Cat On The Run: Desenvolvimento de um jogo Endless Runner

Bianca Alves Costa, Fabrício Tonetto Londero
Curso de Jogos Digitais
UFN - Universidade Franciscana
Santa Maria - RS
bianca.costa@ufn.edu.br, fabriciotonettolondero@gmail.com

***Resumo*—O desenvolvimento de jogos é multidisciplinar, sendo assim, durante a graduação, tentou-se abranger o maior número de áreas e conhecimentos possíveis. Contudo, nem sempre foi viável a experiência prática de todas elas. Este projeto visou o desenvolvimento de um Endless Runner 2D, denominado *Cat On The Run*, com o uso de shaders e armazenamento local de dados, permitindo a experiência de praticar o desenvolvimento de conteúdos vistos ao longo do curso, mas ainda não aplicados a um projeto. Para a produção foi utilizada a metodologia de Chandler. Ao final deste projeto resultou em um jogo Endless Runner funcional, assim como, um acréscimo ao portfólio e mais conhecimento nas áreas previamente citadas.**

***Palavras-chave* : tecnologia; jogos; shader; Endless Runner;**

I. Introdução

Este trabalho tem como sua proposta o desenvolvimento de um jogo digital 2D de gênero Endless Runner chamado *Cat On The Run*, que utilize conteúdos aprendidos durante a graduação, dentre eles, encontam-se shaders e armazenamento de dados. Para isso, foi utilizada a metodologia de Heather Maxwell Chandler [3], proposta em seu livro "Manual de Produção de Jogos Digitais", sua divisão em etapas e iteratividade auxiliam no controle do processo de desenvolvimento.

Para a realização deste trabalho, fez-se necessário um estudo sobre o uso de shaders no motor gráfico Unity, assim como em técnicas de salvamento de dados locais. Também foram analisados outros quatro jogos do gênero Endless Runner, para uma melhor compreensão do que é preciso para o desenvolvimento de um jogo deste gênero.

A. Justificativa

A criação de um jogo envolve diversos aspectos como: arte, som, música, programação, game design, entre outros. E, dentro de cada um desses, há ainda diversas subdivisões, pois quanto maior o jogo, mais subdivisões são criadas para melhor repartir o trabalho. Desta forma, necessita-se de um maior número de pessoas para que o desenvolvimento do jogo aconteça.

Durante a graduação foi abrangido o máximo possível destas áreas e, também, a prática do desenvolvimento de jogos. No entanto, por ser um conteúdo extremamente extenso, nem sempre houve tempo de pô-lo totalmente em prática em um projeto, seja por falta de tempo, por não haver um jogo em desenvolvimento no momento ou até mesmo pelo fato do conteúdo em questão não combinar com o escopo do projeto em andamento.

Desta forma, este trabalho mostrou a oportunidade de por em prática alguns destes temas abordados durante a graduação, assim como, a chance de criar um jogo de um gênero ainda não desenvolvido no decorrer do curso.

Assim, este projeto é tanto uma maneira de testar os conteúdos aprendidos, como também, é um desafio próprio vindo da implementação de um Endless Runner.

B. Objetivos

Este trabalho tem por seu objetivo principal desenvolver um jogo digital do gênero Endless Runner em 2D para a platafoma Microsoft Windows utilizando conhecimentos previamente aprendidos na graduação e o motor gráfico Unity para implementação e a metodologia de Chandler durante seu desenvolvimento.

Além disso, têm-se como objetivos específicos do projeto: fazer uso de shaders para realizar transições de telas; criar um sistema de armazenamento de dados locais para guardar as pontuações do jogador e fases desbloqueadas previamente.

C. Estrutura do trabalho

Nas seções II, III e IV serão abordados os principais focos deste projeto: shaders, armazenamento local e o gênero endless runner. Após, na seção V, será apresentada a metodologia de Chandler e como ela influenciou o desenvolvimento do jogo.

Na seção VI, é descrito como será o desenvolvimento do projeto, o qual divide-se nas subseções: pré-produção, produção, testes e pós-produção. Por fim, na seção VII está localizada a conclusão deste trabalho.

II. Shaders

Um shader pode ser definido como código que contém uma lista de instruções e cálculos matemáticos a serem executados na GPU (*Graphics Processing Unit*) do computador [4]. Esse código pode definir muitas coisas sobre um objeto, como: a aparência da superfície dele, maneira de se mover,

se deve possuir uma animação, como deve se mesclar na cena, entre numerosos outros parâmetros possíveis.

Freya Holmér [5] diz que por causa do modo como shaders funcionam, e por existirem diferentes tipos, cada um com especialização em um aspecto, estes são onipresentes nos jogos das mais diversas formas.

Figura 1. Overwatch da Blizzard utiliza shaders, tanto na interface como nos efeitos de habilidades [5].

Há duas maneiras de se utilizar shaders na Unity: uma faz uso do Shader Graph, sistema baseado em *nodes*. E a outra é com programação em HLSL (*High-Level Shading Language*) [13].

Shader Graph é uma ferramenta da própria Unity para facilitar a criação de shaders para não programadores, ela funciona com um sistema de nós (*nodes*) onde, através de conexões criadas entre funções, altera o shader pouco a pouco. O Shader Graph permite visualizar a saída de cada etapa realizada, assim como o produto final, tornando fácil a visualização e acelerando as iterações da criação do shader como pode ser visto na Figura 2[12].

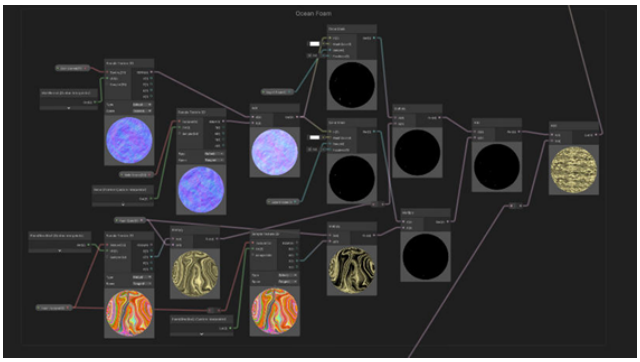

Figura 2. Shader Graph: uso de nodes na criação de shaders [12].

A HLSL é a linguagem de programação oficial da Unity para shaders e, com exceção do Shader Graph, todos os shaders da Unity são escritos na linguagem declarativa ShaderLab. Esta linguagem permite que as variáveis sejam vistas no inspetor, assim permitindo sua manipulação em tempo real [4].

III. Armazenamento local

Quando os primeiros jogos surgiram não havia como "salvar o jogo", em caso de morte o jogador era obrigado a retornar e jogar tudo desde o início. Mas os jogos eram mais simples, praticamente uma repetição da mesma fase com um aumento de velocidade para dificultar, não sendo necessária a capacidade de retornar o jogo de onde parou [1].

No entanto, com o avanço das tecnologias, os jogos foram se expandindo e tornando-se maiores e mais complexos, assim sendo, inviável a cada morte o jogador voltar ao início. Hoje em dia, até o mais simples dos jogos conta com algum tipo de salvamento de dados [9].

Há muitas formas de lidar com o armazenamento e serialização (processo de transformar dados dispersos em dados com estrutura e independentes de arquitetura) de dados, cada uma com suas vantagens e desvantagens. Alguns exemplos são: XML, JSON, banco de dados, PlayerPrefs da Unity, entre outros.

O PlayerPrefs é um sistema próprio da Unity para armazenamento de dados sem serialização. Por não utilizar serialização e atuar em alguns tipos de varáveis pré-definidos este é mais utilizado para salvar as preferências do jogador entre as sessões de jogo, tais como volume da música, qualidade gráfica escolhida, nível de dificuldade, e outras configurações [11].

XML (*Extensible Markup Language*) tem como um de seus usos primários a serialização e transferência de dados entre aplicações. Sua estrutura é constituída de marcações criadas pelo próprio usuário de acordo com sua necessidade, podendo se mostrar de difícil legibilidade como pode ser visto na figura 3. Por ser antiga é fácil encontrar conteúdo sobre ela e ferramentas compatíveis [7] [10].

Figura 3. Exemplo de XML [10].

O JSON (*JavaScript Object Notation*) possui uma sintaxe de fácil leitura aos humanos e também para os computadores, o que facilita seu uso. A princípio, era mais adequado para aplicações que utilizavam JavaScript, pois recebia suporte dentro desta linguagem. Atualmente, possui suporte completo em diversas linguagens por causa de sua popularização [7].

```json
{
    "name": "Dr Charles",
    "lives": 3,
    "health": 0.8,
    "level": 1,
    "timeElapsed": 47.5,
    "playerName": "Dr Charles Francis"
}
```

Figura 4. Exemplo de JSON [10].

IV. Endless Runner

Endless Runner é um subgênero de *Plataformer*, que se originou com o jogo Canabalt (2009). Esse gênero veio da exigência de uma adaptação à era dos smartphones onde a falta de botões e a necessidade de segurar o celular deixavam poucas opções de controles.

Sua premissa é simples: um personagem que sempre está avançando e cabe ao jogador desviar de obstáculos no caminho e, apesar do que o nome sugere, não necessariamente ele seja infinito, podendo ser separado em fases com objetivos a serem completados.

Por sua simplicidade em questão de controles e mecânicas é um gênero extremamente acessível e casual, mas ainda assim, um desafio de se dominar [8].

A. Jogos correlatos

Para a realização deste trabalho foram analisados quatro jogos, cada um possuindo aspectos desejados para estarem presentes no jogo confeccionado ao longo deste trabalho. Estes jogos são: Subway Surfers, Miraculous Ladybug & Gato Noir, Geometry Dash e Spring Ninja.

1) Subway Surfers: Lançado pela Kiloo Games em 2012 é um Endless Runner 3D, o personagem está sempre avançando e cabe ao jogador fazê-lo desviar de obstáculos enquanto pega itens que o auxiliam e moedas, estas são a pontuação. O jogo somente termina quando o jogador falha em desviar de algo, sendo assim, incitando o jogador a rejogar sempre buscando ampliar o seu tempo, conseguindo mais moedas e buscando pontuações melhores do que as jogadas anteriores.

2) Miraculous Ladybug & Gato Noir: Criado por Crazy Labs, este jogo é semelhante ao jogo anterior, sua maior diferença vem do fato de não ser uma única fase infinita, e sim, diversas fases, cada uma com um objetivo a ser concluído para avançar para a próxima fase como pode ser visto no topo da Figura 6. Também há diferentes personagens para serem escolhidos pelo jogador, mas isto é somente estético e não há influência na gameplay.

Figura 5. Subway Surfers (Fonte: print screen da página do jogo na loja Google Play).

Figura 6. Miraculous Ladybug & Gato Noir (Fonte: print screen da página do jogo na loja Google Play).

3) Geometry Dash: Desenvolvido pela RobTop Games, se difere dos jogos citados anteriormente por ser 2D em vez de 3D, limitando os movimentos do jogador a apenas evitar barreiras deslocando-se para cima e para baixo. Também é dividido em fases e possui forte ênfase na trilha sonora, pois o jogador pode utilizá-la para seguir o ritmo e desviar dos obstáculos.

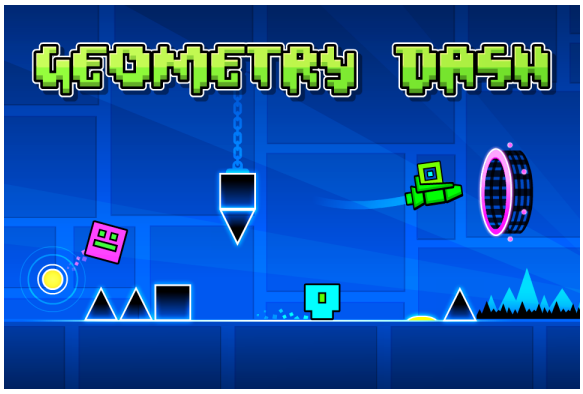

Figura 7. Geometry Dash (Fonte: print screen da página do jogo na loja Google Play).

4) Spring Ninja: Lançado pela Ketchapp, diferentemente dos jogos citados anteriormente, este foi escolhido não por mecânicas, mas sim pelo seu estilo artístico chamado de *Flat Design*. Seu visual é simplista, porém, se bem elaborado, pode ser completo e limpo. Além disso, por ser uma ilustração plana e de detalhes mais simples, demanda menos tempo para ser confeccionado em comparação a outros estilos [6].

Figura 8. Spring Ninja (Fonte: print screen da página do jogo na loja Google Play).

B. Conclusão sobre jogos correlatos

Da análise realizada dos jogos citados anteriormente, as características almejadas para o jogo desenvolvido neste trabalho foram, de Geometry Dash, um mundo 2D onde o jogador somente move-se para cima e para baixo.

Assim como em Miraculous Ladybug & Gato Noir e Geometry Dash, o jogo confeccionado buscou possuir divisão por fases (três especificamente), aumentando em dificuldade. Cada fase somente será concluída quando o jogador conseguir cumprir o objetivo da mesma.

De Subway Surfers, buscou-se utilizar a capacidade de manter um recorde, no modo infinito, o objetivo do jogador será conseguir pontuações cada vez mais altas do que as anteriores.

Finalizando com Spring Ninja que, como dito anteriormente, trás consigo a inspiração para o estilo de arte do jogo. Buscou-se um estilo gráfico vetorial simples, inspirado no *Flat Design*.

V. Metodologia

Para este trabalho foi escolhida a metodologia de Heather Maxwell Chandler [3], retratado em seu livro "Manual de Produção de Jogos Digitais". O processo de produção de jogos descrito neste livro é iterativo e possui quatro etapas: pré-produção, produção, testes e pós-produção. Ao final de cada iteração um novo protótipo do produto é completado, e este ciclo repete-se até a versão final do produto ser alcançado.

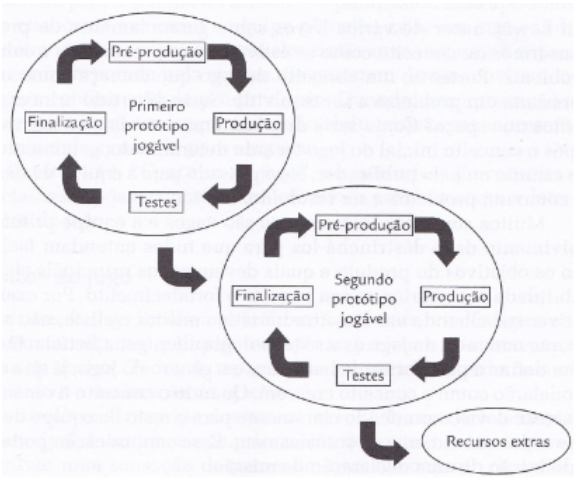

Figura 9. Ciclo de produção [3].

Na etapa de pré-produção são definidas as bases do projeto como, por exemplo, conceito, gênero, plataforma, público-alvo, GDD (*Game Design Document*), recursos necessários, orçamento e cronograma.

Durante a etapa de produção, aquilo que foi planejado na etapa anterior é de fato aplicado, o conteúdo necessário começa a ser criado e implementado. Também nesta etapa é

mantido o controle da conclusão de tarefas, sendo possível realizar o rastreamento do progresso do projeto.

Na fase de testes é verificado se tudo funciona como deveria e, caso haja erros, corrigi-se o necessário. É uma fase crítica, já que é nela que problemas podem ser identificados e tratados antes de se tornarem uma "bola de neve".

Na pós-produção é encerrado o desenvolvimento, pós e contras do projeto são analisados e tudo é arquivado para referência futura. Desse modo, a experiência adquirida ao fim do projeto pode ser utilizada em outros trabalhos.

Essa produção cíclica permite um maior controle ao processo de desenvolvimento, dividindo-o em pequenas partes facilitando a administração e validando-as antes de começar outro ciclo. Assim permitindo maior tempo de planejamento, o que torna possível a visualização e correção de erros antes destes ocorrerem, evitando retrabalho.

VI. DESENVOLVIMENTO

O desenvolvimento foi dividido em etapas, seguindo a metodologia de Chandler [3]. Nesta seção é descrito o processo de criação do jogo dividido nos estágios citados anteriormente: pré-produção, produção, testes e pós-produção.

A. Pré-produção

Nessa etapa são realizadas as definições básicas do jogo (conceito, estilo artístico, mecânicas, plataforma, entre outros) e o planejamento do escopo do projeto.

1) Conceito: O jogador é um gato que comeu a carne dada a um cachorro, agora o cão está furioso com você. Assim começa a perseguição pela vizinhança, correndo entre calçada, rua e muros para evitar ser pego.

A colisão com obstáculos no caminho causará a perda de uma vida ao gato e, a cada vida perdida, o cão estará mais próximo de pegá-lo. Somente adquirindo o número de peixes necessários na fase o gato conseguirá escapar do cachorro.

2) Escopo: Foi planejada a confecção de três fases, um modo infinito e um menu inicial. Como pode ser visto na Figura 10 o fluxo do jogo terá início no menu, onde o jogador pode optar por jogar a fase 1 ou o modo infinito. Após a fase 1, será liberada a fase 2, caso o jogador tenha tido êxito na conclusão do objetivo da fase anterior. Se não ele terá que tentar novamente e, assim, se repete na fase 2 para desbloquear a fase 3.

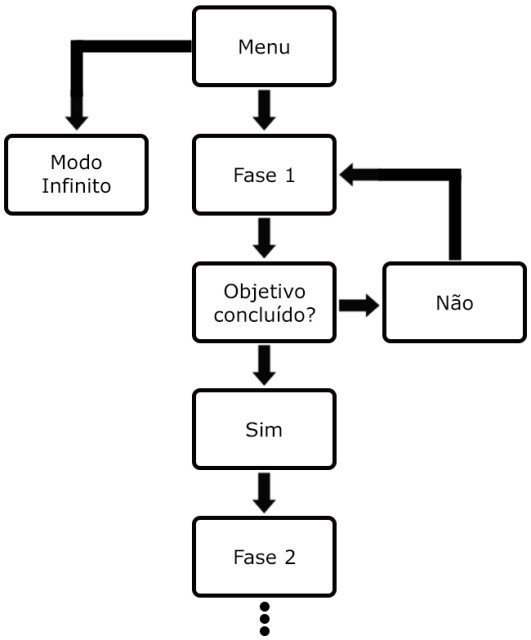

Figura 10. Fluxo de jogo.

3) Mecânicas: A principal mecânica do jogo é o desvio dos obstáculos, o jogador utilizará as setas do teclado (para cima e para baixo) para alternar entre os caminhos que o gato pode se movimentar, ou seja, entre três possíveis caminhos: rua, calçada ou muro.

Na Figura 11 encontra-se o *loop* desta mecânica. Durante a gameplay, obstáculos aparecerão no caminho, cabendo ao jogador desviar deles, caso desvie com sucesso o *loop* retorna ao começo.

Entretanto, se o jogador falhar e colidir com algum obstáculo perderá uma vida. Caso ainda haja vidas restantes, o *loop* voltará ao início, mas se o jogador já tiver perdido todas as vidas, então o jogo acaba.

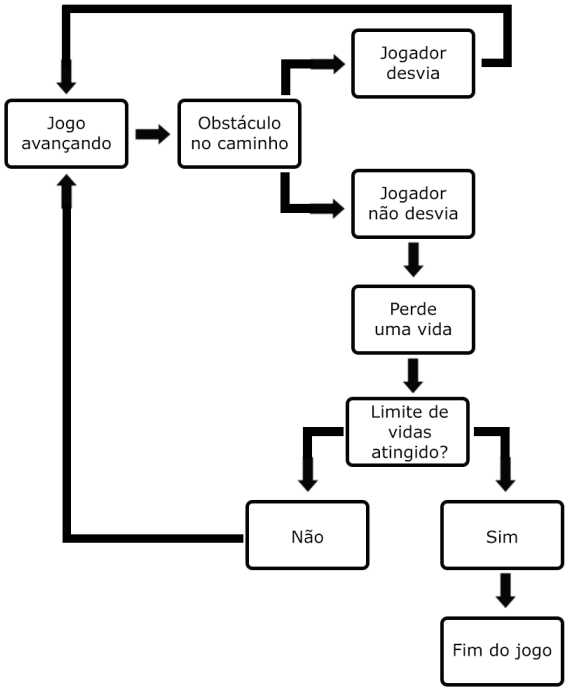

Figura 11. *Loop* dos obstáculos.

Há também a mecânica dos itens. Há dois tipos de itens no jogo: ração e peixe. A ração aumenta a pontuação do jogador durante a fase, já o peixe é necessário para completar o objetivo da fase e é obrigatório para conseguir desbloquear a próxima fase.

A Figura 12 ilustra o *loop* do que acontece quando o jogador pega ou não um item e demonstra a diferença entre o peixe e a ração.

4) Estilo artístico: Como dito na seção de jogos correlatos, foi utilizado um estilo gráfico vetorial simples com inspiração no *Flat Design* [6] [2] para criação das artes do jogo. Esse estilo artístico é minimalista e faz uso de cores sólidas e formas geométricas simples como base, trazendo consigo um aspecto limpo e marcante.

5) Músicas e sons: Foram utilizados áudios de licença *Creative Commons* (CC) tanto para músicas, quanto para efeitos sonoros do jogo.

6) Plataforma: O jogo foi desenvolvido para Microsoft Windows e está disponível no itch.io para download de forma gratuíta.

7) Programas: O desenvolvimento fez uso da Unity como game engine e da linguagem de programação C#. Para a criação dos *assets* foi utilizado o Adobe Illustrator.

B. Produção

A produção do jogo começou pelo aspecto mais básico de um Endless Runner, um personagem com animação de correr, um cenário que movimenta-se e obstáculos e

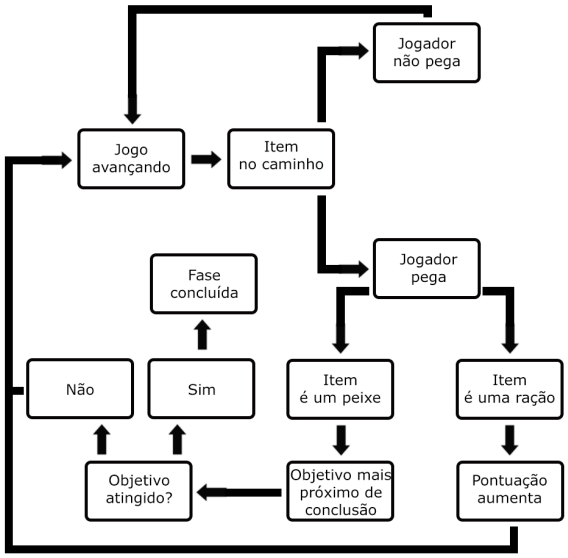

Figura 12. *Loop* dos itens.

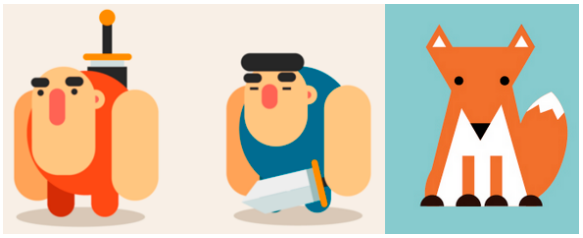

Figura 13. Exemplos de *Flat Design* [6] [2].

itens gerados aleatoriamente. Utilizando o programa Adobe Illustrator a criação desses *assets* foi iniciada.

Durante a fase de pré-produção foi estabelecido que a localização em que o jogo se passa é uma rua, assim, como pode ser visto na Figura 14, foi criado uma calçada, estrada e muro como as três possíveis rotas que o gato pode correr.

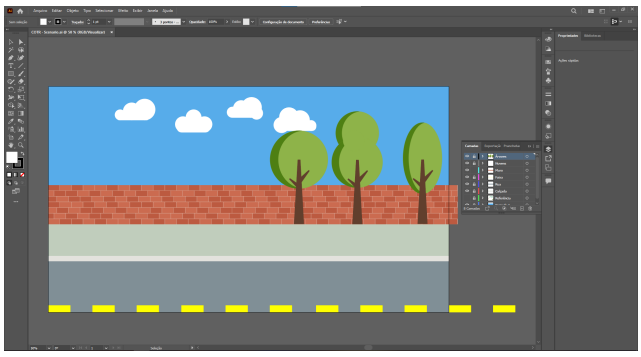

Figura 14. Cenário do jogo no Adobe Illustrator.

Tanto o muro quanto a faixa amarela na estrada foram feitas de modo *seamless* (sem emendas), pois estas duas foram

utilizadas para criar o efeito de *parallax* [1]. Para ajudar nesse efeito também foram desenhados três modelos de árvores e quatro modelos de nuvens, estes modelos tornaram-se uma segunda camada mais ao fundo do *parallax*.

O próximo item criado foi o gato e sua animação de corrida. Como o cenário estava finalizado e utilizava tons de cinza, para calçada e estrada, e tons avermelhados para o muro, as cores do gato precisavam criar um contraste entre ele e o cenário.

Então decidiu-se por fazer um gato preto e branco com olhos verdes, assim contrastando bem com o cenário e evitando que o jogador não conseguisse diferenciá-lo deste. Também foi criada uma sombra embaixo do gato para ajudar na percepção de sua posição exata na cena. A animação é composta por quatro *frames* que podem ser vistos na Figura 15.

Figura 15. O gato em sua animação de corrida.

Para os obstáculos a serem evitados, foi decidido a criação de três tipos, um para cada caminho: cones de trânsito para a parte da rua, uma lixeira caída para a calçada e um portão com adornos na parte do muro.

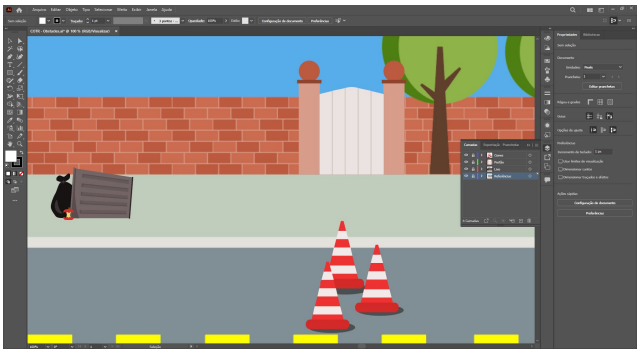

Figura 16. Os três obstáculos encontrados no jogo.

A seguir, veio a criação dos itens (peixe e ração) em suas versões como itens e como interface. Também foi criado uma arte de coração, para representar o número de vidas do jogador. No caso da ração e do peixe foram utilizadas camadas diferentes dentro do Adobe Illustrator para separá-los do contorno branco que a versão para interface possuía.

Assim, no momento de exportar a versão final das imagens, somente foi necessário ocultar a camada de contorno branco e se obteve a versão de item do peixe e da ração, sem necessidade de criar outra versão.

Figura 17. Ícones da interface.

Após a criação destes primeiros assets partiu-se então para a fase de implementação na Unity. Primeiramente, foi desenvolvido o código necessário para o efeito de *parallax* do cenário e a movimentação do gato entre as três rotas.

O céu, calçada e estrada são elementos fixos no cenário, para eles não foi necessário a implementação de nenhum código. Para o muro e a faixa amarela foi criado um *script* cuja função era reposicionar ambos, isso juntamente com um *script* para fazê-los se moverem para a esquerda, compõem a primeira camada do *parallax*.

```
[SerializeField]
private string type = "object";

Mensagem do Unity | 0 referências
private void LateUpdate()
{
    if (type == "Wall")
    {
        if (this.transform.position.x < -33.0f)
        {
            this.transform.position = new Vector2(10.12f, this.transform.position.y);
        }
    }
    else if (type == "Strip")
    {
        if (this.transform.position.x < -33.0f)
        {
            this.transform.position = new Vector2(18.15f, this.transform.position.y);
        }
    }
}
```

Figura 18. Código *Parallax* do muro e da faixa amarela.

Na Figura 18 está este código. Ele funciona para ambos os objetos, pois a *string type* serve como uma identificação para dizer qual dos dois objetos o *script* está atrelado e então, no método *LateUpdate* é verificado se este objeto já passou do limite estabelecido, caso sim, ele é posto em sua posição definida.

Para camada mais ao fundo do efeito foi criado o código da Figura 19. Nele, em um tempo pré-determinado, um número é sorteado, caso seja igual ou menor do que quatro uma nuvem é gerada no cenário em uma posição estipulada previamente, mas em uma altura aleatória dentro de um intervalo. Caso o número seja maior do que quatro é uma árvore que é gerada na posição fixa já escolhida.

Tanto as nuvens quanto as árvores também movem-se da direita para a esquerda, porém estas mais lentamente do que os objetos de primeiro plano e, ao atingirem o limite fora da tela à esquerda, são destruídos ao invés de reposicionados.

[1]Técnica em que as imagens de plano de fundo movem-se mais lentamente que as imagens em primeiro plano, criando uma ilusão de profundidade e movimento em uma cena 2D.

```
IEnumerator Background()
{
    while (true)
    {
        yield return new WaitForSeconds(delay);

        number = Random.Range(1.0f, 10.0f);

        if (number ≥ 4.0f)
        {
            SpawnCloud();
        }
        else { SpawnTree(); }
    }
}

1 referência
public void SpawnCloud()
{
    var position = new Vector2(10.0f, Random.Range(2.0f, 4.4f));
    Instantiate(clouds[Random.Range(0, clouds.Length)], position, Quaternion.identity);
}

1 referência
public void SpawnTree()
{
    Instantiate(trees[Random.Range(0, trees.Length)], treePosition, Quaternion.identity);
}
```

Figura 19. Código de geração das nuvens e árvores.

A seguir foi codificada a produção aleatória dos obstáculos e a dos dois itens (ração e peixe), assim como a conexão entre estes e as informações dispostas na interface, cada vez que o jogador colidisse com um destes sua vida, pontuação e andamento do objetivo são atualizados na tela.

Com a finalização de um menu inicial básico, contendo opções para jogar, tutorial e sair implementadas primariamente, o jogo atingiu seu primeiro ciclo de desenvolvimento. Após isso, testes foram realizados antes do prosseguimento do desenvolvimento como estipula a metodologia de Chandler.

Figura 20. Primeira versão do menu inicial.

No segundo ciclo de desenvolvimento o foco inicial foram as correções e mudanças necessárias após o recolhimento do *feedback* dos formulários. Depois disso passou-se para a implementação dos *shaders* do jogo.

Foram confeccionados dois *shaders*: um para transições de tela e o outro para *feedback* visual a perda de vidas do jogador. Para o *shader* de transição foi criado uma imagem preta que em posição fica a frente de todo o resto do jogo.

A este objeto foi atribuído o *shader graph* da Figura 21. Os *node* UV se refere as coordenadas da textura do objeto a que foi atribuído, já o Distance juntamente com o Vector 2 faz com que o ponto do UV se desloque para o centro da textura, criando uma forma circular em seu meio.

O *node* a seguir, chamado Step, torna o resultados dos *nodes* anteriores em um valor binário (0 ou 1). A variável chamada *Progress* é a responsável por definir o valor do Step e este se conecta ao valor alpha da textura do objeto.

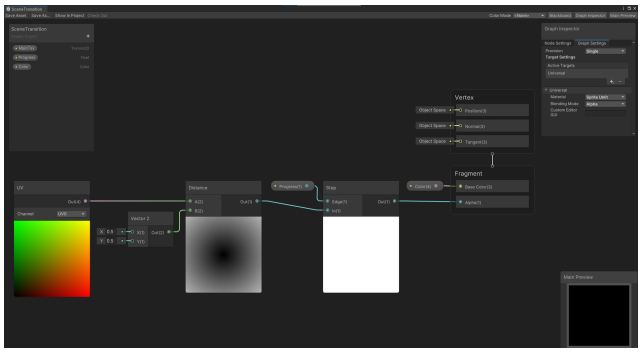

Figura 21. Shader de transição de telas.

Através do código a variável *Progress* pode ser acessada e alterada. Assim, quando há mudança de cenas, ao fazer com que este valor aumente do 0 até o 1, como pode ser visto na Figura 22, há o efeito de revelação do que está atrás da imagem preta, do centro até as bordas.

Caso *Progress* comece em 1 e vá até 0, ocorre o oposto. A imagem começa a escurecer a telas das bordas até o centro, ocultando todo o resto.

Figura 22. Shader de transição em execução.

O segundo shader foi atrelado ao personagem gato, com ele, cada vez que o jogador colidisse com um obstáculo e perdesse uma vida o gato ficaria vermelho por um breve período para indicar esse dano.

Apesar dos muitos *nodes*, como pode ser visto na Figura 23, o funcionamento básico é: colocar uma camada de cor vermelha em cima da textura padrão do gato com alguma transparência, para que ainda seja possível visualizar a arte abaixo dessa camada.

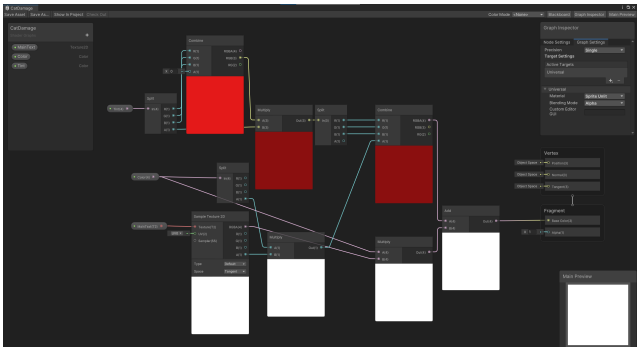

Figura 23. Shader de efeito de dano.

Na Figura 24 encontra-se o aspecto final deste *shader* durante o jogo. Este efeito aparece no exato momento da colisão com obstáculo e dura poucos segundos antes de desvanecer.

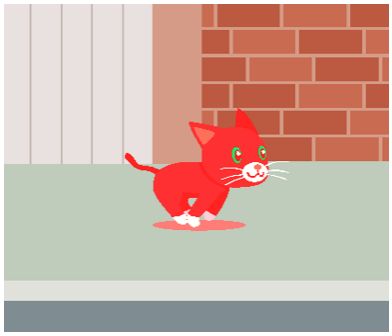

Figura 24. Shader de efeito de dano em execução.

Com o acréscimo dos *shaders* e algumas outras implementações como criação da tela de resultados finais, implementação do modo infinito, balanceamento pós testes das fases e adição de música, o jogo atingiu o fim de mais um ciclo de produção. Uma nova rodada de testes foi então efetuada antes do prosseguimento do desenvolvimento.

O último ciclo de desenvolvimento focou no armazenamento de dados, utilizando tanto o sistema de PlayerPrefs da Unity quanto o JSON para isto.

O PlayerPrefs foi usado para guardar um *string* que diz se o jogador já completou a fase anterior, caso o tenha feito, então a fase seguinte é liberada, do contrário, o botão da fase seguinte fica inacessível e com um pequeno ícone de cadeado ao seu lado para representar esse bloqueio.

Outro uso do PlayerPrefs foi para guardar o volume escolhido pelo jogador, assim evitando que o mesmo tenha que alterá-lo a cada vez que for jogar. Para isso, foi criado um botão que se encontra tanto no menu principal, assim como no menu de pausa durante o jogo e também na tela de resultados final, possibilitando a mudança de volume a qualquer momento do jogo.

O JSON foi utilizado para salvar o nome do jogador e sua pontuação no modo infinito. Durante a tela de resultados o código na Figura 25 é chamado e as informações necessárias são passadas, caso já exista um arquivo com informações anteriores este é lido e as novas informações são acrescentadas a essa lista já existente, então tudo é serializado novamente.

```
if (!File.Exists(jsonFile))
{
    List<PlayerData> list = new List<PlayerData>();
    list.Add(player);

    string jsonContent = JsonUtility.ToJson(list);

    using (System.IO.StreamWriter file = new System.IO.StreamWriter(jsonFile))
    {
        file.WriteLine(jsonContent);
    }
}
else
{
    string fileContent = File.ReadAllText(jsonFile);

    PlayerDataList list = new PlayerDataList();

    list = JsonUtility.FromJson<PlayerDataList>(fileContent);
    list.data.Add(player);

    PlayerDataList listOfData = new PlayerDataList();
    listOfData.data = list.data;

    string jsonContent = JsonUtility.ToJson(listOfData);

    using (System.IO.StreamWriter file = new System.IO.StreamWriter(jsonFile))
    {
        file.WriteLine(jsonContent);
    }
}
```

Figura 25. Código para salvar informações.

Para a apresentação dos recordes é realizado a leitura do conteúdo do JSON que nada mais é do que uma lista contendo o nome e a pontuação do jogador. Esta lista é então passada para outro *script* que realiza a ordenação decrescente e então exibe os cinco maiores recordes.

Figura 26. Tela de exibição dos cinco melhores recordes.

Para finalizar, também foram acrescentados efeitos sonoros ao jogo para quando se é clicado nos botões e ao pegar itens durante o jogo, assim como, um polimento em etapas anteriores. Com isso o jogo atingiu a conclusão de seu desenvolvimento, após sua última fase de testes para correção de bugs nada mais foi modificado.

C. Testes

Durante o desenvolvimento do jogo foram realizadas três etapas de testes em diferentes fases de desenvolvimento.

Cada uma contou com formulários de *feedback* focando em diferentes aspectos do jogo que necessitavam de validação no dado momento.

Em seu primeiro teste o objetivo era verificar tamanho das fontes, problemas em diferentes resoluções de tela, dificuldade das fases, erros diversos encontrados e recolhimento de sugestões.

De modo geral, todas as fases se encontravam fáceis demais nesse estágio e necessitavam mudanças para elevar sua dificuldade. No entanto, o tamanho das fontes escolhido foi constatado de boa legibilidade e sem graves problemas com diferentes resoluções de tela. Alguns erros foram reportados e então corrigidos.

Na segunda rodada de testes foi novamente pedido avaliações sobre a nova dificuldade das fases, erros encontrados durante o teste, assim como, sugestões de melhoria. Dessa vez, também foi acrescentado uma pergunta referente a dificuldade do modo infinito, se este deveria ter velocidade própria ou ter a mesma da fase mais difícil.

De acordo com os resultados obtidos, a primeira fase ainda precisava de ajustes, mas as outras duas se encontravam com uma dificuldade adequada.

Quanto a questão sobre a dificuldade do modo infinito, 66,7% votou que esta deveria ser a mesma da terceira fase. Novamente, bugs relatados foram corrigidos e sugestões foram analisadas e levadas em consideração.

Durante a terceira e última fase de testes o jogo já se encontrava em sua versão final. O objetivo desta última testagem foi o de encontrar problemas novos, assim como, àqueles que passaram sem serem notados nos testes anteriores. Após encerrada esta fase de testes, somente foram corrigidos os bugs relatados, o jogo não recebeu nenhuma grande mudança neste ponto.

D. Pós-produção

Com a conclusão do projeto foi possível entregar o prometido. Então, a versão final foi publicada no itch.io gratuitamente[2], pois o intuito deste jogo é o aprendizado e a prática de desenvolvimento, assim como um acréscimo ao portfólio.

VII. Conclusão

Cat On The Run atingiu seu objetivo, sendo um Endless Runner funcional e um bom acréscimo ao portfólio de desenvolvimento.

O uso de *shaders* provou-se útil e muito flexível, podendo ser usado para a criação dos mais diversos efeitos. Este conhecimento se mostrou conveniente e benéfico de ser utilizado em jogos.

Tanto o PlayerPrefs quanto o JSON se mostraram boas formas de armazenamento de dados, cada qual em sua proposta de uso.

O desenvolvimento deste jogo provou ser uma experiência proveitosa. Sem dúvida, os conhecimentos adquiridos com este projeto serão bem utilizados em futuros jogos desenvolvidos.

[2]https://azuryka.itch.io/cat-on-the-run

Referências

[1] Christopher Boyd. “A brief history of video game saves and data modification”. Em: (2020). Disponível em <https://blog.malwarebytes.com/cybercrime/2020/06/a-brief-history-of-video-game-saves-and-data-modification/>.

[2] C2TI. “Você sabe o que é o estilo Flat Design?” Em: (2018). Disponível em <https://c2ti.com.br/blog/voce-sabe-o-que-e-o-estilo-flat-design-design>.

[3] Heather M Chandler. *Manual de produção de jogos digitais*. Bookman Editora, 2009.

[4] Fabrizio Espíndola e Pablo Yeber. *The Unity Shaders Bible*. Jettelly, 2021.

[5] Freya Holmér. *Shader Basics, Blending Textures • Shaders for Game Devs [Part 1]*. 2021. URL: https://www.youtube.com/watch?v=kfM-yu0iQBk&t=1001s (acesso em 08/03/2022).

[6] Arthur M. “O que é Flat Design? Vale a pena usar em Jogos?” Em: (2022). Disponível em <https://outofmeta.com/o-que-e-flat-design-vale-a-pena-usar-em-jogos-bonus-no-final/>.

[7] Nurzhan Nurseitov et al. “Comparison of JSON and XML data interchange formats: a case study.” Em: *Caine* 9 (2009), pp. 157–162.

[8] Simon Parkin. “Don’t Stop: The Game That Conquered Smartphones”. Em: (2013). Disponível em <https://www.newyorker.com/tech/annals-of-technology/dont-stop-the-game-that-conquered-smartphones>.

[9] The Strong National Museum of Play. “Saving in Video Games”. Em: (2011). Disponível em <https://www.museumofplay.org/2011/07/14/saving-in-video-games/>.

[10] Ítalo Tobler Silva e Rafael Miranda Lopes. “Serialização e Save”. Em: (2020). Disponível em <https://edisciplinas.usp.br/pluginfile.php/5843488/mod_resource/content/1/Aula 18 - Serialização e Save.pdf>.

[11] Unity. “PlayerPrefs”. Em: (2020). Disponível em <https://docs.unity3d.com/ScriptReference/PlayerPrefs.html>.

[12] Unity. “Shader Graph”. Em: (2018). Disponível em <https://unity.com/pt/features/shader-graph>.

[13] Unity. “Unity-Manual: Shaders”. Em: (2018). Disponível em <https://docs.unity3d.com/Manual/Shaders.html>.