UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE ARQUITETURA, ARTES E COMUNICAÇÃO - CAMPUS BAURU
DEPARTAMENTO DE DESIGN
BACHARELADO EM DESIGN

ROCCO DE OLIVEIRA CAVENAGHI

OFICINA LÖVE

BAURU
Fevereiro/2021

ROCCO DE OLIVEIRA CAVENAGHI

OFICINA LÖVE

Trabalho de Conclusão de Curso do Curso de Design da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Arquitetura, Artes e Comunicação, Campus Bauru.
Orientador: Prof. Dr. Dorival Campos Rossi

BAURU
Fevereiro/2021

Rocco de Oliveira Cavenaghi Oficina LÖVE/ Rocco de Oliveira Cavenaghi. – Bauru, Fevereiro/2021- 41 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Dr. Dorival Campos Rossi
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Arquitetura, Artes e Comunicação
Design, Fevereiro/2021.
1. Design de Jogos 2. Ensino de Programação 3. Löve2d 4. Lua 5. Scratch

Rocco de Oliveira Cavenaghi

Oficina LÖVE

Trabalho de Conclusão de Curso do Curso de Design da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Arquitetura, Artes e Comunicação, Campus Bauru.

Banca Examinadora

Prof. Dr. Dorival Campos Rossi
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Arquitetura, Artes e Comunicação
Departamento de Design

Guilherme Cardoso Contini
Mestrando em Mídia e Tecnologia pelo PPGMIT
FAAC/UNESP - Bauru

Nicholas Bruggner Grassi
Doutorando em Mídia e Tecnologia pelo PPGMIT
FAAC/UNESP - Bauru

Bauru, 26 de fevereiro de 2021.

Dedico este trabalho aos meus professores. Foi apenas através deles que pude chegar até aqui, e a eles sou muito grato.

Agradecimentos

Tenho muito a agradecer a todas as pessoas que me ajudaram no decorrer deste trabalho, que é fruto de uma sequência de experiências pessoais e acasos felizes que ocorreram ao longo dos dois últimos anos.

Agradeço aos meus amigos, as maiores constantes da minha vida; a Prof. Ana Beatriz Pereira de Andrade, que apesar de não ser a orientadora deste trabalho, muito me orientou; ao Sergio Komori, pela força e instrução quando eu estava começando a programar; ao Guilherme Cardoso Contini, pelas conversas esclarecedoras; e aos meus pais, que sempre me apoiaram na minha busca pelos meus sonhos.

Gostaria, também, de registrar um agradecimento especial ao meu orientador, Prof. Dorival Campos Rossi, sem o qual a realização deste trabalho não seria possível; e aos demais participantes da banca examinadora.

Resumo

Oficina LÖVE foi uma oficina online de desenvolvimento de jogos digitais, ministrada na UNESP-Bauru no segundo semestre de 2020. O foco da oficina foi o ensino de conceitos centrais de programação de jogos para designers, através da realização de um jogo "Pong"com a framework de desenvolvimento de jogos Löve2d.
Palavras-chave: Design de Jogos, Ensino de Programação, Löve2d, Lua, Scratch.

Abstract

Oficina LÖVE was an online game development workshop taught at UNESP-Bauru in the second semester of 2020. The workshop was aimed at the teaching of core game programming concepts to designers, through the completion of a "Pong" game with the game development framework Löve2d. **Keywords:** Game Design, Programming Education, Löve2d, Lua, Scratch.

Lista de figuras

Lista de abreviaturas e siglas

GUI	Graphic User Interface
IDE	Integrated Development Environment
MIT	Massachusetts Institute of Technology

Sumário

1 Introdução

Os Jogos sempre foram uma parte integral de todas as culturas humanas. Muitos dos que jogamos ainda hoje, como Xadrez, Mahjong e Go, são resultados de um design folclórico que vem os transformando desde os tempos antigos. Embora sempre estivessem presentes, foi apenas no fim do século XX que o interesse acadêmico pelos jogos começou a aumentar, em consequência da evolução das tecnologias informáticas que permitiram a concepção dos primeiros jogos digitais como conhecemos hoje. Não mais limitados a campos como antropologia e história, os jogos como objetos de estudo se popularizaram nas mais diversas áreas, dentre elas o design (FULLERTON; SWAIN; HOFFMAN, 2004).

Veículos de uma nova cultura emergente, os jogos digitais amadureceram rapidamente no decorrer das últimas décadas, e hoje rivalizam com outras formas populares de entretenimento como a televisão. Na medida que a indústria dos jogos digitais cresceu em proporções bilionárias, a demanda por campos de estudos especializados aumentou, e cada vez mais pessoas consideram o design de jogos como uma possível carreira profissional. Contudo, a disponibilidade de formações específicas ainda é pequena no Brasil, e opções mais consolidadas como o design, em especial a sua vertente gráfica, acabam se tornando polos para indivíduos com afinidades criativas e artísticas que desejam poder aprender a criar seus próprios jogos. Infelizmente, o meio acadêmico do design é, no geral, pouco preparado para atender a essa demanda. A natureza transdisciplinar da ciência de criação de jogos digitais requer que qualquer aplicação prática seja pautada em uma diversa gama de habilidades e ferramentas, que muitas vezes requerem, por si só, estudos específicos.

Devido à ausência de uma estrutura acadêmica que facilite o acesso dos alunos de design à disciplina de criação de jogos digitais, é comum que os estudantes decidam buscar seus próprios meios para adquirir esses conhecimentos. A fim de superar as dificuldades inatas do processo de criação, uma alternativa recorrente é buscar colegas de outros cursos (em especial cursos da área da computação) para que se criem grupos de estudo e desenvolvimento de projetos. Através da aplicação das habilidades individuais de cada integrante, o produto final toma forma, resultado de um esforço coletivo. A universidade é um ambiente ideal para essa colaboração, pois favorece o encontro destas pessoas.

Uma configuração bastante observada por mim, e da qual já fiz parte, é a união de alunos de design gráfico e ciência da computação (ou sistemas de informação). Como designers, era esperado que nos responsabilizássemos principalmente pelas tarefas ligadas ao campo artístico: criação de concept arts, produção de sprite sheets, texturas, modelos 3D, artes finais e animações. Aos programadores, como costumávamos chamá-los, ficava o encargo de implementar todas essas ideias e recursos em um produto real e interativo, um jogo de fato.

Enquanto artistas e programadores, era possível que trabalhássemos separados e encapsulados dentro de nossas próprias funções, mas fazer um jogo digital muitas vezes requer que sejamos mais que isso. Como designers de jogos, saber apenas ilustrar ou apenas programar em um contexto isolado não era o suficiente. Era necessário que nos estendêssemos para além dos limites dessas habilidades para entender, em um contexto maior, como essas diferentes áreas interagiam entre si para criar o Jogo. Era apenas através da dialogação constante entre os integrantes do grupo que elementos cruciais como gameplay, interface de usuário e level design, para citar alguns, podiam ser idealizados com sucesso.

Ao decidirmos realizar nosso primeiro projeto individual, a lacuna em nossas habilidades, antes preenchida por outras pessoas, fica mais aparente que nunca, e compreender todas as etapas do processo passa a ser uma necessidade crucial. Longe de ser um inconveniente, este exercício permite que nossas perspectivas sejam ampliadas, nos tornando mais capazes como criadores de jogos, indiferente de quaisquer funções específicas que possamos vir a desempenhar quando parte de uma equipe profissional.

Como designers, criar nosso primeiro jogo de maneira individual pode ser um empreendimento intimidador ou confuso. É difícil saber por onde começar, o que aprender e quais ferramentas utilizar. Por isto idealizei a oficina LÖVE, que é um compilado dos estudos e experiências pessoais que vivenciei ao longo do meu trajeto, e que tem a finalidade de facilitar o acesso de designers (em específico aqueles que tiveram pouco contato com programação) à prática intelectual e lúdica do desenvolvimento de jogos digitais, de maneira sucinta e objetiva, por meio da realização de um jogo "Pong" com a framework de desenvolvimento de jogos Löve2d.

2 Princípio e problemática

Fundamentalmente, não é possível que exista um jogo sem haver, antes, uma metodologia de design que o concebeu. A metodologia guia o processo criativo do designer, que é responsável por idealizar as regras do jogo e garantir que elas sejam justas, divertidas e à prova de falhas.

Da mesma forma que não existe jogo sem design, não é possível que exista um jogo digital sem programação. Enquanto o design é responsável por desenvolver o jogo em um âmbito abstrato, assim como estipular o entrosamento dos sistemas dinâmicos que o formam em um nível estrutural, a programação o concretiza, permitindo que o jogo exista em uma realidade virtualmente construída. Design e programação se misturam, pois é fundamental que as lógicas implementadas pelo programador reflitam de maneira fiel os princípios idealizados pelo processo de design.

De maneira geral, o designer de jogos não é necessariamente uma pessoa com experiência formal na área de design. Uma pessoa pode se tornar designer de jogos partindo de diversos campos diferentes (história, antropologia, psicologia, ciência da computação, design, entre outros), mas é preciso distinguir que, para atingir os seus objetivos, este trabalho foca em indivíduos previamente contextualizados na área do design formal e sem qualquer experiência na área de programação.

A ausência de uma base teórica em práticas de programação é, portanto, a maior problemática que precisa ser solucionada para que o indivíduo consiga realizar com sucesso seu primeiro jogo digital de maneira completamente individual. A aquisição deste conhecimento é relevante para todos que desejam seguir carreira na área de design de jogos, pois dará ao indivíduo maior independência, e fará com que este esteja mais preparado para lidar com a realidade naturalmente transdisciplinar da área.

Programadores iniciantes sofrem com uma variada gama de dificuldades, e aprender a programar é uma tarefa considerada difícil pela maioria. Esse trabalho não se propõe a ser uma substituição para meios de aprendizado formais de programação, visto que estes são muito mais complexos e compreensivos em sua totalidade, mas, sim, habilitar os indivíduos para utilizarem as ferramentas necessárias para a execução do projeto proposto, em específico a framework de desenvolvimento de jogos Löve2d e a linguagem de programação Lua, e em um curto período de tempo. Fica a critério dos participantes a necessidade de expandir seus conhecimentos na área através de fontes paralelas.

2.1 O design na programação

O design é parte intrínseca da ciência da computação. Encontra-se nos fundamentos das linguagens de programação, na concepção de arquiteturas e sistemas, na criação de interfaces de usuário e muitas outras áreas. O bom programador deve ser, antes de tudo, um bom designer, ou seja, um solucionador de problemas. Um problema, no contexto da computação, pode ser entendido como um programa ou até mesmo um jogo. Como podemos criar sistemas interativos, através dos quais o usuário possa realizar tarefas como jogar, desenhar ou ouvir uma música?

Embora extremamente engenhosas, as linguagens de programação são apenas meras ferramentas. Elas "traduzem" as instruções dadas pelo programador para uma "linguagem de máquina". Em outras palavras, elas são como pontes que conectam o homem ao computador; mas cabe ao programador, aqui desempenhando seu papel como designer, inventar as estratégias necessárias para resolver os problemas existentes.

Um fenômeno fundamental à ciência da computação, e que auxilia no processo de idealização de soluções, é a abstração. Abstrair problemas significa compartimentaliza-los em partes menores, para que suas respectivas soluções também se simplifiquem e possam ser reutilizadas de maneira modular. Imagine como seria a nossa vida se tivéssemos que reinventar a eletricidade todas as vezes que fossemos acender uma lâmpada. Eu tenho certeza que viveria toda minha vida à luz de velas, pois não posso dizer que compreendo como a eletricidade funciona. Felizmente eu não preciso desse conhecimento para apertar um interruptor, pois esta é uma solução facilmente disponível e tão abstraída que nem mesmo eu, que não possuo nenhum conhecimento técnico, poderia falhar em aplicá-la.

Linguagens de programação são um exemplo clássico de abstração. A única linguagem que máquinas são efetivamente capazes de compreender é o binário (presença ou ausência de eletricidade, zero e um), também conhecido como código de máquina. Programar em binário é, no entanto, uma tarefa extremamente exaustiva e propensa a erros. Buscando resolver este problema, foram criadas as linguagens *assembly*, que eram significativamente mais legíveis e fáceis de utilizar. Após terminar de escrever uma série de instruções em *assembly*, o programador acionaria um programa chamado *assembler*, que então traduziria essas instruções para código de máquina, permitindo que o computador fosse capaz de executá-las.

Mas mesmo as linguagens de baixo nível como as linguagens *assembly*, que já possuem certo grau de abstração quando comparadas com o código de máquina, são complicadas. Subsequentes abstrações foram criadas a partir delas, resultando nas linguagens de programação como conhecemos hoje. Lua, a linguagem utilizada para a realização do projeto final desta oficina, é considerada uma linguagem de alto nível, pois é extremamente abstraída e legível. Através de um processo complexo que utiliza um compilador e um programa intérprete, ambos escritos na linguagem C (que acrescenta mais uma camada de abstração), Lua é traduzida

para código de máquina e finalmente executada pelo computador.

No entanto, não é necessário compreender todos os processos inferiores que permitem o funcionamento de Lua para utilizá-la com sucesso, e esse é o principal benefício da abstração, que nos permite focar no design das soluções que desejamos implementar sem que nos preocupemos com as questões técnicas que poderiam ser, outrora, impedimentos.

Não existe maior prova do poder da abstração que a criação da Interface Gráfica de Usuário (GUI), criada a fim de acessibilizar o uso dos dispositivos eletrônicos, e a verdadeira chave para o sucesso massivo dos computadores pessoais e celulares que hoje são utilizados sem dificuldades pela maioria das pessoas, até mesmo aquelas que possuem pouco ou nenhum conhecimento técnico na área.

Como estudantes do design formal, pressupõe-se que tenhamos uma grande vantagem ao aprendermos a programar, pois já possuímos grande parte dos modelos mentais necessários para compreender os processos de design inerentes à programação. Mas isso não significa que a tarefa é de todo simples, pois existem outras dificuldades que também precisam ser consideradas.

3 O ensino da programação para iniciantes

Em uma análise geral, du Boulay (1986) descreve cinco domínios sobrepostos e potenciais fontes de dificuldades que precisam ser dominados para o aprendizado da programação. São eles:

a) **Orientação geral:** o que são programas e o que pode ser feito com eles;

b) **A máquina nocional:** um modelo do computador e como ele se relaciona com a execução dos programas;

c) **Notação:** a sintaxe e a semântica de uma linguagem de programação em particular;

d) **Estruturas:** blocos cognitivos utilizados no design e compreensão de programas;

e) **Pragmáticas:** as habilidades de planejar, desenvolver, testar, depurar, entre outros.

Nenhum desses problemas são separáveis dos outros em sua totalidade, e a maior parte do "choque" [...] dos primeiros encontros entre o aprendiz e o sistema são devido às suas tentativas de lidar com todas essas diferentes dificuldades de uma vez só. (BOULAY, 1986, p.58, tradução nossa)

Rogalski e Samurçay descrevem a tarefa de aprender a programar da seguinte maneira:

> Adquirir e desenvolver conhecimento sobre programação é um processo altamente complexo. Ele envolve uma variedade de atividades cognitivas, e representações mentais relacionadas ao design de programas, entendimento de programas, modificação e depuração (e documentação). Até mesmo no nível de compreensão da informática, ela requer a construção de conhecimentos conceituais, e a estruturação de operações básicas (como loops, afirmações condicionais, etc) em esquemas e planos. Ela requer o desenvolvimento de estratégias flexíveis o suficiente para auferir benefícios de auxílios de programação (ambientes de programação, métodos de programação). (ROGALSKI; SAMURÇAY, 1990, p.170, tradução nossa)

Green (1990, p.170) sugere que a programação não deve ser vista como uma "transcrição de uma representação interna", ou no contexto da "teoria pseudo-psicológica da 'programação estruturada' ", mas sim como processo exploratório no qual programas são criados "de maneira oportunista e em pequenos incrementos". Uma conclusão similar é tida por por Davies:

> ... Modelos emergentes de comportamentos de programação indicam um processo incremental orientado à resolução de problemas, onde a estratégia é determinada por episódios localizados de resolução de problemas e reavaliação frequente de problemas. (DAVIES, 1993, p.265, tradução nossa)

Portanto, é necessário dar ênfase a processos exploratórios e oportunistas no que se diz respeito ao ensino de programação para iniciantes . Isso significa que técnicas específicas

e recursos de uma determinada linguagem de programação devem ser ensinados na medida do necessário, conforme o estudante avance na resolução dos problemas propostos, a fim de proporcionar um ensino gradual que respeite o processo natural de aprendizagem dos estudantes. Felizmente, o desenvolvimento de jogos é um contexto ideal, pois o jogo se complexifica proporcionalmente ao nível do conhecimento técnico necessário para concretizá-lo através da programação.

3.1 Modelos mentais

Escrever programas requer a estruturação de diversos modelos mentais que vão além daqueles necessários para compreender uma linguagem de programação específica ou técnicas gerais da ciência da computação. Programas são geralmente escritos para um propósito, e entender este propósito é necessário para que as soluções adequadas sejam devidamente aplicadas através da programação. Portanto, a criação de um modelo mental do problema em questão deve preceder qualquer tentativa de escrever um programa que, de fato, o solucione.

Outros modelos mentais importantes podem ser identificados. Muitos estudos apontam para a existência de um papel central desempenhado por um modelo abstraído do computador, por vezes chamado de "máquina nocional" (CANAS; BAJO; GONZALVO, 1994; BOULAY, 1986; BOULAY; O'SHEA; MONK, 1981; HOC; NGUYEN-XUAN, 1990; MENDELSOHN; GREEN; BRNA, 1990).

O propósito da máquina nocional é prover os fundamentos necessários para que o estudante compreenda os comportamentos dos programas que ele está aprendendo a escrever. A máquina nocional é definida de acordo com a linguagem de programação utilizada e suas características variam de linguagem para linguagem. Para ser útil, ela deve ser simples, concreta e observável, uma "caixa de vidro" e não uma "caixa-preta".

No contexto da oficina LÖVE, a máquina nocional pode ser compreendida como uma mistura das características da linguagem de programação Lua e da framework de desenvolvimento de jogos Löve2d. Além de ensinar a sintaxe e a técnica necessária para o uso destas ferramentas, é necessário expor aos estudantes como elas funcionam "por trás dos panos".

3.2 As dificuldades de programadores iniciantes

Muitos estudos focados no ensino de programação concordam que existe um conjunto de falhas comuns à maioria dos programadores iniciantes. Winslow (1996) fez uma revisão destes estudos e concluiu que iniciantes: são limitados a conhecimentos superficiais (e organizam conhecimentos baseados em similaridades superficiais); falta-lhes os modelos mentais com maior riqueza de detalhes; falham ao aplicar conhecimentos relevantes; usam estratégias gerais de resolução de problemas (no lugar de estratégias orientadas ao problema em questão ou à

programação); e possuem uma abordagem "linha à linha" quanto à programação, ao invés de considerar os blocos significativos do programa e suas estruturas.

Ao contrário de programadores mais experientes, iniciantes passam muito pouco tempo planejando e testando códigos, e tentam corrigir erros a "nível local" ao invés de propor soluções que reformulam significativamente os programas (LINN; DALBEY, 1989). Eles são frequentemente ruins em rastrear e monitorar códigos (PERKINS et al., 1986), e possuem pouco domínio quanto a natureza sequencial da execução de um programa: "O que muitas vezes é esquecido é que cada instrução opera no ambiente criado pelas instruções anteriores" (BOULAY, 1986, p.68, tradução nossa).

Além dessas dificuldades relacionadas à incapacidade dos iniciantes em compreender programas em seu âmbito total, outro problema frequente é a antropomorfização dos sistemas que executam os programas escritos. "A noção de que o sistema dá sentido ao programa de acordo com suas próprias regras muito rígidas é uma ideia crucial que o aluno precisa entender" (BOULAY, 1986, p.61, tradução nossa).

Isso significa que iniciantes muitas vezes falham em compreender que as interpretações que eles têm sobre a execução de um programa não é compartilhada pelo computador, levando à ocorrência de erros de execução. Isso não quer dizer que a interpretação inicial do aluno seja falha, apenas que ele não soube como traduzi-la para um conjunto de instruções que fosse lógico dentro do contexto rígido do sistema para o qual ele está escrevendo o programa.

> [Um ponto importante] é o grande número de estudos concluindo que os programadores iniciantes conhecem a sintaxe e a semântica das declarações individuais, mas não sabem como combinar essas características em programas válidos. Mesmo quando sabem como resolver os problemas à mão, têm dificuldade em traduzir a solução em um programa de computador equivalente.(WINSLOW, 1996, p.17, tradução nossa)

Em uma revisão de diversos estudos na área, Robins, Rountree e Rountree, concluem que "a causa subjacente aos problemas enfrentados pelos novatos é sua falta de (ou fragilidade) de conhecimentos e estratégias específicas de programação." (ROBINS; ROUNTREE; ROUNTREE, 2003, p. 153, tradução nossa), e que essa falta manifesta-se principalmente em falhas de planejamento e design, e não em equívocos relacionados a construções ou sintaxe de uma determinada língua de programação.

3.3 Conclusão

De acordo com o objetivo proposto por este projeto, o contexto do público para o qual ele se direciona, e as dificuldades previamente elencadas e discutidas, defino, enfim, o seguinte conjunto de tarefas a serem completadas durante a realização da oficina:

a) Ensino robusto de conceitos e estruturas gerais de programação, pensamento computacional e práticas de programação;

b) Desenvolvimento do conhecimento técnico necessário para a utilização e compreensão das ferramentas utilizadas (Lua e Löve2d);

c) Ensino de técnicas de identificação e resolução de problemas, seguindo diretrizes de design e programação, e orientadas para o desenvolvimento de jogos.

4 A oficina

4.1 *Pong*, e porque devemos começar pequeno

Como vimos anteriormente, são muitas as dificuldades que precisamos considerar quando formos realizar um primeiro projeto individual de desenvolvimento de jogos. É preciso escolher as ferramentas certas e aprender a utilizá-las de maneira eficiente. No entanto, existe um outro fator, que é muitas vezes negligenciado, que faz com que muitas pessoas falhem antes mesmo de começar: o escopo do projeto em si.

Jogos são programas complexos por natureza, e até mesmo o mais simples deles é compostos por diversas camadas lógicas complexas. Além das regras inerentes ao jogo, é preciso que se programem as estruturas gerais que permitem a execução do programa pela máquina, que se criem as ferramentas necessárias para a manipulação gráfica dos elementos na tela, que se estabeleça um canal de comunicação entre o programador e o computador, e muitas outras coisas.

Decerto que muitas das ferramentas disponíveis no mercado já implementam uma boa parte dessas lógicas para nós (e ainda bem, senão, aprender a fazer jogos seria ainda mais difícil), mas nem mesmo um motor de jogos poderoso como a Unreal Engine é capaz de programar um jogo sozinho. Cabe, portanto, a nós, designers do jogo, a implementação de todas as lógicas de programação que tornarão uma mera janela em branco em um jogo de verdade.

E é por isso que quando vamos realizar nosso primeiro jogo *solo*, é necessário começar com um projeto pequeno. Infelizmente, é preciso muitos anos de prática e experiência para criar um jogo como "Skyrim" ou "World of Warcraft", não importa o quão motivados estejamos, ou quanto tempo livre estamos dispostos a gastar. As tarefas a serem completadas são muitas, e se nossas expectativas não forem realistas, estaremos apenas nos sabotando.

Quando estamos iniciando, nossos projetos não precisam ser perfeitos, eles precisam ser finalizados. É só quando criamos um jogo do início ao fim que ganhamos a experiência necessária para ter sucesso ao realizar projetos maiores.

Portanto, ao criar jogos, é imprescindível que os escopos dos projetos sejam rigidamente definidos, e que estes estejam de acordo com os objetivos propostos e com o nível técnico dos participantes envolvidos. Essa é a razão para o jogo *Pong* ter sido o escolhido para a oficina LÖVE. Embora aparente ser muito simples, a quantidade de "problemas" que precisam ser resolvidos são muitos, e conceitos muito cruciais à programação de jogos precisam ser aprendidos.

Além de um projeto bem planejado, as ferramentas que serão utilizadas precisam ser escolhidas com cautela. As soluções disponíveis são muitas, umas mais ou menos abstraídas que outras, sendo necessário uma avaliação cuidadosa dos prós e contras de cada uma delas.

4.2 Porque Löve2d?

Löve2d é uma framework para desenvolvimentos de jogos 2D em Lua. Uma framework é um conjunto de bibliotecas de código que simplificam o processo de desenvolvimento de um determinado programa. Diferente de motores de jogos (Unity, Godot, Unreal Engine, para citar alguns), frameworks são muito menos abstraídas, oferecendo apenas o mínimo necessário para a realização da tarefa em questão. Elas geralmente não possuem interface gráfica e nem mesmo um editor de nível. Isso significa que todo o trabalho de desenvolvimento é realizado através de programação textual em um editor de texto paralelo. A framework, que neste caso pode ser entendida como um programa, é apenas responsável por executar o jogo escrito pelo usuário.

Ao optarmos por utilizar uma framework ao invés de um motor de jogos, estamos trocando facilidade por flexibilidade. Além disso, devido à natureza pouco abstraída da framework, o processo de desenvolvimento é muito mais observável, pois menos coisas ocorrem "por trás dos panos". Isso é ideal para o processo de aprendizado que foi idealizado para a oficina LÖVE.

Löve2d possui muitas outras qualidades que a tornam uma excelente escolha. Ela é gratuita, open-source e pode ser executada em diversas plataformas diferentes como Windows, Linux, Mac OX, Android, iOS e até mesmo em portáteis como o 3DS e em navegadores da web. Sua comunidade é ativa e a sua documentação é extensa, garantindo que o usuário sempre encontre suporte quando necessário.

Outras ferramentas foram cogitadas durante a conceitualização da oficina, em específico os motores de jogos Unity, Godot e Game Maker Studio 2, que possuem maior relevância no mercado quando comparados com Löve2d. Godot, que também é gratuita e open-source, quase foi a escolhida, mas, por questões de didática, optei pela ferramenta menos abstraída. A exposição extra a conceitos básicos de programação garante que os conhecimentos sejam melhor fixados pelos estudantes. Além disso, a mudança de um ambiente menos abstraído para um mais abstraído não comporta grandes dificuldades. O mesmo não pode ser dito para mudanças no sentido contrário.

Por fim, a linguagem de programação utilizada em Löve2d, Lua, é legível, amigável e possui uma curva de aprendizagem suave. Seu uso é bastante difundido, especialmente no mercado de jogos digitais, onde é frequentemente utilizada como uma linguagem de *scripting* tanto por programadores como por designers.

> Lua tornou-se uma linguagem de programação extremamente popular, de tal forma a atingir uma massa crítica de desenvolvedores na indústria de jogos,

> o que significa que as habilidades de Lua são transferíveis de empresa para empresa. Isso se deve em parte à sua velocidade e à facilidade com que os desenvolvedores podem incorporar Lua em um motor de jogos. (WHERE..., 2013, tradução nossa)

4.3 Estrutura geral e conteúdos abordados

A oficina de desenvolvimento de jogos digitais foi realizada no segundo semestre de 2020, dentro da disciplina optativa "Introdução ao desenvolvimento de Jogos Digitais - Game Design" do curso de design da Unesp/Bauru, sob a orientação do professor docente, Dr. Dorival Campos Rossi, que também orientou este trabalho. Devido à situação da pandemia de COVID-19, as aulas foram ministradas em caráter estritamente online, com suporte de ferramentas como o Google Meets e o Google Drive.

Em respeito à realidade individual de cada estudante durante a pandemia, e à natureza livre da disciplina em questão, a participação da oficina não foi obrigatória, e atividades paralelas poderiam ser desenvolvidas desde que isto fosse previamente acordado entre o aluno e o professor responsável. Em qualquer momento os alunos poderiam escolher abandonar as atividades da oficina para desenvolver uma outra atividade relevante à disciplina, caso assim desejassem.

No segundo semestre de 2020 as aulas onlines do curso de design ainda operavam em caráter provisório, o que significa que muitos alunos tinham irregularidades em suas grades que precisariam ser resolvidas ao longo do semestre. Portanto, a pedido do Prof. Dr. Dorival, o cronograma da oficina foi elaborado com intuito de realizar aulas espaçadas e com término previsto para o fim de novembro. Assim, foi-se estipulado que as aulas da oficina ocorreriam ao longo de nove encontros síncronos, entre os meses de setembro e novembro de 2020, separadas em três etapas diferentes.

Todos os encontros foram gravados através do Google Meets e disponibilizados posteriormente no Google Drive, em um drive específico da disciplina, para que todos os alunos tivessem acesso ao conteúdo da aula caso não conseguissem participar dos encontros. Os materiais de apoio (slides), também foram disponibilizados, e um grupo de WhatsApp foi criado para que os alunos pudessem manter contato com o Prof. Dorival e comigo fora do ambiente da sala de aula digital.

4.3.1 Primeira etapa: Scratch

Dentre as diversas dificuldades elencadas no capítulo primeiro, a primeira que me propus a enfrentar ao idealizar o programa da oficina foi o ensino da programação.

> A teoria atual sugere um foco não no ensino do instrutor, mas no aprendizado do aluno, e na comunicação efetiva entre professor e aluno. O objetivo

> é fomentar o aprendizado "profundo"de princípios e habilidades, e criar aprendizes para a vida toda, independentes e reflexivos. Os métodos envolvem metas e objetivos claramente definidos, estimulando o interesse e envolvimento dos alunos com o curso, envolvendo ativamente os alunos com o material do curso, e avaliação e feedback apropriados. (ROBINS; ROUNTREE; ROUNTREE, 2003, p.157, tradução nossa)

Linn e Dalbey (1989), citados por Robins, Rountree e Rountree (2003, p. 157) propõem uma "cadeia de realizações cognitivas" que deve acontecer através do bom ensino da programação. Esta cadeia começa com as características da linguagem que está sendo ensinada (sintaxe e semântica). O segundo elo são as habilidades de design, incluindo modelos e estruturas gerais, e as habilidades procedurais de planejamento, teste e reformulação de código. O terceiro elo é a habilidade de resolver problemas, conhecimentos e estratégias (incluindo o uso das habilidades procedurais) abstraídos da linguagem específica ensinada e que podem ser aplicados a novas linguagens e situações. Esta cadeia de realizações forma um bom resumo do que poderia ser entendido como aprendizado profundo da programação introdutória.

No entanto, este projeto não foi contextualizado como uma aula formal de introdução à ciência da computação, mas como uma introdução à prática de desenvolvimento de jogos digitais direcionada a designers. Dessa forma, eu optei por alterar a fórmula desta cadeia, vinculando cada uma das etapas a marcos específicos relacionados ao desenvolvimento de jogos, e me aproveitando do conhecimento prévio dos participantes nas teorias do design.

Ao invés de começarmos com aulas direcionadas à sintaxe e semântica da linguagem de programação Lua, representados pelo primeiro elo da cadeia, escolhi iniciar a oficina pelo segundo elo, expondo os alunos imediatamente às estruturas gerais da programação, por meio de uma abordagem criativa que utiliza a linguagem de programação Scratch, desenvolvida pelo MIT.

De acordo com o "Currículo de Computação Criativa" (CREATIVE..., 2021), desenvolvido pela Harvard School of Education, ciência da computação e campos relacionados são, há muito tempo, introduzidos às pessoas de uma forma desconectada de seus interesses e valores - enfatizando detalhes técnicos ao invés do potencial criativo. A computação criativa apoia o desenvolvimento de conexões pessoais com a computação, aproveitando a criatividade, imaginação, e interesses de cada indivíduo, e enfatizando seus papéis como designers e criadores, e não como consumidores passivos.

Em Scratch, o usuário cria seus próprios programas com "blocos de comando" (Figura 1), que se encaixam uns aos outros como bloquinhos de construção. Através de atividades expositivas, é possível introduzir os alunos aos conceitos chaves da programação (sequenciamento, *loops*, paralelismo, eventos, condicionais, operadores, dados) e às práticas de programação (experimentação, *debugging*, reutilização e remixagem, abstração e modularização). Esses conceitos foram então resgatados nas etapas seguintes, quando os alunis passaram para um ambiente de programação textual com a linguagem de programação Lua.

A primeira etapa ocorreu no decorrer de quatro encontros e foi estruturada da seguinte maneira:

a) **Primeiro encontro:** apresentação da oficina e discussão com temática livre;

b) **Segundo encontro:** aula teórica - Scratch;

c) **Terceiro encontro (presença facultativa):** atendimento da atividade proposta;

d) **Quarto encontro:** apresentação e discussão da atividade.

Figura 1 – Exemplo de um programa em Scratch

Fonte: Elaborada pelo autor.

A aula teórica teve caráter majoritariamente expositivo. Conceitos básicos da ciência da computação foram brevemente discutidos, e então dez programas foram construídos em Scratch e comentados em tempo real, introduzindo os conceitos necessários de forma gradual. Os alunos foram estimulados a criarem os programas em seus próprios computadores. Fiz algumas perguntas aos alunos com o propósito de levar a atenção deles a pequenos detalhes ou possíveis bugs pouco aparentes.

Os programas desenvolvidos, e os conceitos que eles introduzem, são, resumidamente:

a) **Hello, World:** pontos de entrada, impressão de *Strings*;

b) **Hello, Rocco:** concatenação de *Strings*;

c) **Scratch miando:** *loops* infinitos, funções de espera, tocando sons;

d) **Scratch segue o mouse:** manipulação e movimentação de sprites;

e) **Scratch conta até o infinito:** variáveis;

f) **Scratch miando quando o mouse encosta nele (e variações):** condições (*if/else*), colisão com sprites;

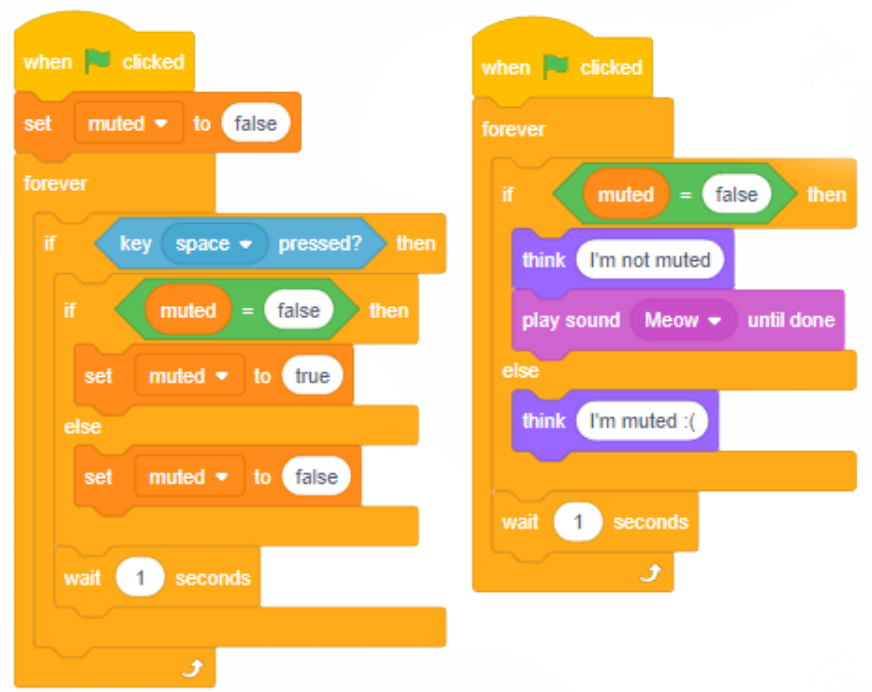
g) **Scratch caminhando:** animação de sprites;

h) **Silenciando Scratch com a barra de espaço:** *inputs* de usuário, paralelismo;

i) **Marco Polo:** eventos;

j) **Scratch tossindo:** funções, modularização, design.

Figura 2 – Silenciando Scratch com a barra de espaço



Fonte: Elaborada pelo autor.

Um material de apoio foi desenvolvido através de slides para facilitar a compreensão de determinados conceitos. No fim da aula, foi sugerido aos alunos que criassem um jogo com tema livre usando a linguagem de programação Scratch, para que concretizassem o conhecimento adquirido com criatividade e liberdade.

4.3.2 Segunda etapa: Lua

Na segunda etapa, os alunos foram introduzidos a um ambiente de programação textual com a linguagem de programação Lua. Antes de tudo foi preciso garantir que todos conseguiram instalar e configurar as ferramentas necessárias em seus respectivos computadores: os binários de Lua e uma IDE adequada. Por razões de padronização, sugeri a utilização da ZeroBrane, uma IDE específica para desenvolvimento em Lua, ou então o Visual Studio Code.

Três encontros foram realizados nesta etapa, e foram estruturados da seguinte maneira:

a) **Primeiro encontro:** aula teórica - Lua I;

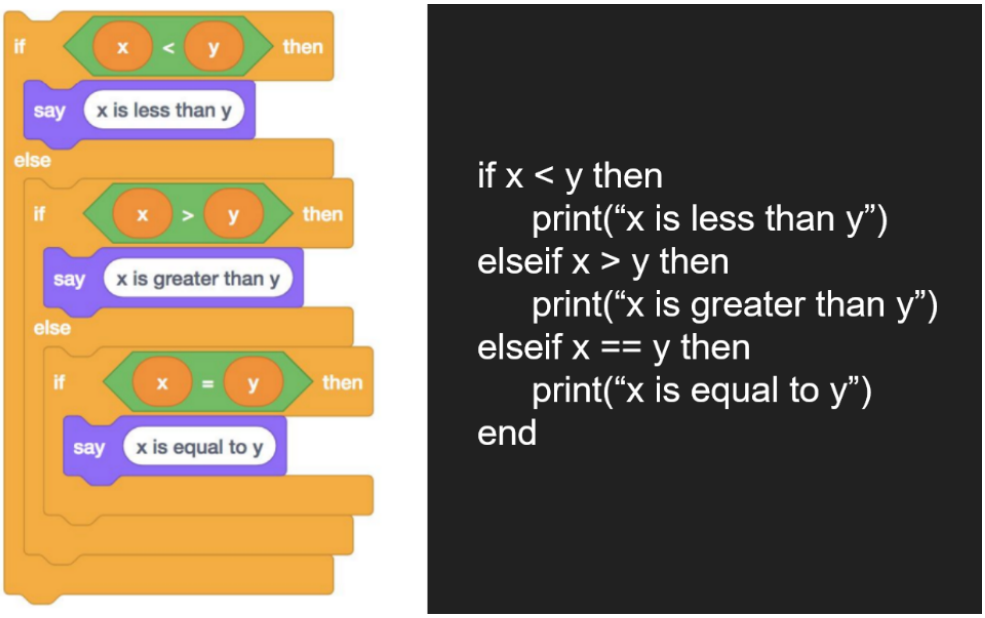
b) **Segundo encontro (presença facultativa):** atendimento da atividade;

c) **Terceiro encontro:** resolução comentada da atividade e aula teórica - Lua II.

O uso de material de apoio foi crucial nesta etapa. Programas previamente escritos por mim foram disponibilizados no drive da disciplina, e pedi para que os alunos os baixassem para que pudéssemos estudá-los durante a aula. Procurei utilizar exemplos dinâmicos e estimulei os alunos para que escrevessem e executassem os programas estudados em seus próprios computadores.

O objetivo principal era fazer com que os alunos traduzissem os conhecimentos aprendidos na etapa anterior para o novo ambiente de programação textual, a fim de estabelecer mais um elo na "cadeia de realizações cognitivas". Para isso, foram feitas comparações entre os blocos da linguagem de programação Scratch e as estruturas da linguagem de programação Lua(Figura 3). Nenhum conhecimento novo foi introduzido no primeiro encontro, mas a natureza complexa da programação textual faz com que conceitos já aprendidos sejam pouco reconhecíveis.

Figura 3 – Comparação entre blocos Scratch e Lua



Fonte: Elaborada pelo autor.

A atividade sugerida nesta etapa foi um desafio elaborado pelo professor David J. Malan para o curso de introdução à ciência da computação CS50x da universidade Harvard. O objetivo deste desafio é incentivar o uso dos conhecimentos aprendidos de uma maneira inovadora. Nele, os alunos devem implementar um conjunto de *for-loops* aninhados para imprimir uma pirâmide de hashtags no terminal. O uso de loops aninhados é uma prática extremamente comum no desenvolvimento de jogos.

Duas versões da atividade foram disponibilizadas, uma para aqueles que se consideram mais confortáveis com os conceitos explorados na aula, e uma para aqueles que se consideram menos confortáveis. A auto-avaliação é crucial para o bom desenvolvimento do aluno, que deve

estabelecer uma relação descontraída com o conteúdo sendo estudado, evitando frustrações desnecessárias.

Código 4.1 – Resolução do desafio Mario More

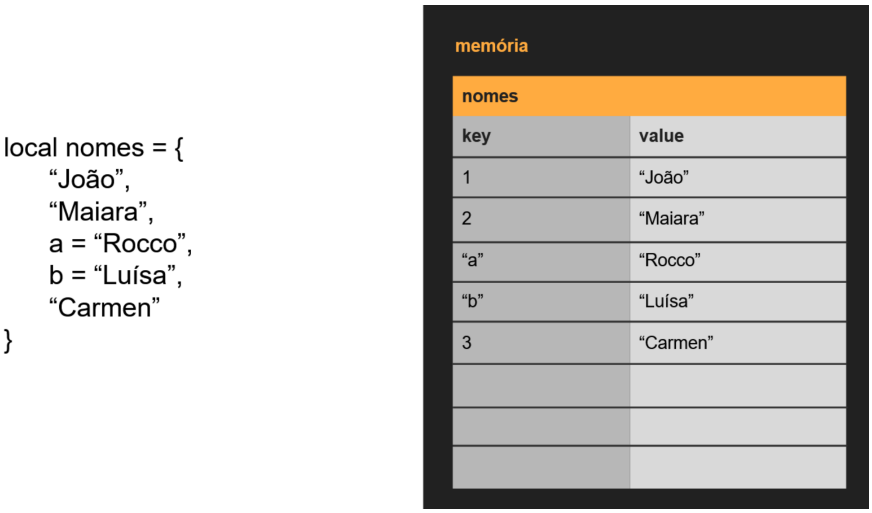
```
local h
repeat
  io.write("Altura: ")
  h= io.read("*n")
until h > 0 and h < 9

for i = 1, h do
  for j = 1, h do
    if j <= h - i then
      io.write(" ")
    else
      io.write("#")
    end
  end

  io.write("  ")

  for j = 1, i do
    if j == i then
      print("#")
    else
      io.write("#")
    end
  end
end
```

Na última aula teórica foi necessário introduzir alguns conceitos novos que ainda não tinham sido explorados em Scratch, em específico, estruturas de dados e ponteiros. A linguagem de programação Lua possui uma única estrutura de dados chamada *Table*, que funciona como uma matriz associativa, e seu uso é recorrente em todos os projetos feitos em Lua. Por isso, é muito importante que os alunos desenvolvam um modelo mental adequado que lhes permita visualizar e entender seu funcionamento.

Embora muito menos relevantes que *Tables* dentro do ambiente de programação Lua, ponteiros também precisaram ser mencionados, pois *Tables* em Lua são sempre passadas por referência (ou seja, através de ponteiros), o que pode levar a muitos bugs e diversas frustrações caso o programador não se atente a isso. Porém, não é necessário estender-se muito neste

assunto, visto que Lua não trabalha com a manipulação direta de ponteiros.

Figura 4 – Funcionamento de uma *Table* em Lua



Fonte: Elaborada pelo autor.

4.3.3 Terceira etapa: Löve2d

Na última etapa, os alunos foram introduzidos à framework de desenvolvimento de jogos Löve2d. Seguindo o modelo das etapas anteriores, as aulas desta etapa focaram na exposição das técnicas e conceitos necessários para o desenvolvimento de soluções dos problemas recorrentes na programação de jogos. Aqui que estabelecemos, por fim, o último elo da “cadeia de reações cognitivas” como idealizada por Linn e Dalbey.

No total, quatro encontros foram realizados nesta etapa, estruturados da seguinte forma:

a) **Primeiro encontro:** aula teórica - Introdução a framework de desenvolvimento Löve2d;

b) **Segundo encontro:** aula teórica - Pong I;

c) **Terceiro encontro:** aula teórica - Pong II;

d) **Quarto encontro:** apresentação da atividade final e finalização da oficina.

Esta etapa foi a que contou com a maior quantidade de aulas teóricas, devido à complexidade da ferramenta utilizada e da atividade proposta. No primeiro encontro, foi necessário garantir, em primeiro lugar, que todos os alunos conseguiram instalar a framework Löve2d em seus respectivos computadores. Então, foi necessário ensinar a estrutura básica de um projeto Löve2d e como poderíamos executar os jogos que estivéssemos desenvolvendo. Para facilitar este processo, preparei e disponibilizei aos alunos uma série de sete programas, que foram executados e comentados durante a aula.

Cada programa explorou uma funcionalidade diferente da framework, expandindo os conhecimentos adquiridos nas etapas anteriores e os consolidando em sistemas com complexidade gradual. As técnicas estudadas nesta aula podem ser entendidas como os verdadeiros "blocos de construção" que permitiram o desenvolvimento da atividade final. Os seguintes domínios e capacidades foram abordados:

a) Escrevendo na tela;

b) Desenhando formas na tela;

c) Desenhando sprites;

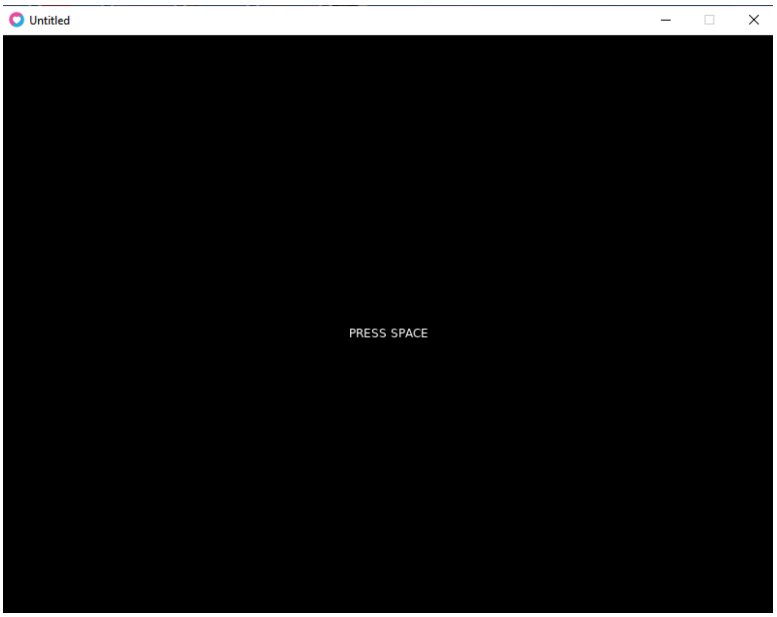
d) Captando inputs do teclado;

e) Movimentando um personagem;

f) Reproduzindo sons;

g) Manipulando fontes customizadas.

Figura 5 – Captando inputs do teclado



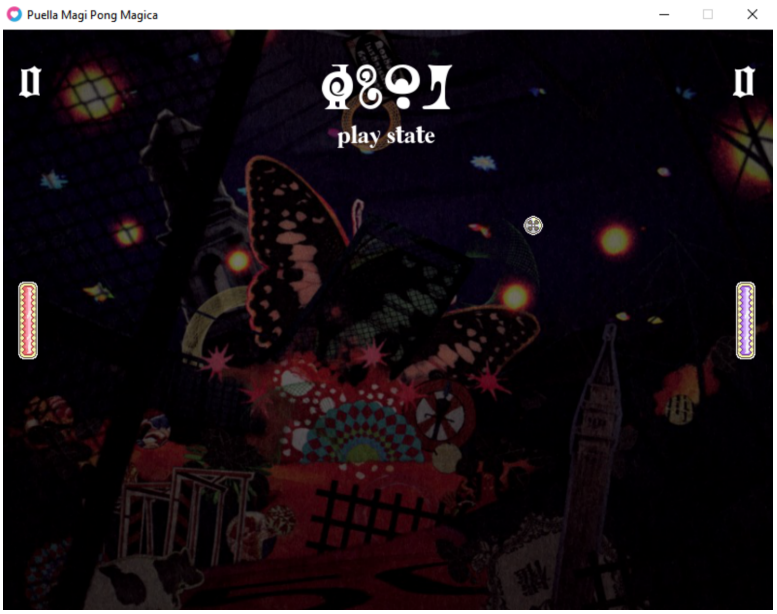
Fonte: Elaborada pelo autor.

No fim da primeira aula, os alunos foram orientados a prepararem os recursos gráficos que seriam utilizados em suas próprias atividades finais, e a desenvolver um "*mockup*" de seus jogos finalizados utilizando a framework Löve2d. Os alunos tiveram a liberdade de escolher qualquer tema que desejassem para a elaboração dos seus jogos.

Para a realização das duas últimas aulas teóricas, eu preparei um jogo *Pong* finalizado de minha autoria, mas o separei em diversos projetos menores, onde funcionalidades foram implementadas gradualmente. Pedi aos alunos que baixassem estes projetos para que eles pudessem acompanhar a explicação durante as aulas. "Pong-0", por exemplo, era apenas um

mockup estático, com os recursos renderizados na tela. Em "Pong-1", a movimentação das pás foi implementada. Em "Pong-2" introduzimos a movimentação da bola (Figura 6), e assim por diante, até "Pong-11", no qual o jogo estava finalizado. Conceitos básicos de programação orientada à objetos também foram estudados.

Figura 6 – Pong-2: movimento da bola



Fonte: Elaborada pelo autor.

Durante as aulas, os alunos foram estimulados a refletirem sobre as problemáticas existentes antes de serem expostos às suas soluções. Este tipo de exercício reforça as capacidades de reconhecimento de problemas dos alunos, permitindo que eles possam idealizar suas próprias soluções futuramente.

A última atividade sugerida da oficina foi a finalização individual dos jogos *Pong* e a adição de uma funcionalidade extra ao jogo finalizado, como um estado de pausa ou uma mecânica de jogo especial. O objetivo desta atividade é dar liberdade ao aluno para que ele possa colocar em prática tudo que aprendeu no decorrer da oficina, através da personalização de seu projeto final.

Assim como nas etapas anteriores, o último encontro desta etapa foi direcionado para a apresentação dos trabalhos desenvolvidos. Cada aluno apresentou e comentou sua atividade final. Após todas as apresentações, abriu-se espaço para uma última discussão, onde cada participante pôde comentar sua experiência geral com a oficina e sua avaliação dos conteúdos abordados.

4.4 O início e a expectativa dos participantes

Antes do início das aulas, um formulário de participação foi enviado aos alunos matriculados na disciplina de "Introdução ao desenvolvimento de Jogos Digitais - Game Design", para que se pudesse averiguar quantos deles tinham, de fato, interesse em participar das atividades da oficina. Junto com este formulário foi disponibilizado um cronograma discriminado das atividades que seriam oferecidas, para que os alunos pudessem avaliar se o conteúdo lhes interessava. Dez alunos de trinta se inscreveram.

O formulário também continha perguntas que buscavam sondar as expectativas dos alunos participantes quanto à oficina, os sistemas operacionais utilizados em seus computadores e suas percepções sobre a programação de jogos.

Oito alunos relataram que gostariam de aprender os conceitos básicos da programação de jogos digitais, um relatou que gostaria de conhecer a framework de desenvolvimentos de jogos Löve2d e um relatou não ter qualquer expectativa. Dois alunos alegaram possuir experiência prévia com a programação, e que não se intimidavam com a perspectiva de programar um jogo. Os demais alunos não possuíam qualquer experiência prévia na área, cinco alegaram que se intimidavam com a perspectiva de programar um jogo, e três não sabiam se se intimidavam ou não.

4.5 Avaliação geral das etapas

4.5.1 Primeira etapa

Figura 7 – Metier or Meteor

Fonte: Jogo feito por Letícia Mestre.

A primeira etapa da oficina ocorreu sem problemas. A taxa de participação foi adequada e os alunos permaneceram engajados durante todos os encontros. Nove alunos entregaram a atividade proposta e os resultados foram excepcionais: os jogos desenvolvidos (exemplo na Figura 7) foram complexos em sua maioria, indo muito além dos conceitos explorados durante as aulas, o que demonstra que os alunos adquiriram um excelente domínio da ferramenta e do material estudados.

Em uma avaliação posterior, a maior parte dos alunos relatou ter se sentido propriamente desafiada no decorrer da etapa.

4.5.2 Segunda etapa

Na segunda etapa foi possível observar uma queda significativa do engajamento e participação durante as aulas, e apenas dois alunos entregaram a atividade proposta, um número bem menor do que o esperado. Das duas atividades entregues, uma foi desenvolvida após extensa orientação individual, o que indica que o nível de dificuldade estava acima do ideal.

Em uma avaliação posterior, esta etapa foi a menos bem avaliada pelos alunos, e a baixa quantidade de entregas da atividade impede uma real avaliação do nível de compreensão dos conteúdos estudados. Uma das hipóteses levantadas por mim, que talvez explique este resultado bastante desanimador, é que esta etapa falhou em estimular o interesse e a imaginação dos alunos, tendo sido demasiadamente focada em aspectos teóricos da linguagem de programação Lua, e com uma atividade proposta pouco lúdica. Encontrar uma solução para isso não é uma tarefa fácil, visto que não existem muitos projetos interessantes condizentes com o nível teórico alcançado pelos alunos ao fim da etapa.

No entanto, alternativas melhores devem ser estudadas para edições futuras da oficina, pois a má performance de qualquer etapa compromete toda a estrutura da "cadeia de realizações cognitivas", impedindo o bom aprendizado dos alunos.

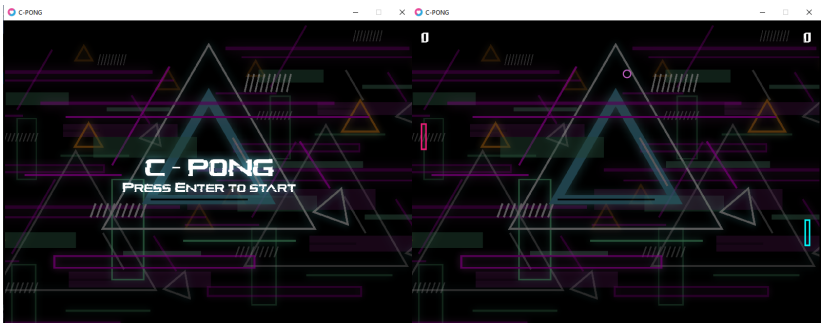
4.5.3 Terceira etapa

Após o pouco engajamento da etapa anterior, estava receoso que uma grande parte dos alunos fosse abandonar as atividades da oficina. No entanto, a maioria frequentou os encontros normalmente. As interações durante as aulas teóricas foram pouco significativas, quase nenhuma pergunta foi feita, e os alunos falaram muito pouco, mesmo quando estimulados. Não é certo se isso está de alguma forma relacionado à oficina em si, ou à situação atípica das aulas online, ou até mesmo à fadiga do fim do semestre.

Cinco alunos finalizaram com sucesso os seus jogos Pong, mas apenas um implementou uma funcionalidade extra ao jogo finalizado. Apesar disso, considero os resultados desta etapa bastante satisfatórios, pois todos os cinco trabalhos demonstraram grande esforço por parte

de seus idealizadores, especialmente nas soluções gráficas e sonoras. Isso mostra que todos estavam devidamente envolvidos com os seus projetos, mesmo que de uma maneira diferente da esperada.

Figura 8 – C-Pong



Fonte: *Pong* feito por Mateus Augusto Tavano.

Após a apresentação de seus jogos *Pong*, os alunos dividiram suas experiências com o restante da turma, e todos relataram estarem satisfeitos com o projeto desenvolvido e com conteúdo estudado.

4.6 Post mortem: após o fim

Após o fim da última etapa, uma pesquisa foi realizada para averiguar a opinião dos alunos participantes quanto à oficina como um todo. As perguntas focaram em diversos pontos: avaliação da experiência geral, maiores dificuldades encontradas, auto-avaliação dos resultados e avaliação de cada etapa específica. Seis alunos responderam a esta pesquisa, dois dos quais não entregaram o trabalho final.

Todos relataram terem gostado da experiência como um todo, e todos que realizaram o projeto final alegaram estarem satisfeitos com os resultados adquiridos. A maior dificuldade encontrada pela maioria foi com relação à estruturação dos códigos e à programação em si (assim como a matemática envolvida nestes processos). Apenas um aluno afirmou possuir vontade de continuar utilizando a programação em projetos futuros relacionados à área do design.

Foi pedido aos alunos que dessem uma nota entre um e cinco para cada uma das três etapas da oficina. A avaliação média de cada etapa é a seguinte, da maior para a menor.

a) Terceira etapa - Löve2d: **4,8**;

b) Primeira etapa - Scratch: **4,5**;

c) Segunda etapa - Lua: **3,66**.

Estas notas vão de acordo com os resultados adquiridos em cada uma das etapas, sendo a segunda etapa aquela com a pior nota e com os piores resultados aparentes.

Aos alunos que não completaram a oficina até o final, foi perguntado o motivo da desistência, mas apenas uma única resposta foi coletada: o aluno achou que "não daria conta" de completar o trabalho final, devido a dificuldade do mesmo. Isto é preocupante, pois indica que ele possivelmente não se sentiu confortável para buscar ajuda, optando pelo abandono da atividade. Isto pode ter sido causado por uma falha de comunicação de minha parte, o instrutor, e deve ser considerado em edições futuras da oficina, para evitar ocorrências semelhantes.

Embora a quantidade de participantes tenha sido pequena, impedindo uma interpretação mais séria dos resultados encontrados e da metodologia de ensino desenvolvida, considero a oficina um sucesso. Todos os participantes relataram terem aprendido coisas novas, e os trabalhos finais foram realizados com muita dedicação e atenção.

5 Conclusão

A Oficina LÖVE é fruto de uma amálgama de experiências pessoais vivenciadas por mim no decorrer dos meus dois últimos anos da graduação, experiências estas que me levaram até a programação, na qual encontrei uma verdadeira paixão. Este projeto é minha primeira tentativa de repassar os conhecimentos que adquiri, na expectativa de despertar nos meus colegas de profissão o interesse por esta ciência, que considero tão semelhante à ciência do design.

Finalizo o projeto com muita satisfação, não apenas pelas minhas conquistas pessoais, que permitiram que ele se concretizasse, mas também pelos excelentes resultados adquiridos. As aplicações práticas para os conhecimentos aqui ensinados são infinitas, e espero que todos possam ter aprendido algo novo ou, quem sabe, encontrado um interesse para a vida toda.

De acordo com os princípios que inspiraram a realização deste projeto, eu e os demais participantes da oficina disponibilizamos os materiais por nós produzidos, assim como a gravação dos nossos encontros, em um site, para que qualquer pessoa possa acessá-los, seja para fins de estudo ou apenas diversão.

Referências

BOULAY, B. D. Some difficulties of learning to program. *Journal of Educational Computing Research*, SAGE Publications Sage CA: Los Angeles, CA, v. 2, n. 1, p. 57–73, 1986.

BOULAY, B. du; O'SHEA, T.; MONK, J. The black box inside the glass box: presenting computing concepts to novices. *International Journal of man-machine studies*, Elsevier, v. 14, n. 3, p. 237–249, 1981.

CANAS, J. J.; BAJO, M. T.; GONZALVO, P. Mental models and computer programming. *International Journal of Human-Computer Studies*, Elsevier, v. 40, n. 5, p. 795–811, 1994.

CREATIVE Computing Curriculum. 2021. Disponível em: <<http://scratched.gse.harvard.edu/guide/index.html>>. Acesso em 04 de Fevereiro de 2021.

DAVIES, S. P. Models and theories of programming strategy. *International journal of man-machine studies*, Elsevier, v. 39, n. 2, p. 237–267, 1993.

FULLERTON, T.; SWAIN, C.; HOFFMAN, S. *Game Design Workshop: Designing, Prototyping, and Playtesting Games*. [S.l.]: CMP Books, 2004. ISBN 1578202221.

GREEN, T. Programming languages as information structures. In: *Psychology of programming*. [S.l.]: Elsevier, 1990. p. 117–137.

HOC, J.-M.; NGUYEN-XUAN, A. Language semantics, mental models and analogy. In: *Psychology of programming*. [S.l.]: Elsevier, 1990. p. 139–156.

LINN, M. C.; DALBEY, J. Cognitive consequences of programming instruction. *Studying the novice programmer*, p. 57–81, 1989.

MENDELSOHN, P.; GREEN, T.; BRNA, P. Programming languages in education: The search for an easy start. In: *Psychology of programming*. [S.l.]: Elsevier, 1990. p. 175–200.

PERKINS, D. N.; HANCOCK, C.; HOBBS, R.; MARTIN, F.; SIMMONS, R. Conditions of learning in novice programmers. *Journal of Educational Computing Research*, SAGE Publications Sage CA: Los Angeles, CA, v. 2, n. 1, p. 37–55, 1986.

ROBINS, A.; ROUNTREE, J.; ROUNTREE, N. Learning and teaching programming: A review and discussion. *Computer science education*, Taylor & Francis, v. 13, n. 2, p. 137–172, 2003.

ROGALSKI, J.; SAMURÇAY, R. Acquisition of programming knowledge and skills. In: *Psychology of programming*. [S.l.]: Elsevier, 1990. p. 157–174.

WHERE Lua is used. 2013. Disponível em: <<https://sites.google.com/site/marbux/home/where-lua-is-used>>. Acesso em 07 de Fevereiro de 2021.

WINSLOW, L. E. Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin*, ACM New York, NY, USA, v. 28, n. 3, p. 17–22, 1996.

Glossário

A | **B** | **C** | **E** | **F** | **G** | **I** | **L** | **M** | **P** | **S** | **T** | **U** | **V** | **W** | **Z**

A

assembler

Programa que traduz código *assembly* para código de máquina. 14, 37, *Veja também:* assembly

assembly

Linguagem de programação de baixo nível, bastante similar ao código de máquina, porém mais abstraída, de forma a ser mais legível para programadores. O código *assembly* é traduzido para código de máquina por um programa chamado *assembler*. 14, 37, *Veja também:* assembler

B

binário

Sistema de numeração com apenas dois dígitos, geralmente zero e um. Nos dias de hoje a maior parte dos computadores modernos utilizam o binário para encodificar dados e instruções. 14, 37

C

C

C é uma linguagem de programação com propósitos gerais, criada com o intuito de proporcionar acesso de baixo nível à memória e construções que são eficientemente mapeadas para código de máquina. De acordo com o TIOBE Index, C é a linguagem de programação mais popular do mundo em abril de 2021. 14

concept art

Artes de pré-produção, utilizadas para criar, definir ou consolidar uma ideia ou tom que poderão ser utilizados no produto final. 11

código de máquina

Código de máquina é uma linguagem de programação de baixo nível e estritamente númerica, utilizada para controle do CPU (Central Processing Unit, ou Processador) de um determinado computador. Foi criada com a intençã de ser executada o mais rápido possível. O código de máquina pode ser representado por qualquer sistema númerico (decimal, hexadecimal, binário), porém todas as instruções são eventualmente traduzidas

para binário, para que possam ser devidamente "compreendidas" pela máquina. 14, 37, *Veja também:* binário

E

estrutura de dados

Em ciência da computação, uma estrutura de dados é um formato de organização, administração e armazenamento de dados, que permite o fácil acesso e manipulação desses dados na memória do computador. 27

F

framework

Conjunto de bibliotecas de código que simplificam o processo de desenvolvimento de um determinado tipo de programa. 13, 17, 21, 28, 31

G

Game Maker Studio 2

Motor de jogos comercial, utilizado no desenvolvimento de jogos 2D. <https://www.yoyogames.com/en/gamemaker>. 21

Godot

Motor de jogos open-source e gratuito. Pode ser utilizado tanto para a criação de jogos 2D como jogos 3D. <https://godotengine.org/>. 21, *Veja também:* motor de jogos

I

IDE

Uma IDE (Integrated Development Environment) é um programa que providencia ferramentas úteis para o desenvolvimento de programas. Geralmente IDEs possuirão um editor de código fonte (*source code*), um *debugger*, para que bugs possam ser facilmente encontrados e corrigidos, e uma variedade de ferramentas de compilação para que o código escrito seja facilmente compilado e executado. 25

L

Lua

Linguagem de programação de altíssimo nível, desenvolvida por Roberto Ierusalimschy, Waldemar Celes e Luiz Henrique de Figueiredo na PUC-Rio. 13, 14, 17, 19, 21, 23, 25, 27, 32, 39–41

Löve2d

Framework de desenvolvimento de jogos 2D, gratuita e open-source. Utiliza a linguagem de programação Lua. 12, 13, 17, 19, 21, 28, 31, *Veja também:* framework

M

mockup

Em design, um mockup é um protótipo ou modelo de um produto a ser desenvolvido, que não necessariamente possui todas as funcionalidades do produto final. 29

modelo mental

Um modelo mental é uma representação interna de um processo externo. Uma explicação do processo de pensamento de alguém sobre como algo funciona no mundo real. É uma representação do mundo, das relações entre suas diversas partes e da percepção intuitiva de um indivíduo sobre seus próprios atos e suas consequências. Modelos mentais podem ajudar a moldar comportamentos, estabelecer uma abordagem para a resolução de problemas (semelhante a um algoritmo pessoal) e realizar tarefas de todos os tipos. 17, 27

motor de jogos

Programa que busca facilitar o desenvolvimento de jogos de todos os tipos. Motores de jogos possuem uma grande gama de funcionalidades previamente implementadas, como por exemplo editores de nível, interface gráfica de usuário, suporte para animações e interpolações, sistemas de física, entre muitas outras. 20, 21, *Veja também:* framework

máquina nocional

A máquina nocional é um modelo mental abstraído do computador, que define o seu funcionamento e comportamento. 17, *Veja também:* modelo mental

P

Pong

Jogo arcade inspirado no tênis de mesa, lançado em 29 de novembro de 1972 e criado por Nolan Bushnell e Ted Dabney. O jogador controla uma raquete, movendo-a verticalmente no lado esquerdo da tela, e compete contra o computador ou outro jogador que controlam uma segunda raquete no lado oposto. O objetivo é usar as raquetes para acertar a bola e mandá-la para o outro lado. 12, 20, 29, 33

ponteiro

Em ciência da computação, um ponteiro é um objeto que armazena um endereço específico da memória do computador, para fácil acesso e manipulação do dado armazenado neste endereço. 27

S

Scratch

Linguagem de programação visual criada pelo MIT. <https://scratch.mit.edu/>. 23, 26, 27

Skyrim

RPG de mundo aberto desenvolvido pela Bethesda e lançado em 2011. Bastante aclamado pelo público e pela crítica. <https://elderscrolls.bethesda.net/en/skyrim/>. 20

sprite

Em computação gráfica, um sprite é uma imagem bidimensional que integra uma cena maior (um cenário, por exemplo). Sprites são mais comumente utilizados na produção de jogos 2D. 24, 40

sprite sheet

Arquivos de imagem com diversos sprites a serem utilizados na produção de jogos 2D. 11, *Veja também:* sprite

T

Table

Única estrutura de dados presente na linguagem de programação Lua. Funciona como uma matriz associativa. Cada entrada em uma table possui uma chave específica, que pode ser utilizado para o rápido acesso da entrada. 27, *Veja também:*

textura

Em computação gráfica, uma textura é uma imagem bidimensional que é aplicada a uma superfície tridimensional, com finalidade de proporcionar maior riqueza de detalhes, cores e texturas. 11

U

Unity

Motor de jogos comercial e bastante popular. Pode ser utilizado tanto para a criação de jogos 2D como jogos 3D. <https://unity.com/>. 21, *Veja também:* motor de jogos

Unreal Engine

Motor de jogos comercial e bastante popular. Pode ser utilizado tanto para a criação de jogos 2D como jogos 3D. <https://www.unrealengine.com/>. 20, 21, *Veja também:* motor de jogos

V

Visual Studio Code

IDE desenvolvida pela Microsfot, voltada para o desenvolvimento de programas em diversas linguagens de programação. <https://code.visualstudio.com/> 25, *Veja também:* IDE

W

World of Warcraft

MMORPG criado pela Blizzard Entertainment e lançado em 2004. É o MMORPG mais jogado atualmente no mundo. <https://worldofwarcraft.com/>. 20

Z

ZeroBrane

IDE voltada para o desenvolvimento de programas com a linguagem de programação Lua. <https://studio.zerobrane.com/> 25, *Veja também:* IDE