

JOGOS DIGITAIS

Princípios, conceitos e práticas

Luiz Fernando Reinoso
Giovany Frossard Teixeira
Renan Osório Rios



Edifes



Luiz Fernando Reinoso
Giovany Frossard Teixeira
Renan Osório Rios

JOGOS DIGITAIS

Princípios, conceitos e práticas



Edifes

Vitória, 2018



Edifes

Editora do Instituto Federal de Educação,
Ciência e Tecnologia do Espírito Santo
R. Barão de Mauá, nº 30 – Jucutuquara
29040-689 – Vitória – ES
www.edifes.ifes.edu.br
editora@ifes.edu.br

Reitor

Jadir Jose Pela

Pró-Reitor de Administração e Orçamento

Lezi José Ferreira

Pró-Reitor de Desenvolvimento Institucional

Luciano de Oliveira Toledo

Pró-Reitora de Ensino

Adriana Pionttkovsky Barcellos

Pró-Reitor de Extensão

Renato Tannure Rotta de Almeida

Pró-Reitor de Pesquisa e Pós-Graduação

André Romero da Silva

Coordenador da Edifes

Giovani Zanetti Neto

Conselho Editorial

Glória Maria de Farias Viegas Aquije
Felipe Zamborlini Saiter (suplente)
Francisco de Assis Boldt
Pedro Vitor Morbach Dixini (1º Suplente)
Ediu Carlos Lopes Lemos (2º Suplente)
Aldo Rezende
Maria das Graças Ferreira Lobino (suplente)
Karine Silveira
Nelson Martinelli Filho (suplente)
Marize Lyra Silva Passos
Viviane Bessa Lopes Alvarenga
Rossanna dos Santos Santana Rubim (suplente)

Revisão de texto

Roberta Patrocínio de Amorim

Projeto Gráfico e Diagramação

Semíramis Aguiar de Oliveira Louzada

Imagem da Capa

<http://opengameart.org/users/kenney>

Dados Internacionais de Catalogação na Publicação (CIP)

Bibliotecária Rossanna dos Santos Santana Rubim – CRB6- ES 403

E373 Reinoso, Luiz Fernando
 Jogos digitais : princípios, conceitos e práticas / Luiz Fernando Reinoso, Giovany Frossard
 Teixeira e Renan Osório Rios. – Vitória, ES : Edifes, 2020.
 140 p. : il.

ISBN: 978-85-8263-357-1 (e-book.).

1. Jogos eletrônicos – Desenvolvimento. 2. Jogos eletrônicos – Programas de computador. I.
Reinoso, Luiz Fernando. II Teixeira, Giovany Frossard. III. Rios, Renan Osório. IV. Título.

CDD 22 – 004.4

@ 2017 Instituto Federal do Espírito Santo

Todos os direitos reservados.

É permitida a reprodução parcial desta obra, desde que citada a fonte.

O conteúdo dos textos é de inteira responsabilidade do autor.

Sumário

Lista de figuras	07
Sobre os autores	10
Prefácio	12
Agradecimento	17
1. Introdução	18
2. Jogos digitais	19
2.1. Definições	21
2.2. Ludificação e gamificação	21
2.3. Serious Games: games e aprendizagem	24
2.4. Exercícios	25
3. Metodologias e técnicas de desenvolvimento para jogos	26
3.1. Brainstorm	28
3.2. Idea Sheets	29
3.3. Flowcharts	31
3.4. MDA Framework	32
3.5. Design/Play/Experience Framework	35
3.6. Desenvolvimento ágil de software	39
3.7. Processo de software pessoal (PSP)	40
3.8. UML – Diagrama de estado	41
3.9. Exercícios	44
4. Design de Jogos	45
4.1. Conceito de um game	45
4.2. Mecânica de um jogo	47
4.3. Regras e balanceamento	47
4.4. Roleplaying e motivação do personagem	48
4.5. Explicação vs imaginação	49
4.6. Sinopse e história	50
4.7. Character Brief	50
4.8. Etapas e processos	51
4.9. Game Design Document	52

4.10. Game Design Canvas	52
4.11. High Concept for Flow	53
4.12. Profissões	54
4.13. Exercícios	54
5. Desenvolvimento de Jogos	56
5.1. Arquitetura de software	56
5.2. Teste de Desempenho	58
5.3. Postmortem	62
5.4. Exemplo - Buga: A aventura de um Neandertal	62
5.5. Conhecendo as ferramentas	64
5.6. Pixel arte com Gimp	64
5.6.1. Interface	64
5.6.2. Configuração para pixel arte	66
5.6.3. Desenhando pixel arte	68
5.7. Arte vetorial com Inkscape	70
5.7.1. Interface	71
5.7.2. Criando formas no Inkscape	72
5.7.3. Rasterizando imagens com Inkscape	75
5.8. Conhecendo o Godot Game Engine	76
5.8.1. Nódulos ou Nó	77
5.8.2. Cenas	78
5.8.3. Configurando um novo projeto	78
5.8.4. Editor	80
5.8.5. Criando nossa primeira cena	82
5.8.6. Configurando um projeto	85
5.8.7. Instâncias	86
5.8.8. Linguagem de Design	89
5.8.9. Aprendendo programação com GDScript	90
5.8.10. Programando nossa primeira cena	91
5.8.11. Manipulação de um sinal	93
5.8.12. Processamento	95
5.8.13. Grupos	96
5.8.14. Funções substituíveis	97
5.8.15. Criando nós	98
5.8.16. Instanciando cenas	99

6. Desenvolvendo o game 2D: ‘Astra’ - O herói do espaço	100
6.1. Configurando o projeto	101
6.2. Script de controle	102
6.3. Criando nossa nave	103
6.4. Programando nossa nave	104
6.5. Criando nossos tiros para a nave	105
6.6. Atirando com a nave	106
6.7. Criando um meteoro	108
6.8. Spawner de meteoros	109
6.9. Colisões	109
6.10. Identificando objetos e atirando neles	111
6.11. Plano de fundo	113
6.12. HUD	115
6.13. Exercícios	116
7. Atmosfera musical	118
7.1. BFXR	118
7.2. Audacity e LMMS	120
7.3. Inserindo músicas e sons	121
7.4. Exercícios	123
8. Jogos Móbile	124
8.1. Controles touch screen	125
8.2. Exercícios	126
9. Conclusão	127
Apêndice	128
Apêndice A. Goa’s reign paths, paper game	129
Apêndice B. Game Design Document	131
Apêndice C. High Concept for Flow	136
Referências	137

Lista de figuras

Figura 1. Captura da proteção de tela Johnny Castway do Windows	12
Figura 2. Jogo de Senet	19
Figura 3. Estudantes testando o jogo “Spacewar!”	20
Figura 4. Pepsi Man	23
Figura 5. Pirâmide de equilíbrio para um projeto de jogo	27
Figura 6. Fluxo de um brainstorm e seu ciclo	28
Figura 7. Fluxograma de um jogo	31
Figura 8. Produção e consumo de artefatos de um jogo	32
Figura 9. Componentes de consumo de um jogo	33
Figura 10. Contrapartes a partir do consumo de um jogo	33
Figura 11. Design e jogador em diferentes perspectivas	33
Figura 12. O coração do design dos “Serious Games”	36
Figura 13. DPE Framework	36
Figura 14. Processo iterativo de design	37
Figura 15. DPE Framework expandida	37
Figura 16. Processo interativo do desenvolvimento ágil de software	39
Figura 17. Ciclo para obtenção de resultados no DAS	40
Figura 18. Diagrama de estado exemplo	42
Figura 19. Diagrama de estado	43
Figura 20. Representação da visão funcional do game design para um jogo eletrônico	46
Figura 21. Representação visual do balanceamento de regras de um jogo	47
Figura 22. Exemplo de um balanceamento ruim	48
Figura 23. Game Design Canvas	53
Figura 24. Arquitetura de sistema cliente-servidor	57
Figura 25. Representatividade do teste de performance e sua ação	59
Figura 26. Menu de contexto para seleção do Inspect (Inspetor/Inspeccionar)	59
Figura 27. Teste de performance com o Google Inspect	60
Figura 28. Captura de tela do inspetor modo device	61
Figura 29. Captura de tela do Inspetor em visão modo device	61
Figura 30. Captura de tela do game BUGA! A aventura de um Neandertal	63

Figura 31. Interface em janela única do GIMP	65
Figura 32. Cirando uma nova cena no GIMP	66
Figura 33. Configuração do lápis para pixel arte	67
Figura 34. Janela de Configuração de grade	67
Figura 35. Base ou esboço do desenho	68
Figura 36. Esboço colorido com cores base	69
Figura 37. Desenho com iluminação	69
Figura 38. Figura realçada com exposição de cores	70
Figura 39. Interface do Inkscape	72
Figura 40. Formas geradas no Inkscape	73
Figura 41. Passos para criação de um emotico	74
Figura 42. Janela de operações para rasterização	75
Figura 43. Imagem resultante da rasterização	76
Figura 44. Estrutura de nódulos no Godot Engine	77
Figura 45. Hierarquia de nós para uma cena no Godot Game Engine	78
Figura 46. Janela de gerenciamento de projetos, aba Project list	79
Figura 47. Janela de gerenciamento de projetos, aba Templates	79
Figura 48. Janela de criação de um novo projeto	80
Figura 49. Editor do Godot	80
Figura 50. Painel Scene	82
Figura 51. Painel de criação de nós	83
Figura 52. Campo “Text” do Label	84
Figura 53. Janela Save Scene As	84
Figura 54. Janela de configurações do projeto	85
Figura 55. Instâncias e cenas	86
Figura 56. Configuração da cena bola	87
Figura 57. Configuração da cena arena	87
Figura 58. Configuração da cena arena com instâncias de bola	88
Figura 59. Esquema de um jogo de tiro sob ótica da linguagem de design do Godot	89
Figura 60. Diagrama de jogo complexo sob a óptica de linguagem de design do Godot	90
Figura 61. Painel com rótulo e botão	91
Figura 62. Menu de anexação de script a um nó	91
Figura 63. Janela de configuração de Script	92

Figura 64. Modo de edição de Scripts	93
Figura 65. Painel de conexão de Signals	94
Figura 66. Painel Node com a opção Groups	96
Figura 67. Sprites para o game Astra	100
Figura 68. Configurações de controles do Godot	101
Figura 69. Criação de Singleton	102
Figura 70. Estrutura da cena nave.tscn	103
Figura 71. Inserindo nave por instância no centro da cena principal	103
Figura 72. Posicionamento e estrutura de cena para o tiro	105
Figura 73. Estrutura de cena do meteoro	108
Figura 74. Modificação na estrutura da cena do meteoro	110
Figura 75. Desenhando colisão com o nó CollisionPolygon2D	110
Figura 76. Alterando o tipo de um nó	110
Figura 77. Criação de um Signal via UI	111
Figura 78. Composição de um background com parallax	113
Figura 79. Inserindo espelhamento com a opção Mirroring	114
Figura 80. Importação de fontes	115
Figura 81. Selecionando uma fonte para um rótulo	116
Figura 82. Exercício de desenho com Inkscape	117
Figura 83. BFXR	118
Figura 84. Mixer de sons do BFXR	119
Figura 85. Interface do Audacity	120
Figura 86. Interface do LMMS	121
Figura 87. Criando biblioteca de sons	122
Figura 88. Sample Library de músicas	122
Figura 89. Controles Touch	125
Figura 90. Configurando o Touch Screen	125
Figura 91. Ativando a emulação para os touch	126
Figura 92. Tabuleiro Goa's Reign Paths	130

Sobre os autores

	<p>Luiz Fernando Reinoso, formado em Tecnologia de Análise e Desenvolvimento de Sistemas pelo Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo (IFES), campus Santa Teresa(2012); pós-graduado em Novas tecnologias na educação pela Escola Superior do Brasil (ESAB), em Vila Velha (2014); e mestre em Informática na educação, pelo LIEd/UFES, Vitória (2016). Em relação a jogos, publicou sete games em eventos de Game Jam internacionais e artigos em eventos e periódicos nacionais da área. Professor desde 2013, iniciando na rede estadual de ensino e em cursos do Pronatec. Professor EBTT (Ensino Básico, técnico e tecnológico) Substituto do Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo, campus Colatina até junho de 2018. Professor EBTT efetivo no Instituto Federal do Pará, campus Óbidos até a presente data.</p> <p>Currículo: http://lattes.cnpq.br/5476483807420402</p> <p>Sites: https://sites.google.com/site/portifolioluizfreinoso http://cuppixel.com.br</p>
	<p>Giovany Frossard Teixeira, graduado em Ciência da Computação pela UFES (2004), mestre em Informática pela UFES (2006) e Doutor em Educação pela UniNorte (2015). Professor Efetivo do Instituto Federal do Espírito Santo, campus Colatina, nos cursos Técnico em Informática, Tecnólogo em Redes de Computadores e Bacharel de Sistemas de Informação. Tem experiência na área de Ciência da Computação, com ênfase em Linguagens de Programação, Otimização e Software Básico. Na área de educação, tem foco de estudo em ensino a distância e em avaliação da aprendizagem.</p> <p>Currículo: http://lattes.cnpq.br/7406806998563478</p>



Renan Osório Rios, técnico em Informática pelo Centro de Federal de Educação Tecnológica do Espírito Santo (CEFET-ES), graduado em Sistemas de Informação pelo Centro Universitário do Espírito Santo (UNESC), mestre em Modelagem Matemática e Computacional pelo Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) e doutor em Ciências da Educação pela Universidade do Norte (Uninorte).

Currículo:

<http://lattes.cnpq.br/3555360133532677>

Prefácio

Minhas primeiras impressões sobre criação de jogos não poderiam ser menos otimistas do que as pertencentes a você leitor que está permitindo que minha produção faça parte de sua jornada, houve muitas dúvidas, receios e até mesmo dificuldades em ir em frente. Iniciei a caminhada no mundo dos jogos digitais acreditando em crescimento científico, acadêmico e, claro, profissional. Não estive enganando-me em nenhum momento, depois de participar de edições do Global Game Jam, Ludum Dare e outros eventos de Game Jam¹ e projetos com amigos profissionais da educação e desenvolvedores independentes, notei o quanto são vastos os ramos de um profissional ou mesmo conhecedor de jogos digitais, estes, por sua vez, abrem seus olhos para o ramo criativo.

A primeira paixão por computador, surgiu quando eu ainda era novo e meu pai mostrou-me uma proteção de tela no Windows 95 do Johnny Castway, tratava-se de o naufrago. Diante daquilo, percebi que adorava computador e suas possibilidades, meu primeiro interesse foi o paint² e suas ferramentas de desenho e escrita, afinal, podia fazer o que minha mente possibilitasse. De fato, inicialmente não pensava em jogos diretamente, mas em animações, adorava histórias.



Figura 1. Captura da proteção de tela Johnny Castway do Windows

Fonte: TechTudo (2010).

Meus trabalhos e pesquisas acadêmicas fundamentaram-se com a paixão aos jogos e desenhos animados. As primeiras publicações em anais e periódicos com o uso das tecnologias de jogos em ambiente educacional surgiram durante a graduação em “Análise e Desenvolvimento de Sistemas”, em meados de 2012 e 2013.

Em constante evolução, atualmente, invisto em estudos em inteligência arti-

1 Game Jam – são eventos que ocorrem na modalidade presencial ou remota (online) em que desenvolvedores, objetivando criar um jogo em um curto intervalo de tempo, reúnem-se.

2 Paint – Software de edição gráfica da Microsoft®.

ficial, em tecnologias com realidade virtual³ e aumentada⁴ e em jogos assistivos⁵ (bem como tecnologias assistivas⁶), esse último estudo permite a pessoas com deficiências diversas a interagir com um jogo eletrônico.

Hoje, sou professor e ministro cursos de jogos digitais nas instituições de ensino que faço parte, graças à colaboração e aos bons resultados de trabalhos desenvolvidos nos últimos anos, estamos caminhando para tornar a indústria de jogos em nosso país mais visível e interessante, despertando o interesse dos mais variados tipos de pessoas, por isso, faça bom uso deste material, tem muito trabalho e dedicação minha e dos colegas envolvidos.

- *Luiz Fernando Reinoso*

Este material foi criado como um recurso de nível profissional e acadêmico, expondo metodologias, conceitos e prática para desenvolver um jogo digital. O livro foi produto de experiências, leituras e referências importantes, portanto, é crucial que você leia todas as notas e dicas, faça as atividades e siga as instruções criadas, o intuito é fornecer formação suficiente para sua independência criativa.

O objetivo principal é introduzir os leitores no desenvolvimento de jogos, dando capacitação mínima para compreender o universo dos games (Jogos digitais/eletrônicos). Trabalharemos diversos conceitos e técnicas que vão possibilitar a criação de jogos digitais e o entendimento do que é um jogo, suas características, história, carreiras, processo de concepção e designer, prototipagem, funcionamento de dispositivos móveis, marketing e comércio digital.

O estudante, com uso deste material, desenvolverá habilidades de implementação de jogos digitais, podendo programar, criar arte e inserir toda a atmosfera musical. Logo, englobaremos as atividades mais comuns na prática de desenvolvimento de um jogo. O livro está dividido da seguinte forma:

Capítulo 1: Introdução

Expõe-se as instruções de uso, as impressões e apresentação formativa do conteúdo abordado, a descrição dos sinais e as caixas de diálogo apresentadas nos demais capítulos e sessões. Capítulo destinado à orientação de professores e alunos, com defi-

3 Realidade virtual – é uma tecnologia de interface que permite a um usuário interagir com um sistema operativo. O objetivo é reproduzir a sensação de realidade para uma pessoa, permitindo obter interação como uma de suas realidades temporais.

4 Realidade aumentada – integra a visão virtual e real. Geralmente ilustra o mundo real através de captura de imagem, processando-as e ampliando as suas capacidades.

5 Jogos assistivos ou jogos acessíveis (Assistive Games or Accessible Games) – são jogos produzidos para pessoas com deficiência, criados para incapacidades específicas ou gerais.

6 Tecnologias assistivas – toda tecnologia e/ou serviço desenvolvido para ampliar as habilidades funcionais de pessoas com deficiência.

nição das dinâmicas abordadas e acesso ao material digital dos projetos que acompanham este livro, disponíveis via internet.

Capítulo 2: Jogos Digitais

É apresentado um breve resumo da historicização⁷ dos jogos, abordando sua criação e uma introdução da era digital com o desdobramento dos jogos digitais. O capítulo segue expondo conceitos de uso no desenvolvimento com ludificação, gamificação e debate sobre como aprender com jogos.

Capítulo 3: Metodologias e técnicas de desenvolvimento para jogos

É ensinada a produção de artefatos importantes para criação de seu game, este capítulo é prático, contém diversas atividades e demonstrações de técnicas e metodologias aplicáveis e úteis para o desenvolvimento de um jogo.

Capítulo 4: Design de jogos

Este capítulo apresenta ao leitor o universo de criação e fundamentação de um jogo, a definição de conceitos gerais e de regras e balanceamento, o conhecimento de metodologias de desenvolvimento e a prototipagem de um game e sua documentação. O capítulo foca na criação de artefatos importantes para criar um jogo digital e em como criar um canal de comunicação entre membros do projeto.

Capítulo 5: Desenvolvimento de jogos

Estuda um projeto de exemplo, o game “BUGA!”, passando pela sua definição, arquitetura, testes de desempenho, além de conhecer a documentação pós-projeto, o postmortem. Neste capítulo, iremos iniciar o desenvolvimento da documentação dos projetos que acompanham este livro, replicando o que é exemplificado no “BUGA!”.

⁷ Historicização[s. f. – Ato ou efeito de colocar em contexto histórico, de historicizar (Dicionário Priberam da Língua Portuguesa, 2008-2013).

Capítulo 6: Desenvolvendo o game 2D: ‘Astra’ - O herói do espaço

Trabalha de maneira prática a criação de um game em 2D, fazendo uso das ferramentas Godot Game Engine, Gimp 2.6 e Inkscape.

Capítulo 7: Atmosfera musical

Expõe a criação de atmosfera musical e a utilização de ferramentas de autogeração, a partir das quais podemos adiantar a composição sonora de nossos jogos.

Capítulo 8: Jogos mobile

Evidencia as etapas de conversão dos jogos criados para que sejam habilitados para dispositivos móveis, o perfil desse tipo de jogador, as vantagens e desvantagens de um jogo móvel, o funcionamento de um mobile e suas tecnologias.

Capítulo 9: Conclusão

É apresentado um fechamento do livro com passos para dissolução da obra e de problemas encontrados na jornada pelos autores. Além de indicar pontos de vista e dicas para sua complementação didática, links importantes da internet e outras questões.

Referências

Livros, artigos, sites e mídias utilizadas para compor o livro, colaborações importantes, indicações e inserção de crédito a terceiros de maneira apropriada.

Ambiente virtual de aprendizagem (AVA)

Para melhor desenvolvimento com o livro, os autores criaram um ambiente virtual de aprendizagem, contendo materiais do livro, como músicas, sons, imagens, esclarecimentos e projetos.

Você não precisa estudar sozinho, além dos recursos ligitados, existem aqueles para interação assíncrona, como fóruns em que você pode postar dúvidas, conhecer novidades e interagir com outras pessoas. O ambiente disponibiliza atividades e complementações ao seu estudo. O endereço de acesso é: <https://www.cead.cuppixel.com.br/>

Para acessar a sala intitulada “LIVRO – Jogos Digitais: princípios, conceitos e práticas”, insira a senha “#QueroAprenderMais” (sem as aspas).

Dicas

Dica: As caixas de dicas compreendem conhecimentos dos autores para a evolução do leitor e para o aprimoramento de informações presentes na teoria desenvolvida. Sendo opcional ao leitor sua consideração.

Notas

Nota: As notas completam diretamente o que foi escrito, sendo diferentes das dicas no sentido de aperfeiçoarem o que foi colocado pelos autores diretamente e porque devem ser consideradas.

Códigos

São trechos de código de computador completos ou parciais.

Agradecimento

Agradeço primeiramente a Deus, pois tenho me colocado a segui-lo e com suas bênçãos tenho conseguido enxergar cada vez mais longe.

A meus pais que nunca desistiram de acompanhar a minha jornada, pelo amor e companheirismo eternos.

A minha noiva Suzane Zinger, que com paciência e em concordância com meu trabalho foi sempre compreensiva acerca de minha ausência e me confortara nos momentos de medo e descontentamento.

Aos meus amigos, alunos e colegas de trabalho, especialmente a Coordenadoria de Informática do IFES, campus Colatina, que apoiaram esta iniciativa.

- Luiz Fernando Reinoso

1. Introdução

Os novos modelos de negócio e de plataformas de distribuição de mídia digital, como a Google Play⁸, Apple Store⁹, Steam¹⁰ e outras, oferecem uma maneira diferenciada de comercializar a criatividade.

Entrar no mercado de jogos digitais fascina até mesmo quem não joga, pois envolve tecnologia e profissionalização de ponta. Diversos estudiosos e mestres da tecnologia envolvem-se profissionalmente com games.

Devemos entender os games como uma ponte entre o entretenimento e a ciência. Pois por trás das atividades divertidas que eles nos proporcionam, temos linguagens de programação, arte, música, literatura, ciências e outras. Um jogo é um produto multicultural, multidisciplinar e sociocultural, seja no panorama estritamente social, educacional ou de negócios.

Todavia, viver como um desenvolvedor de jogos é um desafio, principalmente em países subdesenvolvidos, onde, por vezes, o reconhecimento cultural e profissional de uma pessoa que se dedica a criar e a comercializar jogos digitais não é valorizado ou reconhecido em âmbito social ou mesmo político, tendo esta que se arriscar sem custeio, leis ou apoio.

Para avançar de maneira segura e com confiança nesse ramo, tem-se atualmente tecnologias acessíveis a desenvolvedores independentes, bem como metodologias e técnicas que permitem mensurar um projeto e assim fornecer dados para composição de uma representação orçamentária, custo de tempo, estatísticas de empenho e desempenho para um projeto.

Logo, lançando luz a essas possibilidades, resta entender como introduzir uma ideia dentro do mercado. Tal conhecimento é importante para ganhar tempo e economizar recursos, evitando, assim, desperdícios, ociosidade e principalmente frustração, que podem culminar na sua falência moral e de autoestima. Portanto, para começar algo, devemos ter a mente aberta e saber iniciar do zero, não pular etapas e aprender o necessário ou o suficiente para criar algo na medida.

Vamos aprender a desenvolver um jogo baseando-se em processos de engenharia de software e design, trabalhando desde a parte conceitual à prática, em que criaremos jogos nos moldes comerciais, focando no entendimento e no uso de técnicas de projeto e desenvolvimento, você poderá, a partir da experiência que teremos no livro, gerar seu produto de jogo inteiramente.

8 Google Play – Disponível em: <<https://play.google.com/store>>.

9 Apple Store – Disponível em: <<https://www.apple.com/itunes/>>.

10 Steam – Disponível em: <<http://store.steampowered.com/?l=portuguese>>.

2. Jogos digitais

A humanidade desenvolve jogos desde os primeiros registros históricos. Um dos jogos mais antigos que se tem registro é o Senet, apresentado na Figura 2. Jogo de Senet, encontrado em algumas relíquias arqueológicas egípcias que datam de 3500 anos antes de Cristo (THOMPSON, BERBANK-GREEN, CUSWORTH, 2007, p. 12). A humanidade tem utilizado jogos desde seus primórdios, objetivando tanto seu aperfeiçoamento cognitivo, quanto seu domínio sobre situações problema. Na Roma antiga, campeões de jogos eram valorizados e enriqueciam e não estamos falando apenas de gladiadores, mas de grandes comerciantes. Confuso?

Jogamos o tempo inteiro, o advento de jogos vai muito além do que imaginamos, jogos de cartas no computador, por exemplo, utilizam recursos muito semelhantes aos seres humanos, como probabilidade e estatística para obter a melhor jogada possível, vendedores também jogam conosco, tentando acertar nas propostas, diálogos, preços e negociações, tudo isso em sincronia correta o possibilita vender um produto, logo, ele ganha o jogo.



Figura 2. Jogo de Senet

Fonte: University of Waterloo (2010).

Durante a guerra fria (1947-1991), em meados dos anos 60, as pessoas viviam cercadas pelas cenas de horror e morte dos massacres que estavam ocorrendo no Vietnã e pela pressão global de uma guerra nuclear. Willy Higinbotham, um pesquisador da Brookhaven National Laboratories, no estado de Nova York, criou em suas horas vagas um game a partir de um osciloscópio, processado por um computador analógico um game de “tênis para dois” em que era possível jogar uma partida de tênis de forma virtual entre dois jogadores.

Consta no site do conceituado Massachusetts Institute of Technology (MIT) que o primeiro software de entretenimento criado com a função de ser um jogo na história foi o “Spacewar!”, desenvolvido em 1961 por Martin Graetz, Stephen Russell e Wayne

Wiitanen, que foram inspirados nos livros de ficção científica do autor (já falecido) E.E “Doc” Smith. (FIKSUSSA, 2012). Na Figura 3 temos uma foto de estudantes testando o game.



Figura 3. Estudantes testando o jogo “Spacewar!”

Fonte: Massachusetts Institute of Technology (2011).

2.1. Definições

Encontrar definições para a palavra “game”, dado que estamos estudando jogos eletrônicos de forma orientada, é um exercício abrangente, visto que existem muitas definições diferentes. De maneira simplória e geral, podemos ter a seguinte definição: “Um game pode ser entendido como qualquer interatividade que proporcione ao usuário, mesmo que limitadamente, controle sobre ações, ou seja, deixe-o ter poder de escolha em ambiente artificial” (REINOSO et al., 2012).

O conceito colocado é substancial, útil para este primeiro momento, mas existem jogos que podem modificar facilmente essa formulação. Os jogos evoluem muito rápido e tentar fechar um conceito em volta deles atualmente é difícil, principalmente se compararmos seu surgimento com outras áreas, como matemática e física, que são estudadas a mais de 10.000 a.C., estas têm conceitos muito mais definidos, enquanto os jogos digitais (games) constituem uma área nova para a humanidade que tem muito ainda a mostrar.

Para Mitchell e Savill-Smith (2004 apud SAVI; ULBRICHT, 2008) os jogos proporcionam uma experiência estética visual e espacial muito rica, sendo capazes de atrair os jogadores para um mundo fictício despertando o sentimento de aventura e prazer.

Puramente físico ou biológico? Huizinga (2001, p. 3-4) diz-nos que o jogo é mais do que um fenômeno fisiológico ou um simples reflexo psicológico, e que ultrapassa os limites da atividade puramente física ou biológica. É uma função significativa, isto é, encerra um determinado sentido. No jogo, existe alguma coisa “em jogo” que transcende as necessidades imediatas da vida e confere um sentido à ação. Todo jogo significa alguma coisa.

2.2. Ludificação e gamificação

Quando estamos criando algo (objeto) é comum querermos que a mensagem sobre o que estamos produzindo seja entendida, pois criar um canal de informação entre esse objeto de consumo e uma pessoa é necessário para que esta possa se apropriar deste.

A aplicabilidade da gamificação ou ludificação está em conceber um produto de software que satisfaça um objetivo corporativo, seja uma escola, empresa, governo ou qualquer tipo de instituição, utilizando a temática de um jogo digital.

Apesar das terminologias diferentes, fato que tem gerado debates entre muitos

autores, elas não se apresentam como um impedimento para você chegar a uma conclusão: ambas podem ser aplicadas ou mesmo confundidas com Serious Games (Jogos sérios, veremos a seguir) ou games publicitários. Porém existe insistência em dividir a abordagem em termos. Utilize a expressão que te satisfaça, pesquise além do escopo deste livro, por hora utilizaremos o termo gamificação.

A gamificação faz uso de técnicas de design de jogos que utilizam mecânicas e pensamentos orientados para enriquecer contextos diversos, estes, normalmente, não relacionados a jogos na maioria das vezes. Tipicamente emprega-se a gamificação em processos e aplicações com o objetivo de incentivar as pessoas a adotarem esses contextos ou de influenciar a maneira como são usados.

Nesse processo, podemos criar afetividade entre o consumidor e um objeto que produzimos, um teste interessante que você pode fazer mentalmente é pensar, neste exato momento, em “refrigerante”, aposto que você pensou em uma marca específica e bastante famosa e conseguiu visualmente enxergá-la, você acabou de consumir um produto de marketing (objeto) dessa empresa.

Obviamente, após o teste retratado, você deve estar pessimista sobre essa introdução ao assunto, quando conseguimos esse tipo de efeito prolongado de consumo com um game, ou seja, quando a pessoa vê ou imagina algo em referência ao nosso game, existe um efeito colateral da gamificação, assim, vislumbra-se aspectos e informações do mundo real que têm alusão dentro dos jogos e vice-versa.

O exemplo representado na Figura 4 apresenta o game “Pepsi Man”, desenvolvido em 1999 para o console Playstation® One¹¹, o game objetiva apresentar e incentivar o consumo de produtos da Pepsi®, empresa norte-americana fornecedora de refrigerante.

A gamificação está presente no game apresentando a marca ao povo japonês como “Receptiva”, pois, imitando as propagandas da época, o personagem do game, nomeado “Pepsi Man”, entrega garrafas de refrigerante na casa das pessoas e corre rápido, além disso, mesmo se machucando, cumpre com a responsabilidade de entregar a bebida e fazer as pessoas felizes.

11 Playstation One – Console de videogame desenvolvido pela Sony no Japão em 1994, sua produção parou em 2006, vendendo mais de 100 milhões de unidades.



Figura 4. Pepsi Man

Fonte: Google Images (2016).

Essa prática é muito utilizada por empresas para ampliar a visibilidade de suas marcas e serviços, existem cursos para aprimoramento específicos em gamificação, entender como aplicar essa técnica pode gerar um grande ganho de receita para um jogo, como popularizar personagens, conceitos ou mesmo venda de produtos terceirizados da sua proposta, a exemplo, você tem diversos games que vendem camisetas, canecas, chaveiros, bonecos colecionáveis e outros itens advindos da popularização do jogo.

Além das técnicas de design, a gamificação pode utilizar componentes da mecânica de jogos (iremos estudar melhor adiante) em contexto sério, como por exemplo, o uso de mecânicas de jogos para motivar estudantes a buscarem compreensão de assuntos complexos como Matemática e Computação, ou seja, podemos usar componentes lúdicos, motivadores e facilitadores do aprendizado presentes nos jogos em ambiente escolar, empresarial, entre outros.

Logo, projetos que utilizam em sua modelagem técnicas de gamificação procuram além de gerar um produto, trazer expressões concretas ou mesmo subjetivas, acima do próprio produto. Existem treinamentos e cursos para aperfeiçoamento destas práticas de desenvolvimento.

Dica: Para ampliar suas habilidades e mesmo obter capacitação em gamificação, a rede de cursos online Coursera oferece o curso “Gamificação”, com o professor Kevin Werbach, desenvolvido pela Universidade da Pensilvânia. Consulte no link: <https://www.coursera.org/learn/gamification>.

2.3. Serious Games: games e aprendizagem

O termo Serious games (Jogos sérios) surge justamente para identificar uma categoria de jogos focada em passar algo a mais ao jogador, algo que de fato proporcione melhoramento de habilidades e competências e que, até mesmo, possa ajudá-lo a encontrar o significado para algo na vida real. Trata-se de softwares que fazem uso dos princípios dos jogos convencionais, porém embutem um conteúdo com o intuito de transmitir uma informação, seja de cunho educativo, publicitário, informativo ou de treinamento.

Clark Abt descreveu um esboço do termo ainda na década de 70 em seu livro *Serious Games*, publicado pela Viking Press. A ideia foi primeiramente utilizada em jogos de cartas, mas também bastante aplicável a jogos eletrônicos. Segundo Michael e Chen (2006, pg. 21):

Os jogos podem ser jogados a sério ou de forma casual. Estamos preocupados com jogos sérios no sentido de que estes jogos são explícita e cuidadosamente pensados para fins educativos e não pretendem ser jogado principalmente por diversão. Isso não significa que os jogos não são sérios, ou não deverão ser divertidos.

Jogos desse tipo tem recebido forte apoio de governos ao redor do mundo, muitos países investem bastante na abordagem com intuito de adiantar e melhorar o ensino, reduzir custos com treinamento profissional e até mesmo ampliar a área de testes, uma vez que games são fortes em tecnologias gráficas para representação de simulações.

Nota: Matar zumbis pode te tornar um ser humano melhor sabia? Uma pesquisa de Daphne Bevelier e C. Shawn Green, publicado na revista *Scientific American* de Agosto de 2016 explica melhor essa questão, leia em: <https://www.scientificamerican.com/article/serious-brain-benefits-from-computer-games/>.

2.4. Exercícios

1. Leia o texto “Opinião: Mi-mi-mi meus jogos não são cultura, e eu não sou culto” no link: <https://jogos.uol.com.br/ultimas-noticias/2013/04/05/opiniaio-mi-mi-mi-meus-jogos-nao-sao-cultura-e-eu-nao-sou-culto.htm>. Debate acerca dos jogos e sua difusão cultural;

Dica: Não estude sozinho, fique a vontade e poste sua posição no fórum da sala virtual criada para este livro, conforme apresentado na introdução.

2. Faça avaliação e revisão de filmes abaixo. O intuito da atividade é você ter referência de design e de como jogos influenciam pessoas e o dia a dia;

2.1. Assista “Spore, o nascimento de um jogo”, uma palestra de Will Wright sobre o game Spore, uma bela apresentação de conceito e design de um jogo criativo. Link: https://www.ted.com/talks/will_wright_makes_toys_that_make_worlds?language=pt-br;

2.2. Assista “Jane McGonigal: Jogando por um mundo melhor”, uma palestra da pesquisadora Jane McGonigal, uma apresentação de um ponto de vista muito rico e reflexivo acerca das possibilidades que temos com os jogos. Link: https://www.ted.com/talks/jane_mcgonigal_gaming_can_make_a_better_world?language=pt-br.

3. Procure pela internet alguns nomes que listarei abaixo, são designers de jogos conhecidos e famosos por suas produções. O intuito dessa atividade é você conhecer a história e os conceitos criados pelos autores para que você crie referências interessantes para seu conhecimento;

- Will Wright;
- Sid Meier;
- Richard Garriott;
- Toshihiro Nishikado;
- Toru Iwatani.

4. Defina em suas palavras, o que é um game e se esforce para defender e diferenciar o que seria um “jogo” e um “jogo eletrônico”;

5. Busque dois jogos digitais que fazem uso de gamificação e tente retratar como o fazem.

3. Metodologias e técnicas de desenvolvimento para jogos

Provavelmente, o primeiro capítulo que você esperava era o “design de jogos”, mas tão importante quanto compreender a estrutura de um jogo é entender sua arquitetura e como projetá-lo. Um jogo digital ainda é um software, logo, muitas técnicas e metodologias aplicadas à construção de sistemas computacionais influenciam e são subvertidas para o desenvolvimento de jogos digitais.

Estudaremos neste capítulo algumas técnicas e metodologias para “projetar” jogos digitais, um projeto, qualquer que seja, tem de ser delineado no mais alto nível possível, ou seja, deve possibilitar clareza no entendimento de sua ideia por completo, sem deixar escapar detalhes, principalmente se tiver uma equipe envolvida.

Conheceremos o uso dos chamados “Idea Sheets” (Folha de ideias) que são documentos que expressam as ideias principais de um game. Podemos criar esses artefatos com materiais mais simples como post-it¹². Aprenderemos a criar e utilizar “Flowcharts” (Diagramas de fluxo), através desses gráficos é possível visualizar as mudanças e o fluxo de um jogo por completo. Uma das ferramentas metodológicas mais interessantes que iremos estudar neste livro é o “MDA Framework”, por meio desse Framework¹³, podemos analisar como o design, a mecânica e a afetividade estética de um jogador estão balanceadas e como tirar proveito disso.

As metodologias de desenvolvimento de software são utilizadas com intuito de aperfeiçoar e tornarem-se mais produtivas o processo de definição, desenvolvimento, distribuição, suporte e manutenção de um software ou um jogo. Estudaremos ao longo deste livro o “Desenvolvimento ágil de software” e o “Processo de Software pessoal” (PSP). A última técnica estudada neste capítulo submete-se a ideia da Linguagem de Modelagem Unificada (do inglês, UML – Unified Modeling Language) que é o diagrama de estado e de visão funcional, a técnica passada para você é a composição gráfica com os atores e suas operações/responsabilidades dentro de um game.

Posterior a este capítulo, você irá aprender conceitos de design para projetos de jogos e estará instruído para criar documentação técnica específica para jogos, o GDD, “documento de design de jogos” (do inglês, GDD – Game Design Document). Por hora, entenderemos de forma prática e teórica o gerenciamento, o ciclo de vida e os processos da execução de um projeto.

____ Uma forma de enxergar como temos que equilibrar um projeto de um jogo é exem-

12 Post-it – Post-it ou folhas adesivas, são papéis adesivos que você pode destacar e colocar em locais diversos. É um material de escritório fácil de encontrar em papelarias.

13 Framework – No design, um framework dita o fluxo de controle do projeto, esclarecendo as camadas, os aspectos e os processos de criação. O termo pode ser encontrado em desenvolvimento de software como biblioteca de recursos (códigos de computador) e em outras áreas.

plificada na Figura 5, na qual equilibramos quatro processos principais: Planejamento (1), Coordenação (2), Execução (3) e Resultados (4). Veja que, desde a base até o topo da pirâmide, temos que construir uma base bem sólida que comporte as camadas superiores. O conhecimento dessa pirâmide de equilíbrio serve para todas as técnicas de processos deste livro, mantenha-a em mente.

Nota: As metodologias e técnicas apresentadas a seguir foram adaptadas para fácil compreensão e uso, logo, mantenha atenção às notas e dicas apresentadas adiante, elas irão ajudar em seu aprofundamento nas temáticas abordadas.

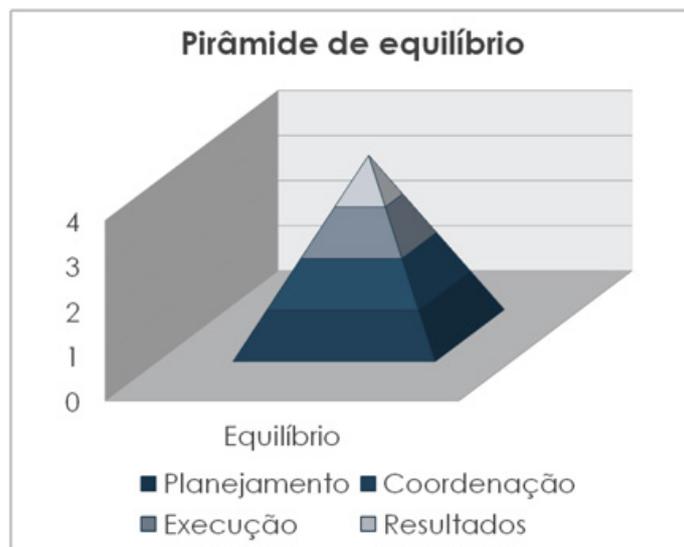


Figura 5. Pirâmide de equilíbrio para um projeto de jogo

Fonte: próprio autor.

Em resumo, tudo que é bem planejado é mais bem coordenado e executado, os resultados aparecerão se estas camadas forem bem estruturadas e complementares, mas claro que qualquer uma delas, caso esteja mal elaborada ou tenha sido construída sem cuidado, afetará a camada superior diretamente, inflando nos resultados ao final da pirâmide.

3.1. Brainstorm

Um brainstorm, em tradução literal significaria “tempestade de ideias”, ocorre de forma bem comum em nossa área de desenvolvimento de jogos, trata-se de momentos onde grupos de pessoas em prol de um objetivo comum podem expor todas suas ideias, mesmo as mais esdrúxulas (AHMAD et al., 2017).

A Figura 6 apresenta o fluxo do brainstorm. Perceba que o ciclo de vida é contínuo, as ideias e soluções podem evoluir constantemente a cada interação, logo, a duração ou quantidade de informação necessária para encerramento de uma reunião com o brainstorm varia de acordo com o tipo de projeto e objetivos a serem alcançados.

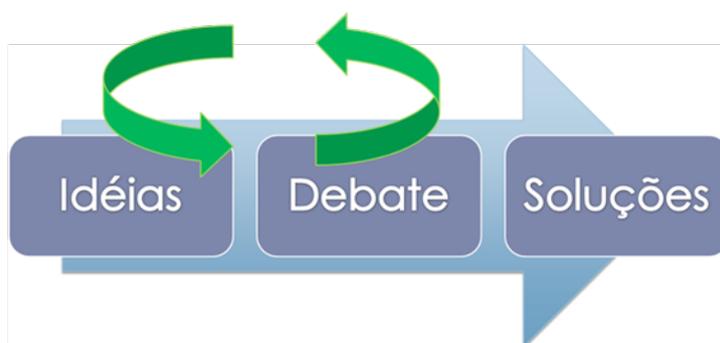


Figura 6. Fluxo de um brainstorm e seu ciclo

Fonte: próprio autor.

O choque entre ideias pode lapidá-las e gerar novas de maneira bem permissiva e livre. É interessante tirar tempo para fazer isso, expor suas ideias, permitindo que outras pessoas façam alterações nestas e exponham suas próprias ideias para que você faça o mesmo. Por mais informal que essa técnica possa parecer, ela é extremamente útil e vale a pena ser registrada, você pode criar uma espécie de “ata” para esse tipo de reunião.

Uma ata é um documento (existem diversos modelos na internet) em que se registram todos os benefícios, soluções, ideias e conclusões de uma reunião ou conversa, o objetivo é manter um registro para ajudar-nos a não esquecer nada após um evento.

3.2. Idea Sheets

O uso de Idea Sheets ou, simplesmente, no português, “Folha de Ideias”, é uma técnica bem simples, trata-se de um cartão com objetivos a serem alcançados ou mesmo definidos (ROBERTS et al., 2017). Você pode usar de forma bem livre em diversas aplicações nos processos de elaboração de um projeto, a exemplo:

- Folha de Ideia dos objetivos do jogo – expor a saga do herói no jogo descrevendo como o jogo será do início ao fim;
- Folha de ideias do Protagonista – apresentar como o herói se sente sobre suas metas, desenvolvimento em caráter amplo;
- Folha de Ideia do antagonista – revelar as metas e motivações para o antagonista;
- Folha Ideia das Configurações e Níveis – descrever definições de ambientes onde a história se passa com visões e orientações espaciais.

No escopo deste livro, você não precisa seguir a risca o uso das Folhas de Ideias, fique à vontade para criar quantas forem necessárias. Mas, afinal, como usamos isso? Na prática, cada folha, que vai de um pequeno “post-it” a um extenso documento, dependendo da complexidade do projeto, deverá elencar os passos a serem seguidos, dentre os quais podemos indicar:

1. Defina seus objetivos;

Divida os objetivos em categorias, como ‘Personagens’, ‘Mundo’, ‘Missões’, etc.;

Cada objetivo deve ser descrito e sua importância (prioridade) deve ser identificada.

A seguir mostramos um modelo de Ficha de Ideias, neste é fornecido um número de identificação para seu controle, cada uma dessas folhas trata um objetivo específico, veja que existe uma coisa importante no modelo, a identificação da prioridade e das estratégias, pois se deve dentre todos os seus objetivos, enxergar os prioritários e o modo como vai atingi-los. Essa ficha pode ser apresentada a sua equipe para que ela faça observações e, até mesmo, revise os planos traçados.

Tabela 1. Modelo de Ficha de Ideia

	OBJETIVO	DESCRIÇÃO	PRIORIDADE
N° 001	Definir o estilo do mundo do jogo.	Criar a definição de tipos de elementos do mundo.	1
<p>DESCRIÇÃO: O jogo se passa em uma terra distante, fantasiosa, misturando mitologia nórdica, estaremos em uma era medieval; Logo teremos elementos como castelos, montanhas, cavernas, torres, pântanos e masmorras.</p> <p>ESTRATÉGIAS: Estudar mitologia nórdica, avaliar as construções, roupas, regras e sociedade da época; Verificar em filmes e jogos as ambientações existentes e comuns; Desenhar um mapa do mundo, indicando as cidades, dungeons, rotas e caminhos; Estabelecer as missões em cada cidade.</p>			
DATA DE CRIAÇÃO:		DATA DA ÚLTIMA REVISÃO:	
01/06/2017		01/11/2017	
DATA DE INICIAÇÃO:		DATA DE FINALIZAÇÃO: ___/___/_____	
___/___/_____			
AUTORES: Luiz Fernando Reinoso		CONCLUÍDO? () SIM () NÃO	
OBSERVAÇÕES: Nenhuma.			

As estratégias descrevem as ações (geralmente verbos – grifados em vermelho) que você irá tomar para fazer com que a ficha seja satisfeita. A ficha de ideias ainda pode conter desenhos, esquemas gráficos, fotos, figuras e tudo que você desejar, o modelo proposto é apenas uma ideia simples para te ajudar a ter controle sob suas produções, metas e estratégias. Uma ficha é concluída quando se consegue atender seu objetivo e várias podem ser criadas e trabalhadas simultaneamente, por mais de uma pessoa o por grupos diferentes em um projeto. Essas fichas podem ser criadas dentro

3.3. Flowcharts

Um flowchart ou simplesmente fluxograma é um diagrama para representação esquemática de um processo e seu ciclo, tenta de forma ilustrativa descomplicar a transição de informações entre os elementos que o compõem, ou seja, é a sequência operacional do desenvolvimento de um processo, o qual se caracteriza pelos seguintes traços: como o trabalho está sendo realizado; qual o tempo necessário para sua realização e a distância percorrida pelos documentos; quem está realizando o trabalho; e como ele flui entre os participantes do processo (ARELLANO et al., 2017).

O uso do fluxograma aqui define todo o processo do início ao fim de um jogo, mostrando toda sequência e desvio de fluxo possível que o jogador terá possibilidade de executar. Esse tipo de gráfico consegue demonstrar a dimensão de um jogo e seus desdobramentos, além de conseguirmos balancear o jogo e distribuir de forma ordenada nossas tarefas com esse mapeamento.

A Figura 7 mostra um exemplo de um fluxograma para um jogo, neste o jogador inicia o game e tem de pegar uma estrela na tela, com o cumprimento da operação, uma animação (cena não jogável) é apresentada a ele, após isso ele tem uma missão opcional, pode alimentar seu gato e, quando opta por alimentá-lo, a animação dele alimentando o gato aparece. No caso de decidir não alimentar o gato não há alteração no fluxo do jogo, a missão principal, sendo demarcada pela linha vermelha, mostra que o jogador deve fazer a missão de resgatar uma escova para seu gato, ao tentar fazer isso, portanto, uma animação dele jogando uma bola onde a escova se encontra é apresentada, ao derrubá-la ele pode coletá-la.



Figura 7. Fluxograma de um jogo

Fonte: próprio autor.

O exemplo proposto foi de um jogo em que o jogador assume papel de uma criança e tem que ajudar seu gato, que adora ser escovado. Perceba que criamos ainda uma

legenda que ajuda a identificar os tipos de elementos que temos dentro do processo do jogo. Esse gráfico, quando temos diversas missões (objetivos), fica maior, note que, por isso, é importante enxergar o fluxo principal do game (marcado em vermelho), que mostra a transição do game do início ao fim. Caso seu projeto seja extenso, você pode criar um fluxograma para cada parte, fase, capítulo ou transição do game, o importante é possibilitar a visão do fluxo do jogo por completo.

Esta técnica consegue fornecer, de maneira visual e interativa, ao desenvolvedor uma forma de mensurar o tamanho de um jogo, a fase ou os processos a serem organizados e desenvolvidos e as principais interações dentro de uma sequência operacional.

3.4. MDA Framework

O MDA, abreviação para “Mechanics, Dynamics & Aesthetics”, em português com tradução livre, “Mecânicas, Dinâmicas e Estéticas”, é um framework utilizado em game design para a análise e criação de jogos, seus elementos têm características distintas e ajudam-nos a identificar e analisar cada parte de nosso jogo para projeto e desenvolvimento.

O que você tem que ter em mente é que a pessoa intitulada design de jogos é a pessoa que constrói um jogo que irá ser consumido por um público, por jogadores. A Figura 8 representa essa interação.



Figura 8. Produção e consumo de artefatos de um jogo

Fonte: Hunicke, LeBlanc, Zubek (2004).

Adaptada pelo autor.

O framework MDA formaliza o consumo de jogos, dividindo-os em componentes distintos: regras, sistema e diversão, conforme apresentado na Figura 9. Quando o jogador começa a jogar um game, conforme ilustra a figura, é neste momento que ele tem uma perspectiva do jogo, ele interage com o jogo, entendendo e adaptando-se às regras por meio do sistema de jogo que é apresentado e, posteriormente, começa a se divertir.



Figura 9. Componentes de consumo de um jogo

Fonte: Hunicke, LeBlanc, Zubek (2004).

Adaptada pelo autor.

Partindo da análise e da perspectiva do designer, somos levados a três contrapartes: Mecânica, Dinâmica e Estética, conforme a Figura 10:

- Mecânica: Refere-se aos componentes bases de um jogo, como regras, movimento do jogador, algoritmos, sistemas, estrutura de dados, etc.;
- Dinâmica: Diz respeito ao ato de “jogar” quando as regras e sistemas são postas em movimento, assim, trata-se da interação do jogador com os engenhos e estratégias que emergem do game. A dinâmica é a parte da mecânica com a qual o jogador pode realmente interagir;
- Estética: A estética descreve as respostas emocionais desejadas evocadas no jogador, quando este interage com o sistema de jogo.



Figura 10. Contrapartes a partir do consumo de um jogo

Fonte: Hunicke, LeBlanc, Zubek (2004).

Adaptada pelo autor.

A Figura 11 mostra a perspectiva entre jogador e desenvolvedor. O jogador inicia a sua experiência a partir da parte estética do jogo e interage com a dinâmica, mas nunca com a mecânica. O designer começa a partir da mecânica, definindo regras, estrutura de dados e sistema do jogo como um todo, em seguida, ele irá trabalhar no desenvolvimento do aspecto dinâmico do jogo para o jogador. Ao trabalhar com jogos é muito útil considerar as perspectivas do designer e do jogador e tentar, dessa forma, oferecer a melhor experiência para seu público. Observamos que podemos adequar uma camada a outra, tentando encaixá-las, mas cada mudança afeta as contrapartes entre criadores e consumidores.

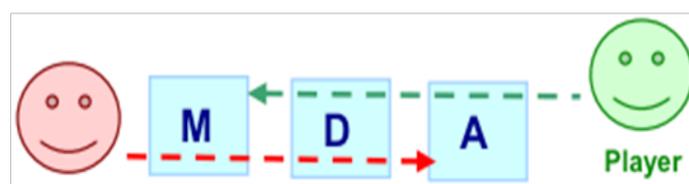


Figura 11. Design e jogador em diferentes perspectivas

Fonte: Hunicke, LeBlanc, Zubek (2004).

Taxonomia é o processo que descreve a diversidade dos seres vivos. Esse processo é feito usando artifícios como a classificação e a nomenclatura. Existe uma taxonomia criada para MDA que busca descrever algumas sensações principais que o jogo pode passar (HUNICKE, LEBLANC e ZUBEK, 2004). O intuito desse tipo de estudo objetiva uma classificação acerca das possíveis sensações que um jogo pode passar aos jogadores:

- Sensação: Jogo como prazer sensorial, aquele jogo que você joga pelos visuais ou pela música por exemplo;
- Fantasia: Jogo como faz de conta, troca de papéis, permite ser algo que não é na vida real, como um cavaleiro ou um médico;
- Narrativa: Jogo como drama, com uma história com engajamento ou que o jogador possa construir a sua própria história;
- Desafio: Jogo como um obstáculo, trata-se de um obstáculo a ser superado pelo jogador, não confunda com dificuldade como razão absoluta para esse tipo de jogo;
- Amizades: Jogo como ambiente social, neste há a dependência de outros jogadores para atingir um objetivo;
- Descoberta: Jogo como conquista de um novo território, vigora o prazer de descobrir algo novo, seja um território, um item ou um personagem secreto;
- Expressão: Jogo com o intuito de mostrar a individualidade do jogador, baseia-se na construção de casas, personagem ou na expressão através da escolha de um equipamento, como armas e outros tipos de ferramentas;
- Submissão: Jogo como passatempo, não requerem um esforço do jogador, revela-se como um ambiente para relaxar.

Essas sensações são previstas em um game, sendo proposital na maioria das vezes. Objetiva maior imersão do jogador, chamamos isso de **metas estéticas**. Existem muitos artigos e blogs especializados sobre esse estudo em game design. Pesquise análises e leituras desse nicho para ampliar seu entendimento.

Exemplo de jogos e suas metas estéticas:

- Charades: amizades, expressão, desafio;
- Quake: desafio, sensação, competição, fantasia;
- The Sims: descoberta, fantasia, expressão, narrativa;
- Final Fantasy: fantasia, narrativa, expressão, descoberta, desafio, submissão;
- Doom: sensação, desafio, descoberta, fantasia;
- Metal Gear Solid: expressão, narrativa, fantasia, desafio;
- Pokemon: expressão, amizades, desafios, fantasia, descoberta.

Podemos perceber que o MDA Framework pode ser utilizado para produzir jogos focados no engajamento de seus jogadores, a utilização da abordagem é muito signi-

ficativa, demonstra também a importância do balanceamento de todas as partes e a influência de cada uma destas. As alterações em uma delas influenciam diretamente as outras, logo, o comportamento de toda a estrutura do jogo pode ser interpretado e descrito a partir do uso da MDA. Tentar encaixar o projeto de um jogo neste modelo coloca-o em uma abordagem formal e iterativa para a concepção e afinação do seu produto (jogo).

Dica: Leia um artigo acerca deste tópico: “The MDA Framework” de Luiz Claudio Silveira Duarte, disponível em: http://www.gamasutra.com/blogs/LuizClaudioSilveiraDuarte/20150203/233487/Revisiting_the_MDA_framework.php.

Dica: O artigo original “MDA: A Formal Approach to Game Design and Game Research”, por Robin Hunicke, Marc LeBlanc, Robert Zubek, pode ser lido em: <http://www.cs.northwestern.edu/~hunicke/MDA.pdf>.

3.5. Design/Play/Experience Framework

O Design, Play, Experience (DPE) Framework ou, em tradução, Estrutura de Design, Jogar e Experiência, foi criado como uma ferramenta metodológica de design para criação de jogos sérios com potencial de mercado em uma abordagem de design formal para concepção de aprendizagem, narração, jogo, experiência do usuário e componentes de tecnologia de um jogo sério. A abordagem criada fornece ao desenvolvedor uma linguagem comum para **discutir** o design de **jogos sérios**.

Em um projeto de jogo, devemos ter a teoria, o conteúdo e o game design compatíveis e complementares, a sobreposição existente entre essas partes foi apelidada por Winn (2008), em sua pesquisa, como “o coração dos jogos sérios”, apresentado na Figura 12. O conceito do coração do design sério apresentado por Winn, trata-se de um paralelo ao “technological pedagogical content knowledge model” (TPCK), em tradução livre, “modelo tecnológico de conhecimento de conteúdo pedagógico”, proposto por Mishra e Koehler (2006 apud WINN, 2008, p. 1012).

O modelo TPCK define a sobreposição dos conhecimentos sobre tecnologia, pedagogia e conteúdo, e é uma forma emergente de conhecimento que vai além dos três componentes individuais, objetivando a produção de um resultado maior que a soma de suas partes.



Figura 12. O coração do design dos “Serious Games”

Fonte: Winn (2008).

Adaptada pelo autor.

O DPE surge como uma extensão do MDA, possui estrutura semelhante, retrata a relação entre o designer e o jogador, conforme apresentado na Figura 13. O designer projeta o jogo para um jogador jogá-lo, o que resulta na experiência do jogador e tem controle direto sobre o próprio projeto. Para projetar um jogo de forma eficaz, o designer deve primeiro apresentar metas para a experiência resultante.

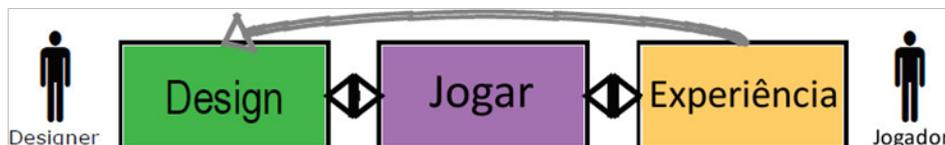


Figura 13. DPE Framework

Fonte: Winn (2008).

Adaptado pelo autor.

A flecha que retorna da etapa da experiência para o design (Figura 13) representa tanto a influência dos objetivos no design original quanto à iteração no projeto, quando um protótipo do jogo é **testado** em relação aos objetivos da experiência do usuário (Playtest¹⁴) podemos ver isso. Segundo Salen e Zimmerman (2004 apud WINN, 2008, p. 1014), isso reflete o processo inerentemente iterativo de design de jogos, incluindo o design, a prototipagem, o teste de reprodução (Playtest) e a iteração de volta ao projeto com base na experiência do teste de reprodução (Figura 14).

14 Playtest – Refere-se à testar um jogo desenvolvido recentemente ou mesmo em desenvolvimento com a intenção de jogá-lo ou tê-lo jogado, trata-se do teste de reprodução do game/protótipo. O Playtester é a pessoa que joga objetivando o Playtest.

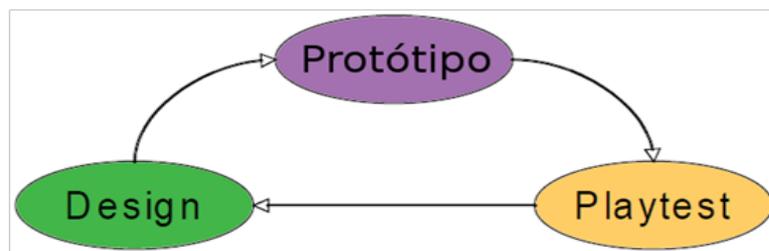


Figura 14. Processo iterativo de design

Fonte: Winn (2008).

Adaptado pelo autor.

A estrutura do DPE expandida (Figura 15) descreve os subcomponentes do design dos jogos sérios, incluindo as camadas de aprendizado, narração, jogo e experiência do usuário. Cada camada possui um aspecto de design, jogo e experiência. A tecnologia é representada na camada inferior. Embora designer não designe necessariamente a tecnologia, o próprio projeto é idealizado (ou não) na tecnologia.

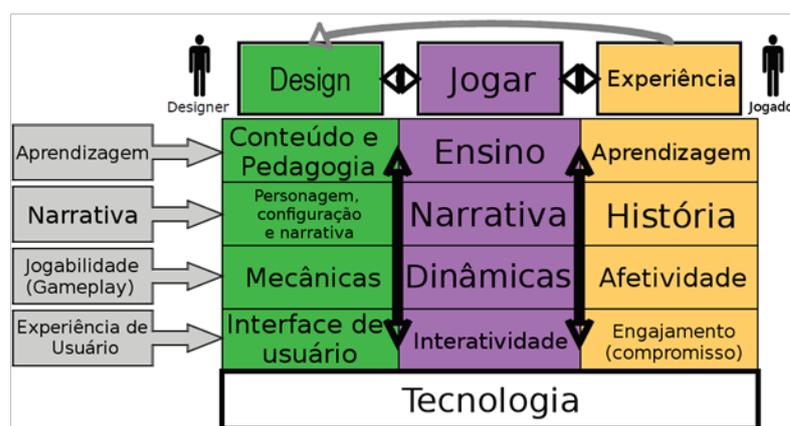


Figura 15. DPE Framework expandida

Fonte: Winn (2008).

Adaptado pelo autor.

Logo, devemos desenvolver cada camada do DPE em cima da base tecnológica existente, reocupando-nos sempre na compatibilidade entre elas, Deborah Lieberman (2006 apud WINN, 2008, p. 1011) lista oito benefícios para a aprendizagem a partir dos jogos:

- Fornecem ao jogador uma experiência;
- Encorajam o jogador a aprender fazendo;
- São um meio social de um humano para humano, como a interação de respostas emocionais;
- São participativos, fornecendo ao jogador feedback personalizado e rápido;
- São cativantes. A participação faz com que o jogador preste muita atenção, o

que requer planejamento criterioso e tomada de decisão. Também exige aprendizagem para ter sucesso (se você não aprende, então você não pode ter sucesso);

- Promovem o aprendizado comportamental, pois dá ao jogador recompensa pelo comportamento (pontos, poder, posição, etc.). Esse Feedback positivo no jogo pode encorajar os comportamentos desejados na vida real;
- Oferecem consequências. Estas não são abstratas ou hipotéticas, estão representadas no jogo diretamente. O jogador joga com um personagem e identifica-se com ele ou ela. O sucesso e o fracasso mapeiam diretamente as ações do jogador; ademais, o ego e a autoimagem são investidos na experiência;
- Fornecem modelos para o jogador, o qual pode aprender com o jogo e compreender seu comportamento e experiências.

Utilize o DPE com sabedoria. Um jogo sério, como exemplificado neste capítulo, é muito comumente confundido com um jogo unicamente educacional. Cuidado! Um jogo comercial pode ensinar história, geográfica e outras matérias e ser tão desafiante e educativo quanto um game estritamente educacional.

Quando falamos em aprendizagem, estamos estabelecendo que o jogo tem a capacidade de ensinar ou encadear o entendimento de algo que pode ser educativo, profissional, um aperfeiçoamento ou mesmo um treinamento. Tudo irá depender do esforço do designer nas camadas em que ele é responsável e nos resultados obtidos na experiência resultante, pois o modo como estes serão tratados poderá melhorar os resultados objetivados a cada ciclo do processo interativo.

Dica: Leia o artigo original utilizado para composição deste capítulo: “The Design, Play, and Experience Framework” de Brian M. Winn no link: http://gel.msu.edu/winn/Winn_DPE_chapter_final.pdf.

Para ampliar seu conhecimento acerca do uso e entendimento do DPE, estude e pesquise acerca do tema. Cada camada existente na estrutura expandida apresentada merece um estudo detalhado para aperfeiçoar sua visão como designer.

3.6. Desenvolvimento ágil de software

O desenvolvimento ágil de software (DAS) é uma metodologia que utilizamos com objetivo de aumentar nossa produtividade dividindo as tarefas, para que sejam menores e completadas em curtos espaços de tempo. Cada tarefa compreende ações que possibilitam sua conclusão. Cada tarefa completa um ciclo ou uma interação, um incremento da totalidade do projeto., gerando resultados de forma rápida para seus desenvolvedores (PRESSMAN, 2011).

A Figura 16 apresenta o ciclo de vida das tarefas com o DAS, demonstrando que você pode possuir diversas tarefas em um projeto. Cada uma destas pode abranger uma ou mais ações, que são as atividades a serem executadas para completar ou cumprir a tarefa. Porém, uma tarefa tem de ser bem planejada, pois compreende recursos humanos e materiais para que um objetivo seja alcançado (requisitos).

Tarefas podem ser executadas em paralelo, o que vai requerer um planejamento minucioso para que não falem recursos, ou podem ser predecessores, nesse último caso as tarefas só podem ser executadas mediante o cumprimento de uma tarefa anterior, por exemplo, programamos as ações de uma fase no game mediante artes necessárias encomendadas de um artista, caso este não elabore os recursos gráficos, não podemos programar a fase. Nessa perspectiva exemplificada, os desenhos do artista é predecessor da tarefa de programar a fase.

O **projeto** de uma tarefa descreve as estratégias que garantirão a execução desta, como sua estrutura, passos e controles (gerenciamento), respeitando seu planejamento. A **codificação** (criação do código-fonte do game, desenvolvimento propriamente dito) é realizada mediante a definição do projeto, após isso, o produto da tarefa é **testado** para garantir se está de acordo com o projeto e os objetivos da tarefa.

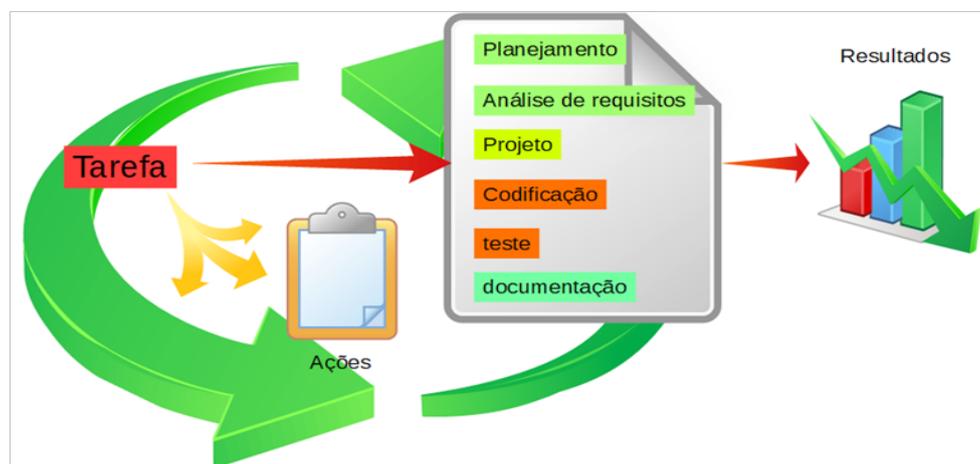


Figura 16. Processo iterativo do desenvolvimento ágil de software

Fonte: próprio autor.

Ao se obter um produto da fase de codificação ou desenvolvimento, testamos para melhorar ou ajustar os **resultados**, que podem ser esperados, devido sua definição em nossos objetivos para a tarefa. A Figura 17 apresenta do ciclo de desenvolvimento das tarefas à obtenção de resultados, perceba que, no exemplo, fazemos um conjunto de ações que resultam em artefatos diferentes, apesar desse ciclo não se mostrar interativo, tenha em mente que ele o é, mas ao utilizarmos as resultantes de cada fase, conforme a Figura 16, por exemplo, contemplamos um novo ciclo, tendo suas etapas dinâmicas umas ligadas as outras, logo, o ciclo de **tarefa>ação>resultado** pode ocorrer em cada fase do planejamento até a documentação de uma tarefa.

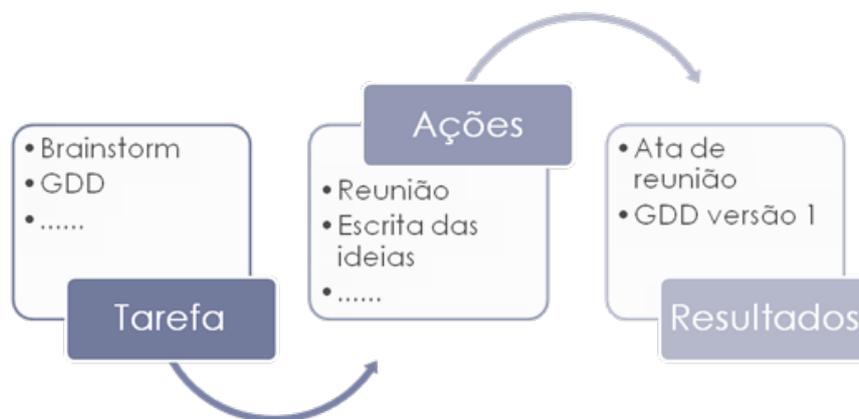


Figura 17. Ciclo para obtenção de resultados no DAS

Fonte: próprio autor.

Por fim, temos a etapa de **documentação**, trata-se da fase em que você descreve todas as funcionalidades do programa (jogo) para ajudar a sua equipe, ou a você mesmo, a entender o que cada membro realizou (Sommerville, 2007).

3.7. Processo de software pessoal (PSP)

Assim como as organizações, o profissional desenvolvedor de software também deve se preocupar com seus processos de trabalho. Estabelecer, aprimorar, aperfeiçoar e até mesmo documentar os próprios métodos de trabalho.

O objetivo dessa metodologia é conseguir prevenir erros, custos, entre demais as-

pectos, objetivando entregar um produto isento de erros, com prazos e preço oportuno.

O Processo de Software Pessoal (PSP) pode sofrer influência de diversos fatores, como responsabilidades, métricas, indicadores, previsibilidade, controle, entre outros. Muitos desenvolvedores de jogos iniciam com a criação de jogos independentes, nesse sentido, obter controle sobre suas ações e determinar uma faixa para alguns dos fatores mencionados pode ajudá-lo a mensurar projetos que se qualifica a executar.

Para utilizar o PSP, profissionais criam um roteiro de atividades baseado no projeto que irão executar como um manifesto do quanto podem contribuir, validando as etapas, habilidade e atividades a serem realizadas. Esse roteiro é importante, pois ajuda a encontrar lacunas operacionais, como pessoas que faltam, mão de obra necessária, definição de tempo, entre outras coisas.

3.8. UML – Diagrama de estado

Segundo especificações da OML (Grupo de Gerenciamento de Objeto), a UML (Unified Modeling Language) – em tradução, Linguagem de Modelagem Unificada – “é uma linguagem para especificação, construção, visualização e documentação de artefatos de um sistema de software intensivo” (LIMA, 2009).

Com essa linguagem gráfica podemos analisar, especificar e construir todo um sistema orientado a objetos utilizando uma notação comum, ou seja, uma forma de representação universalizada para área de sistemas.

Na UML, temos diversos tipos de diagramas, cada um destes permite aos desenvolvedores obter uma visão acerca de um ponto de vista particular do sistema, possibilitando uma representação em menor escala de toda a aplicação que está sendo desenvolvida (LARMAN; LIMA, 2004, 2009).

Para nosso trabalho com jogos, apresentaremos apenas um diagrama que ajuda a ter uma visão sobre o comportamento interno do núcleo de nossos sistemas de jogo ou sobre o jogo por completo, o diagrama de estado. Com esse tipo de diagrama podemos visualizar os estados possíveis que teremos em nosso game. A Figura 19 apresenta um diagrama de estado, o qual foi criado na aplicação Astah Community, ferramenta gratuita para criação de diagramas UML.

A Figura 18 mostra de forma simplificada, para seu entendimento, dois estados possíveis em um game fictício, o jogo pode estar em execução ou pausado, sendo assim,

cada estado pode ser acionado pelo jogador através da tecla 'ESC' do teclado. O jogador pode executar isso infinitamente, até que pressione a tecla 'Backspace', que faz o game se encerrar.

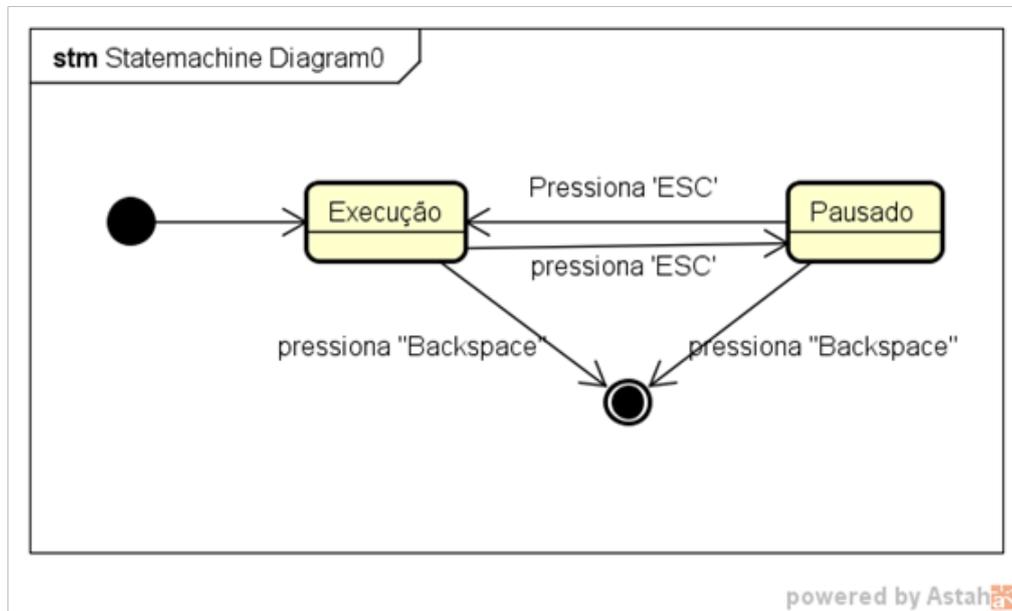


Figura 18. Diagrama de estado exemplo

Fonte: próprio autor.

Dica: Não estude sozinho, fique a vontade e poste suas atividades nos fóruns indicados da sala virtual criada para este livro, conforme apresentado na introdução.

Dica: Complemente seu conhecimento acerca de UML e diagramas de estado no endereço: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/diagramas/estado/diag_estados.htm

3.9. Exercícios

1. Crie duas fichas de ideia para um jogo de seu interesse. Se possível, crie os mesmos através de um brainstorm com colegas. Após isso, utilize o MDA para reestruturar as fichas;
2. Crie um diagrama de fluxo para o ciclo de uma fase ou jogo, encontre sua linha crítica, ou seja, fluxo principal e alternativo;
3. Elabore uma planilha com tarefas e as subdivida em ações, mensure os resultados/indicativos de cumprimento para cada. A seguir, crie uma folha nesta planilha com sua análise utilizando o PSP, colocando suas habilidades e identificando as tarefas que você precisaria de mais ajuda ou não possui habilidade para cumprir;
4. Crie um diagrama de estado para um game que você conhece. Após isso, crie metas estéticas ou as identifique;
5. Você já deve ter jogado ou conhecido um game educacional, correto? Se não conhece, pesquise sobre um e o analise perante o MDA.

4. Design de Jogos

O **processo criativo** nos jogos envolve as possibilidades e limites de nossa imaginação e suas intrarrelações (subjetividade) que assumem forma dentro do mundo digital. Essa tarefa desprende investimento de tempo, estudo e dedicação. Podemos trabalhar em equipe para dividir tarefas e aumentar a produtividade de nossos projetos, por exemplo, dividindo uma parte destes para cada integrante da equipe.

Podemos marcar reuniões, anotar ideias, definir regras – sobre como será o layout do game, por exemplo – e delinear outras práticas e técnicas para representar o **pensamento**. Essas etapas podem ser melhor organizadas e definidas, como você aprenderá a seguir. As metodologias abordadas no **capítulo anterior** já podem ser aplicadas nos artefatos que iremos aprender a construir neste capítulo.

A primeira pergunta que podemos fazer no ponto que estamos é “O que é design de games?” Segundo Schuytema (2008), o design de games pode ser entendido como a planta baixa de um game. A pessoa com o papel de design é quem criar essa planta e é a partir dela, com talento, esforço e dedicação, que um game irá se tornar realidade.

Neste capítulo iremos aprender a ser um design de jogos, iremos conhecer técnicas e metodologias para criar um conceito de jogo e a partir dele desenvolver um projeto com documentação própria, sendo este o chamado Game Design Document (Documento de design de jogo), bem como modelos representativos que auxiliam em seu entendimento e composição.

4.1. Conceito de um game

Conceituar algo está na faculdade do intelectivo ou cognitivismo do ser humano, em estado de pensamento. **É a arte do pensar!** Todo processo de criação humana se inicia a partir de uma ideia, um pensamento que nos permite enxergar um determinado fim. Quando temos uma ideia tentamos torná-la cada vez mais palpável, desenhamos, escrevemos, esquematizamos, equacionamos, fazemos de tudo para tentar inseri-la no papel para que se torne realidade.

O termo “conceito”¹⁵ tem origem a partir do latim “conceptus” (do verbo concipere), que significa “coisa concebida” ou “formada na mente”. Quando usamos essa palavra para defender uma ideia, estamos querendo transmitir que entendemos completamente as qualidades dessa ideia.

15 Conceito – Significação. Disponível em: <<https://www.significados.com.br/conceito/>>.

Por exemplo, quando dizemos que algo é “diferente perante outras coisas”, “inovador devido a isso e aquilo”, temos uma representação mental de um objeto e muitas vezes podemos ter uma visão abstrata dele. Um conceito de jogo é a definição em alto nível de uma ideia completa e compreendida, quando temos um conceito para um game em mãos, temos uma ideia de como ele “é” de fato ou mesmo como “será” (ADAMS, 2010).

Os designers de jogos não tentam apenas criar um game, mas criar conceitos novos, buscar algo que seja significativo ao jogador, que o faça inclusive formar ou reformar seus próprios conceitos para o jogo (no capítulo anterior, falamos de metas estéticas, as quais completam essa ideia). A Figura 20 apresenta uma representação funcional da visão que um game designer deve levar em consideração para desenvolver um produto de jogo digital.

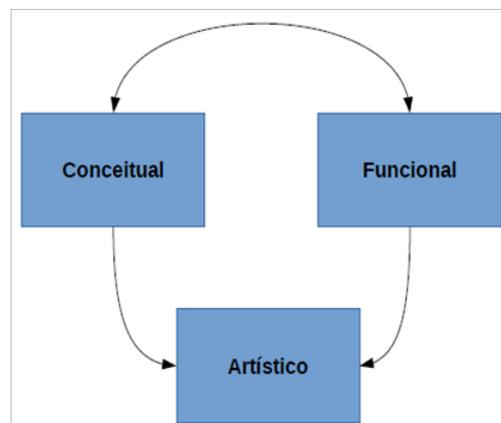


Figura 20. Representação da visão funcional do game design para um jogo eletrônico

Fonte: Próprio autor.

A representação apresentada não possui setas de retorno, pois parte do ponto que o game já foi conceituado e sua visão funcional já está estabelecida. É uma forma de ajudar a visualizar o equilíbrio do ponto de vista **conceitual**, que transmite o jogo como um todo, e do ponto de vista dos **aspectos funcionais**, que é como o game será de fato jogado e interativo. Esse equilíbrio entre o conceitual e o funcional influenciará diretamente na concepção **artística**, que representa os estilos de desenho, gráficos, ambientação, sons, música e outros aspectos. Um game depende dessas visões para existir, logo, um conceito de jogo é uma ideia de produto, definido como um todo e entendido por completo.

Dica: Leia o livro “The art of computer game design” escrito por Chris Crawford, principalmente o capítulo acerca de “Game design sequence”.

4.2. Mecânica de um jogo

A mecânica de um jogo compreende um conjunto de **regras**, possibilidades de **ações**, **decisões** e variedades de respostas do **sistema** do game. Regras são o conjunto lógico que permite e/ou impede ações dos usuários e fazem parte de uma norma estabelecida por quem quer impor um padrão geral. Sobre sistema, Crawford (1997, pg. 8) acerca de jogos eletrônicos define:

O termo 'sistema' é muitas vezes mal utilizado, mas neste caso, sua aplicação é bastante apropriada. Um game é uma coleção de partes que interage uma com as outras, muitas vezes de maneiras complexas. É um sistema.

A mecânica descreve as regras do jogo (implícitas e explícitas) e é o modelo do universo no qual o jogo funciona. Pense nisso como a **simulação do mundo do jogo** e em como todos os pedaços interagem entre si.

4.3. Regras e balanceamento

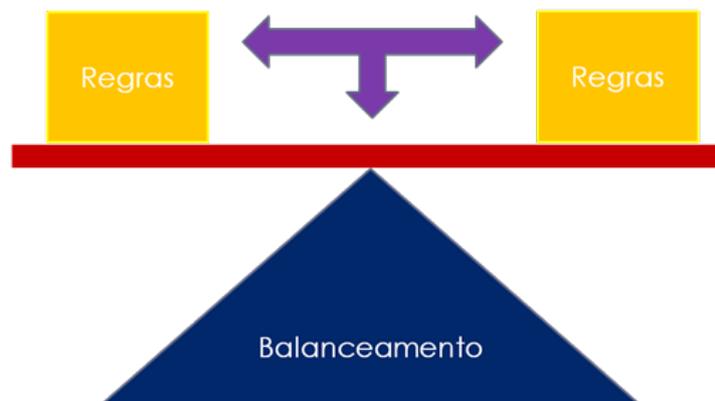


Figura 21. Representação visual do balanceamento de regras de um jogo

Fonte: próprio autor.

O desafio é a verdadeira motivação que faz um jogador ir em frente no mundo do jogo. As formas com que ele socializa com o universo do game são fundamentais para que torne sua experiência significativa.

Um jogo tem que responder de forma eficiente às ações do usuário e apresentar com clareza o resultado de tais ações, seja positivo ou negativo, de acordo com o ponto

de vista das regras colocadas. Logo, como representado na Figura 21, as regras têm de ser balanceadas entre si e ter equivalência de peso e equilíbrio para o pleno funcionamento de todo o jogo.

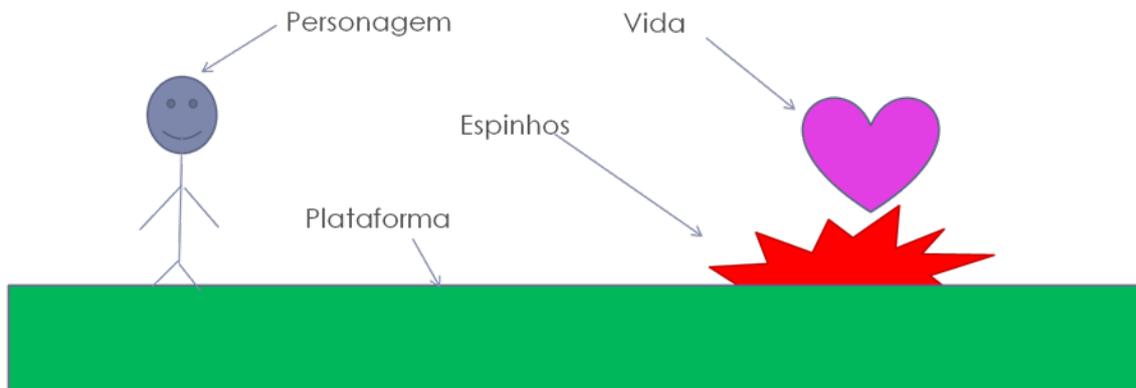


Figura 22. Exemplo de um balanceamento ruim

Fonte: próprio autor.

A Figura 22 exemplifica um balanceamento mal elaborado: você tem uma regra que diz que o personagem morre/perde ao encostar-se aos espinhos vermelhos e que o coração rosa o concede mais uma vida/chance, assim, colocar o coração em cima dos espinhos impede o contato com o coração.

No exemplo, caso a ideia não seja “encorajar a morte eminente” do jogador, seria um mau balanceamento da regra de “garantir uma vida com os corações”. Porém, caso o personagem possa pular, ajustando onde vai cair e podendo passar por cima dos espinhos, você balanceia a regra em cima da nova mecânica.

4.4. Roleplaying e motivação do personagem

Roleplaying trata-se da situação em que o jogador assume o *papel* (role) do personagem jogável. Logo, quando o jogador é responsável pelas ações de um personagem ele pode estar representando o papel deste no game. Muitos RPGs agregam essa funcionalidade de assumir papéis de forma profunda e acabam tornando-se referência para o termo. Enquanto assumimos um personagem devemos pensar: “O que faz esse

personagem ir em frente?”.

Imagine um game em que o personagem simplesmente exista, sem um plano de fundo, nome ou raça? Devemos ter em mente o porquê de um pônei estar atacando pessoas, certo?

A **motivação** do personagem deve ser identificada e intensificada para que o jogador possa entender quem ele é. A exemplo, seria ruim o jogador confundir em um game um policial antiterrorista com o próprio terrorista, em uma situação em que aquele tenta impedir que este implante uma bomba em sua base. A motivação do jogador vem do próprio ato de jogar, à medida que vai se aprofundando no jogo, ficando cada vez mais imerso (MARINS et al., 2012), quanto mais ele joga, mais o jogo consegue vender seu conceito.

4.5. Explicação vs imaginação

Um game pode apresentar detalhes interessantes acerca do seu conceito, seja com recurso audiovisual ou na imaginação do jogador. A apresentação, por exemplo, com lendas do mundo do seu game pode usar a mente do jogador como recurso para maior imersão e profundidade.

Na prática, só precisamos mostrar ou explicar ao jogador o que ele não conhece ou o que é de difícil dedução ou assimilação, dados esses que ultrapassam a comunicação com a própria mecânica, como jogos anteriores, histórias passadas, informações externas ao game, entre outras coisas. Por exemplo, você não precisa explicar todo o funcionamento de uma arma, a menos que esta seja inventada por você e que tenha um funcionamento único.

O ideal é tentar passar informações e explicações por meio do gameplay, com a própria mecânica. Comumente, os desenvolvedores tentam reduzir exemplificações à ícones e figuras minimalistas facilmente associativas, bem como tem transformado as falas e emoções em pequenos sons e expressões faciais e/ou corporais, a exemplo, o game The Sims¹⁶.

16 The Sims® - The Sims é uma franquia de jogos de simulação de vida real criado pelo designer Will Wright e produzida pela Maxis® e The Sims Studio® e atualmente distribuído pela Electronic Arts®.

4.6. Sinopse e história

Escrever uma história não é fácil, a escrita é uma das formas comuns para expressar uma ideia e os games entendem isso muito bem, existem jogos que são basicamente baseados em texto, trabalhando com entradas e saídas de frases, falas e palavras. Criar um bom roteiro para jogo é tão difícil quanto criar um para filme.

Por isso, antes de aprofundar-se no universo da escrita, leia bastante acerca dos temas que deseja trabalhar em seu jogo, comece com uma sinopse, que é um resumo breve ou uma síntese do livro/filme expondo todo seu conteúdo, abrangendo-o do início ao fim (veja o Apêndice A. Goa's reign paths, paper game). Uma sinopse deve possuir minimamente a descrição e/ou a resposta para as perguntas (ADAMS, 2010):

- Onde estamos: onde/quando é que a história acontece?
- Quem estamos a seguir na história (o protagonista) e por quê?
- Quem e/ou que força se opõe ao herói da história e por que estão nessa oposição (chamamos este de antagonista)?
- O que esses personagens querem realizar e o que está em jogo para eles?
- Qual é o conflito que se segue e por que ele existe?
- Como é a ascensão de ação/crise e como é que as coisas ficam mais tensas?
- O que é a crise final e como é que o jogador sai dela?
- Como as coisas se resolvem e se encerram?

Dica: Leia o capítulo 7 “Storytelling and Narrative” do livro “Fundamentals of game design”, segunda edição de Ernest Adams.

4.7. Character Brief

O Character Brief, em tradução livre, “Sumário do Personagem”, é um documento breve que contém as características dos personagens, principalmente a relevância deles para o público alvo. Segundo Derakhshani (2016), um Character Brief deve conter:

- Quem é ele/ela? É um personagem primário ou secundário e como ele se relaciona com a história do jogo?
- O que é que tem em comum o outro personagem? Quais os itens importantes

que pode usar ou levar consigo?

- Qual é a sua personalidade e como é que essa característica pode ajudar a cumprir seu papel na história do jogo?
- Como é que ele se relaciona com outros personagens no jogo?
- Quais são os seus pontos fortes e fracos e como isso pode afetar a história do jogo?
- Quais são os objetivos de caráter de motivação para esse personagem?

Nota: Os processos abordados foram testados em um curso de desenvolvimento de jogos com 20 alunos, no IFES, campus Colatina, ajudando os alunos na confecção do Game Design Document e na organização de suas atividades, bem como na produção de seus jogos durante o curso.

4.8. Etapas e processos

Não existe uma receita totalmente correta para criarmos jogos, apesar das metodologias mencionadas. Um jogo pode envolver algumas influências incomuns, como uma tecnologia diferente, que pode agregar hardware e diferenças de uso que tornam seu desenvolvimento um tanto quanto diferente.

Para nosso estudo seguiremos os seguintes processos, por onde irei guiá-los:

- **Conceito:** Onde definiremos a ideia de um jogo. Com criação de conceitos visuais e pequenos protótipos, idealizados e documentados.
- **Pré-produção:** Criação de protótipos e elicitação de funcionalidades para o desenvolvimento, definição de tarefas bem como suas ações.
- **Produção:** Criação de recursos audiovisuais, codificação, testes, correções e geração de produto.
- **Documentação:** Onde registramos nossos artefatos, manuais de uso bem como modelos significativos ao projeto.

Para organizar seu trabalho sobre esses processos, iremos seguir um modelo simples de Game Design Document, que você irá aprender a criar neste capítulo, na seção a seguir. O importante é entender a divisão desses processos, principalmente se você estuda sozinho.

4.9. Game Design Document

O Game Design Document (GDD) é um documento amplamente aceito no ambiente de desenvolvimento de jogos digitais. Seu conteúdo contém dados importantes para compreender o projeto e o conceito de um jogo a ser desenvolvido ou que já esteja em desenvolvimento (ADAMS, 2010, p. 54-63).

Não existe um modelo fixo ou único para um GDD, ele pode ser adaptado e modificado de forma interativa ao projeto em planejamento ou mesmo em produção quando necessário. Para nossas atividades, elaboramos um modelo disposto no Apêndice B. Game Design Document, utilize-o para documentar seus jogos.

A legibilidade de um GDD deve ser suficiente para que uma pessoa, mesmo que leiga em desenvolvimento de jogos, possa entender o que é o game e como jogá-lo, pois, geralmente, essa documentação trafega e é modificada durante todos os processos e etapas de desenvolvimento, tendo de ser lido, redigido e interpretado por todos os envolvidos no projeto.

Character brief, sinopse, regras, roleplaying, motivações, avaliações do DPA, DPE, folhas de ideias, cronograma, lista de tarefas e tudo que você aprendeu pode agregar informações e ajudar na confecção do GDD. O mesmo vale para o Game Design Canvas, que será mostrado posteriormente.

4.10. Game Design Canvas

O GDD é uma das formas mais populares de documentar seu game, diversas metodologias e técnicas foram desenvolvidas nos últimos anos, inovando seu uso e implementação. Uma dessas formas possibilita que o GDD seja subvertido em uma folha que molda seu projeto, é ela o Game Design Canvas (GDC).

Vale ressaltar que o uso de canvas (traduzido como tela de pintura) já é difundido na área de gestão há muito tempo, trata-se de exportar o projeto do jogo para uma forma de visualização em planta baixa, com estrutura e distribuição concisa, possibilitando uma visão esquemática do projeto (FINOCCHIO Jr., 2013).

Cada bloco ou partição do canvas comporta informações que completam todo o quadro, a ideia é preencher tudo. A Figura 23 apresenta tal proposta de canvas para o cenário de jogos, o Game Design Canvas (CARVALHO, 2014).

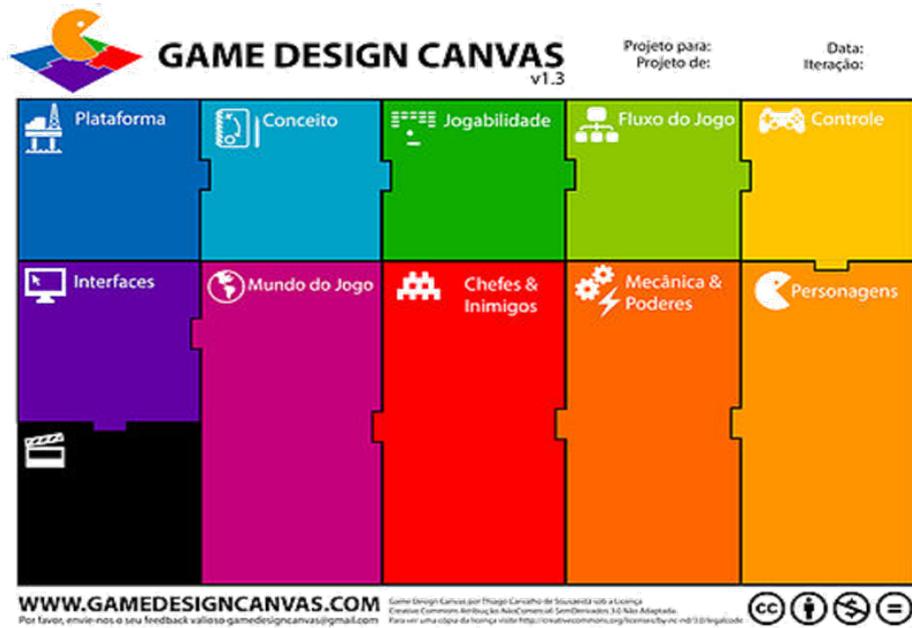


Figura 23. Game Design Canvas

Fonte: Carvalho (2013).

4.11. High Concept for Flow

O High Concept for Flow é um recurso simples e segue as diretrizes do GDD, do GDC e dos processos e etapas abordados. Foi criado pelos autores deste livro para dar conformidade ao desenvolvimento ágil e é utilizado na criação do game “ChildHood Fears” (REINOSO et al., 2015), sendo este do gênero plataforma de suspense.

O High Concept for Flow (traduzido como alto conceito para o fluxo) pode ser obtido no ambiente virtual disponibilizado para este livro e estudado no Apêndice C. High Concept for Flow. Como informado, a forma de uso é similar ao canvas, porém foi estruturado com base nos conceitos estudados neste livro e criado dentro de uma planilha com duas folhas principais, são elas:

- **Folha de Game Design:** possibilita completar toda a estrutura de seu jogo, desenvolvendo suas ideias e descrevendo todos os componentes expostos pela folha.
- **Folha de funcionalidades (Feature Set):** baseada na folha de Game Design, permite criar as tarefas e suas funcionalidades, com prioridade em seu anda-

mento. Essa folha conta com métricas de exemplo, que não são fixas, mas foram testadas em sete projetos de game jam, o que as torna importantes. Ademais, o número de tarefas varia de acordo com cada projeto.

Nota: O High Concept for Flow foi utilizado em um curso de desenvolvimento de jogos com 20 alunos, no IFES, campus Colatina e na criação de sete games em game jams funcionando inteiramente.

Dica: Confira os projetos que utilizaram o High Concept for Flow, estão na página “Sobre” da Cup Pixel, no link: <http://www.cuppixel.com.br/pages/sobre/page/>. Os eventos de Jam são o Global Game Jam e o Ludum Dare.

4.12. Profissões

Diversos tipos de profissionais podem fazer parte de uma equipe de desenvolvimento de jogos, desde programadores a roteiristas de Hollywood. Algumas destas profissões têm grande relevância para criar um produto de software interativo. Aqui listaremos quatro que acreditamos serem fundamentais em uma equipe:

- **Programador:** desenvolve a parte lógica do jogo, ou seja, suas regras, o funcionamento e a interatividade com o usuário. Elabora o código que faz tudo funcionar.
- **Designer Gráfico (Artista):** gera toda a arte, desde conceitos à arte final, que é animada e inserida em um jogo, desse modo, produz a parte visual do game.
- **Sound Designer/Compositor (Músico):** responsável por criar desde efeitos sonoros a músicas, aspectos que compõem a atmosfera de um jogo. Sendo assim, cria todo o áudio necessário.
- **Game Designer (Engenheiro de jogos):** projeta sistemas funcionais, como mundos e interfaces, e formula narrativas e mecânicas únicas e inovadoras.

4.13. Exercícios

1. Nesta atividade você irá criar um jogo com as seguintes limitações (exemplo no Apêndice A):

- 1.1 O jogo deverá ser criado respeitando os limites de uma única **folha** de A4, sendo essa folha o próprio jogo;
 - 1.2 Para jogar você terá apenas **dois** dados;
 - 1.2.1. Crie e **balanceie** as **regras** conforme estas limitações.
 - 1.3 O game deverá possibilitar que uma pessoa sozinha (modo **single-player**¹⁷) o jogue.
2. Após a atividade anterior, você irá criar o modo multijogador (**Multiplayer**¹⁸) para o jogo criado, nesse caso o jogo terá a opção de ser jogado por duas ou mais pessoas;
 - 2.1 Atenha-se que esse novo modo não irá substituir o modo singleplayer, mas adicionar mais um modo de jogo ao game. Logo, **balanceie** e crie **regras** para conformidade dos dois modos.
3. Para o jogo da atividade 1 e atividade 2, crie uma **história**, resumindo-a em uma **sinopse** e contextualizando-a, além disso, defina um **nome** para o jogo;
4. Elabore um **GDD** para um jogo qualquer ou para um jogo existente, obedecendo ao modelo disponível no Apêndice B. Você pode fazê-lo para o jogo criado nas atividades anteriores;
 - 4.1 Organize seu **GDD** depois de organiza-lo no **High Concept of Flow**, conforme exposto no Apêndice C, e tente mensurar o número de tarefas que você possivelmente deverá realizar.

17 Singleplayer - Em tradução significa “único jogador”. Modo de jogo, onde temos apenas um controlador/jogador para o game.

18 Multiplayer - Em tradução, “multijogador”, são games que permitem que vários jogadores participem simultaneamente em uma mesma partida.

5. Desenvolvimento de Jogos

Neste capítulo iremos criar jogos eletrônicos utilizando metodologias, técnicas e processos desenvolvidos no livro, para tanto, iremos conhecer ferramentas tecnológicas para desenvolvimento e manipulação dos artefatos que utilizaremos em um jogo, como desenhos, animações, modelos 3D, códigos de computador, sons, música e outros complementos.

Como suporte aos trabalhos que desenvolveremos, existem comunidades e sites dedicados a fornecer material de apoio a quem está começando ou prototipando um jogo. Muitos destes possuem recursos com licenças bem permissivas e, até mesmo, estão disponíveis em domínio público, dessa forma você pode utilizá-los sem nenhum receio. À medida que nos aprofundarmos, notas e endereços para estes diretórios de complementos serão disponibilizados para obtenção de recursos e para uso.

Antes de iniciar, você irá aprender conceitos importantes para o desenvolvimento de jogos, como a definição de arquitetura, o entendimento de desempenho e o uso do postmortem (documentação pós-projeto).

5.1 Arquitetura de software

A arquitetura de software é um conceito que os engenheiros de sistema entendem de forma geral, mas expor uma noção com precisão é difícil. Geralmente se define aspectos acima da estrutura algorítmica e gráfica do software, portanto, consiste na definição dos componentes de software, de suas propriedades externas e de seus relacionamentos com outros softwares. Ademais, o termo também se refere à documentação da arquitetura de software do sistema, que geralmente é a forma de representação da arquitetura propriamente dita.

Muitos pesquisadores definem a **etapa de definição e arquitetura** como bem parecida com as etapas de composição de design de um produto, pois em ambas são tratados aspectos relacionais e inter-relacionais do produto.

A Figura 24 apresenta a arquitetura de sistema cliente-servidor, a partir da qual poderíamos criar um jogo em que o jogador pode jogar no computador, no celular ou no tablet e pode fazer uso da internet para salvar seu game ou mesmo para jogar com amigos. Tais possibilidades se fazem reais pois temos um servidor que se trata de uma máquina responsável por disponibilizar serviços que podemos usar como banco de dados¹⁹, arquivos e outros recursos.

¹⁹ Banco de dados – Banco de dados ou base de dados é um conjunto de arquivos relacionados entre si, como uma empresa tem todos os formulários de seus clientes.

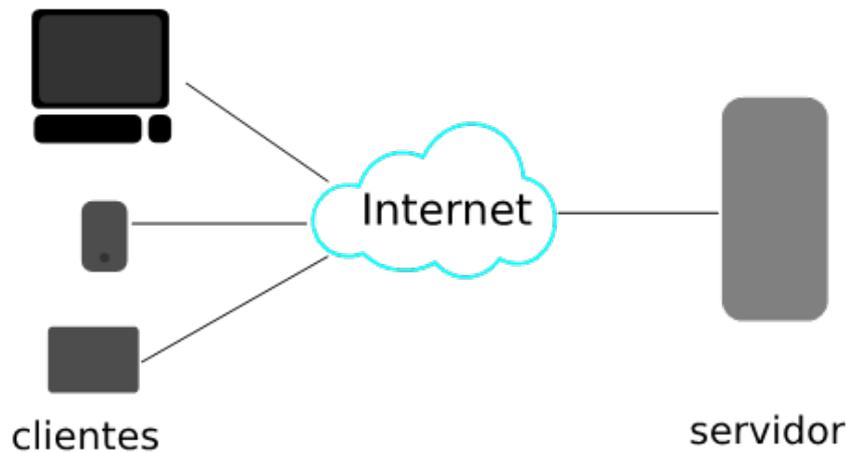


Figura 24. Arquitetura de sistema cliente-servidor

Fonte: próprio autor.

Existem diversos modelos de arquiteturas de software, destes, no mundo dos jogos, destacam-se:

- **Cliente-Servidor:** modelo que distribui tarefas e cargas de trabalho entre fontes de recurso ou de serviço (servidores) a seus consumidores (clientes);
- **Computação distribuída:** interliga diversos nós de processamento, de forma que um processo grande/pesado possa ser executado por diversos nós (computadores), quando disponíveis, ou subdividido entre os computadores. Essa solução garante que uma tarefa qualquer possa ser divisível em várias subtarefas em paralelo;
- **P2P (Peer-to-peer, em tradução, ponto-a-ponto):** modelo em que cada ponto (computador) em uma rede está interligada, funcionando tanto como cliente quanto servidor;
- **Aplicação monolítica:** arquitetura de sistema operacional mais comum e antiga, na qual cada componente do sistema operativo está contido no núcleo do sistema. Trata-se de uma aplicação formada por vários módulos que são compilados separadamente e depois interligados, formando assim um grande sistema onde os módulos podem interagir;
- **MVC (Model-view-controller, em tradução, modelo-visão-controlador):** padrão de arquitetura de software que separa a aplicação em três camadas distintas. O modelo (model), que consiste nos dados da aplicação, nas regras de negócios, na lógica e nas funções. A visão (view) pode ser qualquer saída de representação dos dados, como uma tabela, menu, cenário com personagem, etc. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. Como terceira e última camada tem-se o controlador (controller), que faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do

MVC são a reusabilidade de código e a separação de conceitos, essa arquitetura é famosa e comum no desenvolvimento orientado a objetos, sendo um paradigma de programação que conheceremos neste capítulo.

O estudo de arquitetura de software é extenso, mas pode ajudar a entender ou mesmo a definir como um jogo deve ser. Esta etapa vai ajudá-lo a construir a infraestrutura completa do seu projeto, ou seja, a arquitetura é um modelo para construção ou implantação da infraestrutura do jogo. Como utilizamos a maioria das vezes ferramentas de editoração de jogos, que processam código de computador, vídeo e imagem em um mesmo ambiente, a acoplagem dentro dos modelos supracitados tem sido facilitada.

Dica: Duas referências da engenharia de software são os autores Roger S. Pressman e Ian Sommerville. Ambos são especialistas no assunto.

5.2. Teste de Desempenho

Em engenharia de software, testes de desempenho/performance são, em geral, testes realizados para determinar qual a capacidade de resposta e de estabilidade de um sistema, sob uma carga de trabalho específica. Pode servir para investigar, medir, validar ou verificar outros atributos de qualidade do sistema, tais como escalabilidade, confiabilidade e uso de recursos.

A Figura 25 exemplifica o teste de performance e sua representatividade, se enxergamos dessa forma, percebemos que a cada teste conseguimos identificar maneiras de balancear nosso projeto e tornar cada funcionalidade mais estável e funcional. A figura é simples, mas revela um intenso trabalho de desenvolvimento.

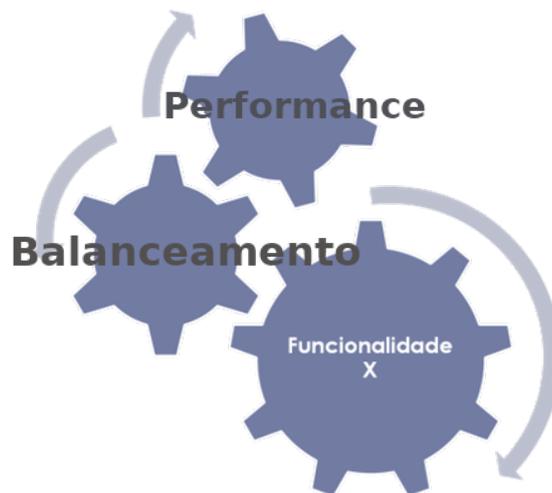


Figura 25. Representatividade do teste de performance e sua ação

Fonte: próprio autor.

Existem diversas ferramentas de teste de desempenho, aqui apresentaremos uma comum no mercado, geralmente utilizadas para sites, smartphones e tablets, trata-se do mecanismo de Inspeção da Google. Muitos editores de jogos atualmente exportam para HTML5²⁰ ou OpenGL²¹, abrindo suas aplicações em navegadores de internet.

Para essas aplicações, a Google, no navegador Google Chrome, possibilita abrirmos o “Inspector” (Inspetor) – que tem dois modos de funcionalidade, os quais serão explicados nos próximos parágrafos –, recurso da web, que mede vários tipos de dados de performance em tempo real. Para acessá-lo basta usarmos o atalho de teclado “Control + Shift + I” ou optar pelo botão direito do mouse na página que se está navegando, a partir do qual será exibida uma tela em que é possível selecionar o “Inspector” (Figura 26).

Voltar	Alt+Seta para a esquerda
Avançar	Alt+Seta para a direita
Recarregar	Ctrl+R
Salvar como...	Ctrl+S
Imprimir...	Ctrl+P
Transmitir...	
Traduzir para o português	
Exibir código fonte da página	Ctrl+U
Inspeccionar	Ctrl+Shift+I

Figura 26. Menu de contexto para seleção do Inspect (Inspetor/Inspeccionar)

Fonte: próprio autor.

20 HTML5 – (Hypertext Markup Language, versão 5) é uma linguagem para estruturação e apresentação de conteúdo para internet.

21 OpenGL – É uma biblioteca de rotinas gráficas e de modelagem, bi (2D) e tridimensional (3D), extremamente portátil e rápida, podendo ser aplicada a interfaces web.

No Inspetor, na aba “Performance”, você pode fazer testes para verificar o uso de memória, a alocação de banda, a taxa de frames por segundo e diversos outros dados, os quais podem ser vistos na Figura 27. Após o teste, temos um sumário de tempo e uso, onde podemos visualizar o que é mais envolvido no processamento do conteúdo na maior parte do tempo com seus picos.

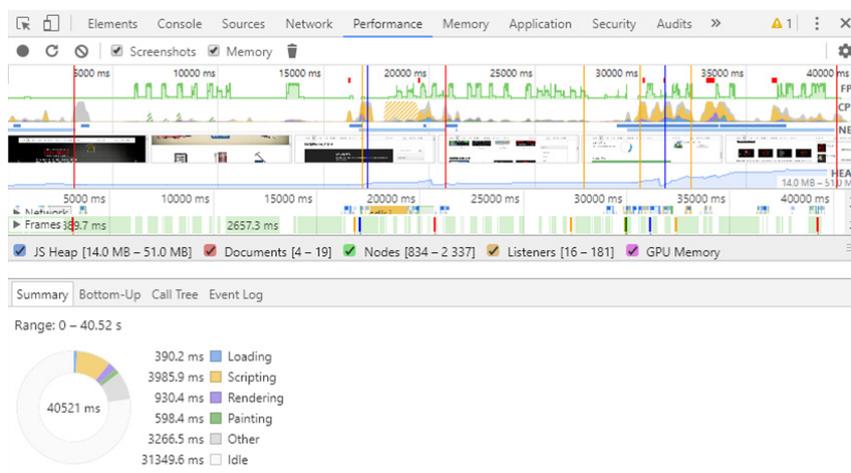


Figura 27. Teste de performance com o Google Inspect

Fonte: próprio autor.

Desse modo, podemos enxergar os momentos de maior consumo de memória ou de banda de internet e entender fluxos mais rotineiros e pesados. Com isso é possível tentar melhorar, em nosso projeto, esses processos, por exemplo, identificando e evitando falhas ou corrigindo problemas de infraestrutura e de programação baseados na performance.

Outra forma de utilizar o Inspetor é utilizando o modo de análise **device**, com esse mecanismo você pode testar a performance de sua aplicação em dispositivo mobile diretamente, ou seja, você conecta seu celular, tablet ou outro dispositivo no computador e começa a trocar informações entre seu computador e o dispositivo para testar sua performance. Nessa funcionalidade, a tela de seu dispositivo passa a ser transmitida no computador.

Para ativar o modo **device**, no browser Google Chrome, coloque o seguinte caminho na barra de endereço: “chrome://inspect/#devices”. Ao plugar seu celular, este aparecerá na lista (Remote Target), caso você tenha colocado um aplicativo para ser monitorado, será permitido ativar o inspetor, conforme Figura 28.

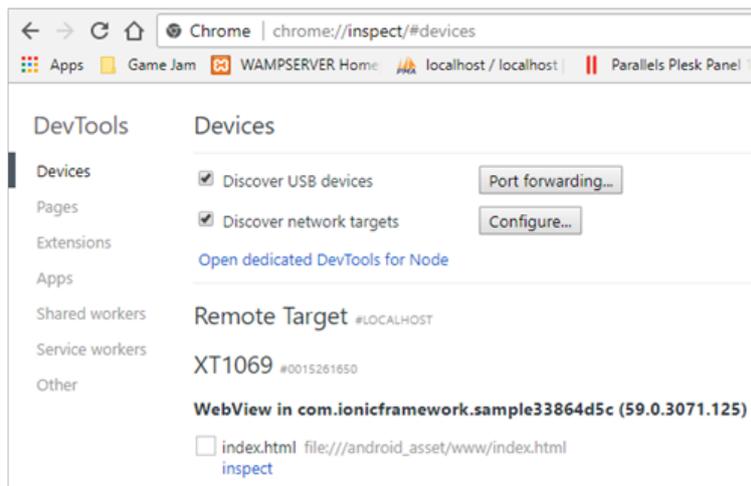


Figura 28. Captura de tela do inspetor modo device

Fonte: Próprio autor.

Ao clicar em “Inspect”, abre-se a tela para testes de performance e outras operações, conforme demonstrado na Figura 29. Note que, quando você interage no celular ou no navegador, os estados da aplicação mudam em tempo real, atualizando o visor tanto do computador quanto do dispositivo remoto.

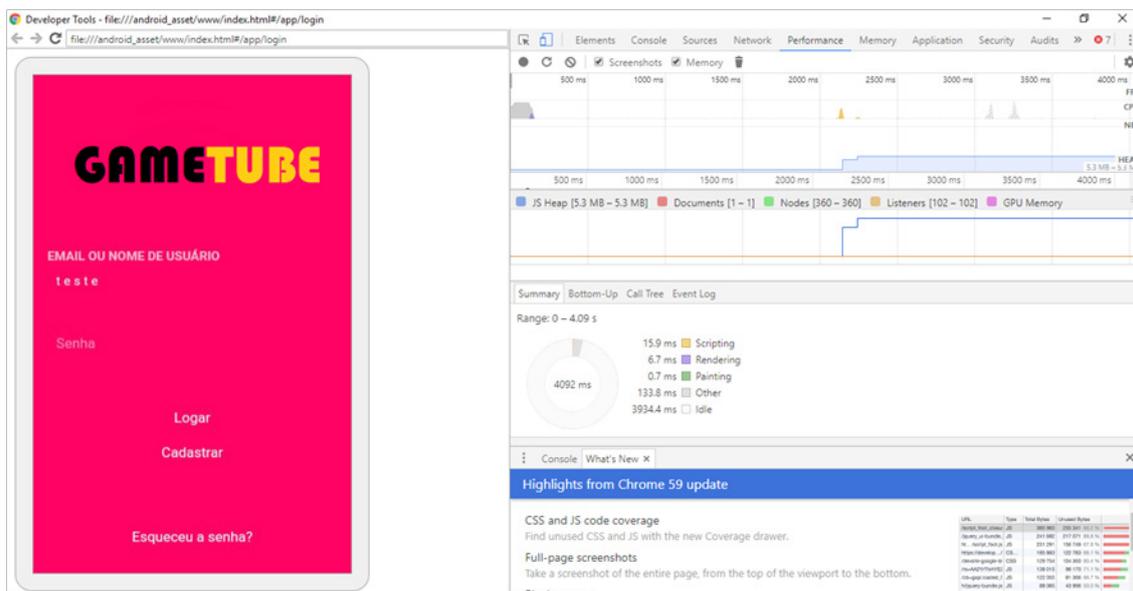


Figura 29. Captura de tela do Inspetor em visão modo device

Fonte: Próprio autor.

Possivelmente você deverá informar a seu dispositivo qual aplicativo está em depuração, a forma de efetuar isso depende da versão e do sistema de seu dispositivo, consulte o manual de desenvolvimento para seu aparelho. O “Inspetor” possibilita testar qualquer aplicação web que rode no browser. Cabe mencionar que muitas ferramentas e editores de jogos digitais possuem tecnologia e módulos para teste de desempenho, o que pode ser bem diferente do mostrado nesta seção, porém, possuem o mesmo intuito.

5.3. Postmortem

O postmortem (pós-morte) é uma documentação de tudo que foi realizado em um projeto. Durante todo ciclo do seu trabalho você irá corrigir problemas, encontrar soluções, criar novos conceitos e técnicas de trabalho, avaliar o que deu certo e errado, entre diversas outras constatações.

Essas ponderações que você pode fazer ao finalizar um projeto podem ser de interesse de outro projeto, assim, podem ser reutilizadas e aumentar a produtividade de seus trabalhos futuros. O postmortem não tem modelo definido, por isso, mapear todas as concretizações ao final e durante o processo de desenvolvimento é crucial para não correr o risco de esquecer etapas importantes.

5.4. Exemplo - Buga: A aventura de um Neandertal

Iremos iniciar nosso desenvolvimento a partir de uma síntese do postmortem do game *BUGA! A aventura Neandertal*²². Trata-se de um jogo de aventura e ação dirigido em terceira pessoa e ambientado em um cenário pré-histórico, no qual o jogador compreende a criação de instrumentos e de tecnologias antigas que ajudaram na definição do homem moderno, como estacas de madeira para proteção, roupas de pele, vivência em cavernas e cabanas, entre outros aspectos.

Segundo os desenvolvedores Reinoso e Moura (2016, p. 396):

O propósito do trabalho é demonstrar o uso de uma ferramenta computacional interativa, “Buga! A aventura de um Neandertal” e sua capacidade para envolver alunos dentro de uma abordagem pedagógica praticada em sala de aula para aprendizagem de história, acerca da evolução da humanidade. As intervenções para com a aplicação objetivam seu uso prático.

No jogo você assume papel de um membro de uma tribo Neandertal e tem que evoluí-la até a contemplação da sociedade moderna. O jogo é simples, você joga a partir de

22 BUGA! A aventura Neandertal – jogo publicado no V Congresso Brasileiro de Informática na Educação (CBIE) por um professor e sua aluna do IFES, campus Colatina. Disponível em: <<http://br-ie.org/pub/index.php/wcbie/article/view/6959/4833>>.

análise de diálogos entre NPCs²³, os quais direcionam a tomada de ações que o ajudam a progredir. O jogo possui 30 missões possíveis de serem trabalhadas de forma aleatória. A Figura 30 apresenta uma captura de tela do jogo.



Figura 30. Captura de tela do game BUGA! A aventura de um Neandertal

Fonte: Reinoso e Moura (2016).

Na Figura 30, temos diversas representações para orientação do jogador, uma HUD²⁴ (Heads-up-display, tela de alerta) como a disposição das missões (1) e a barra de evolução (2), que marca o início e o fim do jogo, os atributos (3) vão aumentando de acordo com a evolução do jogo (3) e são necessários para concluí-lo. O jogador (4) ao se mover, tem essas informações o acompanhando (3) com o movimento da tela, como na figura. À medida que as missões são finalizadas e os atributos aumentam, o jogador irá liberar habilidades e conhecimentos, como o fogo, a pesca, a culinária, domesticação de animais e os princípios da cidadania. Ao completar estes conhecimentos ele encerrará o jogo, completando-o.

Com uma temática simples, os autores registram uma análise acerca da concepção do jogo enquanto instrumento de lazer e de aprendizagem, sua mecânica e os resultados de implementação. O jogo foi testado com jogadores reais, os quais afirmam que o jogo é interativo e possui muitos personagens, missões e um cenário considerável-

23 Non-Player Character (NPC) - Um personagem não jogável/manipulável (em inglês: non-player character ou NPC), ou seja, é um personagem de qualquer jogo eletrônico que não pode ser controlado por um jogador, mas que se envolve de alguma forma no enredo do jogo (REINOSO; MOURA, 2016).

24 HUD (Heads-up-display, em tradução, tela de alerta) - Local em que se insere a barra de status, a quantidade de vidas, os pontos e os informativos diversos importantes ao usuário em tempo de jogo.

mente grande, incentivando a exploração, atendendo, assim, tanto ao lazer, quanto a aprendizagem. O uso de muitas imagens e objetos representativos foi notado pelos jogadores responsáveis pela avaliação, tal característica ajuda os alunos a compreenderem diversos conceitos, correlacionando-os com o mundo real, desta forma, o jogo transmitiu a experiência um pouco mais realista das descobertas feitas em ambiente virtual.

Notoriamente, podemos identificar nestas afirmativas muitas metas estéticas, a Figura 19, apresentada no capítulo anterior, é o diagrama de estado do game *BUGA! A aventura Neandertal*, fazendo parte dessa documentação, a qual possibilita ao desenvolvedor uma melhor visão do gameplay e dos ajustes do fluxo do jogo.

5.5. Conhecendo as ferramentas

Antes de começarmos a programar um jogo, iremos conhecer as ferramentas tecnológicas que você irá aprender a utilizar, começaremos por apresentá-las e explicar seu funcionamento básico, necessário para trabalhar no projeto que iremos desenvolver. Vale ressaltar, que estas ferramentas não o limitam, ou seja, caso você queira usar outros softwares, fique à vontade.

5.6. Pixel arte com Gimp

Nesta seção aprenderemos a interagir com o GIMP, uma ferramenta para composição e editoração de fotos e imagens. Para tanto, iremos desenvolver pixel arte²⁵ como forma de entender o software e, ao mesmo tempo, trabalhar um conceito artístico famoso e comum aos jogos digitais.

5.6.1. Interface

A Figura 31 apresenta a interface do GIMP. Inicialmente você pode ter seus painéis distribuídos em janelas individuais, mas para ativar o modo de janela única, como apresentado, acesse **Janelas > Modo de janela única**. Ao redor de nossa **área de trabalho** ou, simplesmente, “**cena**”, temos os **menus** (em vermelho), que nos permitem ter acesso a todas as operações do programa, são elas:

- **Arquivo:** permite salvar as suas modificações, importar alguma imagem, abrir imagens recentes, fechar o projeto atual, entre outras operações;
- **Editar:** podemos obter o histórico de operações, desfazer a última coisa que realizamos (atalho: CTRL+Z), recortar (atalho: CTRL+X), colar (atalho: CTRL+V),

25 Pixel arte – Formato artístico que integra pontos baseados em pixels, que são os menores pontos de uma imagem digital.

entre outras;

- **Seleção:** possibilita configurar uma seleção, remover seleção (atalho: SHIFT+CTRL+A), “flutuar” uma parte selecionada (atalho: SHIFT+CTRL+L), aumentar ou diminuir uma seleção, etc.;
- **Visualizar:** podemos acender grades para facilitar uma edição, trabalhar em modo tela cheia, mudar o zoom (atalho: CTRL+Scrool/girar o botão do mouse para cima ou para baixo), entre outros;
- **Imagem:** permite ações de redimensionamento para uma imagem ou cena, mudanças nos tons de cores, rotacionamento ou corte de uma imagem para uma seleção, combinação camadas, entre outros;
- **Camada:** possibilita criar camadas, além de agrupá-las e/ou duplicá-las, realizar operações de recorte e redimensionamento, etc.;
- **Cores:** faculta a modificação do contraste, da matiz, da tonalidade, dos ajustes automáticos, entre outros;
- **Ferramentas:** confere acesso a todas as ferramentas do programa e seus painéis;
- **Filtros:** permitem a criação de efeitos visuais em nossas produções, como efeitos de iluminação, distorção, entre outros. Você também pode criar filtros através do Script-fu, mas para isso é necessário ter um bom conhecimento sobre programação;
- **Janelas:** possibilita acessar todas as janelas de diálogos que necessitamos ou que tenhamos fechado;
- **Ajuda:** configura-se como material que oferece suporte ao GIMP e informações do programa.

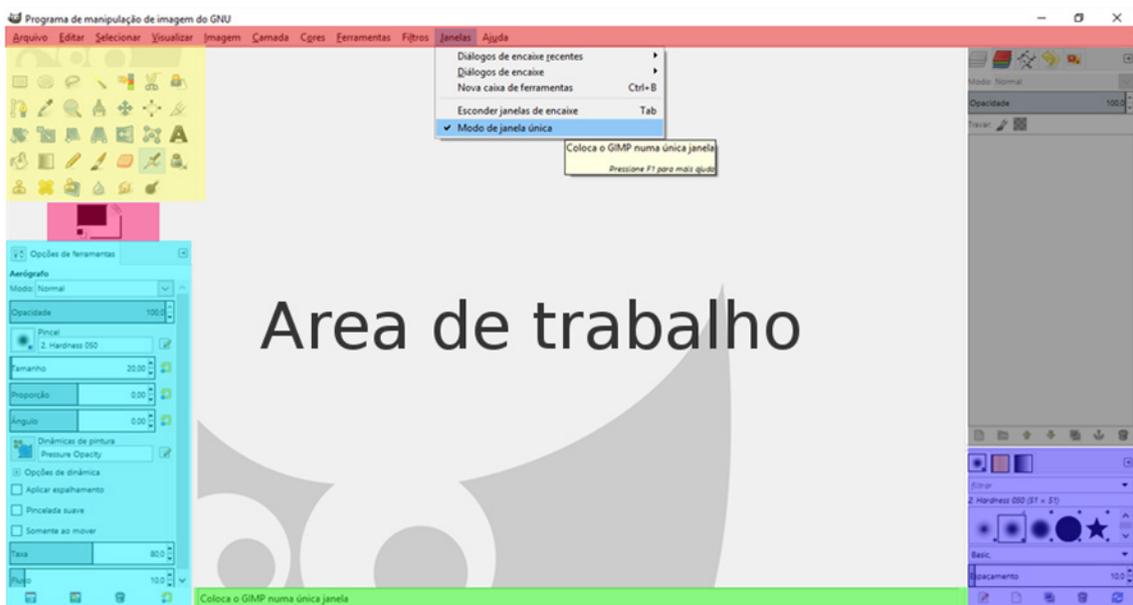


Figura 31. Interface em janela única do GIMP

Fonte: Próprio autor.

A **caixa de ferramentas** (em amarelo) permite a você ter acesso ao conjunto de instrumentos para edição de imagens do GIMP, ao selecionar qualquer um deles as **opções de ferramenta** (em azul claro) mudam. Cada ferramenta pode ter modos de funcionamento ou mais de um modo de operação, essas informações aparecerão **na caixa de informação** ou na **caixa de status** (em verde).

Para alterar ou encontrar uma cor durante a composição gráfica, você pode selecioná-la na **caixa de seleção de cores** (em rosa) e, de acordo com a ferramenta utilizada, podemos ter um pincel adequado ao que queremos editar, para tanto, utilize a **caixa de pincéis** (em azul escuro).

Para nos ajudar a organizar um projeto gráfico, o GIMP possui um sistema de camadas, no qual **podemos** separar nossa produção em partes individuais, isso ajuda na criação de arte para jogos, pois temos que modular cenas e artefatos diversos, para então montar nossos jogos. O **painel de camadas** (em cinza escuro) permite-nos criar, ajustar e visualizar as camadas que compõem nossa cena/projeto.

5.6.2. Configuração para pixel arte

Inicie um projeto novo, acesse o menu Arquivo > Nova (atalho: CTRL+N). Para nosso estudo, a cena deverá ser pequena, com 32 pixels de altura por 32 pixels de largura, conforme apresentado na Figura 32.

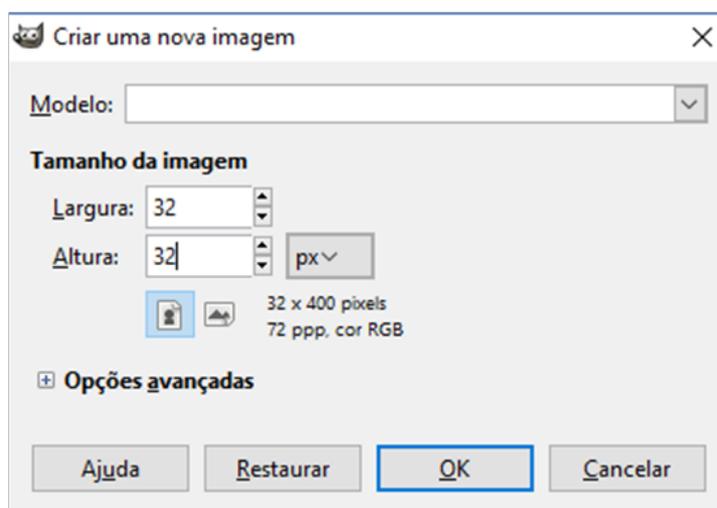


Figura 32. Criando uma nova cena no GIMP

Fonte: Próprio autor.

Selecione a ferramenta **lápiz** e configure o **pincel** para “Hardness 100”, conforme apresentado na Figura 33. Para você entender o conceito de **pincéis**, teste vários deles, compreendendo “caminhos” ou “máscaras” que seu desenho irá seguir quando interagir com as ferramentas em sua cena. Você terá uma visão mais apropriada se testá-los com a **ferramenta pincel** (atalho: tecla P).

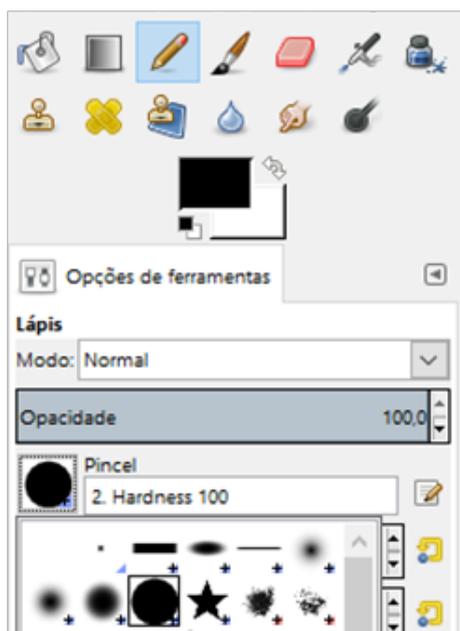


Figura 33. Configuração do lápis para pixel arte

Fonte: Próprio autor.

Para desenhar pixel arte com maior controle, o GIMP possibilita a criação de **guias**, linhas que podemos traçar no desenho sem alterá-lo, para ajudar na organização (atalho: CTRL+Shift+T permite visualizar e esconder as guias). Também podemos acender uma **grade**, dividindo nossa cena em partes menores. Para tal, acesse o menu **Imagem > Configurar grade**, a janela de configuração para a grade, como apresentado na Figura 34, irá ser aberta. Nesta, ajuste o **espaçamento** para 1 de largura por 1 de altura e confirme, clicando em “OK”.

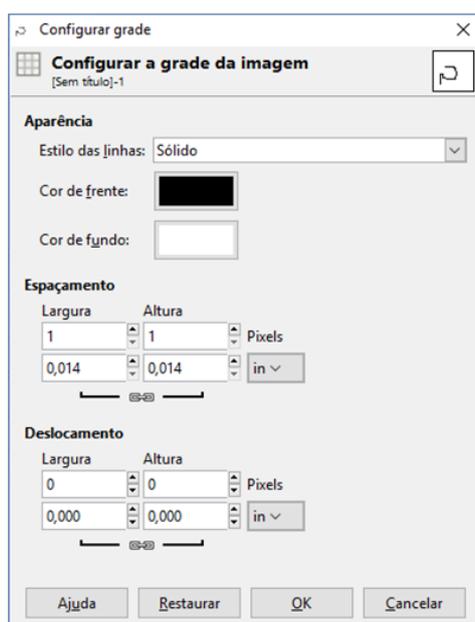


Figura 34. Janela de Configuração de grade

Fonte: Próprio autor.

Após a configuração, acesse o menu **Visualizar > Exibir grade**. Agora você terá sua tela subdividida, dessa forma, poderá desenhar os pixels e identificar sua localização no desenho. Repare que nossa cena é pequena, logo, será preciso usar a **ferramenta de zoom** (atalho: tecla Z, ou podemos utilizar CTRL+Scroll/botão de rolagem do mouse para frente e para trás) para ter maior controle de nossa produção.

5.6.3. Desenhando pixel arte

Agora, com nossa cena configurada, vamos a um exercício de quatro etapas simples para um pixel arte. A primeira é desenhar a **base do seu desenho**, que seria seu esquema/esboço ou esqueleto, para tanto, selecione a ferramenta lápis com o pincel “Hardness 100”, exponha a grade e faça como mostrado na Figura 35. Se necessário, você pode **umentar** e **diminuir** o tamanho/raio de uma ferramenta no GIMP, que seria sua área de atuação na cena (para aumentar use a tecla] e diminuir a tecla [).

Perceba que, no esboço apresentado, temos apenas uma imagem como a sombra do que será a ilustração, isso ajuda a termos controle de tamanho, volume e forma do que está sendo representado.

Durante esse processo, faça correções e ajustes até ficar satisfeito com a configuração da base, pois é a partir desta que teremos um desenho completo. Assim, por mais simples que possa parecer, faça com calma e com atenção aos detalhes.

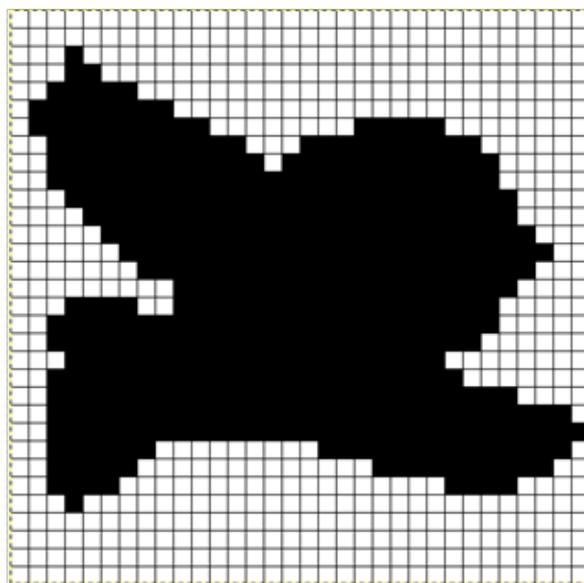


Figura 35. Base ou esboço do desenho

Fonte: Próprio autor.

Quando obtiver um esboço satisfatório, iremos inserir as cores bases, nessa etapa, o desenho passa a ser definido em relação à distribuição espacial, ou seja, profundidade, direção e sentido do que está sendo desenhado.

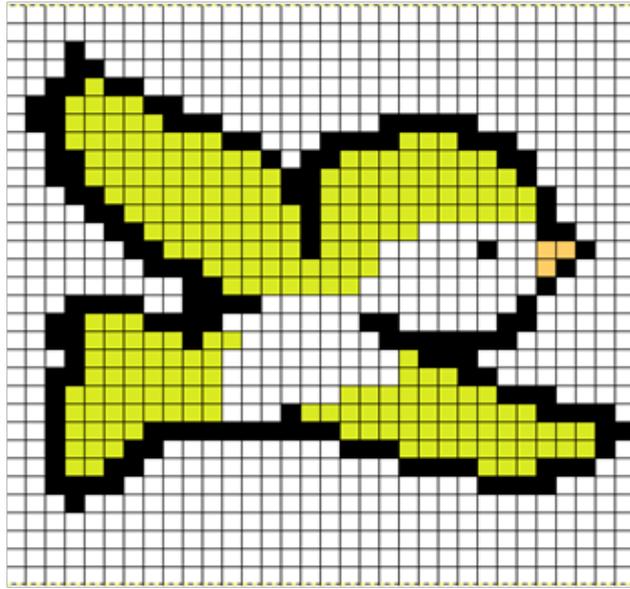


Figura 36. Esboço colorido com cores base

Fonte: Próprio autor.

Na terceira etapa, iremos iluminar a imagem a partir das cores base do desenho, selecione cores mais escuras para sombra e mais claras para brilho, utilize a **ferramenta de seleção de cores** (atalho: tecla O) e ajuste-as na caixa de seleção de cores. Aplique a cor mais clara na direção de onde vem a luz, no exemplo da Figura 37 seria da zona superior direita, nessa perspectiva, a cor mais escura aplicamos onde a luz não tocaria.

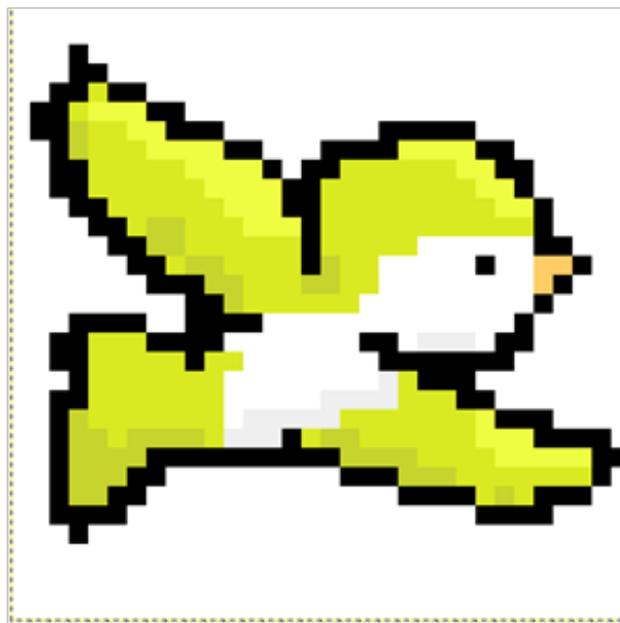


Figura 37. Desenho com iluminação

Fonte: Próprio autor.

Na quarta etapa, realçamos a profundidade do desenho, temos vários pontos de “pressão” para dar sensação de volume à composição. Para realçar ainda mais o desenho, aumentando agora o brilho e contraste entre as cores, você pode usar a **ferramen-**

ta de sub/super exposição (atalho: Shift+D). Ao usar a ferramenta você alterna entre sub (normal) e super (segure CTRL) exposição para realçar a imagem. A Figura 38 apresenta o resultado da aplicação da ferramenta.

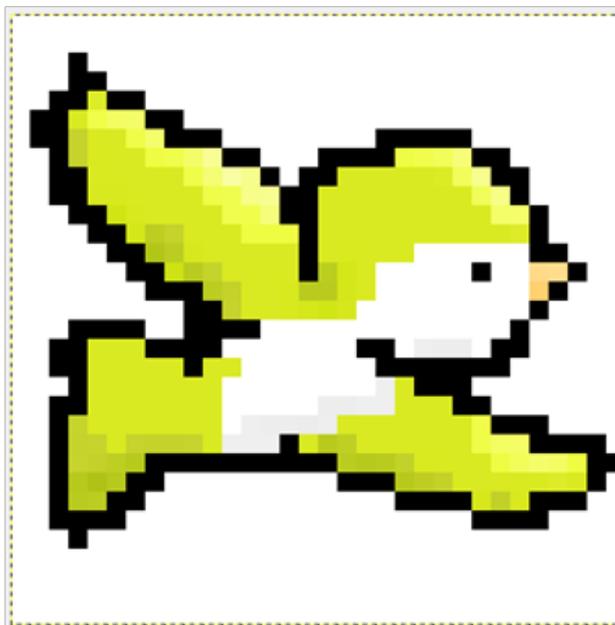


Figura 38. Figura realçada com exposição de cores

Fonte: Próprio autor.

Com as etapas explicitadas você pode desenhar diversos outros elementos para seu jogo, como partes de um cenário, personagens, objetos, entre outros. A prática na composição de pixel arte irá te ajudar a fazer suas ilustrações cada vez melhor. Nessa parte de seu estudo, orientamos que vá para a seção de 6.13 Exercícios, deste capítulo, e faça as atividades relacionadas a pixel arte.

5.7. Arte vetorial com Inkscape

O Inkscape é uma ferramenta que nos permite editar imagens e documentos vetoriais. Nesse ponto, cabe esclarecer que um **vetor**, quando estamos fazendo referência a desenho ou arte gráfica, concerne ao uso de primitivas geométricas, como linhas, curvas, pontos, polígonos ou formas, sendo estas baseadas em expressões matemáticas.

5.7.1. Interface

A interface do Inkscape foi projetada de forma bem comum aos softwares de vetorização encontrados no mercado. Tal recurso é ilustrado na Figura 39 do modo como é apresentado ao iniciar a aplicação.

Observe que, na parte superior da janela, temos a barra de menus (em vermelho), na qual podemos encontrar todas as funcionalidades do programa. Dos menus disponíveis e suas funções, temos:

- **Arquivo:** apresenta funcionalidades para salvar, abrir, importar e exportar arquivos, bem como imprimir o documento em edição, entre outras;
- **Editar:** possibilita desempenhar operações de copiar, colar, recortar, encontrar um elemento, entre outras;
- **Exibir:** contém operações de exibir grades e guias, painéis e funções afins;
- **Camada:** permite-nos adicionar e remover camadas, desprender ações para organizá-las e etc.;
- **Objeto:** oferece operações em cima de vetores (objetos), permitindo-nos agrupá-los, ordená-los, alinhá-los e arranjá-los, e operações de preenchimento e contorno, entre outros;
- **Caminho:** possibilita expandir e combinar formas vetoriais, além de rasterizar imagens e outras funções;
- **Texto:** oportuniza a manipulação de textos, definindo fonte e tamanho, além de permitir convertê-los, ajustar ortografia e operações afins;
- **Filtros:** permitem a manipulação de efeitos visuais em nossas produções. Neste menu temos diversos deles para aplicar em nossos vetores;
- **Extensões:** comportam funcionalidades adicionais à ferramenta, as quais que estendem suas ações convencionais;
- **Ajuda:** apresenta informações da ferramenta, arquivos e links de ajuda e de suporte.

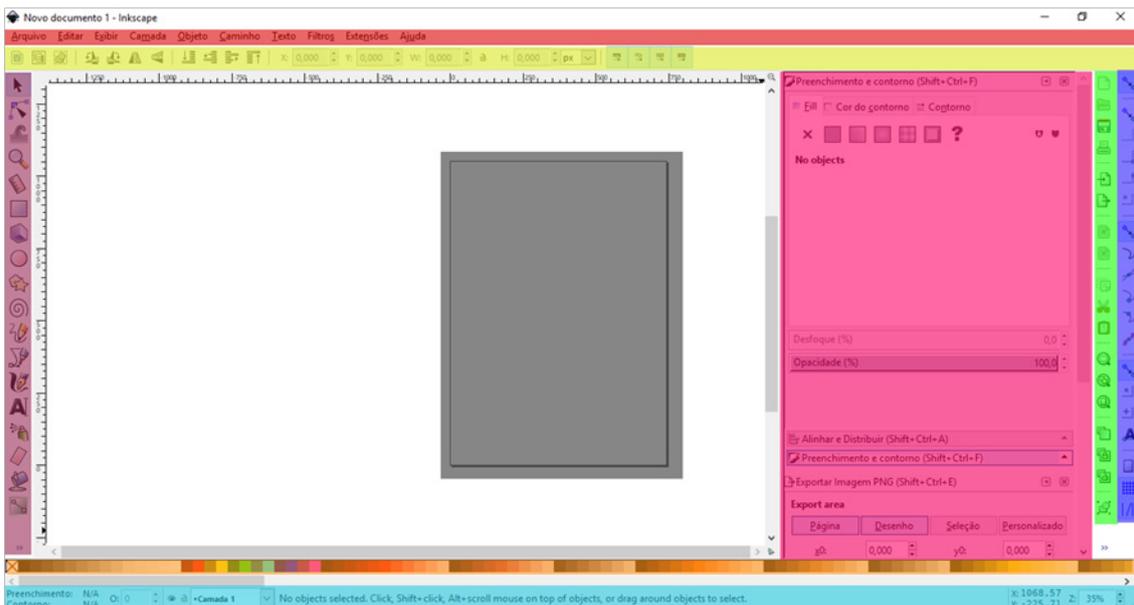


Figura 39. Interface do Inkscape

Fonte: Próprio autor.

Ao iniciar um projeto no Inkscape, temos acesso a uma cena, chamada aqui de *canvas* (área em branco), com uma folha de papel que é nossa *página* (em cinza), tudo que estiver dentro desta fará parte de nossa composição.

Quando selecionamos uma ferramenta no *menu de ferramentas* (em roxo), suas regras e ajustes podem ser visualizados na *barra de configurações de ferramenta* (em amarelo). À medida que interagimos e abrimos menus, para que novos painéis possam ser carregados, estes são inseridos no chamado *dock* (em português, doca, identificado em rosa), que vai expandindo para comportar os inúmeros painéis possíveis.

Para ajustar nossas produções e elementos em tela, combinando-os a uma grade, ou à página que estamos trabalhando, temos a *Snap bar* (em azul). Ao lado desta, temos a *barra de ações* (em verde) que comporta diversas operações em forma de acesso rápido, como copiar, colar, recortar, salvar, entre outras.

Durante a manipulação e edição dos elementos, o programa fornece-nos feedback de tudo que está ocorrendo na *barra de status* (em azul claro), bem como permite que mude a cor de frente e de fundo de qualquer ferramenta de maneira rápida com a paleta de cores disposta (em laranja).

5.7.2. Criando formas no Inkscape

Para criar formas automáticas no Inkscape você pode usar a *ferramenta de retângulo* (atalho: tecla F4), que cria paralelepípedos e quadrados com bordas retas e arredondadas, a *ferramenta elipse* (atalho: tecla F5), que cria círculos e segmentos de um

círculo, a **ferramenta polígono** (atalho: tecla *), que cria desde polígonos regulares à estrelas, e por fim a **ferramenta espiral** (atalho: tecla F9), que gera espiral. A Figura 40 apresenta formas geradas com as ferramentas.

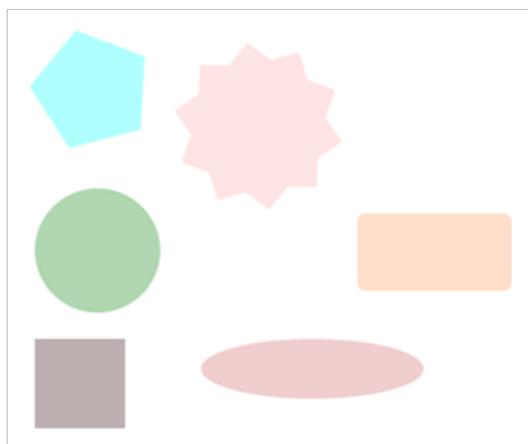


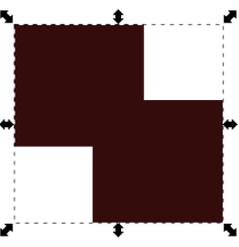
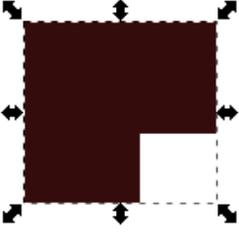
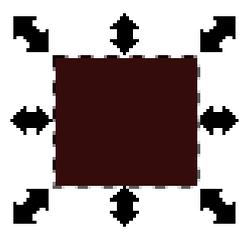
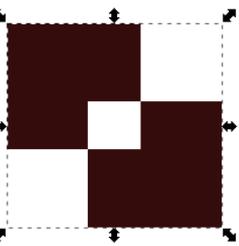
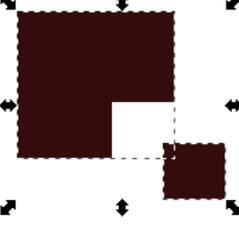
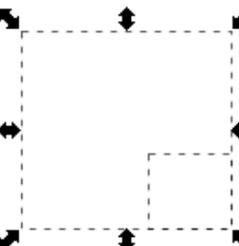
Figura 40. Formas geradas no Inkscape

Fonte: Próprio autor.

É imperativo destacar que, com as ferramentas **mão livre** (atalho: tecla F6), **caneta bézier** (atalho: tecla Shift+F6) e **caneta caligráfica** (atalho: tecla Ctrl+F6), podemos gerar formas diversas em formato livre. Selecione cada uma delas, altere suas configurações e se ambiente. Vamos concentrar-nos aqui em operações com formas e rasterização.

A criação de objetos pode ser feita a partir de formas primitivas (formas simples), as quais irão amparar o desenvolvimento de novas formas. O menu **Caminho** fornece diversas ações para moldá-las.

Tabela 2. Operações com objetos e caminhos

		
União	Diferença	Interseção
		
Exclusão	Divisão (2 partes geradas)	Cortar caminho

Para aprendermos a criar vetores, vamos desenvolver um *emoticon*. Veja as etapas ilustradas na Figura 41. Primeiro, deve-se criar um círculo, selecione a **ferramenta elipse**, segure a tecla CTRL e clique com o mouse na página para iniciar o desenho. Perceba que, com o CTRL pressionado, a ferramenta mantém a proporção. Em seguida, selecione a cor amarela na paleta de cores para a forma gerada.

A próxima etapa consiste em criar mais duas esferas, com elas faremos os olhos de nosso desenho, uma deverá ser branca (globo ocular) e outra azul (íris). Com a **ferramenta seletor** (atalho: tecla F1), disponível em sua **barra de configurações de ferramenta**, além de mover nossos objetos, você pode arrancar e espelhar os itens da maneira correta.

Após criar os objetos junte o globo ocular com a Iris, como um grupo, selecione os itens a que deseja agrupar com a tecla Shift pressionada, agrupe-os através do menu **Objeto > Agrupar** (atalho, teclas CTRL+G para agrupar e Shift+CTRL+G para desagrupar), deste modo, diferentemente das ações de caminho, as partes continuam individuais mas são manipulados como um só membro.

Agora, para a boca do desenho, crie uma elipse, selecione-a e a converta para caminho no menu **Caminho > Converter para caminho** (atalho: Shift+CTRL+C). Selecione a **ferramenta editor de nós** (atalho: tecla F2), com essa ferramenta podemos editar pontos de um caminho ou polígono no programa. Ao alcançar essas etapas, realize atividades relativas ao Inkscape na seção 6.13 Exercícios deste capítulo.

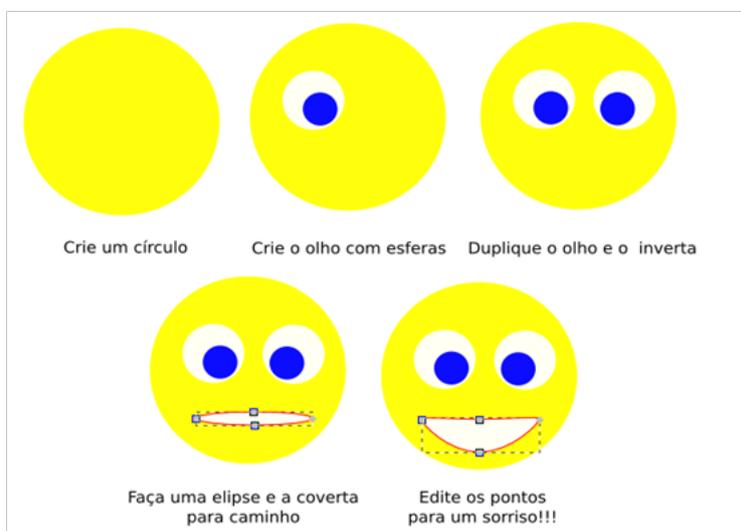


Figura 41. Passos para criação de um emoticon.

Fonte: Próprio autor.

Nota: Sobre a criação de formas com o Inkscape, a documentação oficial da ferramenta disponibiliza o seguinte tutorial: https://inkscape.org/pt-br/doc/tutorials/shapes/tutorial-shapes_pt_BR.html.

5.7.3. Rasterizando imagens com Inkscape

Muitas vezes como artistas, fazemos esboços manuais. Comumente, um esboço manual torna-se arte final, mas o retrabalho é grande, por isso, com o Inkscape podemos transformar desenhos manuais em arte digital com poucos cliques, através da função de rasterização.

A opção de rasterização está em **Caminho > Rasterizar bitmap** (atalho: Shift+Alt+B). Para testar a funcionalidade, crie um novo projeto e arraste uma imagem de seu computador para a página ou copie e cole. Ao fazer isso, selecione essa imagem e acione a opção de rasterização. A Figura 42 apresenta a janela com as opções para a rasterização.

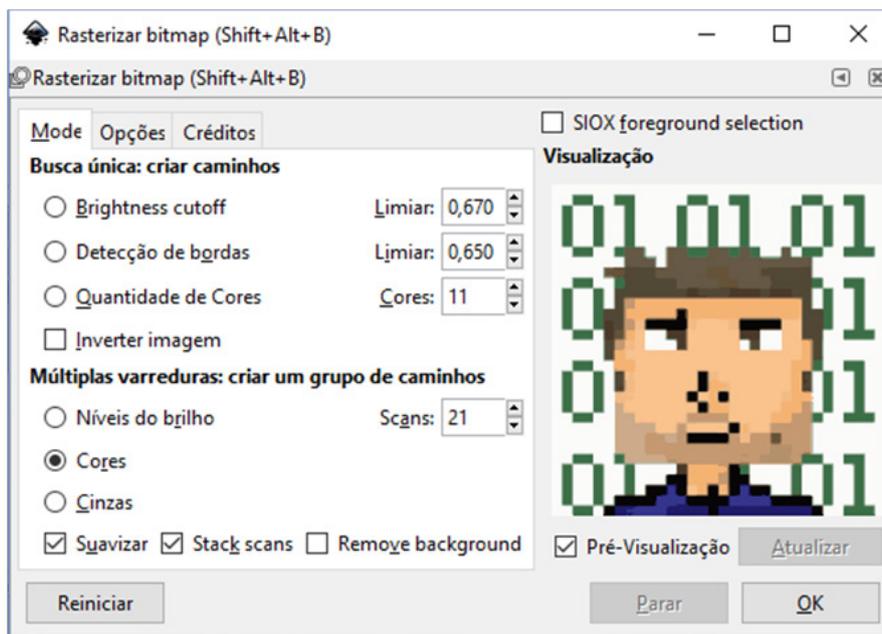


Figura 42. Janela de operações para rasterização

Fonte: Próprio autor.

No exemplo, foi selecionado o modo de múltiplas varreduras, o qual cria um grupo de caminhos por cores, ativada a opção de “Pré-Visualização”, à medida que escolher outras opções, a imagem da janela apresenta o resultado a ser obtido. Ao aceitar a modificação, teremos um conjunto de vetores criados a partir da imagem, com a ferramenta editor de nós. Podemos ver isso com maior clareza na Figura 43. Assim, a partir da imagem resultante, você pode além de obter uma imagem vetorizada, modificar cores e formas desse novo objeto criado.

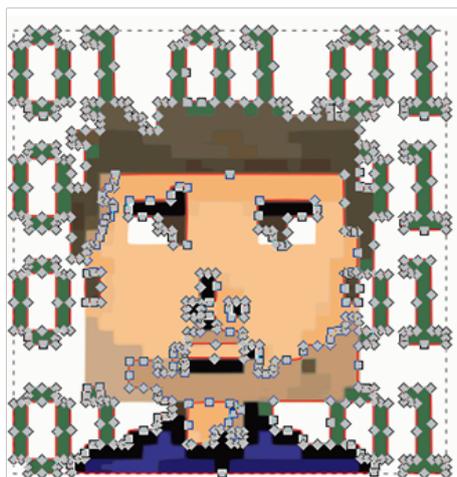


Figura 43. Imagem resultante da rasterização

Fonte: Próprio autor.

Sobre rasterização a documentação do inkscape disponibiliza o seguinte tutorial: https://inkscape.org/pt-br/doc/tutorials/tracing/tutorial-tracing.pt_BR.html

5.8 Conhecendo o Godot Game Engine

Uma engine de jogo (em português, motor de jogo) é uma ferramenta complexa e completa que permite desenvolver e rodar todo um jogo com sua lógica, com renderização²⁶ gráfica e com músicas e sons, tudo em perfeita harmonia. Segundo os desenvolvedores (GODOT ENGINE, 2017):

O Godot Engine é um motor de jogo multi-plataforma, equipado com recursos para criar jogos 2D e 3D a partir de uma interface unificada. Ele fornece um conjunto abrangente de ferramentas comuns, para que os usuários possam se concentrar em fazer jogos sem ter que reinventar a roda. Os jogos podem ser exportados em um clique para uma série de plataformas, incluindo as principais plataformas desktop (Linux, Mac OSX, Windows), além de plataformas móveis (Android, iOS) e baseadas na web (HTML5).

O Godot é completamente gratuito e *open source*, ou seja, possui código aberto, além disso está sob a licença MIT²⁷ muito permissiva. A aplicação não possui restrições ou royalties. Os desenvolvedores garantem que os jogos desenvolvidos são de seus criadores, “até a última linha do código do motor” (GODOT ENGINE, 2017).

O desenvolvimento do Godot Engine é totalmente independente e conduzido pela

²⁶ Renderizar – Resultado final do processamento digital.

²⁷ Licença MIT - Licença de programas criada pelo Instituto de Tecnologia de Massachusetts (MIT). Você pode obter a licença em: <https://opensource.org/licenses/mit-license.php>.

comunidade, capacitando os usuários para ajudar a moldar seu mecanismo de modo que corresponda suas expectativas. O projeto é apoiado pelo Software Freedom Conservancy²⁸ sem fins lucrativos.

Logo, você pode obter uma cópia do software Godot Engine via internet, no endereço oficial: <https://godotengine.org/>. Ao realizar o download da aplicação, obtenha também o pacote “Export Templates” (que permite exportação para várias plataformas) e os “Demos e templates” (que contêm demonstrações e projetos modelo) para conhecer as funcionalidades da ferramenta e se aprofundar.

As seções a seguir foram construídas em cima de um plano de aprendizagem para ensinar desde pessoas leigas à conhecedores de programação e para ambientar e ensinar como usar a Godot Engine. Elas foram retiradas e criadas a partir da documentação oficial da ferramenta, também livre.

Nota: A documentação oficial para a Godot Game Engine encontra-se no seguinte endereço: <http://docs.godotengine.org/>. Essa sessão retirou diversos trechos e partes da documentação citada, fazendo adaptações e reestruturações para seu estudo e entendimento.

5.8.1 Nódulos ou Nó

No Godot, quando nos referimos a um nóculo (nó ou, no caso do programa, “Node”), estamos fazendo referência a um elemento básico de um jogo, que possui as seguintes características (a Figura 44 apresenta a sua forma estrutural):

- Nome;
- Propriedades editáveis;
- Pode receber um retorno de chamada (callback) para processar a cada quadro (frame);
- Pode ser estendido (para ter mais funções);
- Pode ser adicionado a outros nós como filhos.

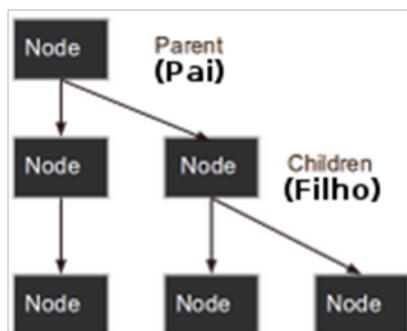


Figura 44. Estrutura de nós no Godot Engine

Fonte: Godot Game Engine (2017).

Adaptado pelo autor.

28 Software Freedom Conservancy – Organização sem fins lucrativos de Nova York que ajuda a promover, melhorar, desenvolver e defender projetos de Software Livre e de Código Aberto. Consulte-a em: <https://sfconservancy.org/>.

Como podemos perceber a estrutura de nós se comporta como uma árvore, logo, cada nó pode ter vários filhos e esses filhos, mais filhos. Essa estrutura é prática e eficiente, ajudando os desenvolvedores a organizar suas cenas.

5.8.2 Cenas

De maneira bem simples, uma **cena** é uma composição ou agrupamento hierárquico de **nós** (vide Figura 45). Podemos editar cenas 2D e 3D no Godot para um mesmo projeto, cada cena é individual. O editor foi criado baseado na ideia de compor cada cena inserindo os **nós** necessários a cada uma delas. Temos as seguintes propriedades de uma cena:

- Sempre tem apenas um **nó** raiz;
- Pode ser salva no disco e carregada de volta;
- Pode ser instanciada;
- Executar um jogo significa executar uma cena;
- Pode haver várias cenas em um jogo, mas, para iniciá-lo, uma delas deve ser selecionada para ser carregada primeiro.

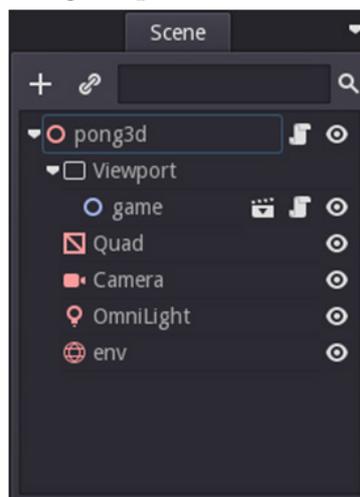


Figura 45. Hierarquia de nós para uma cena no Godot Game Engine

Fonte: Próprio autor.

5.8.3 Configurando um novo projeto

Ao iniciar o programa teremos uma janela como apresentado na Figura 46. Esta é a **tela de gerenciamento de projetos** com os nossos projetos recentes dispostos na aba “Project list” (Lista de projetos). Para recarregar a lista, caso um projeto não apareça ou por algum motivo você o tenha movido, execute a busca do projeto clicando no botão “Scan” (Escanear) ou carregue o projeto com a opção “Import” (Importar).

Ao selecionar um projeto, para carregá-lo, basta selecioná-lo na lista e clicar no botão “*Edit*” (Editar), caso queira executar o jogo sem abrir o projeto utilize a operação “*Run*” (Rodar/Executar), dessa forma não gastamos recurso computacional.

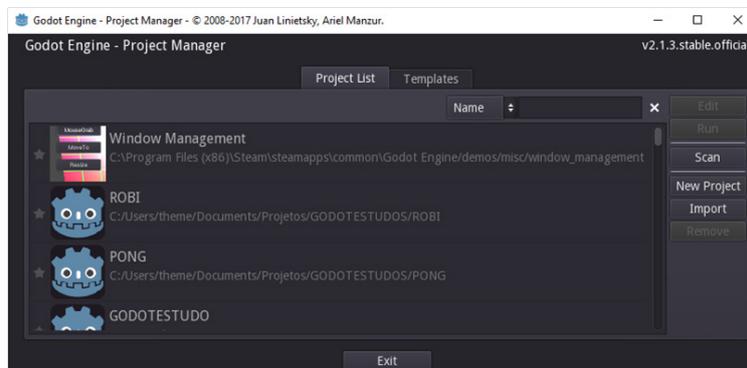


Figura 46. Janela de gerenciamento de projetos, aba Project list

Fonte: Próprio autor.

Outra aba disponível nesta janela é “*Templates*”, que reúne modelos a serem seguidos para construção de um jogo, dessa forma podemos diminuir nossa curva de aprendizagem, uma vez que podemos criar nossos projetos a partir de planejamentos pre-existentis.

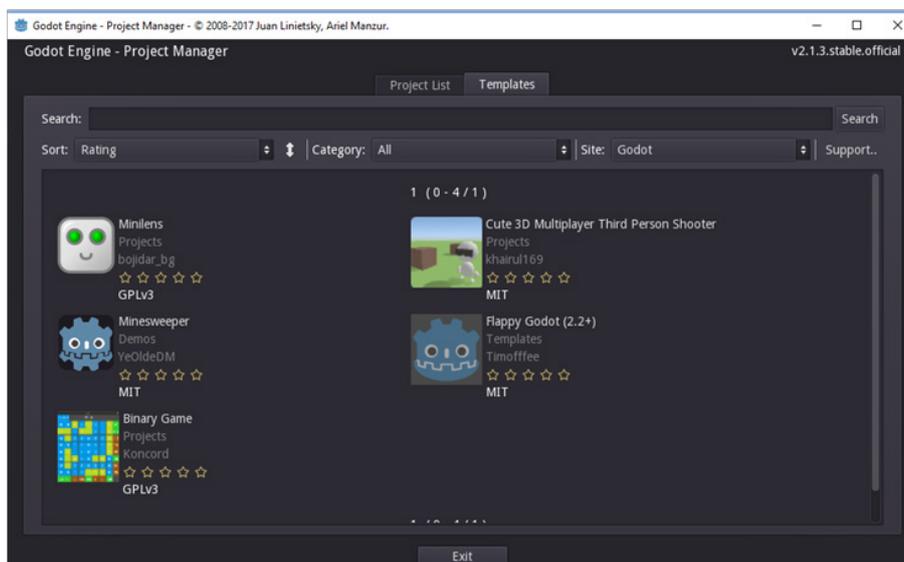


Figura 47. Janela de gerenciamento de projetos, aba Templates

Fonte: Próprio autor.

Para criar um novo projeto, deve-se voltar à aba “*Project list*” e clicar no botão “*New Project*” (Novo Projeto). Uma janela para criação do projeto irá aparecer, nela você irá encontrar um diretório para salvar o projeto em seu computador, para isso clique no botão “*Browser*” (Navegador) e navegue até um local de seu conhecimento, por fim, com o objetivo de manipular o projeto a seu critério, redija um nome para seu trabalho em “*Project Name*” (Nome do projeto) e clique em “*Create*” (Criar).

Na Figura 48 foi criado um projeto chamado “Meu primeiro Jogo” dentro do dire-

tório de documentos do usuário do computador. Após criar um projeto, selecione-o na “Project list” e clique no botão “Edit”.

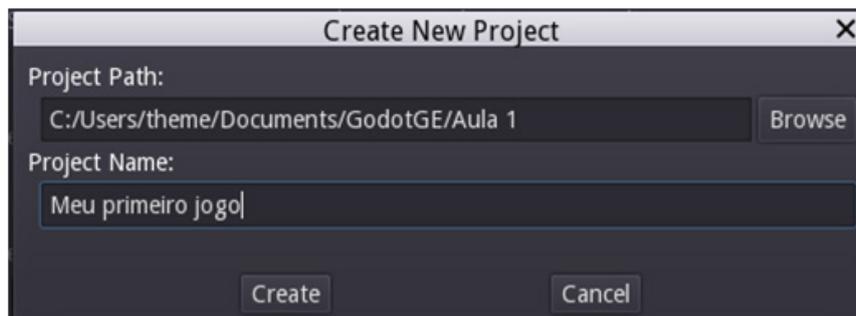


Figura 48. Janela de criação de um novo projeto

Fonte: Próprio autor.

5.8.4 Editor

Agora que criamos um novo projeto, ao abri-lo, o Godot apresenta seu editor, local em que criamos nossas cenas e lógica de todo um jogo. A interface que nos é apresentada é complexa quando não a entendemos, mas, na verdade, é muito bem estruturada.

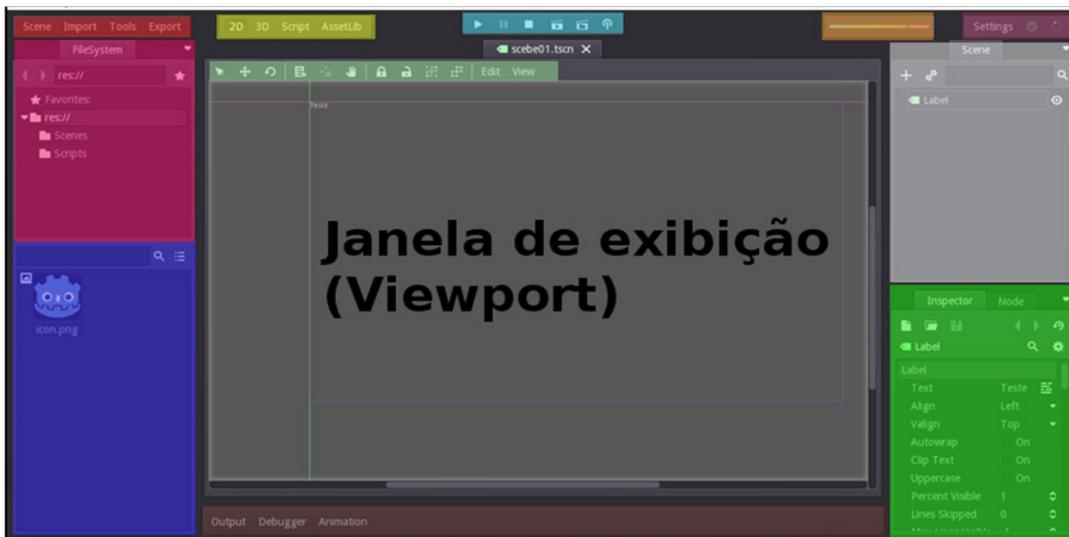


Figura 49. Editor do Godot

Fonte: Próprio autor.

A Figura 49 apresenta o editor do Godot, nela é possível perceber que ao redor de nossa **viewport** (janela de exibição), temos diversos painéis e menus, cada um com sua função. A **barra de menu principal** (em vermelho) tem operações de gerenciamento do projeto, dos menus e de suas funções. Temos portanto:

- **Scene** (Cena): operações para criar, abrir, salvar e fechar uma cena, para configurar o projeto, para voltar ao “Project list” da aplicação, entre outras funcio-

nalidades;

- **Import** (Importar): caminho que permite importar recursos para seu projeto, como imagens, objetos 3D, áudios, etc.;
- **Tools** (Ferramentas): ferramentas que estendem o Godot Editor, podendo ser instaladas por meio de pacotes e de extensões;
- **Export** (Exportar): janela para exportarmos nossos jogos para diversas plataformas, como mobiles (Android, IOS), desktop (Windows, Linux, Mac) e mesmo para web (HTML5), cada plataforma pode ser configurada.

O **menu de modos de edição** (em amarelo) alterna entre composição de cena 2D (opção 2D), composição de cena 3D (opção 3D), edição de script (Script) e biblioteca de recursos (AssetLib) em que podemos baixar ferramentas e funcionalidades para nosso projeto, utilizando o que outros desenvolvedores criaram para obter produtividade.

Ao centro da janela do editor, temos o **menu de execução** (em azul claro), no qual temos os seguintes botões: “Play” (Jogar/Rodar, atalho: F5), “Pause” (Pausar, atalho: F7), “Stop” (Parar, atalho: F8), “Play parent” (Jogar cena atual, atalho: F6), “Play custom” (Jogar cena customizada, selecionada em uma lista, atalho: Shift+CRTL+F5) e “Debug Options” (exibe um menu com caixas de seleção com opções para a caixa de saída de erros).

A **barra de configuração do editor** (em roxo) apresenta uma opção e dois ícones, o ícone verde (estado atual na Figura 49) é um alerta sobre modificações em recursos externos, caracteriza-se por girar toda vez que você move o mouse ou faz algo. Trata-se do nosso “Spinner”, que desempenha essas tarefas para indicar uma renderização do motor, ou seja, para indicar cada ciclo. A opção “Setting” (Configuração) apresenta as sub-operações e funções, são elas:

- **Editor Setting** (Configurações do editor): janela em que podemos fazer diversas mudanças nas configurações no editor, desde cores das janelas à atalhos de teclado. Você pode modificar todo o editor por aqui;
- **Editor Layout** (Layout do editor): caminho que nos permite trocar abas de lugar, alterar tamanhos de painéis, entre outras características, caso desejemos manter esses novos padrões de distribuição, podemos salvá-los e abri-los nesta opção;
- **Install Export Templates** (Instalar templates de exportação): acesso propicia selecionar o pacote “Export Templates” (baixe no site oficial do Godot) e instalá-lo para obter definições que viabiliza exportar seus jogos para múltiplas plataformas;
- **About** (Sobre): opção que oferece informações sobre a ferramenta e seus autores.

O painel “Filesystem” (Sistema de arquivos, em rosa) permite a você gerenciar os

recursos digitais e seus diretórios a partir do diretório raiz do projeto. Ao clicar em qualquer pasta, os elementos internos são apresentados abaixo (em azul escuro), como uma lista. Ademais, existe um campo de busca para ajudar a filtrar essa lista, facilitando encontrar itens específicos em um diretório.

Quando selecionamos qualquer item na cena, as opções ou configurações disponíveis irão aparecer no “*Inspect*” (Inspetor, em verde). Além disso, podemos configurar ações de Script na aba “*Node*” (ao lado), lá temos os diversos “*Signals*” (Sinais) que são eventos que podemos utilizar no Script.

O painel “*Scene*” (cena, em cinza) ou **editor de árvore de cena** apresenta a hierarquia de nós da cena, podemos visualmente entender como está a relação destes por meio dessa área, nela podemos criar, modificar e ordenar nossos nós.

Por fim, temos a **barra de monitoramento** (em marrom), na qual temos o “*Output*” (Console de saída), meio que o Godot expõe mensagens internas, o “*Debugger*” (Console de depuração de erros), que nos ajuda a identificar erros para providenciarmos soluções, e o “*Animation*” (Console de animação), no qual podemos criar e manipular animações no Godot.

5.8.5 Criando nossa primeira cena

A sessão anterior concentrou-se em apresentar toda interface do editor do Godot, agora iremos iniciar a composição de cenas, manipulando e criando nós. No painel “*Scene*”, clique no botão “*Add/Create a New Node*” (Criar/Adicionar um novo nó, atalho: CTRL+A), conforme especificado na Figura 50.

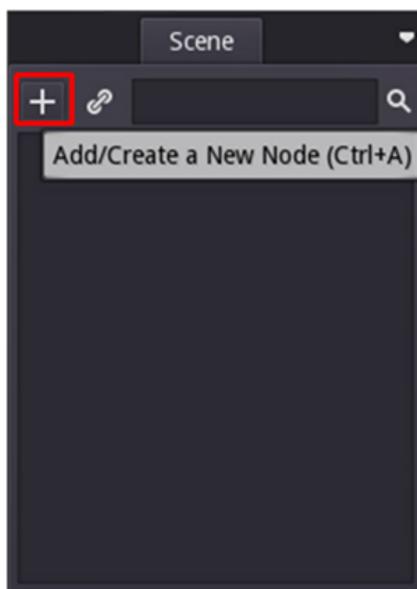


Figura 50. Painel Scene

Fonte: Próprio autor.

Ao clicar no botão, uma nova janela será apresentada com todos os nós do editor que estão disponíveis, por meio do campo “*Search*” (Buscar) você pode filtrar os inúmeros nós, insira neste a palavra “*label*” (Rótulo) para que o nó que precisamos seja encontrado, conforme apresentado na Figura 51. Selecione esse nó e pressione o botão “*Create*” (criar). Ademais, é possível verificar que cada nó tem uma descrição, apresentada no campo “*Description*” (descrição).

O “*label*” é um nó 2D, logo, o nosso modo de edição deve estar selecionado para 2D, tal informação pode ser vista no canto superior esquerdo, com pontos ao redor para você escaloná-lo (alterar seu tamanho), é possível também mover o nó. Perceba que tal elemento foi adicionado em nosso painel “*Scene*” e que, ao selecioná-lo, o painel “*Inspect*” é alimentado com diversas configurações e opções para possibilitar a modificação desse nó.

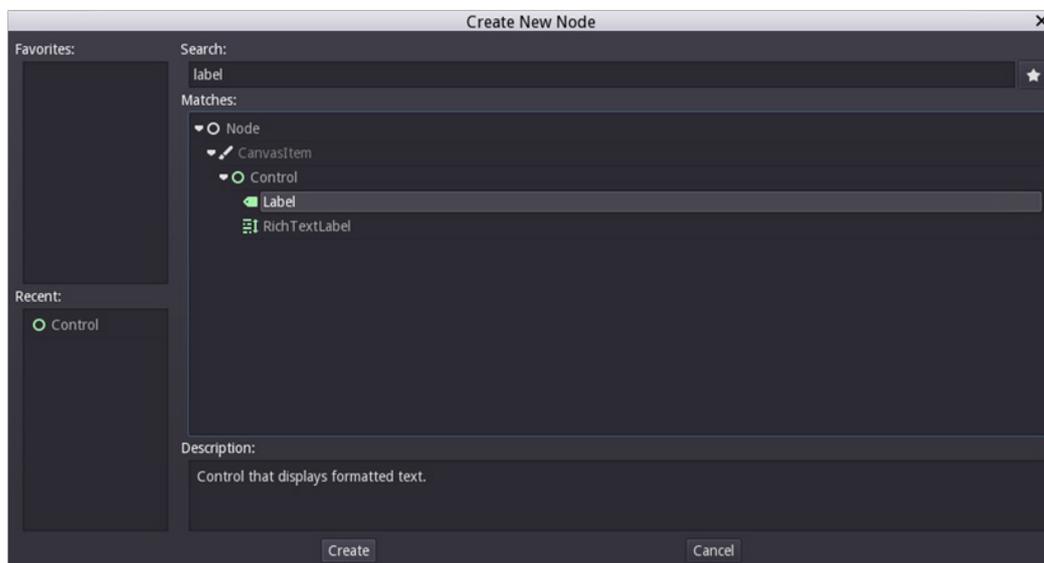


Figura 51. Painel de criação de nós

Fonte: Próprio autor.

Pelo “*Inspect*” podemos fazer modificações em nosso “*label*”. Ao realizar o teste, vamos inserir uma mensagem na tela para o usuário, para isso, no campo “*Text*”, insira a expressão “Olá mundo Godot!!!” (vide Figura 52). Para testar a cena você pode executar a atual, pressionando F6.

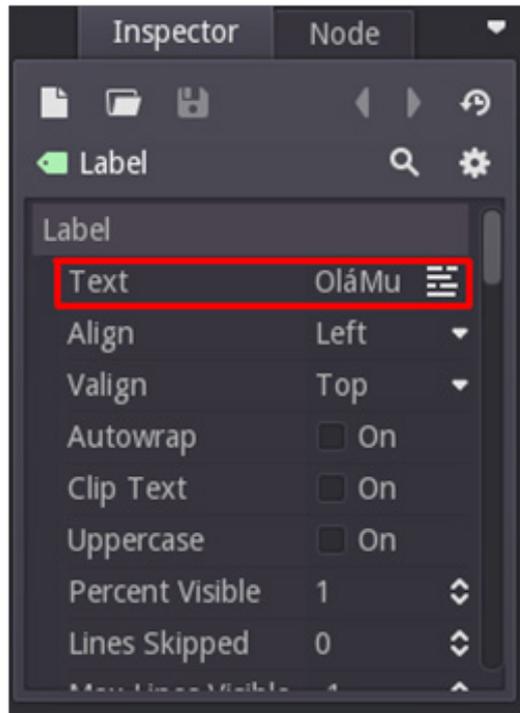


Figura 52. Campo “Text” do Label

Fonte: Próprio autor.

Ao salvar uma cena (atalho: CTRL+S), geralmente o Godot irá lhe direcionar para a pasta raiz do seu projeto, esta é a base de nossos recursos, que o editor chama de “res://” (res como abreviação de *resources*, em português, recursos). Para gerenciar seu projeto, crie na raiz uma pasta chamada “Cenas”, local em que gravaremos todas as cenas. Ao salvar a cena aqui detalhada, nomeie-a de “Label.tscn”, sobre isso cabe destacar que nossa extensão “tscn” refere-se a cenas no Godot (vide Figura 53).

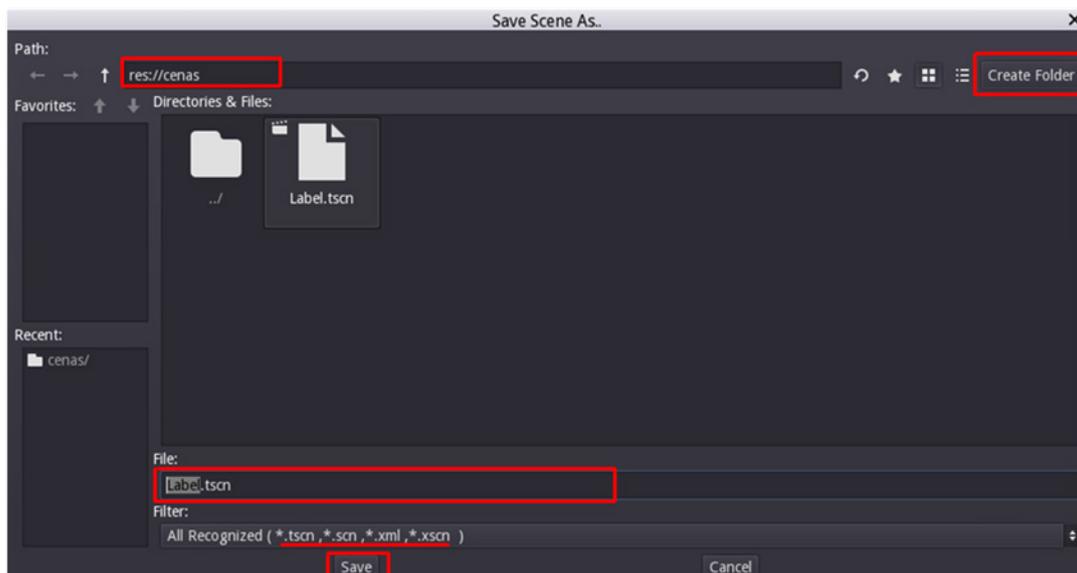


Figura 53. Janela Save Scene As ...

Fonte: Próprio autor.

5.8.6 Configurando um projeto

Um jogo é composto por diversas cenas e, quando o executamos, é inicializado por uma cena principal, a primeira que irá aparecer ao jogador. No exemplo anterior executamos a cena atual, mas precisamos definir a cena principal para organizar o fluxo de execução do jogo, bem como definir para o usuário o nome do game e a *splashscreen* (pode ser junto ou separado)²⁹ de sua empresa, entre outras configurações.

No Godot as configurações são salvas no diretório principal do seu projeto no arquivo “*engine.cfg*”. Para gerenciar suas configurações acesse na **barra de menu principal** o menu **Scene > Project Settings**. Na aba “*General*” (Geral) dentro da “*Section*” (Sessão) “*Application*” (Aplicação) selecione a opção **main_scene** (Cena principal) onde iremos definir a nossa cena principal clicando no ícone da pasta, no menu suspenso escolha a opção “*File...*” (Arquivo...), conforme Figura 54.

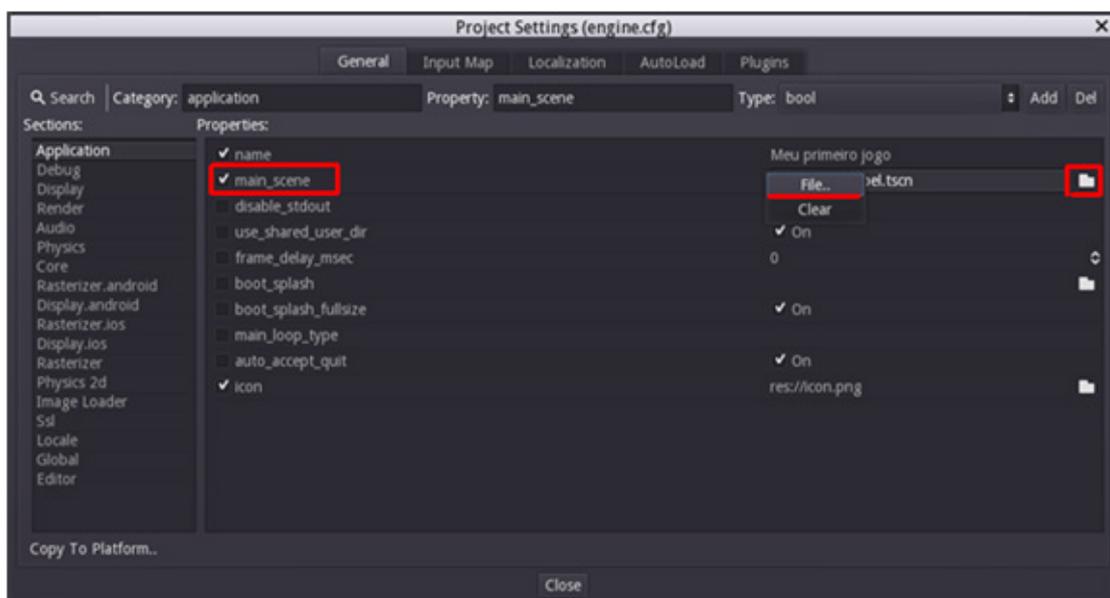


Figura 54. Janela de configurações do projeto

Fonte: Próprio autor.

Agora, ao pressionar a tecla F5, seu jogo irá iniciar a partir da cena configurada, pois nesse modo de execução, o jogo sempre irá rodar do começo e não da cena que está sendo editada, após a inicialização seguirá o fluxo do jogo. Você poderá configurar diversos módulos e operações do motor Godot a partir desta tela de configurações, por isso, vale a pena estudá-la com calma.

5.8.7 Instâncias

Uma cena é constituída de vários nós, agora imagine várias cenas. Para um jogo pequeno, a existência de muitas cenas pode não afetar muito a sua produtividade, mas em games maiores, com cenas mais complexas, inserir vários nós e configurá-los pode ser trabalhoso. O Godot foi desenvolvido para te ajudar a desenvolver seus jogos com reuso de recursos que você produziu e com compartilhamento de objetos de jogo.

Para essa tarefa, ele usa o conceito de “Instâncias”, no qual podemos reutilizar uma hierarquia de nós de outras cenas. Logo, o editor permite-nos separar nosso projeto em várias cenas individuais.

Como aprendemos, uma cena é constituída de uma hierarquia, também conhecida como árvore de nós, em que temos um único nó raiz, conforme apresentado anteriormente na Figura 44. No Godot, para criar cenas mais complexas, podemos subdividir nossa cena em diversas outras, quantas desejarmos, e depois instanciar quantas cenas desejarmos dentro de uma nova cena, conforme representado na Figura 55.

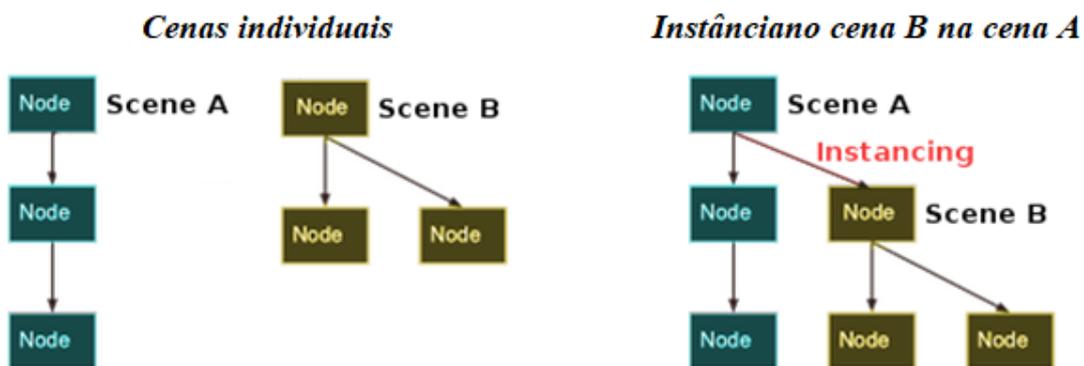


Figura 55. Instâncias e cenas

Fonte: Godot Game Engine (2017).

Adaptado pelo autor.

Para praticar, crie um novo projeto, neste vamos desenvolver duas cenas: a primeira contendo uma bola e a outra uma arena. Primeiro carregue para dentro da pasta do seu projeto recém-criado duas imagens (recurso do livro, vide nota), depois, em uma nova cena, insira o nó “*RigidBody2D*” (nosso corpo físico) e o nomeie de “bola”, dentro deste insira os nós “*Sprite*” (Pequena imagem, fração ou textura) e “*CollisionShape2D*” (Forma de colisão), este último nomeie de “Colisao” (evite acentos, pois o programa tem como idioma o inglês, fato que pode causar a incompreensão das notações em português, gerando erro).

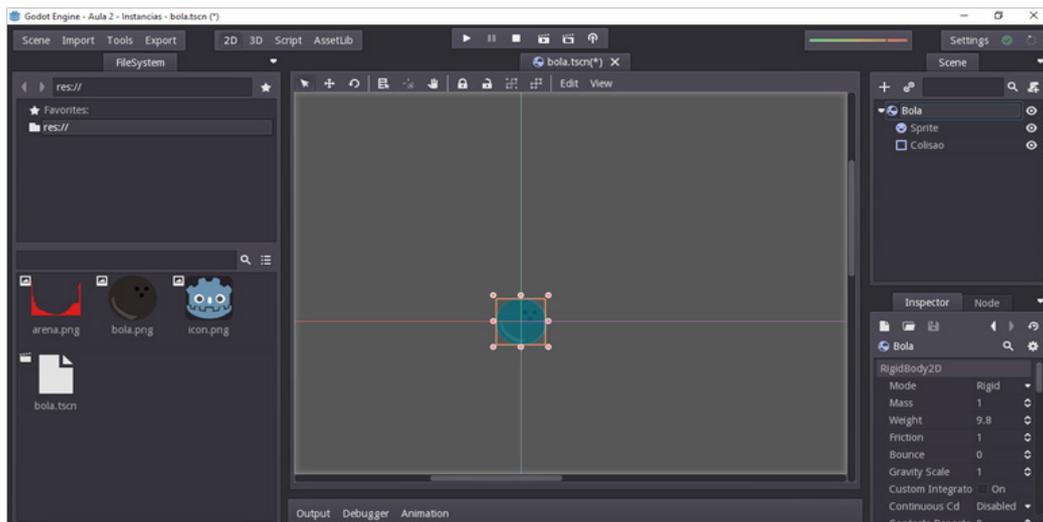


Figura 56. Configuração da cena bola

Fonte: Próprio autor.

A Figura 56 apresenta a configuração da estrutura de nós para a cena. Salve-a com o nome “*bola.tscn*” dentro do diretório raiz. Posteriormente selecione o nó “*Sprite*” e no campo “*Texture*” do “*Inspect*”, ao clicar aparecerá um menu de operações, acesse a opção “*Load*” (carregar) e selecione a imagem de nossa bola. Configure o nó “*Colisao*” e, no campo “*Shape*” (Forma), selecione a opção “*new CircleShape2D*” (esfera azul em cima da bola) e ajuste-a para nossa bola na **viewport** através dos pontos que a circundam. Essa área em que o nó “*Colisao*” atua sobre nossa *Sprite* é sua área de contato (sólida neste caso). Após seguir os passos aqui indicados, salve as modificações da cena.

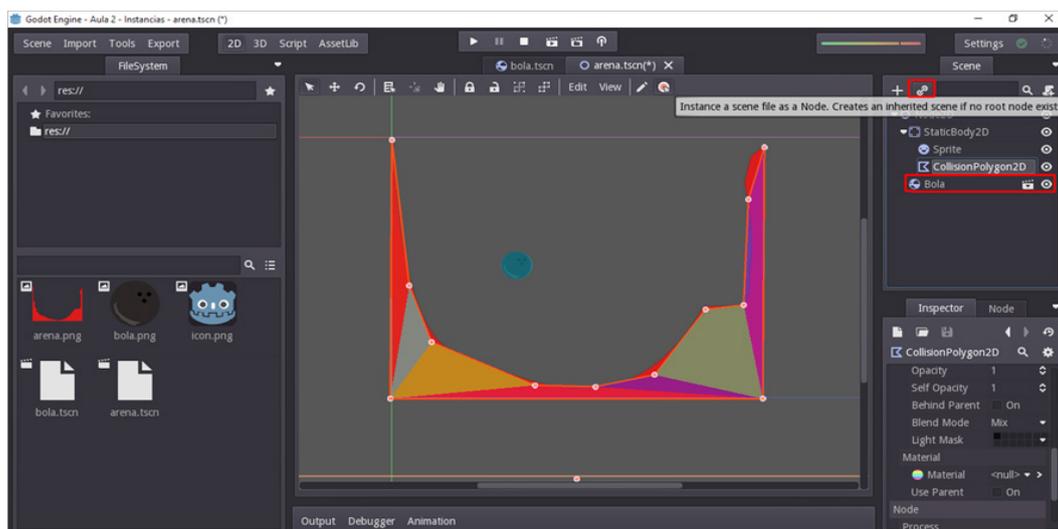


Figura 57. Configuração da cena arena

Fonte: Próprio autor.

Inicie uma nova cena e insira nela o nó raiz “*Node2D*”, neste insira o nó “*StaticBody2D*” (Corpo estático 2D), que terá mais dois nós internos, um “*Sprite*” e um “*Colli-*

sionPolygon2D” (Colisão poligonal 2D). Monte sua cena conforme a estrutura apresentada na Figura 57. Para inserir nossa bola na cena, clique no **botão âncora** no painel “Scene” e selecione nossa cena “bola.tscn” a partir do diretório raiz na janela que irá ser apresentada. Depois de aplicar essas orientações, salve a sua cena com o nome “arena.tscn”.

Para configurar a cena, selecione o nó “Sprite” e no campo “Texture”, apresentado no “Inspect”, defina a imagem para nossa arena. O nó “CollisionPolygon2D” é diferente do que usamos para colisão de nossa bola, pois nele temos que fazer o **vetor de colisão** para nossa arena, para isso, selecione-o e na propriedade “Polygon”, apresentada no “Inspect”, mude o tamanho do “Array” para 12, obtendo-se assim esse número de pontos para criar o polígono. Crie o **polígono** orientando-se pela própria imagem da arena, conforme representado na Figura 57. Salve as modificações da cena.

Ao executar a cena, você perceberá que a bola irá cair, sendo parada pela arena. Agora, selecione a bola e a duplique (atalho: CTRL+D) diversas vezes. Apesar de idênticas, cada instância pode ter propriedades diferentes, para entendermos isso, selecione algumas bolas que você duplicou e altere a propriedade “Bounce” (pulo/elasticidade) para compartilharem a mesma característica no “Inspect” e veja que elas irão pular de maneira diferente.

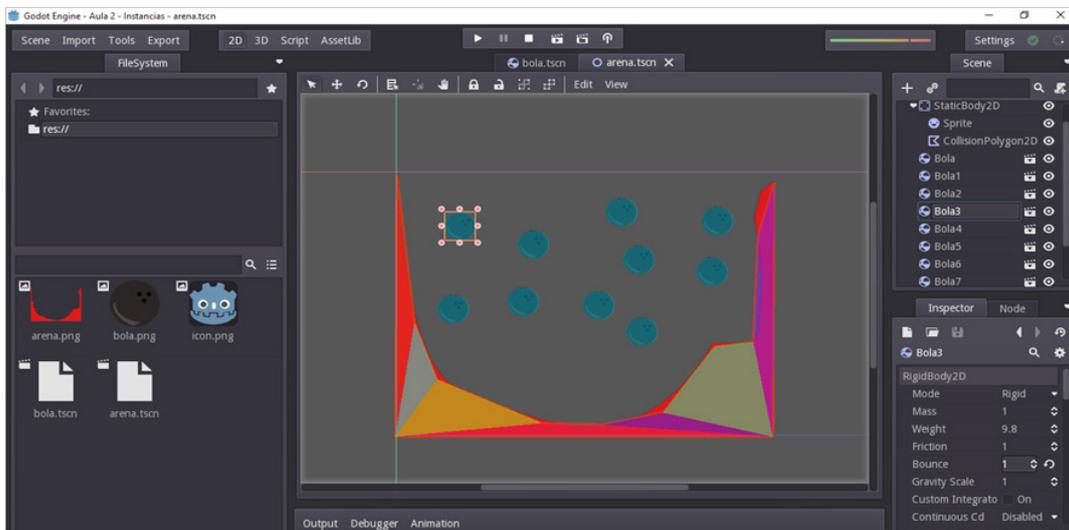


Figura 58. Configuração da cena arena com instâncias de bola

Fonte: Próprio autor.

Você deve ter percebido a grande possibilidade que temos com a dinâmica do recurso de instância, com esse mecanismo podemos criar cenas de forma produtiva. Com as instâncias temos:

- A capacidade de subdividir cenas e torná-las mais fáceis de gerenciar;
- Uma alternativa mais flexível aos pré-fabricados (e muitas instâncias mais poderosas funcionam em vários níveis), comumente chamados pelos desenvolve-

- Uma maneira de projetar fluxos de jogos mais complexos ou mesmo UIs (os elementos de UI também são nós no Godot).

5.8.8 Linguagem de Design

Os desenvolvedores do Godot criaram com o conceito de instâncias uma *linguagem de design* eficiente, a forma de trabalhar no programa é bem diferente de similares existentes no mercado. O motor foi projetado em cima desse conceito por inteiro. Assim, ao fazer jogos com Godot, os desenvolvedores da ferramenta recomendam a seus utilizadores a deixarem de lado outros padrões de design, como os diagramas MVC ou a Entidade-Relacionamento.

Segundo os desenvolvedores (GODOT GAME ENGINE, 2017), devemos começar a pensar nos jogos de forma mais natural, imaginando os elementos de um jogo de forma visual, ou seja, devemos enxergá-los como sendo aqueles que podem ser nomeados não apenas por um programador, mas por qualquer um. O exemplo apresentado na Figura 59 é como um jogo de tiro simples pode ser imaginado:

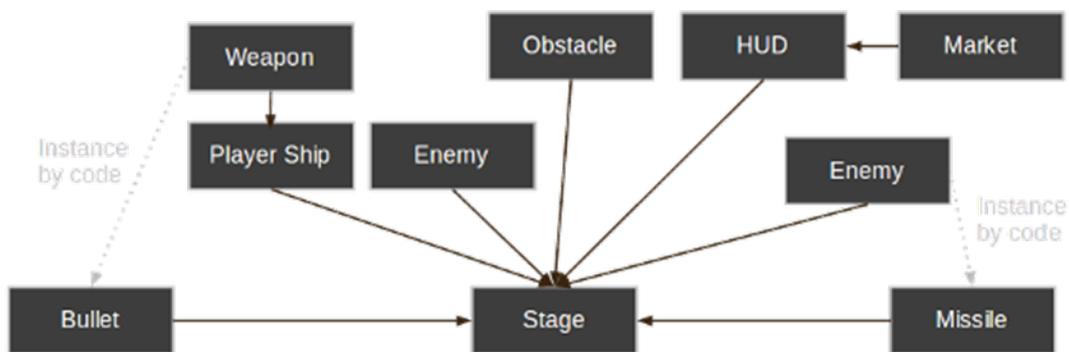


Figura 59. Esquema de um jogo de tiro sob ótica da linguagem de design do Godot

Fonte: Godot Game Engine (2017).

Você pode criar seus jogos imaginando os elementos que os compõem, estabelecendo relações como o diagrama acima, pode inclusive desenvolver esse tipo de representação para seu jogo se necessitar. Uma vez que temos esse diagrama, você pode criar cada nó correspondente a cada elemento existente e realizar suas instâncias montando suas cenas.

Logo, o Godot tem um design baseado em cenas/instâncias, sendo extremamente eficiente, pois os elementos projetam um mapa direto de sua cena. Dessa forma você não precisa preocupar-se com a programação arquitetônica de seu projeto, visto que o motor já o faz para o desenvolvedor. O exemplo a seguir na Figura 60 é mais complexo do que o anterior, trata-se de um jogo com mundo aberto repleto de recursos.

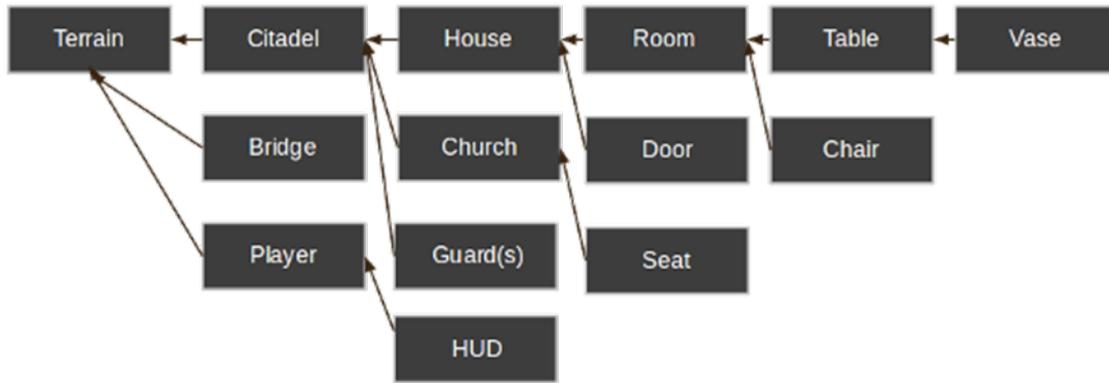


Figura 60. Diagrama de jogo complexo sob a ótica de linguagem de design do Godot

Fonte: Godot Game Engine (2017).

Neste ponto, faça as atividades na seção 6.13 Exercícios deste capítulo relativo ao Godot Editor. Esses exercícios são importantes para seu aprofundamento e aprendizagem no desenvolvimento de jogos e na flexibilidade e produtividade com a ferramenta.

5.8.9 Aprendendo programação com GDScript

GDScript (Godot Script) é uma linguagem de *script* dinamicamente *tipada* (faz controle dos tipos de dados) para caber dentro de Godot. Foi projetada com os seguintes objetivos:

- Tornar a programação simples, familiar e com uma baixa curva de aprendizagem, ou seja, fácil de aprender;
- Possibilitar a composição de um código legível e seguro de erros. A sintaxe idealizada fez um grande empréstimo da existente no Python.

Vale ressaltar que as funções principais foram desenvolvidas dentro do motor na linguagem C++. Dessa forma, para desenvolver as seções mais críticas do Godot, você pode substituí-las ou reescrevê-las em C++. O interessante dessa possibilidade é que podemos substituir uma classe GDScript por uma classe C++ sem alterar seu jogo.

Nota: Para obter mais detalhes acerca de programação no Godot e sobre como funciona o GDScript, estude sua referência, ela é simples e pequena. Isso vai ajudá-lo a progredir neste livro e em seu desenvolvimento, disponível em: http://docs.godotengine.org/en/stable/learning/scripting/gdscript/gdscript_basics.html.

5.8.10 Programando nossa primeira cena

Crie um novo projeto para nosso estudo, será um protótipo 2D (ative o modo 2D) no qual trabalharemos o GDScript com um sistema de alerta. Para sua maior evolução, estude a referência da linguagem pela documentação oficial do Godot Editor, conforme endereços de internet indicados anteriormente. Nesta seção desenvolveremos a linguagem de maneira prática.

Crie em sua cena um nó “Panel” e dentro deste insira um “Label” e um “Button”, modificando em ambos, no “Inspect”, a propriedade “Text” e organizando e escalonando os elementos em sua janela, como apresentado na Figura 61. Salve a cena com o nome “meupainel.tscn” na raiz do seu projeto. Execute a cena com F6.

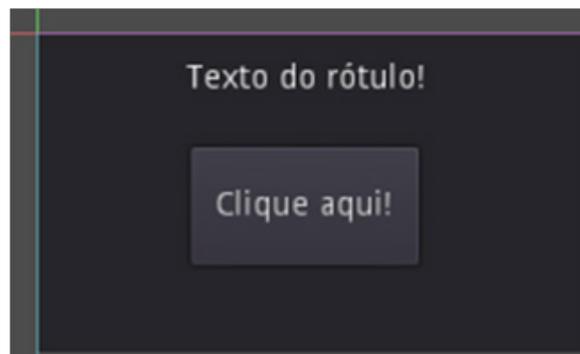


Figura 61. Painel com rótulo e botão

Fonte: Próprio autor.

Ao clicar no botão na cena em execução, nada acontece, mas o evento de clique é percebido. Agora iremos programar o evento. No painel “Scene” clique com o botão direito do mouse no nó “Panel” e selecione a opção “Attach Script” (Anexar Script), conforme apresentado na Figura 62.



Figura 62. Menu de anexação de script a um nó

Fonte: Próprio autor.

Quando a caixa de configuração de script aparecer, você deverá configurá-lo, como apresentado na Figura 63. Cabe elucidar que uma “*Inherits*” (herança) é a classe pela qual a sua cena será criada, ou seja, a herança para a nova classe, automaticamente o campo “*Inherits*” será preenchido com “*Panel*”, pois estamos criando um script a partir do nó *Panel*, assim, mantenha esse dado.

A existência da possibilidade de trocar a linguagem (em *Language*), como comentado anteriormente, atesta o intuito dos desenvolvedores do programa de, no futuro, permitir diversas linguagens para os usuários, mas atualmente na sua versão temos apenas a opção *GDScript*. A entrada “*Path*” (Caminho) define o local em que salvaremos nosso script bem como o nome para ele, nomeie-o como “*acao.gd*”. Para melhor organizar-se dentro do seu diretório raiz, podemos criar uma pasta para inserir os scripts criados.

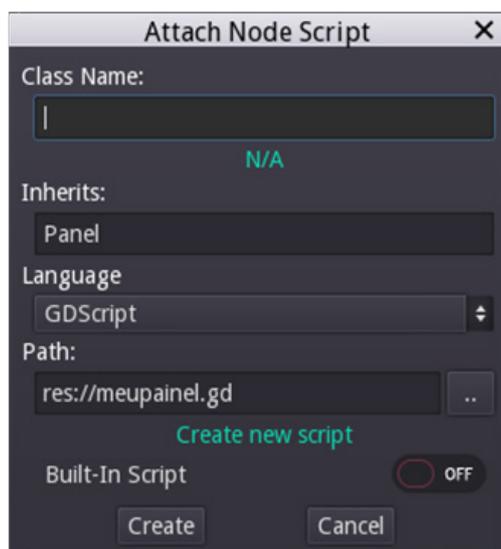


Figura 63. Janela de configuração de Script

Fonte: Próprio autor.

Ao criar o *script*, ele será salvo e poderá ser consultado no **modo script**, no qual poderemos também editá-los, como apresentado na Figura 64. Os scripts têm diversas funções especiais, a função “*_ready()*” gerada, por exemplo, é chamada toda vez que um nó é adicionado em uma cena.

Quando executamos um game, os nós são criados e adicionados na cena, construindo-a. O processo de alocação desses recursos pode ser modificado por meio de funções que mapeiam esses tipos de evento, a exemplo, como mencionado, temos a função “*_ready()*” e existem diversas outras como ela, iremos conhecer algumas à medida que avançarmos em nossos projetos.

Vale ressaltar que a função “*_ready()*” não é a construtora da classe, esta é a função “*_init()*”. Uma função construtora é responsável por gerar um objeto.

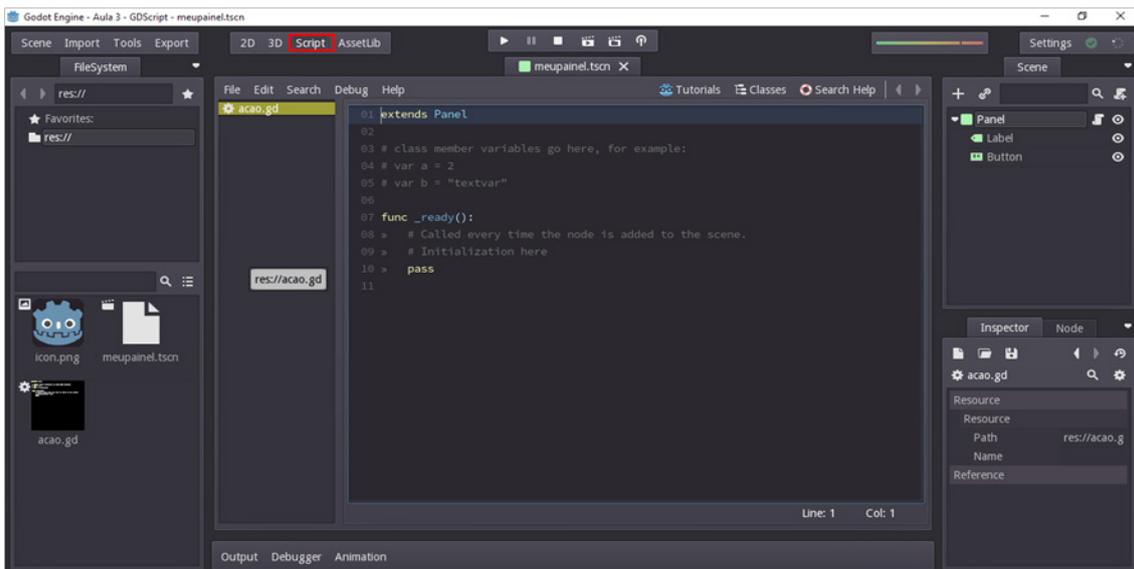


Figura 64. Modo de edição de Scripts

Fonte: Próprio autor.

Um script incorpora comportamentos em um nó, por isso dizemos que anexamos um script ao nó para podermos controlar funções, eventos, bem como os nós filhos. Estudaremos a seguir o conceito de “*Signals*” (Sinais) que completam a forma de programar no Godot.

5.8.11 Manipulação de um sinal

Os *Signals* (sinais) são muito utilizados em nós relativos à GUI’s (Graphical User Interface, em português, Interface Gráfica do Usuário), além de serem “*emitidos*” quando certos eventos específicos ocorrem, podendo ser conectados a qualquer função que criarmos de qualquer instância de *script*. Selecionando um nó em nossa árvore de cena podemos acessar os *Signals* através do painel “*Node*” que é carregado, conforme apresentado Figura 65.

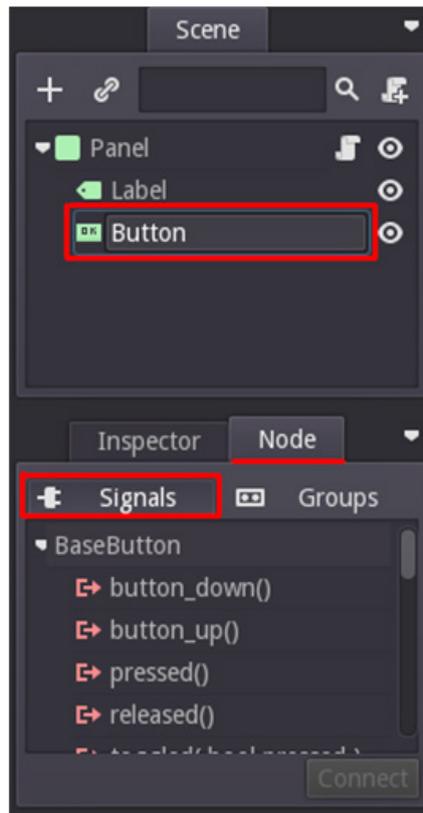


Figura 65. Painel de conexão de Signals

Fonte: Próprio autor.

Agora voltemos ao nosso modo script para trabalhar na codificação do arquivo “acao.gd”. O que faremos é realizar uma ação quando pressionamos o botão em cena. Até o momento você aprendeu que podemos colocar nossos nós em uma hierarquia, para controle e gerenciamento de cena, o que temos que entender agora é que por meio dos scripts podemos não só implementar funções para esses nós, mas, manipulando também a sua árvore, podemos ter acesso aos nós filhos a partir de um nó pai, por isso colocamos nosso Label e nosso Button em um Panel que está associado ao script.

Uma das funções muito utilizadas por desenvolvedores Godot é a “Node.Get_node()” que permite achar nos nós filhos um nó em específico. Utilizamos a função da seguinte forma nos scripts:

```
get_node("Button")
```

Com isso, o Script ficará da seguinte forma (respeite a endentação):

```
#Código para o painel do arquivo 'acao.gd'
extends Panel

func _on_button_pressed():
    get_node("Label").set_text("Clicado!")
func _ready():
    get_node("Button").connect("pressed",self,"_on_button_pressed")
```

Perceba, no trecho de código acima, que o que temos na frente do símbolo ‘#’ (Hashtag) é comentário, textos de anotação do programador, o que é ignorado pelo Godot, opcional e útil para anotarmos informações úteis a nós e a equipe de desenvolvimento. Toda função inicia com “func”, indicação que precede o nome da função (em cor vinho), e é seguida de parâmetros (declarados entre parênteses).

Na função “_on_button_pressed”, capturamos o evento de clique do botão, ou seja, quando o botão é selecionado, caso ocorra, inserimos o texto “Clicado!” no *Label* da cena. Mas, antes disso, como explicado, temos que conectar o *Signal*, nesse caso específico o “pressed” (identifique-o na Figura 65) é a função especificada.

Para isso utilizamos a função “_ready”, que, ao adicionar nosso Painel em cena, busca nosso nó *Button* e conecta nosso *Signal* “pressed” à função “_on_button_pressed”. Dessa forma, quando o botão for pressionado, a função “_on_button_pressed” é ativada e suas instruções executadas. Agora, salve suas modificações e teste a cena.

5.8.12 Processamento

Muitas ações no Godot são ativadas por “callbacks” (Retornos de chamadas) ou funções virtuais. Existem dois tipos de processamento na ferramenta, processamento ocioso e processamento fixo.

O processamento ocioso é ativado com a função “Node.set_process()”. Uma vez ativo, o retorno de chamada de “Node._process()” será chamado a cada quadro (ou frame). Exemplo de uso e implementação:

```
#Código para o painel do arquivo 'processo.gd'
extends Label

func _ready():
    set_process(true) # ativamos o processamento ocioso
func _process(delta):
    # instruções para o processo ocioso, ativo a cada frame
```

O parâmetro da função “_process()” delta refere-se ao tempo decorrido (em segundos, como valor real, ou seja, com casas decimais) desde a chamada anterior para a função “_process()”. O processamento fixo é semelhante, mas só é necessário para a sincronização com o mecanismo de física.

Para que possamos testar isso, voltando a nossa cena, selecione nosso *Label* e crie um novo script, chamado “processo.gd” com o seguinte código para manipular o nó:

```

#Código para o painel do arquivo 'processo.gd'
extends Label

var contador = 0

func _ready():
    set_process(true) # ativamos o processamento ocioso
func _process(delta):
    contador += delta
    set_text(str(contador))

```

Salve suas modificações e teste a sua cena. Claramente entrará em conflito com a ação do Button que fizemos antes, mas, na verdade, a ação de pressionar o botão continua funcionando, o que acontece é que é tão rápido que a função “_process()” modifica o texto do *Label* antes de percebermos a troca.

5.8.13 Grupos

Nossos nós podem ser organizados em Groups (grupos), possibilitando maior organização de nossa cena, principalmente se esta for complexa ou grande. Podemos fazer isso via interface ou código. Por meio da interface, você pode selecionar o seu nó e, por meio do painel “Node”, pode acessar a opção “Groups”, conforme apresentado na Figura 66.

Um nó pode pertencer a vários grupos, a lista de grupos pode ser vista por meio do painel apresentado. Clicando no ícone da lixeira, retiramos o nó de um grupo e, para criar um novo, basta inserir um nome e clicar no botão “Add”.

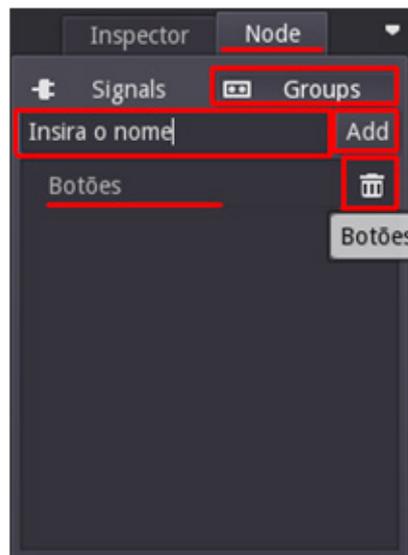


Figura 66. Painel Node com a opção Groups

Fonte: Próprio autor.

Podemos definir qualquer tipo de grupo de objetos em geral, como ‘inimigos’, ‘jogadores’ ou qualquer outro, podemos criá-los via *script* utilizando o seguinte trecho de código:

```
func _ready():
    add_to_group("jogadores")
```

Uma das vantagens em utilizar grupos é que podemos criar diversos tipos de operações, usando a função “SceneTree.call_group()”, você pode, por exemplo, avisar a todos os inimigos de uma fase onde o jogador encontra-se ou avisar ao jogador que inimigos aproximam-se, como em muitos jogos, exemplo:

```
func _inimigo_a_vista():
    get_tree().call_group(0, "soldado", "inimigo_esta_proximo")
```

A inserção “inimigo_esta_proximo” é uma função que será chamada em cada membro “soldado” existente na cena. Podemos ainda capturar a lista de membros de um grupo com a função “SceneTree.get_nodes_in_group()”, veja seu uso:

```
var soldados = get_tree().get_nodes_in_group("soldado")
```

5.8.14 Funções substituíveis

Nesta seção, iremos conhecer as principais funções substituíveis (*Overrideable*), ou seja, funções que redirecionam seu comportamento padrão. A seguir estão disponíveis cada uma delas e sua descrição:

```
func _enter_tree():
    # Quando o nó entra na _Scene Tree_, ele torna-se ativo e essa função é chamada.
    # Os nós filhos ainda não entraram na cena ativa.
    # Em geral, é melhor usar _ready() na maioria dos casos.
    pass

func _ready():
    # Esta função é chamada após _enter_tree, mas assegura
    # que todos os nós filho também tenham sido inseridos em _Scene Tree_,
    # e se tornaram ativos.
    pass

func _unpaused():func _exit_tree():
    # Quando o nó sai do _Scene Tree_, esta função é chamada.
    # Os nós filhos já abandonaram o _Scene Tree_ neste ponto e todos ficaram inativos.
    Pass

func _process(delta):
    # Quando set_process() está habilitado, esta função é chamada a cada quadro..
    pass
```

```

func _fixed_process(delta):
    # Quando set_fixed_process() está habilitado, essa função é chamada a cada quadro de
    # física.
    pass

func _paused():
    # Chamado quando o jogo está em pausa. Após essa chamada, o nó não receberá
    # mais retorno de retorno de processo (callbacks).
    pass

func _unpaused():
    # Chamado quando o jogo sai do modo pausado para “em operação”.
    pass

```

Utilizar essas funções e conhecê-las bem vai ajudá-lo a desenvolver seu projeto de maneira bem mais rápida e simples. Faça testes com essas funções para sua ambientação e aprendizagem.

5.8.15 Criando nós

Para criarmos um nó via script é bem simples, basta utilizarmos a função “*new()*” que as classes possuem. Abaixo, por exemplo, criamos um nó “*Sprite*” dentro da cena atual:

```

var s
func _ready():
    s = Sprite.new() # cria um novo nó Sprite!
    add_child(s) # adiciona o nó como filho do nó atual (nó o qual o Script
    está associado)

```

Para remover um nó, temos a função “*free()*”, que a libera da memória:

```

func _someaction():
    s.free() # a função remove o nó imediatamente da cena e o libera da memória

```

Tenha cuidado ao liberar um nó, pois seus nós filhos também serão removidos. Por muitas vezes, tentar liberar um nó em uso ou mesmo bloqueado pode causar erros. Pensando nisso, é possível lançar mão de uma função com o mesmo intuito de exclusão, mas segura. Trata-se da função “*Node.queue_free()*”:

```

func _someaction():
    s.queue_free() # Remove o nó e o libera quando nada está correndo com o mesmo

```

5.8.16 Instanciando cenas

Podemos instanciar cenas facilmente com nosso script. Para abrir uma cena diretamente do seu computador, utilizamos:

```
var cena = load("res://cena.scn") # carregamos a cena quando o script é carregado
```

Por sua vez, a função “*preload*”, contrária a “*load*”, faz a mesma coisa, porém carrega nossa cena ainda em fase de análise, ou seja, antecipa ou mesmo prepara o carregamento da cena. Como exemplo temos:

```
var cena = preload("res://cena.scn") # será carregado ao analisar o script
```

A variável ‘cena’ ainda não é uma hierarquia de nós. Para conseguirmos trabalhar com os nós existentes nas instâncias carregadas, utilizamos um recurso especial o “*PackedScene*” que nos permite utilizar a função “*instance()*”, a exemplo, poderíamos pegar os nós da “cena” (variável) com o seguinte trecho de código:

```
var no = cena.instance()  
add_child(no)
```

6. Desenvolvendo o game 2D: ‘Astra’ - O herói do espaço

Agora que já aprendemos os principais comandos do Godot, você está preparado para desenvolver um game. Nesta seção, iremos trabalhar em um game de nave, os recursos visuais estão disponíveis no AVA (vide esclarecimento no Prefácio) criado para este livro, faça o download do recurso “Assets do game Astra” (Formato “zip”).

O conteúdo gráfico utilizado é de domínio público, disponibilizado no site OpenGameArt³⁰ e criado pela Kenny Group³¹. Como apresentado na Figura 67, os itens antecipadamente já estão recortados e nomeados em na pasta “png”.

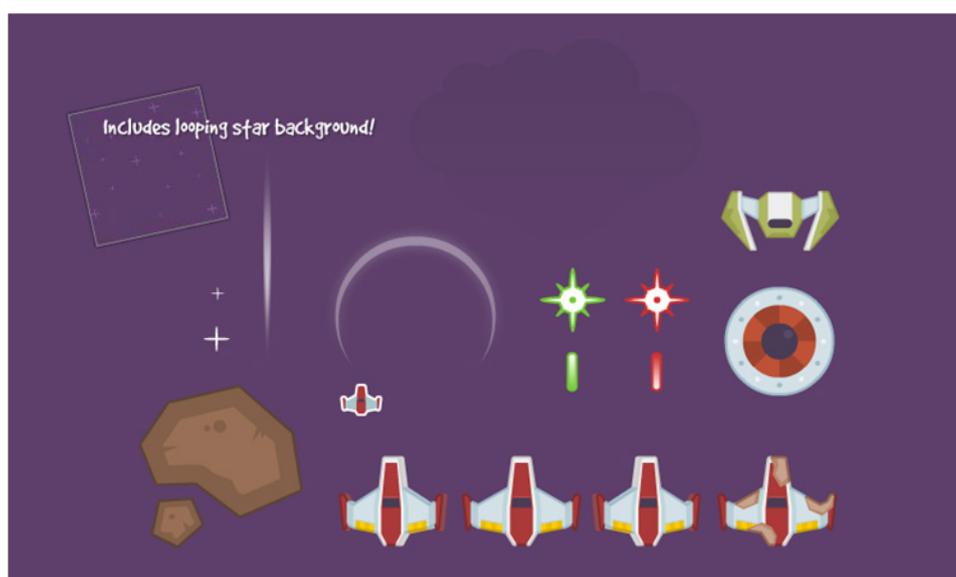


Figura 67. Sprites para o game Astra

Fonte: OpenGameArt (2012)

30 OpenGameArt – site com recursos audiovisuais gratuitos, compreende diversas licenças, disponível em: <<https://opengameart.org/>>.

31 Kenny Group – criam recursos para jogos, disponível em: <<http://www.kenney.nl/>>.

6.1 Configurando o projeto

Vamos portanto criar um novo projeto, nomeado “Astra - O herói do espaço”, antes de abri-lo, copie o diretório “png” para sua pasta raiz do projeto, crie também mais dois diretórios, um com o nome “Scripts”, no qual colocaremos nossos códigos, e outro com nome “Cenas”, neste colocaremos cada uma de nossas cenas. Com as modificações realizadas no seu sistema de arquivos, abra o projeto.

Na **barra de menu principal**, acesse **Scene > Project Settings**, aba **General** seção **Application**, configure o ícone do game na opção **icon** selecionando a imagem “life.png” no diretório “png” existente dentro do diretório raiz do projeto. Ainda na aba **General**, acesse a seção **Display** e configure as opções **width** (largura) para 800 e **height** (largura) para 600, assim, nossa tela será de 800 x 600 pixels. Nessa mesma seção, configure o **stretch_mode** para **2D**, dessa forma ele ajusta nossos pixels caso o usuário modifique o tamanho da tela ou tenha monitor com outra predefinição.

Coloque um nó vazio, o “Node” (classe básica), e altere o nome dele para “principal”. Salve a cena como o nome “principal.tscn”, em seguida, abra novamente o menu **Project Settings** e modifique na seção **Application** a opção **main_scene** apontando para essa cena. Dessa forma, quando o jogador iniciar a partida, é nessa cena que tudo irá começar. Salve suas modificações e execute o jogo com F5.

Agora, configuraremos os botões para jogar, serão os botões do seu teclado: seta direita, seta esquerda e espaço. Para isso, em **Project Settings**, acesse a aba **Input Map** e crie três ações (**Action**) com nome “direita”, “esquerda” e “atirar” (clique no botão **Add**). Para cada uma delas, insira um **input** adequado, em nosso caso serão do tipo “Key”, sendo as devidas setas (clique no ícone “+” para adicionar). Suas configurações devem ficar conforme a Figura 68.



Figura 68. Configurações de controles do Godot

Fonte: Próprio autor.

Perceba que o nome e o ícone para o game aparecem ao topo da janela, como configurado. Com estas configurações começaremos a criar nosso game.

6.2 Script de controle

Teremos um script para controlar a pontuação, os grupos existentes, as vidas e outras regras que nosso jogo pode ter, esse arquivo será global e acessível a todos nossos outros scripts. Vá para o **modo script** e crie um arquivo com o nome “*jogo.gd*” (sem anexar a nenhum nó) salvando-o no nosso diretório “*Scripts*”. Dentro desse arquivo insira, inicialmente, o código abaixo:

```
extends Node

const METEORO = "meteorito"; # nome constante do grupo para os meteoros
var vidas = 3 # iniciamos com 3 vidas
var pontos = 0 # iniciamos com nenhum ponto o game

func _ready():
    # Called every time the node is added to the scene.
    # Initialization here
    pass
```

Uma variável **const** (Constante) não varia, ou seja, seu valor é fixo e não muda durante todo o jogo. Criamos esse arquivo com valores constantes e com funções globais, recursos que serão disponibilizados a todas as outras classes do nosso jogo. Salve suas modificações no script.

Para tornar nosso script acessível a todos os membros do jogo, retorne ao **Project Settings**, abra a aba **Autoload** e, nessa janela, apresentada na Figura 69, navegue para onde nosso script está para selecioná-lo (*Path*), dê um nome a ele (*Node name*) e clique em “*Add*” para adicioná-lo como nó. A esses objetos inseridos no jogo, chamamos de *Singleton* (coisa única), que se refere a algo único, pois teremos apenas um objeto como este.

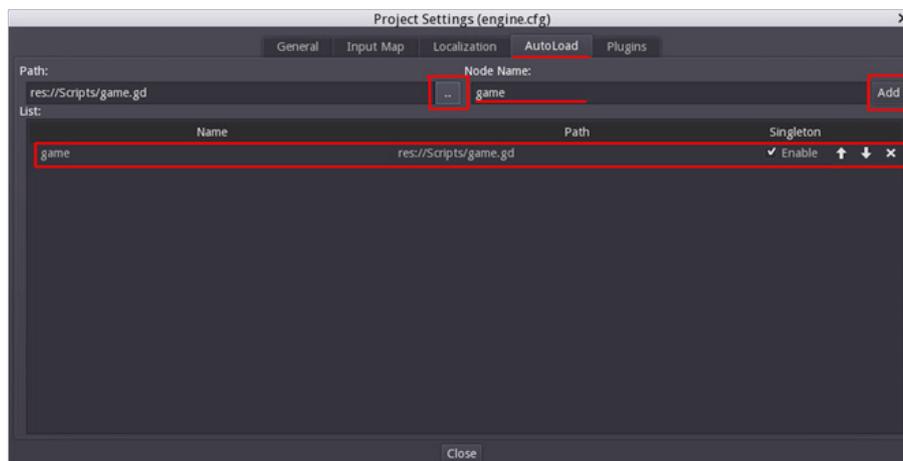


Figura 69. Criação de Singleton

Fonte: Próprio autor.

6.3 Criando nossa nave

Crie uma cena nova, insira um nó “Node2D” e altere seu nome para “nave”. Dentro desse nó, insira um do tipo “Sprite” e carregue, no campo “Texture” deste, a imagem “player.png” do diretório “png”. Salve a nova cena com o nome “nave.tscn” no nosso diretório “Cenas”, a estrutura e o viewport para ela deve estar de acordo com a Figura 70.

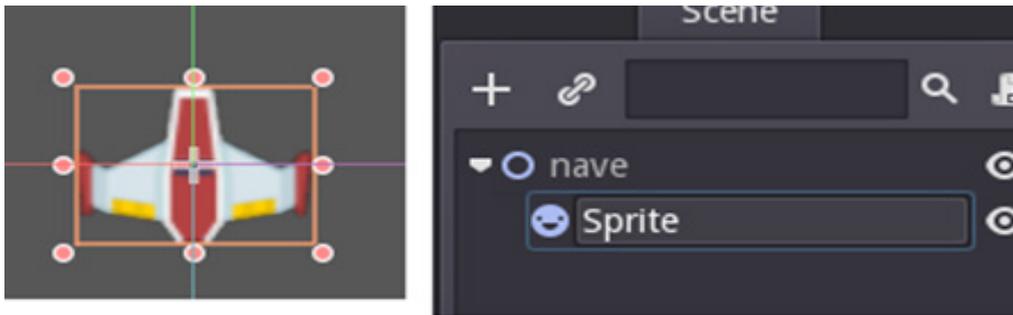


Figura 70. Estrutura da cena nave.tscn

Fonte: Próprio autor.

Retorne a sua cena “*principal.tscn*” e agora insira dentro do seu nó “*Node*” a instância da cena “*nave.tscn*” (como aprendemos anteriormente). Pode acontecer da imagem da nave não aparecer, caso ocorra, salve a cena e a reabra. Coloque nossa pequena nave ao centro sob a linha base inferior da cena, conforme a Figura 71. Salve sua cena e a teste (F5).

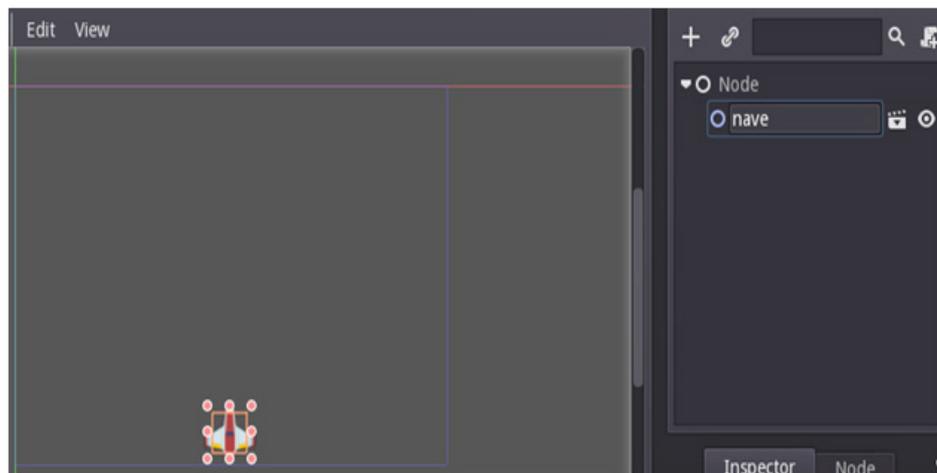


Figura 71. Inserindo nave por instância no centro da cena principal

Fonte: Próprio autor.

6.4 Programando nossa nave

```
extends Node2D

var velocidade = 300 # velocidade da nave em pixels/segundo

func _ready():
    set_process(true)
    pass

func _process(delta):
    # variáveis de verificação
    var d = 0 # 1 para pressionado para direita
    var e = 0 # -1 para pressionado para esquerda, inverte caçulo da posição

    # condição de verificação se a tecla configurada para direita foi ativada
    if Input.is_action_pressed("direita"):
        d = 1 # validamos que direita foi pressionado

    if Input.is_action_pressed("esquerda"):
        e = -1 # validamos que esquerda foi pressionado, -1 par inverter cálculo
    # Cálculo de posicionamento a uma velocidade constante a cada frame
    set_pos(get_pos() + Vector2(velocidade, 0) * delta * (d + e))
    pass
```

Na cena “*nave.tscn*”, selecione nosso nó “*nave*” e adicione um script, salvando-o com o nome “*nave.gd*” no diretório “*Scripts*”, que criamos anteriormente. Insira o código acima.

Muito do que está no código apresentado foi ensinado, o que podemos dar maior atenção é no “*Input.is_action_pressed(“direita”)*”, em que “*Input*” é uma classe que contém funções para tratarmos ações de controle do usuário. Perceba que a função “*is_action_pressed*” da classe recebe como atributo “*direita*”, que configuramos anteriormente.

Logo, para mover nossa nave, verificamos o botão pressionado dentro de um processamento ocioso ativo, que o tempo inteiro verifica se as condições para a nave se mover foram satisfeitas, alterando os valores das variáveis de controle “*d*” (direita) e “*e*” (esquerda).

A função “*set_pos()*” permite-nos modificar a posição do nó ao qual o script está anexado e recebe como parâmetro um objeto *Vector2*. Por sua vez, “*get_pos()*” nos retorna a posição do nó. Cabe esclarecer que o *Vector2* é uma estrutura de elemento que pode ser usada para representar posições no espaço 2D ou qualquer outro par de valores numéricos, o que precisamos saber por hora é que no espaço 2D possuímos valores de posição nos eixos das ordenadas (posição Y, altura) e das abscissas (eixo X, largura), bem como razões vetoriais.

Na linha “*_pos(get_pos() + Vector2(velocidade, 0) * delta * (d + e))*”, somamos “*d + e*” para garantir que caso o usuário tente fazer a nave seguir as duas direções possíveis

simultaneamente a nave anule a manobra e fique parada. O trecho “Vector2(velocidade, 0)” apresenta o valor da variável “velocidade” (300) para X e 0 para Y e, quando multiplicamos por delta, garantimos que a nave se mova 300 pixels por segundo. Após seguir esses passos, salve as modificações e teste seu jogo através da cena principal clicando na tecla F5. Veja sua nave movendo-se e interrompendo manobras não possíveis.

Note que temos um problema, podemos sair da janela do jogo, para evitar isso, após a validação dos “Inputs” e antes de nosso cálculo de posição, valide se o jogador está dentro da tela acrescentando o fragmento de código a seguir:

```
[...]  
if get_pos().x > (800 - 50):  
    d = 0  
  
if get_pos().x < 50:  
    e = 0  
  
# Cálculo de posicionamento a uma velocidade constante a cada frame  
set_pos(get_pos() + Vector2(velocidade, 0) * delta * (d + e))  
pass
```

6.5 Criando nossos tiros para a nave

Nossa nave está funcional, vamos agora criar os seus disparos, de forma que podemos combater meteoros que cairão do topo da janela do jogo. Para tanto, crie uma nova cena, insira um nó “Node2D” nomeie-o como “tiro” e dentro deste instancie um nó “Sprite”, selecionando a imagem “laserGreen.png” do diretório “png” para o campo “Texture”. Sua cena deve estar estruturada como a Figura 72.

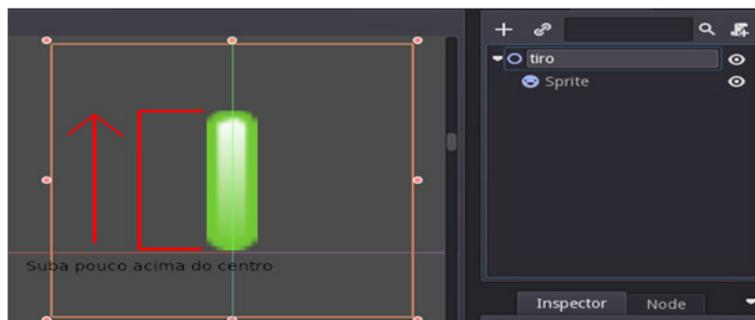


Figura 72. Posicionamento e estrutura de cena para o tiro

Fonte: Próprio autor.

Mova o tiro um pouco acima do eixo das abscissas (eixo X) mantendo-o em cima e ao centro do eixo das ordenadas (eixo Y), como mostrado na imagem acima. Para a movimentação do tiro, tenha em mente que o nó “*Sprite*” está dentro do nó “tiro”, logo, para movê-lo, selecione-o, pois se movermos o “*Sprite*” o tiro trocará de posição dentro do nó “tiro”, ou seja, para trocar a posição da cena, mova sempre o nó pai.

Salve sua cena com o nome “tiro.tscn” no diretório “Cenas”. Para animar o disparo, vamos criar um script para o nó “tiro” com o nome “tiro.gd”, salvá-lo no nosso diretório “Scripts” e inserir o seguinte código:

```
extends Node2D

var velocidade = 500

func _ready():
    set_process(true)
    pass

func _process(delta):
    set_pos(get_pos() + Vector2(0, -1) * velocidade * delta)
    if get_pos().y < -30:
        free()
    pass
```

Algumas constatações se fazem pertinentes, é possível notar, por exemplo, que esse artefato apresenta em relação à nave: programação similar, cálculo de deslocamento idêntico e tiro mais rápido. Perceba que verificamos se o tiro saiu da tela e o destruimos com o uso da função “*free()*”. Com isso, garantimos uma gestão de memória e limpeza de objetos não utilizados no game.

6.6 Atirando com a nave

Temos nosso tiro e uma nave que se move, agora precisamos gerar os tiros em tempo de execução, para tanto, temos que gerar as instâncias do tiro cada vez que o jogador pressiona a barra de espaço do teclado (tecla que configuramos). Primeiro, precisamos pré-carregar nossa cena de tiro, para ela ser preparada para a nave, para isso modifique o script “nave.gd” com o seguinte trecho de código:

```

extends Node2D

var sprite_tiro = preload("res://Cenas/tiro.tscn")
var velocidade = 500
var espera = .2
var disparo = 0

[...]

```

Criamos três novas variáveis, o “sprite_tiro”, que carrega a cena tiro, para que possamos replicá-la sempre que desejarmos, a “espera”, que é um tempo entre um tiro e outro, para que os sprites não saiam sobrepostos um com o outro, a variável “disparo” que valida a execução do tiro sempre que seu valor for ‘0’ ou menor. Acrescente ao final da função “_process”, logo após nosso cálculo de posicionamento da nave, o seguinte trecho de código:

```

[...]
set_pos(get_pos() + Vector2(velocidade, 0) * delta * (d + e))

    if Input.is_action_pressed("tiro"):
        if disparo <= 0:
            var tiro = sprite_tiro.instance()
            tiro.set_global_pos(get_global_pos())
            get_node("../").add_child(tiro)
            disparo = espera

        pass

    if disparo > 0:
        disparo -= delta

    pass

```

Ao pressionar o botão de tiro (barra de espaço, como configurado), verificamos se um disparo é possível, ou seja, se o usuário já aguardou o tempo mínimo para um disparo. Quando permitido, criamos um novo tiro com os nós da cena “tiro.tscn” pré-carregada na variável “sprite_tiro” através da função “instance()”. Pegamos a posição global da nave (local em que ela está na cena no momento do disparo) e colocamos no tiro, por fim, adicionamos nosso tiro à cena principal com a função “add_child()”,

Note que antes utilizamos a função “get_node()” com parâmetro “../”, isso porque nossa nave é nó do nó principal, então, como o script atual faz referência a nave, precisamos encontrar seu nó pai (a cena principal) e adicionar o novo nó tiro neste.

Ao executar um tiro, a variável “disparo” recebe o valor de espera. A instrução “disparo -= delta” subtrai o tempo de espera da variável até ela obter o valor entre ‘0’ ou abaixo deste, liberando para novo disparo. Salve seu script e teste seu game pressionando a tecla F5, você deve ser capaz de disparar projéteis da cena de tiro.

6.7 Criando um meteoro

Agora, vamos criar o desafio do game, um meteoro que teremos que destruir com nossos disparos. Inicie uma nova cena, adicione um nó “Node2d” com nome “meteoro” e, posteriormente, um nó “Sprite”, selecionando a imagem “meteorBig.png” do diretório “png” no campo “Texture”. Salve a cena com o nome “meteoro.tscn” no diretório “Cenas”. Sua cena deve ter a estrutura apresentada na Figura 73.

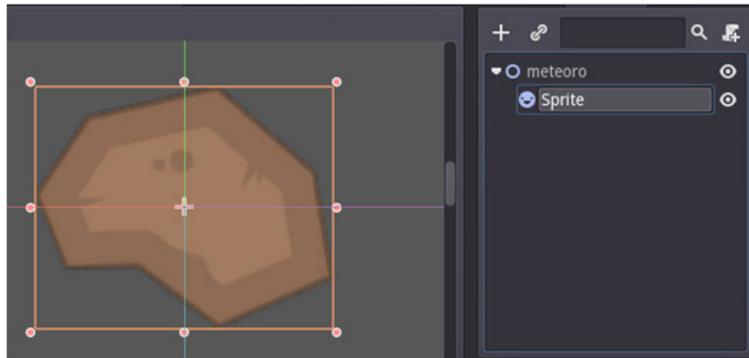


Figura 73. Estrutura de cena do meteoro

Fonte: Próprio autor.

Para nosso meteoro, anexe ao nó “meteoro” um novo *script*, o qual você irá salvar com o nome “meteoro.gd”, insira o seguinte trecho de código neste:

```
extends Node2D

var velocidade = 250
var rotacao = 0

func _ready():
    randomize() # reseta a semente
    set_process(true)
    rotacao = rand_range(-3, 3) # randomiza entre -2 e 2 a rotação
    pass

func _process(delta):
    set_pos(get_pos() + Vector2(0, 1) * velocidade * delta)
    rotate(rotacao * delta) # rotaciona o meteoro
    pass
```

O meteoro irá cair do topo da tela, o jogador pode desviar ou destruí-lo, a função “*randomize()*” gera uma semente de randomização para a função “*rand_range*”, que randomiza um valor entre “-3” e “3” para rotar nosso meteoro. A função “*rotate()*” faz efetivamente a rotação, atualizando o movimento a cada quadro. Salve as modificações do script e a cena do meteoro.

6.8 Spawner de meteoros

Um *spawner* na linguagem dos jogos é um gerador de objetos sob determinada instrução, nesta subseção iremos aprender a desenvolvê-lo. Crie um nova cena e adicione um nó “Node” com o nome “spawner”, salve a cena com o nome “spawner.tscn” e inicie um script com nome “spawner.gd”. Por fim, coloque cada arquivo no seu devido diretório e insira em seu novo *script* o seguinte código:

```
extends Node

var sprite_meteor = preload("res://Cenas/meteorito.tscn")
var espera = 2

func _ready():
    set_process(true)
    pass

func _process(delta):
    if espera > 0:
        espera -= delta
    else:
        espera = rand_range(0.3, 1)
        var meteorito = sprite_meteor.instance()
        meteorito.set_pos(Vector2(rand_range(30, 770), -40))
        get_owner().add_child(meteorito)
    pass
```

Nosso spawner cria e insere meteoros no topo da tela, dentro da largura desta. No código, temos uma função nova “*get_owner()*”, que retorna o nó atual ao qual o script está afiliado. Assim, nossos meteoros são criados e adicionados ao nó “spawner”. Salve as modificações do script e de sua cena e insira a cena “spawner.tscn” como instância no nó “Node” da cena principal “principal.tscn”. Salve a modificação na cena principal e execute o game a partir da tecla F5.

6.9 Colisões

Finalmente trabalharemos nas colisões, nossos projéteis ainda não destroem nada, por isso, iremos aperfeiçoar nossos meteoros. Abra a cena “meteorito.tscn”, selecione o nó pai “meteorito”, adicione a ele o nó “*CollisionPolygon2D*” e o renomeie para “colisão”. Desse modo, obteremos a seguinte estrutura para a cena, conforme apresentado na Figura 74:

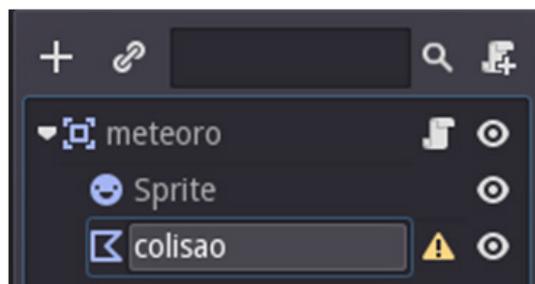


Figura 74. Modificação na estrutura da cena do meteoro

Fonte: Próprio autor.

Aprenderemos agora uma maneira bem fácil e intuitiva de criar colisões com os nós especiais do Godot. Mas algumas inserções ainda são necessárias. Selecione o nó “colisao” e clique no ícone de um lápis acima de sua *viewport*, com ele ativo, desenhe sob o desenho do meteoro sua colisão, como demonstrado na Figura 75. O polígono de representação da colisão é gerado quando você terminar de ligar o primeiro ponto ao último, ativando logo em seguida o modo de edição dos pontos. Tais medidas podem ser percebidas pela indicação do ícone de um ponto com uma seta ao lado do lápis.

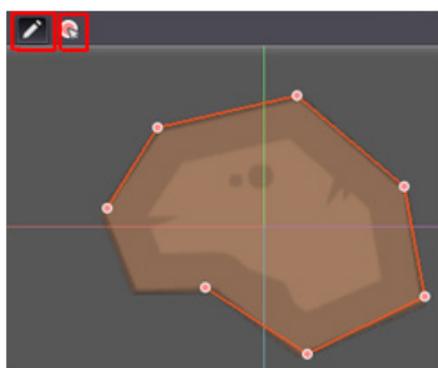


Figura 75. Desenhando colisão com o nó CollisionPolygon2D

Fonte: Próprio autor.

Agora iremos clicar, com o botão esquerdo na árvore de nós, em cima do nó “meteiro” e, no menu de opções, selecione a operação **Change type**, que converte nosso nó para outro tipo. Iremos convertê-lo para “Area2D”, conforme a Figura 76.

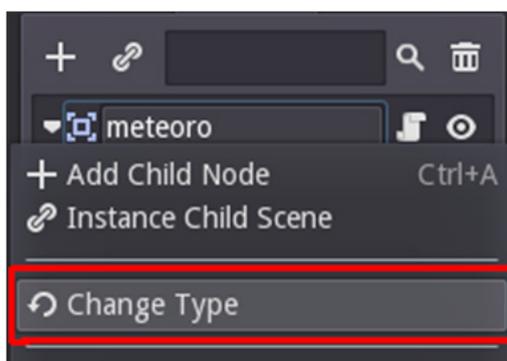


Figura 76. Alterando o tipo de um nó

Fonte: Próprio autor.

A mudança se deve ao fato da classe “Area2D” compreender o universo de objetos

2D e suas colisões, logo, existem diversos métodos e mecanismos para agilizar nosso trabalho, como os “*Signals*”. Feita a alteração volte ao script “*meteoros.gd*” e faça a seguinte modificação:

```
extends Area2D # mudamos a classe herdada de Node2D para Area2D

[...]
```

Abra a cena “*tiro.tscn*” e repita o processo para o tiro, inclusive a adaptação no script, assim, seu nó também será convertido para “*Area2D*”. Opcionalmente você pode utilizar o “*CollisionShape2D*”, como aprendemos anteriormente, pois a forma do tiro é mais simples que o meteoro.

6.10 Identificando objetos e atirando neles

Nesta etapa iremos agrupar os meteoros criados e fazer nossos tiros identificá-los. Para isso, abra sua cena “*tiro.tscn*” e selecione nosso tiro (já como *Area2D*) em nossa hierarquia de nós, no painel “*Node*” selecione um dos *Signals* disponibilizados “*area_enter*” e clique em “*Connect..*”, na janela de conexão de *Signals*, deixe as configurações padrões e clique em “*Connect*”. Você pode alterar o nome para o método do tiro que criaremos. Os passos para criação do *Signal* podem ser identificados na Figura 77.

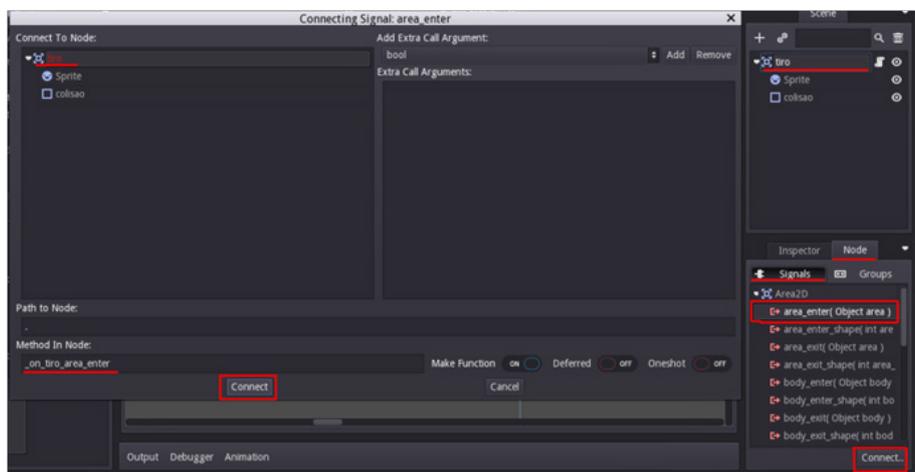


Figura 77. Criação de um *Signal* via UI

Fonte: Próprio autor.

Com nossa conexão criada, abra o script “tiro.gd” e implemente o método criado para ele através do Signal gerado inserindo no final do arquivo o seguinte código:

```
[...]  
func _on_tiro_area_enter( area ):  
    if area.is_in_group(game.METEORO): # identificaremos nosso meteoro por grupo  
        area.queue_free()  
        pass  
pass # replace with function body
```

Inicialmente criamos um arquivo de controle para o game, a saber, o script “game.gd”, que gerou nosso Singleton, ele está presente e iremos utilizá-lo. Abra seu arquivo “meteoro.gd” e modifique-o com a função “_ready()”, conforme a seguir:

```
[...]  
func _ready():  
    add_to_group(game.METEORO) # w adicionamos nosso meteoro a seu devido grupo  
    game.pontos += 10 #cada meteoro destruído o jogador ganha 10 pontos  
    randomize() # reseta a semente  
    set_process(true)  
    rotacao = rand_range(-3, 3) # rndomiza entre -2 e 2 a rotação  
    pass  
[...]
```

A instrução “*add_to_group(game.METEORO)*” adiciona nosso meteoro a um grupo chamado “meteoro” (o nome do grupo é regido pela variável constante “METEORO”). Utilizando nosso *Singleton*, a cada meteoro destruído, o usuário ganha 10 pontos. Dessa forma nosso código de tiro funciona inteiramente. Salve suas modificações e teste seu game (F5), a partir da inserção desses comandos o jogador deverá ser capaz de destruir os meteoros.

Agora temos que aplicar a possibilidade de dano em nossa nave, caso um meteoro a acerte, para tanto, como você aprendeu, converta o seu nó pai para “*Area2D*”, crie uma colisão para a nave e ative um *Signal* para verificar se um meteoro a atingiu. Assim você irá gerar algo em torno do exemplo a seguir no arquivo “nave.gd”:

```
[...]  
func _on_nave_area_enter( area ):  
    if area.is_in_group(game.METEORO):  
        game.vidas -= 1 # retiramos uma vida quando somos acertados  
        area.queue_free()  
    pass # replace with function body
```

Caso as vidas acabem, o game precisará ser reinicializado, o que implicará na perda da pontuação já acumulada e na reinicialização da fase. Quando usamos um *Singleton* para gerenciar nossos estados, é interessante termos funções para resetar valores, isso

ajuda a reiniciar uma fase. Como nosso jogo é pequeno, faremos isso dentro do próprio script da nave, insira, portanto, o seguinte trecho de código na função “_process”:

```
[...]
    if game.vidas <= 0:
        game.vidas = 3
        game.pontos = 0
        get_tree().reload_current_scene() # reinicia a cena atual
[...]
```

6.11 Plano de fundo

Nossa cena ainda é bem estática, para alterar o cenário vamos colorir nosso céu com estrelas. Iremos agora trabalhar no plano de fundo (*background*). Abra sua cena “principal.tscn”, mude o tipo do nó principal (“Node”) para “CanvasLayer” e insira neste o nó “ParallaxBackground”, com esse tipo de nó podemos fazer nosso plano de fundo ter vida e criar efeitos bem interessantes, pois esse componente controla grupos de camadas e imagens e possui diversas configurações.

Dentro de nosso “ParallaxBackground”, deverão ser inseridos nós filhos “ParallaxLayer”, cada um destes refere-se a uma camada dentro de nosso *Background*. Com uma camada selecionada podemos arrastar através do “FileSystem” as imagens do diretório “Background”, que está dentro do diretório “png” na raiz de nosso projeto.

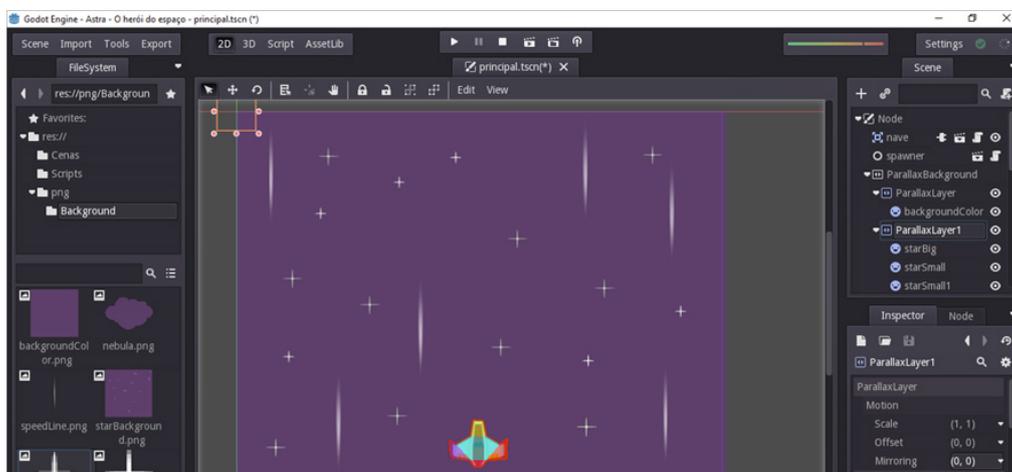


Figura 78. Composição de um background com parallax

Fonte: Próprio autor.

Para nossa composição, criamos duas camadas (dois nós “*ParallaxLayer*”) para o background (“*ParallaxBackground*”), na primeira inserimos um fundo roxo e na segunda, diversas estrelas, perceba que à medida que inserimos nas camadas um item ela automaticamente gera um nó “*Sprite*”. Sua composição será algo em torno do que é apresentado na Figura 78.

Se testarmos, perceberemos que nada se move, mudaremos tal conjuntura a partir da realização de algumas configurações. Insira em sua cena um nó “*camera2D*” alterando a configuração do campo “*Anchor Mode*” no “*Inspect*” para “*Fixed TopLeft*” e marque a opção “*Current*” (Corrente) para torná-la a câmera principal. A câmera se torna pertinente, pois com ela podemos animar nosso plano de fundo, crie um script para o nó “*Camera2D*” com o nome “*câmera.gd*” e insira o código a seguir:

```
extends Camera2D

var velocidade_back = 200

func _ready():
    set_process(true)
    pass

func _process(delta):
    set_offset(get_offset() +Vector2(0, -1) * velocidade_back * delta)
    pass
```

Repare que o cenário passa por nossos olhos seguindo, simplesmente, a programação. Algumas mudanças ainda são necessárias, temos que espelhar o plano para tornar o efeito de continuidade presente, para tanto, selecione os nós camada (“*ParallaxLayer*”) e no “*Inspect*” vá a opção “*Mirroring*”, inserindo em “*Y*” o tamanho de altura da tela (600 pixels, como configuramos). Salve suas modificações e teste seu jogo (F5). Você deverá ser capaz de ver seu plano de fundo animado e contínuo.

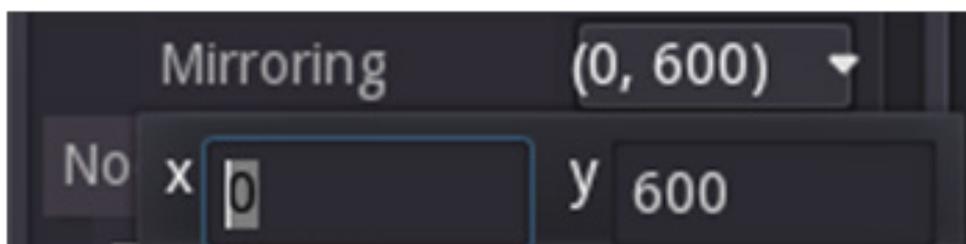


Figura 79. Inserindo espelhamento com a opção Mirroring
Fonte: Próprio autor.

6.12 HUD

Nosso jogo está funcional, agora temos que exibir ao usuário a pontuação em sua HUD, trata-se, como já explicado, de uma tela de alerta que exibe informativos pertinentes ao jogador. Abra sua cena “principal.tcsn” e insira um novo nó do tipo “CanvasLayer” com o nome de “HUD”, dentro deste insira um nó “Label” e ajuste em um dos cantos do topo de sua tela de jogo.

Como aprendemos, um “label” permite a inserção de texto em nosso jogo. Para criar a pontuação do jogador, teremos que possuir uma fonte. Busque na internet uma que o agrade e a insira através da operação **import** na **barra de menu principal**, selecionando a opção **Font**.

Na janela apresentada, conforme Figura 80, você seleciona o arquivo desejado (*Source Font*), o tamanho e o local em que irá salvar a fonte em seu projeto (*Dest Resource*), salve no diretório raiz. O interessante da importação é que você pode testar antes, como apresentado, você tem um campo “Test” onde pode inserir seu texto para visualizar a adequação da fonte, inclusive o comportamento de uma cor.

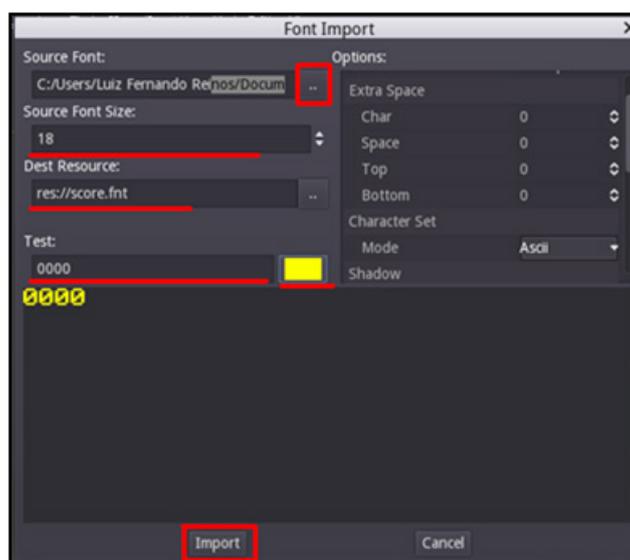


Figura 80. Importação de fontes

Fonte: Próprio autor.

Ao importar sua fonte, selecione nosso “label” e, no “inspect”, abra a fonte na opção “Custom Fonts”, depois modifique sua cor em “Custom Colors”. Seguindo as definições desta seção sua cena está bem parecida com o que é apresentado na Figura 81:

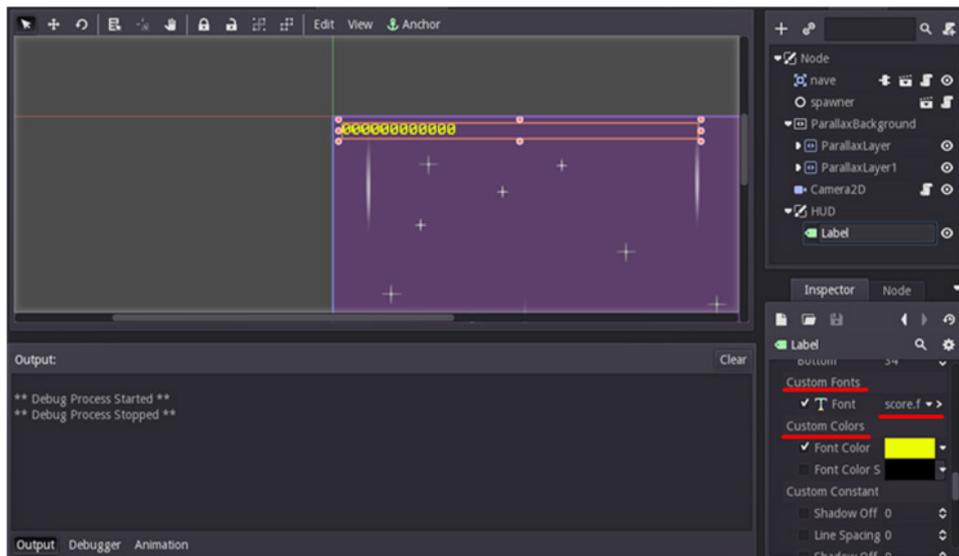


Figura 81. Selecionando uma fonte para um rótulo

Fonte: Próprio autor.

Precisamos atualizar a pontuação do jogador a cada tiro que ele acertar, desta forma, crie para nosso “Label” o script “score.gd” e insira o código a seguir, salve seu game e teste (F5):

```

extends Label

func _ready():
    set_process(true)
    pass

func _process(delta):
    set_text(String(game.pontos)) #convertemos nosso tipo inteiro para texto (string)
    pass

```

6.13 Exercícios

1. Pegue sua foto ou a de um amigo e a redesenhe em pixel arte. Para esta atividade, use uma cena de tamanho 64 x 64.

O método do pixel arte consiste em minimizar o desenho, por isso foque nas principais características a serem apresentadas.

2. Crie telas em pixel arte, comece com uma matriz de pixel pequena, como a do

exercício anterior que foi de 64 X 64. Vá aumentando gradativamente (300 X 200 e assim por diante), seguindo os passos para criação de um pixel arte apresentada neste capítulo. Você pode representar o que quiser.

3. Utilizando o Inkscape, faça um índio, conforme o que aprendeu neste capítulo e nas dicas apresentadas neste exercício, estando estas na Figura 82 abaixo.

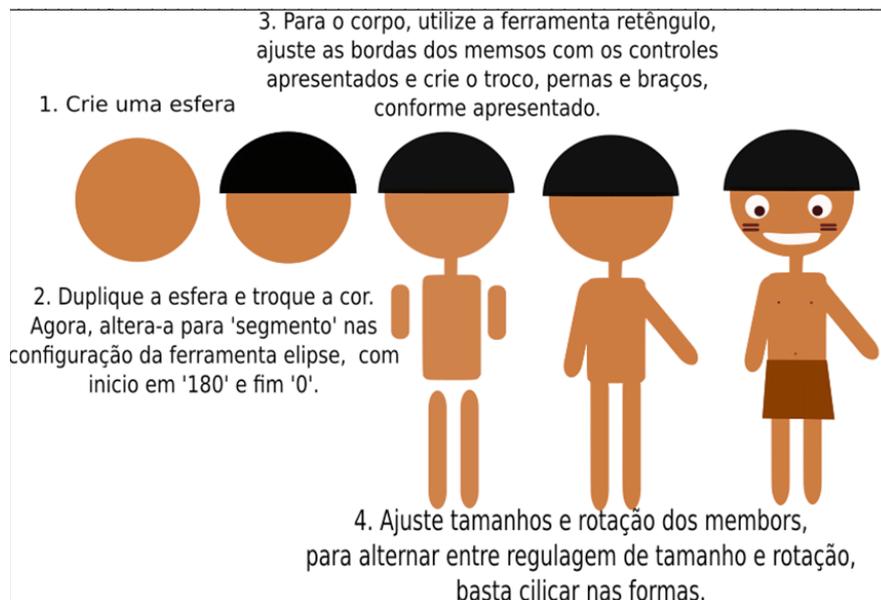


Figura 82. Exercício de desenho com Inkscape

Fonte: Próprio autor.

4. Rasterize desenhos ou imagens que você já tenha feito para praticar a rasterização. Como desafio, estude e se aprofunde sobre a funcionalidade **Trace Pixel Art** disponível no menu **Caminho**.

Nota: Sobre Trace Pixel Art a documentação do inkscape disponibiliza o seguinte tutorial: <https://inkscape.org/pt-br/doc/tutorials/tracing-pixelart/tutorial-tracing-pixelart.en.html>

5. Crie diagramas de acordo com a linguagem de design do Godot para o que é pedido:
 - a. Uma cena do seu quarto, com móveis, ambientes (se tiver), focando nas ações que você pode tomar e em como usar certos objetos.
 - b. Uma cena de jogos situados em uma floresta, com inimigos e itens a serem recolhidos.
 - c. Um jogo de RPG que você conhece.
6. Melhore seu game “Astra’ – o herói do espaço”, para isso, estude e faça uso do AVA criado para este livro.

7. Atmosfera musical

A sonoplastia é um conjunto de efeitos sonoros necessários para a representação de um ambiente e sua execução leva tempo e estudo. Quando temos a intenção de fazer o jogador imergir dentro de um ambiente digital, sentir-se dentro de uma caverna ou em uma floresta, de modo que acredite que este meio existe, o aspecto visual pode ser enriquecido com a riqueza da música, dos sons e dos efeitos sonoros, os quais devem interagir simultaneamente e em harmonia, com isso cria-se uma *atmosfera*.

O trabalho de um músico em um jogo é de extrema importância, nesse sentido, cabe esclarecer que este livro não aborda a composição musical, pois se trata de um estudo técnico profundo, mas os autores conhecem ferramentas que podem ajudar em sua produtividade sonora, trata-se das ferramentas de geração automática.

7.1 BFXR

O BFXR³² é uma ferramenta simples de auto-geração desenvolvida para que qualquer pessoa sem treinamento prévio gere sons. Tal programa oferece os sons e os efeitos mais comuns dentro dos jogos, como pulo, explosão, tiro, entre outros. A Figura 83 apresenta a interface do programa.

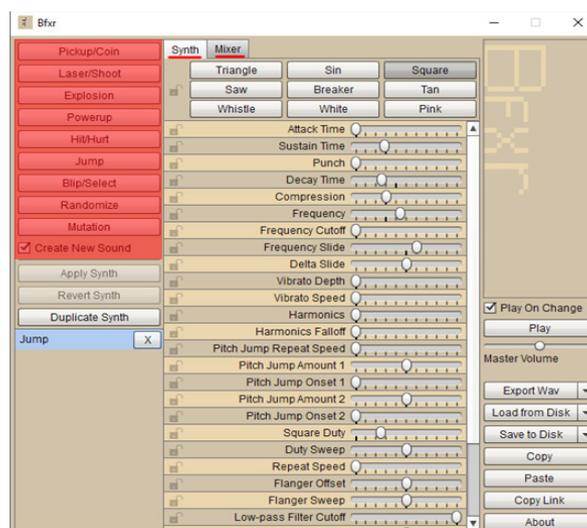


Figura 83. BFXR

Fonte: BFXR (2017)

Adaptado pelo autor.

32 BFXR – Disponível em: <<http://www.bfxr.net/>>.

Nas opções do *menu de criação de sons*, automaticamente, por algoritmos de geração, obtemos um som ou efeito automático, podendo ajustá-lo após obtenção do mesmo. Você ainda pode unir os diversos tipos de sons que gerou para criar algo ainda mais complexo, pois o BFXR acopla tanto o sintetizador quanto um mixer de som. O mixer pode ser visto na Figura 84.

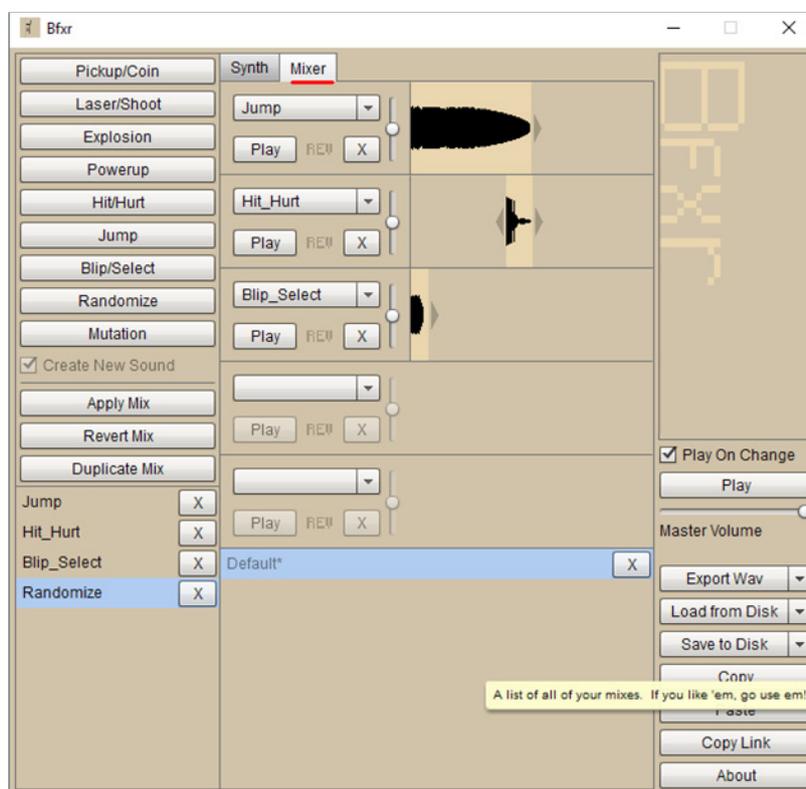


Figura 84. Mixer de sons do BFXR

Fonte: Fonte: BFXR (2017)

Adaptado pelo autor.

O uso do BFXR é simples, após adquirir o que deseja, basta exportar seu som ou efeito e utilizá-lo como desejar. O formato utilizado pelo programa é o “wav”. Vale ressaltar que o BFXR evoluiu de um software anterior ao SFXR³³, também gratuito e disponível na internet, ambos têm a mesma dinâmica.

33 SFXR – Disponível em: <http://www.drpetter.se/project_sfyr.html>.

7.2 Audacity e LMMS

Não poderíamos deixar de apresentar ferramentas de edição profissional para que pessoas que tem afinidade com música também sejam contempladas. O Audacity³⁴ é um editor de música gratuito que permite a gravação, mixagem e adição de recursos para melhoramento de efeitos de áudio. Você pode ter uma noção da interface do programa na Figura 85.

O LMMS³⁵ é uma ferramenta gratuita e *open source*, a comunidade entorno desta aplicação tem aumentado consideravelmente nos últimos anos. Trata-se de um recurso mais complexo do que o Audacity, visto que contempla recursos de criação de sons e efeitos sonoros profissionais e compatibilidade com hardware específico para edição de música com teclados MIDI e placas de som especiais.

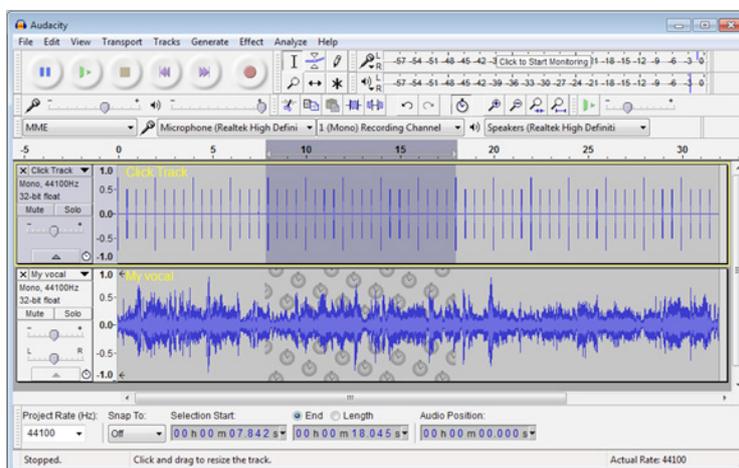


Figura 85. Interface do Audacity

Fonte: Audacity (2017)

O LMMS pode ser utilizado tanto para criações simples como para mais complexas. Sua particularidade a ser abordada aqui é os diversos moduladores que imitam sons e ajudam na editoração e na criação de sons e músicas. Para jogos temos um sintetizador de sons 8-bits, por exemplo. A Figura 86 nos permite obter uma noção de uso e funcionamento da ferramenta.

34 Audacity: Disponível em: <http://www.audacityteam.org/>.

35 LMMS – Disponível em: <https://lmms.io/>.



Figura 86. Interface do LMMS

Fonte: LMMS (2017).

7.3 Inserindo músicas e sons

Em continuidade ao nosso jogo “Astra’ – o herói do espaço”, iremos inserir sons para nossos tiros e para a destruição de meteoros. Você deverá utilizar o BFXR para criar os efeitos e em seguida inseri-los em nosso projeto, para essa ação crie no diretório raiz uma pasta com o nome “Áudios”, colocaremos todos os sons e músicas nesse novo local.

Abra sua cena “tiro.tscn” e insira dentro do nó pai o nó “*SamplePlayer2D*” com nome “som”. Selecione-o e crie uma nova biblioteca de sons (SampleLibrary) através do campo “*Samples*” no “*Inspect*”, após isso clique na seta em frente ao campo. A Figura 87 apresenta o processo descrito.

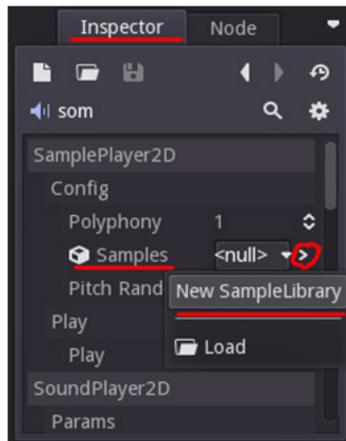


Figura 87. Criando biblioteca de sons

Fonte: Próprio autor.

Ao abrir a biblioteca criada, teremos um painel de controle para inserirmos nossos sons e músicas, conforme Figura 88. Selecione o ícone de uma pasta para abrir o navegador de arquivos e selecione o som produzido para o tiro. Ao inserir um som, ele irá aparecer na biblioteca, compondo-a, nesta etapa você pode alterar o nome e outras configurações do arquivo de som, além de removê-lo se necessário. O nome é importante pois o utilizaremos no código.

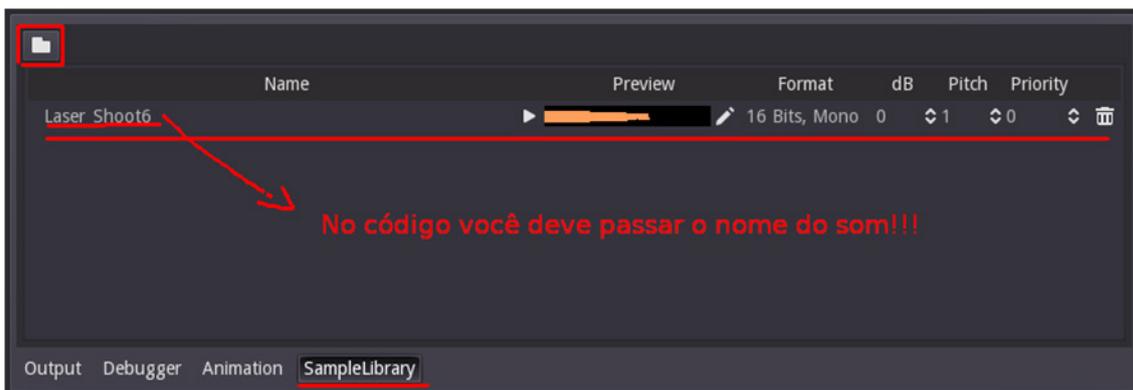


Figura 88. Sample Library de músicas

Fonte: Próprio autor.

Agora, voltando ao script “tiro.gd”, uma vez que temos um nó de som na cena e queremos o som no ato de criação do tiro, iremos modificar nosso código inserindo o seguinte trecho na função “_ready()”:

```
[...]
func _ready():
    set_process(true)
    get_node("som").play("Laser_Shoot6")
    pass
[...]
```

O funcionamento é simples, a linha “get_node(“som”).play(“Laser_Shoot6”)” faz

uso do nosso nó “som” e executa um som da biblioteca, posteriormente, você deverá adicionar na mesma biblioteca o som da explosão, pois tratamos a destruição dos meteoros no script do tiro, adicione a linha antes da destruição do meteoro.

Como exercitação, adicione um nó “*SamplePlayer2D*” na cena principal e insira uma música tema, ela pode ser obtida em um site muito utilizado por desenvolvedores, que contêm músicas com royalty gratuito, trata-se do repositório da Incompetech³⁶. Neste você pode baixar músicas e testar com seu jogo sem nenhum tipo de indulgência, apenas leia as licenças para mais informações, ouça as músicas e selecione as que te agradarem.

7.4 Exercícios

1. Crie sons e efeitos sonoros para a cena principal do jogo que estamos construindo, insira o barulho de dano na nave, reinicie e inicie a fase.
2. Faça mixagem com o BRFX e junte às músicas do Incompetch através do Audacity para criar uma atmosfera para o jogo.

36 Incompetech – Disponível em: <<https://incompetech.com/>>.

8. Jogos MóBILE

A indústria de jogos investe pesado em ferramentas e fórmulas para tornar os jogos cada vez melhores para seu público, o que exige cada criatividade e comprometimento de seus criadores.

A Digi-Capital tem uma estimativa de que os jogos vão bater os US\$ 45 bilhões em faturamento até 2018. Existem previsões mais otimistas que acreditam em um crescimento de 8% ao ano, nessa perspectiva, os jogos alcançariam cerca de US\$ 110 bilhões ultrapassando os consoles.

Uma empresa chamada Swrve fez um levantamento que revelou que acerca de 10 milhões de usuários jogam games *free-to-play*³⁷. Em seus resultados, ela deduziu dados um tanto quanto interessantes. De acordo com a publicação da Re/Code, a Swrve constatou que 19% dos usuários abriam os jogos obtidos uma única vez, 66% destes paravam de acessar o jogo após 24 horas. Verificou-se também que 53% dos usuários compram recursos extras para seu jogo apenas na primeira semana após download e que apenas 2,2% gastam dinheiro com jogos móveis.

Para entrar no mercado de jogos mobile, você deve focar em soluções criativas, desafiadoras e envolventes que consigam fazer as pessoas jogarem e evoluírem em seu game. Pois o estudo da Swrve nos apresenta uma realidade em que o número de downloads não indica que o game é de sucesso, apenas conhecido.

Logo, devemos focar em jogos divertidos e únicos, capazes de transmitir experiências significativas para um usuário ter o prazer de retornar e jogar, sentindo que fez um investimento e não que teve um gasto qualquer. Ademais, é importante salientar que a primeira impressão é de fato a que fica.

Temos características fundamentais que temos que tomar atenção em jogos mobile. Foque no que é mais importante:

- Usabilidade;
- Facilidade;
- Desafio;
- Diversão;
- Imersão;
- Interatividade.

³⁷ Free-to-play – em tradução livre, significa “grátis para jogar”, assim, trata-se de jogos que você pode obter e jogar gratuitamente sem custos.

8.1 Controles touch screen

Para tornar nosso jogo acessível a dispositivos mobile, sem a existência do teclado, primeiro devemos criar uma HUD que dê acesso aos controles, para tanto, faremos uso dos recursos gráficos disponibilizados em nosso AVA. Para este capítulo, faça download do pacote “touchdisplay.zip”, você obterá controles de Paul Wortmann (2011), conforme apresentado na Figura 89.



Figura 89. Controles Touch

Fonte: Wortmann (2011).

Descompacte a pasta “Touch” e a insira dentro da raiz de seu projeto. Em nossa cena “principal.tcsn” do nó “HUD”, adicione três nós “TouchScreenButton” com nome “esquerda”, “direita” e “tiro”.

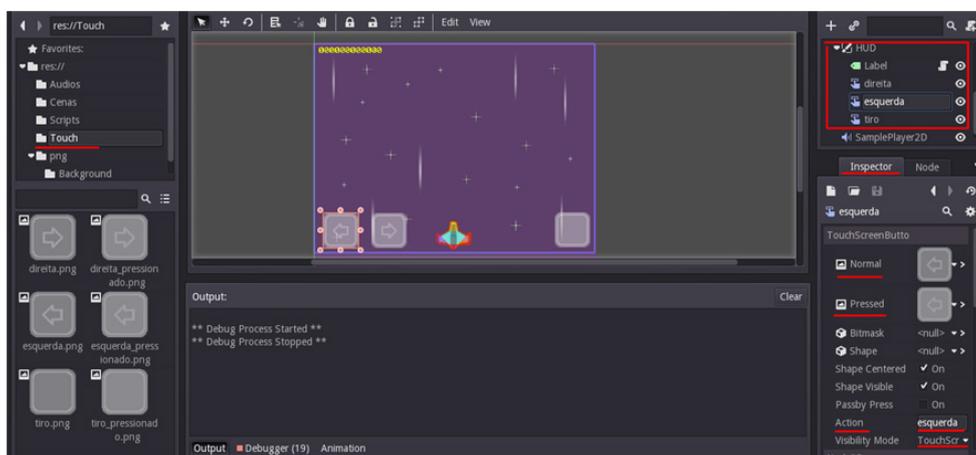


Figura 90. Configurando o Touch Screen

Fonte: Próprio autor.

Selecione cada nó e insira a respectiva figura para os campos “Normal” (quando não pressionado) e “Pressed” (quando pressionado o botão) no “Inspect”, você pode arrastar a imagem até o campo. Insira no campo “Action” de cada botão sua ação respectiva, essas ações foram configuradas no início do projeto, nossos inputs, são elas: “esquerda”, “direita” e “tiro”, insira cada ação em seu botão. Sua cena deve estar parecida com o que está sendo demonstrado na Figura 90.

Salve suas modificações e teste seu jogo (F5). Repare que os botões não funcionam. Verifique que temos no “*Inspect*” para os touch a configuração “*Visibility Mode*” (Modo de visibilidade) com as opções “*always*” (sempre), que faz os botões ficarem visíveis todo o tempo, e “*TouchScreen Only*” (Touch ativo), configure nossos botões para esta última opção.

Repare que com essa configuração selecionada o jogo fica sem os touch, porém, por querermos testá-los, temos que informar ao Godot que queremos simular estes botões. Para tanto, entre em **Project Settings** na aba “*General*”, seção “*Display*”, e acione a emulação ativando a opção “*Emulate Touchscreen*”, conforme a Figura 91, salve e teste seu game novamente, com isso você deverá ser capaz de usar os touch.



Figura 91. Ativando a emulação para os touch

Fonte: Próprio autor.

8.2 Exercícios

1. Angry Birds® 2 foi lançado em 2015, veja os jogos anteriores e repare na evolução do game, bem como a avaliação de seus compradores. Em seguida, trace uma lista para cada jogo que encontrar, reunindo as principais características e metas estéticas deles, relativas às características fundamentais abordadas.

Dica: Tente tornar prazerosa e repetitiva essa atividade, pois, se você realmente quer desenvolver jogos, será uma prática comum e rotineira. O intuito é você aprender como um jogo consegue ter uma continuidade, colocando desafios, interatividade, entre outras características, e ainda sim manter sua usabilidade e facilidade de uso.

Nota: Uma das coisas que você irá descobrir com esta atividade é que o jogo é **SEMPRE** o mesmo, simples e com mecânica padrão. Mas novas funcionalidades apareceram durante sua evolução, **CONSTANTEMENTE**, afetando o gameplay e aumentando a imersão do jogador.

9. Conclusão

Este livro foi desenvolvido para ser utilizado como um guia, servindo como um manual técnico científico. Assim, espera-se que, a partir da leitura, da complementação e da prática de suas atividades e instruções, seus leitores possam desenvolver seu conhecimento, aprendendo conceitos e técnicas importantes para desenvolver jogos eletrônicos.

Diversos métodos e práticas abordadas foram testados pelos autores em ambientes de ensino e profissionalmente, validando alguns ganhos na produtividade e no aperfeiçoamento da criação e modelagem de um jogo.

O uso de softwares livres na abordagem do material se justifica pelo objetivo de trazer maior conhecimento aos leitores de como existem ferramentas de alto nível para trabalhar no ramo de jogos digitais, o qual foi apresentado de maneira sucinta sobre como criar arte, música, sons, efeitos e sobre os próprios jogos com ferramentas comunitárias e/ou open source.

Os estudos apresentados também fazem parte da experiência de cursos de aperfeiçoamento em jogos digitais e internet, ministrados no Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo, campus Colatina, durante o ano de 2016 e 2017, os quais formaram mais de 30 alunos. Muitos das temáticas e estudos contemplados nesta produção foram resultados de aprofundamento em temas diversos como jogos educativos e técnicas de composição de *Serious games*, de jogos comerciais e de mercado, bem como no uso de tecnologias livres para criação de jogos digitais.

Os autores esperam que as dinâmicas e metodologias abordadas para composição e gerenciamento de projetos ajudem aos leitores a obter uma visão mais ampla sobre como planejar e operacionalizar um projeto de jogo eletrônico do início ao fim, de maneira mais profissional e técnica.

Apêndice

O apêndice do livro contém diversos artefatos para seu uso. Fique à vontade para modificá-los e aprimorá-los de forma a te ajudar em seus projetos.

Apêndice A. Goa's reign paths, paper game

GOA'S REIGN PATHS

“O reino de Goa sempre sentiu a falta de um rei. Sofrendo revoltas e conflitos entre algumas pessoas, o reino pede por um peregrino com coragem e uma espada na mão, que atravesse suas planícies e ganhe a paz. Você é um conquistador e tem que tomar todos os castelos de Goa.”

Regras:

- Jogue um dado. O valor do primeiro movimento é o número do seu castelo inicial, que representa o ponto de partida do jogo no tabuleiro;
- Após o primeiro movimento, o valor do dado é o número de posições que você pode avançar no tabuleiro;
- O jogador pode entrar em qualquer direção possível a partir do ponto em que estiver no tabuleiro;
- Se você estacionar ou passar pelas posições roxas e cinzas 4 vezes, você perderá;
- Para ganhar, você deve passar todos os castelos andando de um lugar para outro.

Regras do Multiplayer:

- Dois jogadores competem começando com uma rodada de Jokenpô (rock, papel, tesoura). O vencedor desta começa;
- As regras são as mesmas, mas aqui cada jogador joga uma rodada;
- No modo multiplayer, se um jogador atravessar o castelo do outro, ele poderá ser capturado;
- Se os jogadores colidirem, eles competem com um Jokenpô inicial. Em caso de empate, a rodada se torna um empate e eles continuam. Se alguém ganhar, o jogador perdedor encerra a partida e o vencedor ganha o jogo.

BOARD GAME:

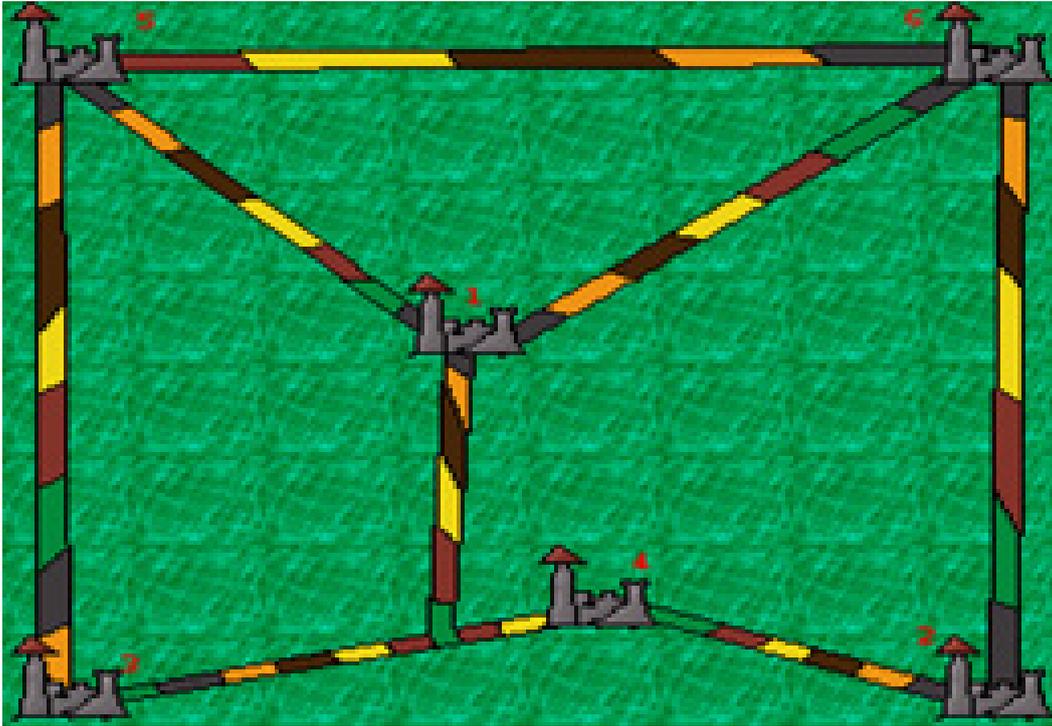


Figura 92. Tabuleiro Goa's Reign Paths

Fonte: Próprio autor.

GDD – Brothers Fighting

Última modificação: 16/11/2016

Autor: Luiz Fernando Reinoso

High Concept (Conceito): O jogo é uma remontagem dos clássicos jogos de luta do Arcade dos anos 90 e filmes de luta dos anos 70 aos 90. O que se quer passar é a sensação de emoção em lutar por justiça em prol do bem das pessoas, enfrentando entidades malignas que querem machucar aqueles que amamos. O game pretende ser bem estilo retro, mas com aspectos imaginários, retratando cidades americanizadas.

Escopo (Mecânica base): O jogo mantém uma estrutura básica de um game “Beat ‘em up”, no qual inimigos vêm em sua direção e o jogador tem que se proteger, utilizando seus recursos e seu ambiente para derrotá-los. O jogador tem que aprender e desbloquear combos de golpes de luta corporal para vencer os inimigos.

1. Visão geral:

a. Resumo:

O jogador é um apreciador de artes marciais, assim como seus amigos de infância. Um destes é sequestrado junto com sua família por um grupo de terroristas que quer montar um exército. Logo, os amigos se reúnem para encontrar o colega em perigo.

b. Aspectos Fundamentais:

- Jogo side scroller.
- Mundo baseado no mundo real em pixel arte.
- O jogador faz missões dadas a ele, completando o game com diversas missões, que contêm muita pancadaria.
- O jogador deve melhorar suas habilidades derrotando inimigos mais fortes e aprendendo novas habilidades.

2. Contexto do Jogo:

a. História do Jogo (Sinopse):

A infância no orfanato nunca foi boa para as crianças no Brooklin, apesar disso, cinco delas, muito espertas, aprenderam a se virar, logo se tornaram inseparáveis

amigos e na adolescência conseguiram ficar fora da criminalidade e se tornarem algo mais. Porém, o Vietnã não as esqueceu, o esquadrão Tales 5, como ficou conhecido, era formado por elas.

Muitas medalhas e histórias marcaram essas pessoas, mas a vida seguia, então elas se viram como folhas a seguir o fluxo do rio. A verdade é que foram treinadas como agentes especiais, lutadores notórios, salvadores da pátria.

Alguns generais presenciaram as ações do Tales 5 e comercializaram ilegalmente informações pelo mundo, chamando atenção da Triade Kane, um grupo terrorista que há anos aumenta seu exército, nomeado o Reich Kane. Com o conhecimento do treinamento e das identidades do Tales 5, eles criam soldados desumanos e sanguinários. Contudo, por saberem que poderiam ser parados pela Tales 5, os terroristas sequestram um de seus membros e a família deste, ameaçando o país e o mundo.

Assim, você, um dos membros da Tales 5, deve pará-los com todas suas habilidades e enfrentar os vilões mais fortes do mundo se necessário, encontrando seu amigo e tirando o mundo da tirania da Triade Kane para sempre.

b. Eventos Anteriores:

Na história o Brooklin foi tomado pela criminalidade e máfia dos anos 50, tráfico, roubo e decadência da cidade total.

c. Principais personagens:

- Jogador – Bruce Walker, lutador de Karatê e Taekwondo
- John Daves – Sargento coronel, melhor amigo de Walker, lutador de boxe.
- Misa Ioshia – Lutadora de Wushu. Agente da contra inteligência americana.
- Ramieres Garcia – Policial em Nova York, DEA. Lutador de Jiu-jitsu. (Raptado pela Triade Kane).
- Luiz Oliveira – Padre católico, lutador de Ninjútsu.

3. Objetos essenciais do Jogo:

a. Personagens:

- Gangue Boy – inimigos comuns
- Biruta – inimigos grandes que usam o peso como arma. Tentam te segurar.
- Assassinos – usam garras para causar mais dano.
- Terroristas – lançam granadas em você, atiram com armas e usam bastão de choque.

b. Armas:

- Revolver .22

- Revolver .38
- Rifle .15
- Espetada .12
- Metralhadora M.16
- Metralhadora Uzi
- Metralhadora AK47
- Basuca .60
- Lancer .42
- Garrafa
- Faca
- Facão
- Tijolo
- Pedra

c. Estruturas:

O jogo se passa dentro de Nova York, nos anos 80, logo, temos um ambiente repleto de prédios e luzes, carros antigos e sons de ambiente de uma cidade movimentada e agitada.

d. Objetos:

Nada a ser declarado. Dados suficientes.

4. Conflitos e Soluções:

Você, no papel de Walker tem de achar na fase, chaves e locais específicos para coletar itens para prosseguir a próxima fase. Logo, você vai ter que bater em muita gente, perguntar, procurar, encontrar itens e entender a história.

Durante as fases, poderá salvar pessoas, encontrar caminhos alternativos, mas nada que mude muito seu trajeto, mas sua pontuação e segredos do game.

5. Inteligência Artificial:

Essa parte será completada quando tiver a equipe de desenvolvimento. Mas podemos usar e definir melhor as funções do MDA aqui:

Mecânica: Sempre que avistado pelos inimigos vocês entram em briga; você avança a medida que os derrota;

Dinâmica: Ir da direita para esquerda, encontrar itens e enfrentar inimigos, conversar com reféns e encontrar informações do Tales 5;

Estética: tensão e nervosismo. Afinal, por qualquer movimento mal calculado você pode ser agarrado e tomar goles e os inimigos podem vir de qualquer lugar. Outro fator é a curiosidade e a entrega à narrativa que queremos que o jogador tenha,

para participar da história.

O projeto será desenvolvido focando na mecânica e nas reações dos jogadores. A cada nível testaremos a progressão do jogador, testando se ele realmente sente vontade de se aprofundar na história e gosta da evolução do gameplay. **Entre os testes, há o cálculo de um espaço de 15 dias para um protótipo.** Tendo esta como uma estimativa inicial.

6. Fluxo do Game:

O ciclo do game faz você seguir da esquerda para a direita e vice-versa, mas a sensação é de sempre ir em frente. A história é linear, mas a cada nova fase, há inimigos e desafios que vão aparecer e depender da compreensão do jogador sobre as habilidades adquiridas e sobre combos de luta.

Durante o game, partes do passado de Walker e do Tales 5 são revelados, para conectar o jogador cada vez mais a Ramires e a sua importância no grupo, além dos outros membros, que fornecem informações durante cada fase para que você conheça todos e entenda melhor a história de amizade.

7. Controles:

Setas direcionais:

- **CIMA:** move o personagem para cima;
- **BAIXO:** move o personagem para baixo;
- **DIREITA:** move o personagem para a direita;
- **ESQUERDA:** move o personagem para a esquerda.

Teclas de comando:

- **E:** Pega item no chão;
- **H:** soco/usa arma;
- **J:** Chute;
- **U:** Agarrar.

8. Variações de jogo:

O jogo tem a opção singleplayer e multiplayer, em ambos os casos, tendo apenas um ou mais jogadores, o usuário se posiciona contra inteligências artificiais.

9. Definições

Nada a declarar. Informações suficientes.

10. Referências

- O poderoso chefão
- Street Fight
- Era uma vez na América
- Bruce lee
- Cadillac Dinosaurs
- Final Fight
- Dragão branco

Apêndice C. High Concept for Flow

Tabela 3. Folha ‘Game Design’ da planilha do High Concepto of Flow

Fonte: próprio autor.

Game Design					
Equipe: Luiz Fernando Reinoso (Programador e Designer), Geraldo Perreira Junior (Audio Engineer),				Data: 23/01/2014	Jogo:
Plataforma	Conceito	Jogabilidade	Fluxo	Controle	Interface
Melo em que o jogo irá funcionar. Ex: Android, PC, MAC, etc.	Descrição do High Concept do game e como o mundo. Descrevendo de fato a experiência do jogador.	Descreve com o jogador irá interagir com o mundo criado. Com o ele irá lidar com o mesmo. Sua mecânica.	Seria com o o jogador irá do início ao fim do game. Como ele evolui? Para onde ele vai? Qual caminho o jogo segue?	Principais comandos e organização mecânica dos controladores.	Descrição da HUD e distribuição de ícones e controladores. São as informações na Tela, bem como os próprios personagens.
HTML, LINUX, MAC, WINDOWS	Ultrapassar obstáculos e resolver problemas com apenas algumas habilidades em mãos e com isso chegar ao objetivo do jogo.	Usará um personagem por vez selecionando através de comandos específicos e irá vencer os obstáculos usando as habilidades de cada um, particularmente.	O jogo trabalha com o crescimento de separação dos diferentes obstáculos e progressão de reconhecendo os mesmos ao decorrer do jogo. O ambiente que os personagens percorrerão são: Casa, Jardim, Estrada, Desfiladeiro, Floresta, Vila e, por fim, hospital sendo que a ordem de fluxo do jogo segue esta mesma respectivamente.	Usará as setas dos direcionais pra mover os personagens; e com comandos X e Y selecionará qual personagem deseja mover; com Z irá usar a habilidade específica do personagem selecionado	
História	Principais NPC's	Personagens	Ambiente/Universo	Motivação	Elementos
Conte a história do jogo.	Seriam os vilões, NPC's, inimigos...	Os personagens ou elementos principais do jogo.	O universo em que o jogador será imerso.	Porque os jogadores irão jogar e o que afeta no desenvolvimento?	Aspectos geográficos, demográficos e psicológicos, entre outros.
Aeron é um pai que se acidentou dentro de casa e precisa de cuidados médicos urgentemente, para isso ele conta apenas com seus dois filhos pequenos que, com suas habilidades e com a coordenação de seu pai, irão ajudá-lo a chegar ao hospital do vilarejo mais próximo, o vilarejo Oak Canyon.	Obstáculos como: pedras, montanhas, locais estreitos e coisas soltas em estalactites.	Aeron (Pai) - "Conselheiro" Moshe (Filho) - Rápido, habilidoso e com lentidão Emy (Filha) - Forte e com corda	Uma casa afastada do vilarejo onde, para chegar até o vilarejo, precisa-se passar por diferentes ambientes. O jogo acontece no período noturno, sendo isso um problema a mais pois tudo fica escuro.	Pelo desafio de se vencer obstáculos.	
Gêneros	Classificação etária	Áudio	Anotações	Observações	Conclusão
O jogo pode ter um som e gêneros ou criar um novo.	Não obrigatório. Mas é recomendado pensar neste quesito.	Descrição de tipo de músicas e sons.	Anotações de grupo.	Observações a serem feitas.	Conclusão do projeto.
Arcade					
Após completar vá ao Feature Set!	Vale lembrar que este documento é atualizado constantemente.				By Luiz Fernando Reinoso

Tabela 4. Folha ‘Funcionalidades’ da planilha do High Concepto of Flow

Fonte: próprio autor.

Nº 1	Feature	Relevância (R) ²	Andamento (%)	Data e hora de conclusão ³	Observação
1	Desenvolvimento Layout BETA	1	100,00%	23/01/2015 22:00:00	Layout Figurativo
2	Desenvolvimento Controles	1	100,00%	23/01/2015 22:00:00	Controles do jogador
3	Desenvolvimento Switch Personagem	1	100,00%	23/01/2015 23:00:00	Alternar entre personagens
4	Story Board	1	20,00%	23/01/2015 23:39:00	Progressão da história
5	Fechamento do conceito	1	100,00%	24/01/2015 00:00:00	
6	HUD	3	0,00%	23/01/2015 00:00:00	
7	Personagens	1	0,00%	23/01/2015 00:00:00	
8	Arte Gráfica	1	0,00%	23/01/2015 00:00:00	
9	Músicas	1	0,00%	23/01/2015 00:00:00	
10	Sons	3	0,00%	23/01/2015 00:00:00	
11	Ajustes finais	4	0,00%	23/01/2015 00:00:00	
12	Exportação Multiplataforma	5	0,00%	23/01/2015 00:00:00	
13			0,00%	23/01/2015 00:00:00	
14			0,00%	23/01/2015 00:00:00	
15			0,00%	23/01/2015 00:00:00	
16			0,00%	23/01/2015 00:00:00	
17			0,00%	23/01/2015 00:00:00	
18			0,00%	23/01/2015 00:00:00	
19			0,00%	23/01/2015 00:00:00	
N

Modo de uso

¹ - 10 features é normal e 20 seria um projeto mais complexo.

² - de 1 a 5, onde 1 é maior relevância e será feito primeiro.

³ - Não obrigatório.

Referências

- ADAMS, E. **Fundamentals of Game Design**, 2. ed. Berkley: New Riders. 700p. 2010.
- AHMAD, F. *et al.* inStorm: a psychosocial game suite design for non-invasive cross-generational cognitive capabilities data collection. In: **Journal of Experimental & Theoretical Artificial Intelligence**, v. 29, 2017, p. 1-13. Disponível em: <<http://dx.doi.org/10.1080/0952813X.2017.1354079>>. Acesso em: 07 ago. 2017.
- ARELLANO, J. L. H., PEREZ, J. N. S., MACIAS, A. A. M. **Identifcation and Assessment of Mental Tasks Using Task Flowcharts**. 2007. Disponível em: <https://www.academia.edu/31739971/Identifcation_and_Assessment_of_Mental_Tasks_Using_Task_Flowcharts>. Acesso em: 21 ago. 2017. 17p.
- AUDACITY – free áudio editor. Disponível em: <<http://www.audacityteam.org/download/>>. Acesso em: 31 ago. 2017.
- BFXR. Disponível em: <<http://www.bfxr.net/>>. Acesso em: 31 ago. 2017.
- CARVALHO, T. **Game Design Canvas 1.3**. 2014. Disponível em: <<http://abxygames.wixsite.com/gdcanvas>>. Acesso em: 08 ago. 2017.
- CRAWFORD, C. **The art of computer game design**, 1. ed., Washington: University at Vancouver. 96p. 1997.
- DERAKHSHANI, D. **Especialização Design de Jogos: Arte e Conceitos: Desenvolvimento de História e Narrativa para Videogames**. Califórnia: Instituto de Artes da Califórnia. Disponível em: < <https://pt.coursera.org/specializations/game-design> >. Acessado em 28 ago. 2016. 2016.
- HISTORICIZAÇÃO. In: Dicionário Priberam da Língua Portuguesa. Disponível em: <<https://priberam.pt/dlpo/historiciza%C3%A7%C3%A3o>>. Acesso em: 01 ago. 2017.
- A HISTÓRIA dos Consoles..., **Fiksussa**. Disponível em: <<http://fiksussa.blogspot.com.br/2012/01/fik-gamer-1-historia-dos-consoles.html>>. Acessado em 02 de Agosto de 2017. 2012.
- FINOCCHIO JUNIOR, J.; **Project Model Canvas: Gerenciamento de Projetos sem Burocracia**. 1. ed. Rio de Janeiro: Elsevier, 2013.
- Godot Game Engine. **Godot Game Engine a platform 2D and 3D open source game engine**. Disponível em:< <https://godotengine.org/>>. Acesso em: 31 ago. 2017.
- HUIZINGA, J. **Homo ludens: o jogo como elemento da cultura**. São Paulo: Perspectiva,

p. 2-3. 2001.

HUNICKE, R.; LEBLANC, M.; ZUBEK, R. MDA: A Formal Approach to Game Design and Game Research. In: **AAAI Workshop - Technical Report**. Research Gate: Berlin. 2004.

LARMAN, C. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientado a objetos e ao processo unificado, 2. ed. Porto Alegre: Bookman, 2004.

LIMA, A. S. **UML 2.0**: do requisito à solução, 4. ed. São Paulo: Érica. 2009.

LMMS. **Let's Make Music**. Disponível em: <<https://lmms.io/>>. Acesso em: 31 ago. 2017.

MARINS, V.; HAUGUENAUER, C. J.; CUNHA, G. Imersão e interatividade em Ambientes Virtuais de Aprendizagem para Educação a Distância, com Uso de Games e Realidade Virtual. In: *Revista AnimaEco*, Rio de Janeiro, v. 3, p. 40-53, 2012. Disponível em: <<http://www.latec.ufrj.br/revistas/index.php?journal=animaeco&page=article&op=view&path%5B%5D=9&path%5B%5D=320>>. Acesso em: 21 ago, 2017.

MICHAEL, D.; CHEN, S. **Serious Games**: Games That Educate, Train, and Inform. Thomson Course Technology: Boston, 2006.

OpenGameArt. **Space Shooter Kit**, 2017. Disponível em: <<https://opengameart.org/content/space-shooter-art>>. Acesso em: 31 ago. 2017.

PRESSMAN, R. S. **Engenharia de Software**: Uma abordagem Profissional 7 ed., Porto Alegre: AMGH, 2011.

REINOSO, L. F., et al. Childhood Fears. In: **Games of Global Game Jam**, Califórnia, 2015. Disponível em: <<https://globalgamejam.org/2015/games/childhood-fears>>. Acesso em: 08 ago. 2017.

REINOSO, L. F., MOURA, C. S. **Buga! A aventura de um Neandertal**: Uma aplicação interativa como recurso pedagógico para aprendizagem de história. In: V Congresso Brasileiro de Informática na Educação, 2016, Uberlândia, Anais..., Uberlândia: Minas Gerais, 1. ed. p.395-398, 2016.

REINOSO, L. F., NETO, C. B., LOPES, L. C. L. Principais Características dos Games para serem Inseridos como Ferramenta Educacional. In: VI Congresso Sul Brasileiro de Informática, 2012, Criciúma, Anais..., Criciúma: Santa Catarina, 2012.

ROBERTS, J. C.; HEADLEAND, C. J.; RITSOS, P. D. **Five Design-Sheets**: Creative Design and Sketching for Computing and Visualisation, Switzerland: Springer International Publishing, p. 27-41, 2017.

SAVI, R.; ULBRICHT, V. R. Jogos digitais educacionais: benefícios e desafios, **Revista**

Novas Tecnologias na educação (RENOTE), Porto Alegre, v. 6, n. 1, 2008. Disponível em: <<http://seer.ufrgs.br/index.php/renote/article/view/14405/8310>>. Acesso em: 08 ago. 2017.

SCHUYTEMA, P. **Design de games**: uma abordagem prática. Tradução de Cláudia Mello Belhassof. São Paulo: Cengage Learning. 2008. 447 p.

SOMMERVILLE, I. **Engenharia de Software**, 8 ed., São Paulo: Pearson Assison-Wesley. 2007.

TechTudo. **Johnny Castaway Screensaver**, 2010. Disponível em: <<http://www.techtudo.com.br/tudo-sobre/johnny-castaway-screen-saver.html>>. Acesso em: 02 ago. 2017.

THOMPSON, J., BERBANK-GREEN, B., CUSWORTH, N. **Game Design**: Principles, Practice, and Techniques - The Ultimate Guide for the Aspiring Game Designer, 1 ed. New Jersey: Wiley, p. 11-12. 2007.

WINN, B. M. The Design, Play, and Experience Framework. In: RICHARD, E. F.; **Handbook of Research on Effective Electronic Gaming in Education**. New York: Information Science Reference, p. 1010-1024, 2008.

WORTMANN, P. **Buttons for touch screen display**, 2017. Disponível em: <<https://open-gameart.org/content/buttons-for-touch-screen-display>>. Acesso em: 31 ago. 2017.

Jogos digitais: princípios, conceitos e práticas

Luiz Fernando Reinoso, Giovany Frossard Teixeira e Renan Osório Rios

Agosto, 2017

Esta produção destina-se a estudantes, professores e profissionais que buscam compreensão e técnicas para desenvolver jogos eletrônicos, sejam para produção independente ou capacitação profissional. A obra reúne conceitos, metodologias, princípios práticos e técnicas para criação de jogos, sendo trabalhados dois projetos reais, ambos com uso de softwares livres. O estudo compreende: designer de jogos, composição de um projeto de jogo, metodologias de desenvolvimento como o desenvolvimento ágil de software e o Processo de Software Pessoal (PSP), bem como suas técnicas. Apresenta também frameworks de designer de jogos, como o MDA (Mecânica, Dinâmica e estética) e o Design/Play/Experience, estudo sobre arquitetura de jogos digitais e sobre atmosfera musical e jogos mobile. Como fundamentação e capacitação prática, os projetos desenvolvidos utilizam a Godot Game Engine como motor gráfico para criação de jogos, o Gimp para tratamento e editoração de imagens e o Inkscape para produção vetorial. Ademais, o estudante irá passar pelo estudo de designer com exercícios de produção gráfica, estudando acerca de pixel arte e arte vetorial e aprendendo a dar vida e forma a suas produções em cima dos conceitos apresentados e desenvolvidos.

Palavras-chave: Jogos digitais, designer de jogos, games, entretenimento.

Imagens utilizadas na capa são de domínio público.
Disponível em <http://opengameart.org/users/kenney>.