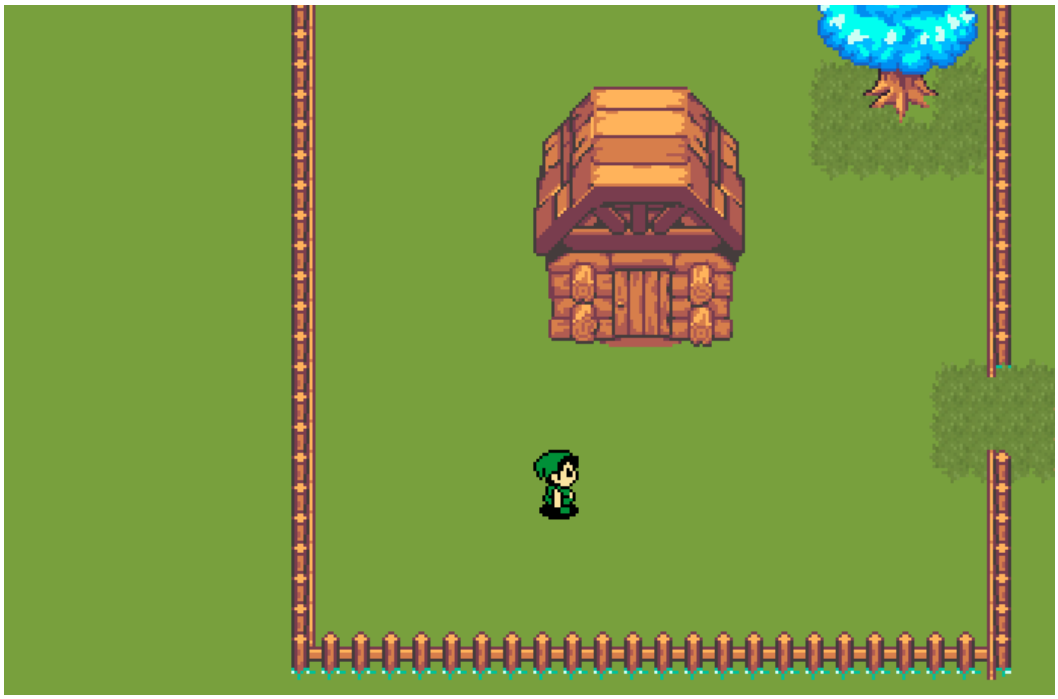
Fontes: https://bakudas.itch.io/generic-rpg-pack
https://assetstore.unity.com/packages/2d/characters/tiny-rpg-forest-114685

Figura 3-Código para criação da tela inicial.

```
public Game() {
    addKeyListener(this);
    setPreferredSize(new Dimension(Width * Scale, Height * Scale));
    initFrame();
    image = new BufferedImage(Width, Height, BufferedImage.TYPE_INT_RGB);
    entidades = new ArrayList<Entidades>();
    sprites = new Assets("/Link.png");
    enviroment = new Assets("/enviroment.png");
    npcs = new Assets("/NPCs.png");
    player = new Player(20, 20, 32, 32, sprites.getSprite(32, 0, 16, 16));
    entidades.add(player);
    Loui = new Loui(Width/2,Height/2,16,16, npcs.getSprite(32, 0, 32, 32));
    entidades.add(Loui);
    placaparalabirinto = new PlacaParaLabirinto(200,150,16,16, enviroment.getSprite(8*32, 5*32, 32, 32));
    entidades.add(placaparalabirinto);
    mapas = new Mapas("/world.png");
    menu = new Menu();
    //Sound.menutheme.loop();


}

public void initFrame(){
     //frame.setIconImage();
    frame = new JFrame("Game"); //título
    frame.add(this);
    frame.setResizable(false);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}

public static void main(String[] args) {
    Game game = new Game();
    game.start();
}
```

Fonte: Elaboração própria.

Figura 4-Tela inicial de Kittens and Bunnies. O fundo verde dá uma ilusão minimalista de gramado com alguns tiles de grama mais alta para dar um contraste.

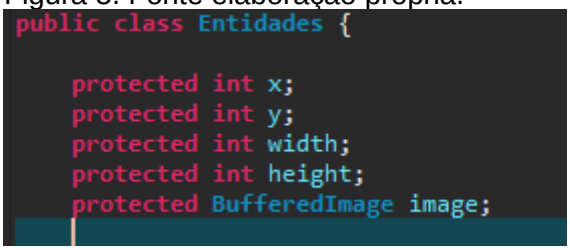


Fonte: Elaboração própria.

A construção do cenário é chamada de *Level Design* e em grandes empresas há uma equipe de profissionais responsável por dar composição aos elementos dispostos nele. Em projetos independentes pode haver pelo menos um profissional para esta função. Este profissional é facilmente confundido com o Game Designer.

Porém, o Game Designer é responsável pelo projeto num todo. É ele quem transmite aos integrantes da equipe toda a atmosfera do jogo da maneira mais convincente possível para que eles possam dar vida às suas ideias. No início, todo esse trabalho era realizado apenas pelos programadores, porém conforme a indústria foi crescendo e amadurecendo, acompanhado da tecnologia que abria para novas possibilidades, surgiu a necessidade de novos profissionais. Naturalmente, esses profissionais devem ter uma visão geral de todo contexto do projeto de um jogo. Isso dá entrosamento a equipe para que as ideias sejam bem executadas e trabalhadas. O ambiente descontraído que as primeiras empresas tinham asseguravam isso.

Figura 5. Fonte elaboração própria.



Fonte: Elaboração própria.

O próximo passo é dar vida ao personagem e fazê-lo movimentar-se pelo cenário. Esse é um processo de duas partes.

A Figura 5 mostra uma classe que receberá os valores das dimensões da imagem e da posição onde ela será renderizada. Tudo que for interativo no jogo passará por essa classe e/ou deverá herdar suas características.

Figura 6 – Classe do Player e seus atributos.

```
public class Player extends Entidades {

    public boolean right, left, up, down, attack, attacking;
    private int right_dir = 0, left_dir = 1, up_dir = 3, down_dir = 4;
    private int dir = right_dir;
    public int velocidade = 2;
    public boolean lastright = false;
    public boolean lastleft = false;

    private int frames = 0, maxframes = 5, index = 0, maxindex = 0;
    private boolean moved = false;
    private BufferedImage[] correrdireita;
    private BufferedImage[] correresquerda;
    private BufferedImage[] subir;
    private BufferedImage[] descer;
    private BufferedImage paradofrente = Game.sprites.getSprite(32, 0, 32, 32);
    private BufferedImage paradoesquerda = Game.sprites.getSprite(32, 32, 32, 32);
    private BufferedImage paradodireita = Game.sprites.getSprite(32, 3*32, 32, 32);
    private BufferedImage paradocostas = Game.sprites.getSprite(32, 2*32, 32, 32);
```

Fonte: Elaboração própria.

A Figura 6 mostra a classe do Player, que contém seus próprios atributos, como suas próprias imagens e métodos com variáveis que cuidarão das trocas de imagens para dar a ilusão de movimento, mas que herde da classe Entidades, para também poder utilizar de suas funcionalidades.

Figura 7 – Os IFs recebem variáveis e métodos para condicionar o movimento.

```
if(right && Mapas.isFree(this.getX()+8, this.getY()+8)){
    for(int i = 0; i < 3; i++){
        correrdireita[i] = Game.sprites.getSprite(0 + (i*32), 3*32, 32, 32);
    }
    moved = true;
    maxindex = 2;
    dir = right_dir;
    x += velocidade;

}
if(left && Mapas.isFree(this.getX()-4, this.getY()+10)){
    for(int i = 0; i < 3; i++){
        correresquerda[i] = Game.sprites.getSprite(0 + (i*32), 32, 32, 32);
    }
    moved = true;
    maxindex = 2;
    dir = left_dir;
    x -= velocidade;

}
```

Fonte: Elaboração própria.

Estes IFs controlam a direção esquerda e direita que o personagem percorre no mapa. Eles recebem um FOR que alterna entre os frames da imagem para dar movimento e adicionam ou subtraem valor da posição X da imagem onde ela está renderizada.

A Imagem é chamada de sprite sheet. Ela recebe vários desenhos em posições e ações diferentes do personagem, como andar, correr e pular, por exemplo. Essa técnica é vantajosa pois você pode usar apenas uma variável para a imagem e depois escolher as partes que irão ser usadas.

Figura 8 – Sprite sheet do personagem utilizado pelo jogado.

Fonte: https://www.spriters-resource.com/

Os IFs, na Figura 7, recebem duas condições: a ação do jogador e se o caminho do personagem está livre para ele percorrer. Ambas devem ser verdadeiras para ativar o FOR e o incremento, ou decremento da variável X da imagem. Vale lembrar que o mesmo se aplica a variável Y se a ação for de subir ou descer.

Como dito na documentação, as teclas A, W, S, D do teclado serão responsáveis por receber o input da ação do jogador para movimentar o personagem pela tela.

Figura 9 – Método que verifica se uma determinada tecla foi pressionada.

```
@Override
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_D || e.getKeyCode() == KeyEvent.VK_RIGHT){
        player.right = true;
    } else if(e.getKeyCode() == KeyEvent.VK_A || e.getKeyCode() == KeyEvent.VK_LEFT){
        player.left = true;
    }

    if(e.getKeyCode() == KeyEvent.VK_W || e.getKeyCode() == KeyEvent.VK_UP){
        player.up = true;

        if(gameState == "Menu"){
            menu.up = true;
        }
    }else if(e.getKeyCode() == KeyEvent.VK_S || e.getKeyCode() == KeyEvent.VK_DOWN){
        player.down = true;

        if(gameState == "Menu"){
            menu.down = true;
        }
    }
```

Fonte: Elaboração própria.

A figura 9 mostra a utilização do método keyPressed e keyRelease, onde pode-se controlar os inputs e outputs do jogador com variáveis booleanas.

Após isso é necessário criar as interações através das colisões que o personagem faz com as entidades dentro do cenário. Isso é necessário para que o personagem não atravesse paredes ou objetos sólidos no jogo.

Figura 10 – Classe que cria um Tile de colisão dentro jogo.

```
public class Colision extends Tile {

    public Colision(int x, int y, BufferedImage tile) {
        super(x, y, tile);

    }

}
```

Fonte: Elaboração própria.

A Figura 10, mostra a classe responsável de receber as imagens que terão colisão na composição do cenário do jogo. Essas imagens podem ser paredes, árvores, postes, portas, etc. Objetos estáticos no cenário que não serão removidos de suas posições.

Figura 11 Fonte: Elaboração própria.

```
public static boolean isFree(int xnext, int ynext){
    int x1 = xnext / Tile_Size;
    int y1 = ynext / Tile_Size;

    int x2 = (xnext+Tile_Size-1)/ Tile_Size;
    int y2 = ynext / Tile_Size;

    int x3 = xnext / Tile_Size;
    int y3 = (ynext+Tile_Size-1) / Tile_Size;

    int x4 = (xnext+Tile_Size-1) / Tile_Size;
    int y4 = (ynext+Tile_Size-1) / Tile_Size;

    return !((tiles[x1 + (y1 * Mapas.WIDTH)] instanceof Colision) ||
            (tiles[x2 + (y2 * Mapas.WIDTH)] instanceof Colision) ||
            (tiles[x3 + (y3 * Mapas.WIDTH)] instanceof Colision) ||
            (tiles[x4 + (y4 * Mapas.WIDTH)] instanceof Colision));
}
```

Fonte: Elaboração própria.

Um método então é criado, como mostra a Figura 11, para analisar os locais de colisão dentro do mapa do jogo. Esse método é chamado quando o personagem se movimenta como vimos na Figura 6.

A partir daí, é disposto um cenário de acordo com a ideia do desenvolvedor como foi elaborado no GDD e protótipo, e adequado ao jogo, adicionando os detalhes da arte que foi escolhida para o jogo. Nesse jogo, foi escolhido pixel arte para criar um visual retrô. Não há desvantagens em se trabalhar com pixel arte, pois ela é simples e fácil de se implementar e sempre deixa um visual satisfatório ao final do projeto.

O projeto foi finalizado com a classe que responsável do som do jogo. Nesse caso há apenas duas músicas de fundo. A classe e seus métodos são mostrados nas figuras 12 (a), (b), (c) e (d).

Figura 12: (a) Classe responsável pela música do jogo e seus métodos (b),(c),(d).

Fonte: Elaboração própria.

A trilha sonora de um jogo é de extrema importância e pode determinar o sucesso do mesmo no lançamento. Para um projeto independente isso pode ser mais difícil, mas pode-se encontrar trilhas e efeitos sonoros, gratuitos, em sites como https://itch.io/, assim como artes para o jogo.

Grandes empresas como a Nintendo e Sega fizeram grandes investimentos nessa área, fazendo com que a primeira fase de Super Mario Bros e Green Hill Zone de Sonic The Hedgehog tornassem icônicas no mundo dos games. Em muitas de suas trilhas para Sonic, a Sega teve participação do grande astro Michael Jackson, que era

um grande entusiasta dos games e fã da Sega. A figura 13 mostra os personagens principais dos jogos.

Figura 13 – Sonic (à esquerda) e Mario (à direita).

Fonte: https://www.deviantart.com/legend-tony980/art/Mario-and-Sonic-395443695

CONCLUSÃO

A parte principal desse projeto está implementa, como trabalhos futuros pretende-se implementar os coelhos e a possível criação de um sistema simples de combate contra alguns inimigos para aprimorar a dificuldade do jogo.

Pode-se concluir que este artigo corrobora que a linguagem Java se apresenta eficaz no processo de desenvolvimento de jogos, embora o seu propósito não seja esse, assim como o de qualquer outra linguagem de programação. Ela possui muitas bibliotecas que podem ajudar no controle dos processos e sua documentação é de fácil disponibilidade e entendimento, possuindo exemplos de classes prontas.

Mas o melhor estudo sobre jogos, ainda é, e sempre será jogar videogames, ou mesmo jogos de qualquer outro tipo. Nolan Bushnell (2021), fundador da Atari, trabalhou em um parque de diversões e viu como as pessoas jogavam os jogos naquele lugar e com qual frequência o faziam. Quando lançou o Computer Space,

entendeu que o axioma básico para um bom jogo é: fácil de aprender, difícil de dominar. Outro grande Game Designer é Shigeru Miyamoto, criador de Donkey Kong, Super Mario Bros, The Legend of Zelda, Star Fox, F-Zero e Pikmin. E ele diz que: “um jogo atrasado pode ser, eventualmente, bom, mas um jogo apressado será ruim para sempre”.

REFERÊNCIA

DANKI CODE. **Desenvolvimento de Games.** Disponível em: <https://cursos.dankicode.com/curso-dev-games >. Acesso em: 01 fev. 2020.

NOVAK, J. Desenvolvimento de Games Tradução da 2ª edição Norte-Americana. Tradução Pedro Cesar de Conti, Revisão Técnica Paulo Marcos Figueiredo de Andrade. São Paulo: Cengage Learning, 2017.

JAVA. **Java™ Platform, Standard Edition 8 API Specification.** Disponível em: <https://docs.oracle.com/javase/8/docs/api/>. Acesso em 20 jun. 2021.

Bushnell, N. A Revolução dos Games. Disponível em: <https://www.youtube.com/watch?v=jnvm7sXrrFk>. Acesso em 20 jun. 2021.