Far Away From Home: Jogo Plataforma com Localização Baseada em Sons

Gabriel B. Buffon[1], Rafael Rieder[1]

[1] Curso de Ciência da Computação, UPF, Campus 1 - BR 285 - Passo Fundo (RS) - Brasil

{152072@upf.br, rieder@upf.br}

Resumo. *Este artigo descreve a implementação de um jogo experimental baseado em sons. Este jogo tem como objetivo mostrar a técnica citada em um jogo plataforma. Para sua implementação foi utilizado a engine Unity 2017.4.0, com a linguagem C# para os scripts. Como resultado tem-se um jogo onde o cenário é exibido na tela do jogador somente quando o personagem, controlado por ele, emite som durante a caminhada no ambiente virtual, em ações de andar, pular e cair - além do disparo de sons ambientes.*

Palavras-chave: Desenvolvimento de jogos. Jogo baseado em sons. Unity.

Abstract. *This paper describes the development of an experimental Audio-based game. The aim of this game is to show the cited technique in a platform game. For its implementation we used Unity 2017.4.0 engine, with C# language for scripts. As a result, we have a game where the scenario is shown on the player's screen only if the character, controled by him, produces sound in its walk on the virtual environment, in actions like walking, jumping and landing - along with ambient sound triggers.*

Keywords: Game development. Audio-Based game. Unity.

1. Introdução

O mercado de jogos *indies* tentam explorar técnicas diferenciadas e desenvolver novas experiências para jogadores, em noticia escrita por Fernandes (2015) "os jogos independentes podem arriscar e ousar fazer o que nenhum estúdio fez antes". Geralmente, eles primam por projetos de baixo orçamento, estimulando o programador a ser criativo e chamar atenção do público pela interface gráfica e pela jogabilidade da aplicação.

A quantidade de materiais disponíveis livremente para auxiliar novos programadores a entrar nesse ramo resultou em diversos jogos que oferecem experiências diferentes dos jogos AAA, "o termo AAA é usado para classificar aqueles jogos feitos por grandes empresas que gastam bastante dinheiro no marketing e criação de seu jogo", escrita por Cardoso (2015). Por isso, os jogos independentes têm que abusar da criatividade e inovação para ganhar espaço no mercado, preenchendo lacunas deixadas pelas grandes desenvolvedoras de games [Fernandes 2015].

Nesse contexto, uma das técnicas que vem sendo trabalhadas pelas pequenas indústrias são jogos baseados em áudio (*audio-based game*). Este tipo de jogo é uma aplicação que explora feedbacks aural ou tátil durante o processo interativo, em detrimento ao feedback visual. De acordo com Rovithis (2012), sua origem é voltada para pessoas com problemas de visão, porém novas soluções para o público em geral têm sido ofertadas por artistas de som, pesquisadores de jogos de acessibilidade e desenvolvedores de dispositivos móveis.

Um *audio-based game* conhecido é o Lurking (2014), um jogo com temática de terror psicológico. Nele, o objetivo do jogador é chegar até o fim do cenário somente se orientando por sons para mostrar o ambiente em sua volta. Além disso, o jogador deve evitar um inimigo que localiza o personagem pelo som emitido decorrente de sua movimentação, interações com o ambiente e seu microfone (caso seja configurado).

Outro jogo é o One Hand Clapping (2018), uma plataforma musical que exige do jogador que cante para resolver quebra-cabeças. Ele utiliza o som captado pelo microfone para modificar o cenário e em desafios de timbre de voz, permitindo o progresso no jogo.

Nesse contexto, o objetivo desse trabalho é explorar a mecânica de sons em jogos plataforma do tipo Side-Scroller para a visualização do cenário. Para sua implementação foi utilizado a engine Unity 2017.4.0, com a linguagem C# para os scripts.

O artigo está estruturado em cinco seções. A Seção 2 apresenta trabalhos relacionados. Na Seção 3, são apresentados os materiais e métodos utilizados. Na Seção 4 é descrita a implementação da aplicação desenvolvida e seu funcionamento. A Seção 5 conclui o artigo.

2. Trabalhos Relacionados

A abordagem proposta por esse trabalho tem relação com os jogos Lurking e One Hand Clapping, e toma por base o artigo Sonic Mechanics: Audio as Gameplay (Oldenburg, 2013), que destaca que o projeto de arte de som deve estar fortemente relacionado a jogabilidade de um projeto de *audio-based game*.

O jogo Lurking é uma experiência tridimensional onde o cenário é mostrado ao jogador apenas quando o personagem controlado pelo mesmo produz som. As formas de interação que permitem isso são: arremesso de objetos, fala em um microfone configurado e a própria movimentação do personagem. A distribuição do áudio pelo cenário alerta inimigos, que buscam localizar o jogador pelo som. O objetivo do jogador é sobreviver o máximo tempo possível, evitando inimigos.

O jogo One Hand Clapping é uma experiência plataforma Side-Scroller onde o jogador deve usar a própria voz para progredir no jogo, utilizando o timbre de sua voz em desafios presentes no cenário. Desenvolvido pela University of Southern California, o objetivo do jogo é completar os desafios para chegar ao final.

Como diferencial, esse trabalho propõe um jogo que dá ao jogador uma experiência Side-Scroller, em conjunto com a aplicação do processo interativo baseado em áudio. A Figura 1 apresenta uma representação geral de cena do jogo intitulado “Far Away From Home”.

Figura 1. Representação geral do projeto desenvolvido.

3. Materiais e Métodos

Para a concepção desse trabalho, foram empregadas as seguintes ferramentas: Unity, uma engine principalmente de criação de jogos; Piskel, uma solução on-line para a criação de *sprites* e animações de 8-bits ou similares (Figura 2); e o Twistedwave, um editor de sons on-line.

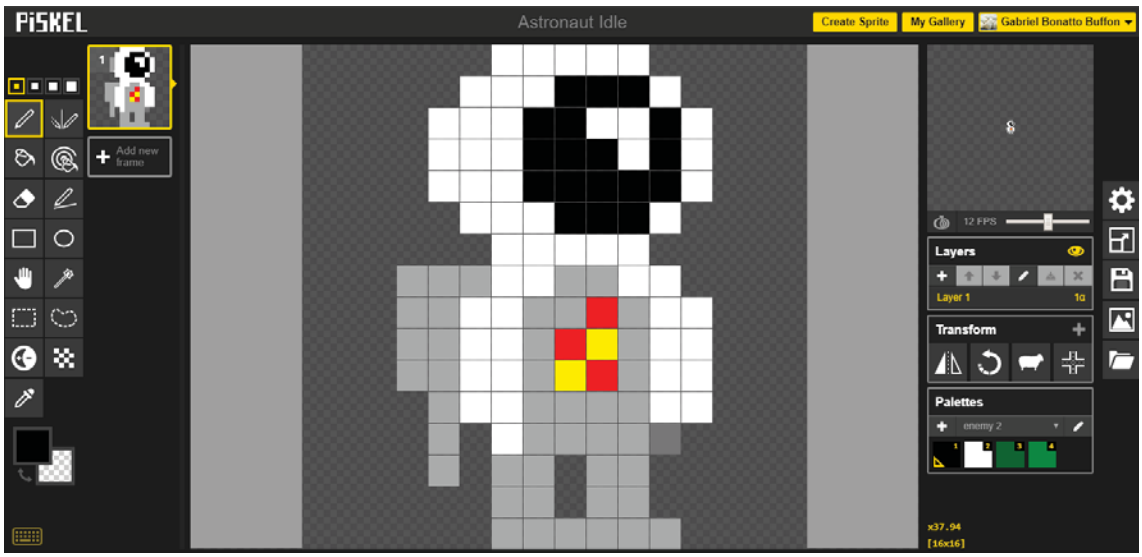

Figura 2. Demonstração da ferramenta online Piskel.

A configuração do computador utilizado para a criação e testes do projeto contém o processador Intel Core I5-4210U, 4GB de memória RAM e placa gráfica dedicada GeForce GT 740M.

Os principais métodos utilizados no projeto foram as técnicas Side-Scroller, onde em um jogo plataforma o personagem se encontra no centro da tela e a movimentação do personagem move, em sua maior parte, o cenário; e a técnica *Audio-Based*, onde o projeto

transforma o som em uma mecânica própria do jogo, podendo se utilizar do som produzido dentro ou fora do jogo.

4. Implementação

A utilização da Unity exigiu a criação de scripts em linguagem C# para o funcionamento do jogo, além de artes pixeladas feitas especificamente para o jogo com o auxílio de editores on-line. Empregou-se também sons sem copyright e diversos guias básicos disponíveis abertamente.

Ao criar uma *build* (gerar o jogo) a partir de um projeto Unity, é necessário definir cenas. Neste projeto, foram implementadas três cenas: o menu principal (cena 0), uma tela de carregamento (cena 1) e a cena principal do jogo (cena 2).

4.1. Comandos

A Tabela 1 a seguir mostra os comandos definidos no software a partir de scripts para navegação no jogo.

Tabela 1. Definição de navegação no jogo "Far away from home".

Teclas	*Comando*
Setas Direcionais (Esquerda e Direita)	Movimentação para Esquerda e Direita
Barra de Espaço	Pular
Tecla ESC	Voltar para o Menu Principal

4.2. Menu Principal

O menu principal é a cena principal do software, escolhida como cena 0 durante a exportação. O menu consiste de um fundo preto, um *canvas* com os botões "Play", "Help" e "Quit" dentro, um *sprite* pixelizado do planeta Terra, o título do jogo escrito com a ferramenta Text e um efeito de partículas no fundo. O menu pode ser visto na Figura 3, que toma por base um *Asset Pack* distribuído oficialmente pela Unity.

O botão "Play" inicia a cena 1, de fundo preto com um *Sprite* no canto inferior esquerdo escrito "LOADING...". Após iniciar esta cena, o jogador imediatamente inicia a cena 2, a do jogo principal, que pode ser vista na Figura 4.

A criação da cena 1 foi feita para demonstrar ao usuário que o jogo estava realmente carregando e não travado na tela inicial, pois o carregamento da cena 2 requer mais tempo por ser mais complexa, sem a cena 1 o jogo ficaria travado na tela inicial até o carregamento estar concluído.

O botão "Help" descarrega o *canvas* do menu principal, carregando outra tela com informações sobre créditos do jogo e os comandos básicos (Figura 5), além de um botão Back, que descarrega este *canvas* e recarrega o anterior.

O botão "Quit" descarrega a cena e fecha o jogo no caso de uma *build* final. No caso de uma *build* de demonstração na plataforma Unity, o botão Quit simplesmente termina a execução do teste.

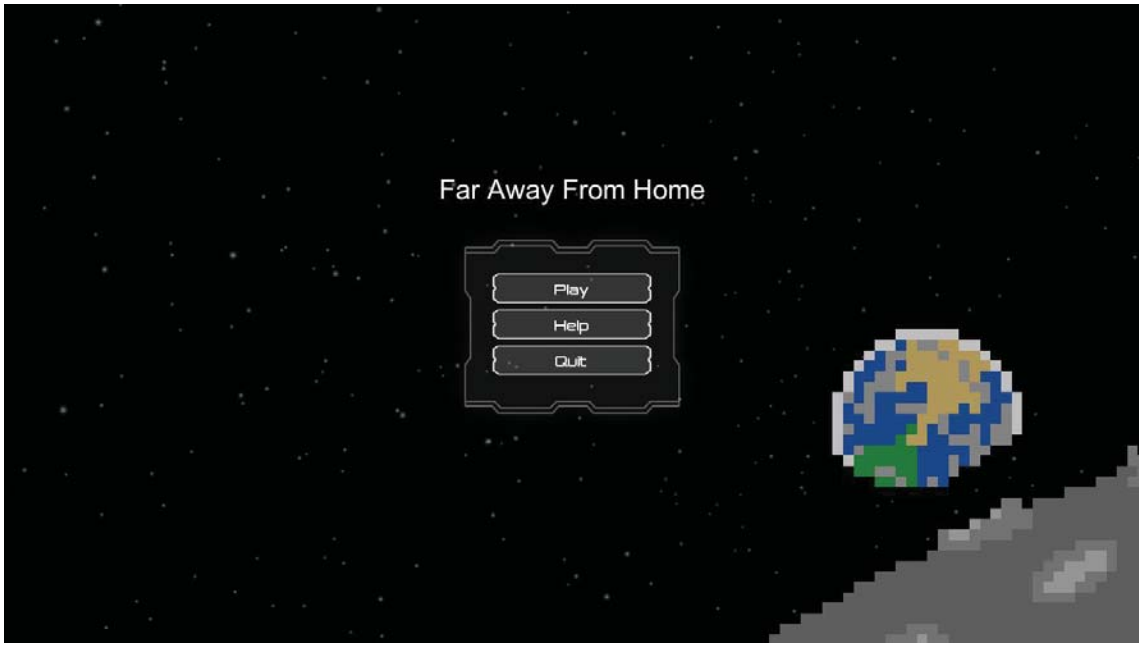

Figura 3. Captura de tela do menu principal.

Figura 4. Captura de tela da tela de carregamento.

Figura 5. Captura de tela do menu de ajuda.

4.3. Jogo Principal

A cena 2 carrega todo o cenário de jogo, bem como o personagem a ser controlado. Scripts e triggers são ativados para iniciar o processo interativo.

Para a movimentação do personagem, ao pressionar qualquer tecla direcional horizontal, um script rodando em background detecta o pressionamento da tecla e realiza a movimentação desejada. O mesmo ocorre para a ação de pular e apertar a tecla ESC para retornar ao menu.

Conectado ao jogador, segue uma iluminação de forte intensidade posicionada próxima ao chão. A luz emanada muda o raio de difusão dinamicamente dependendo das ações do jogador (parado ou andando), como pode ser visto na Figura 6 e na Figura 7. Para tanto, utiliza-se uma função que calcula um valor aleatório (seguindo um limite mínimo, gerando um valor que simula um gráfico de ondas). Este valor é somado ao raio predefinido da luz para se obter um raio dinâmico (o raio predefinido varia de acordo com a ação sendo realizada ou estado do personagem). Dessa forma, evita-se o uso de luz difusa fixa.

O objetivo do jogo é chegar até o final do cenário, tendo como desafios navegar pelo mapa e evitar um inimigo que aparece próximo ao fim do jogo.

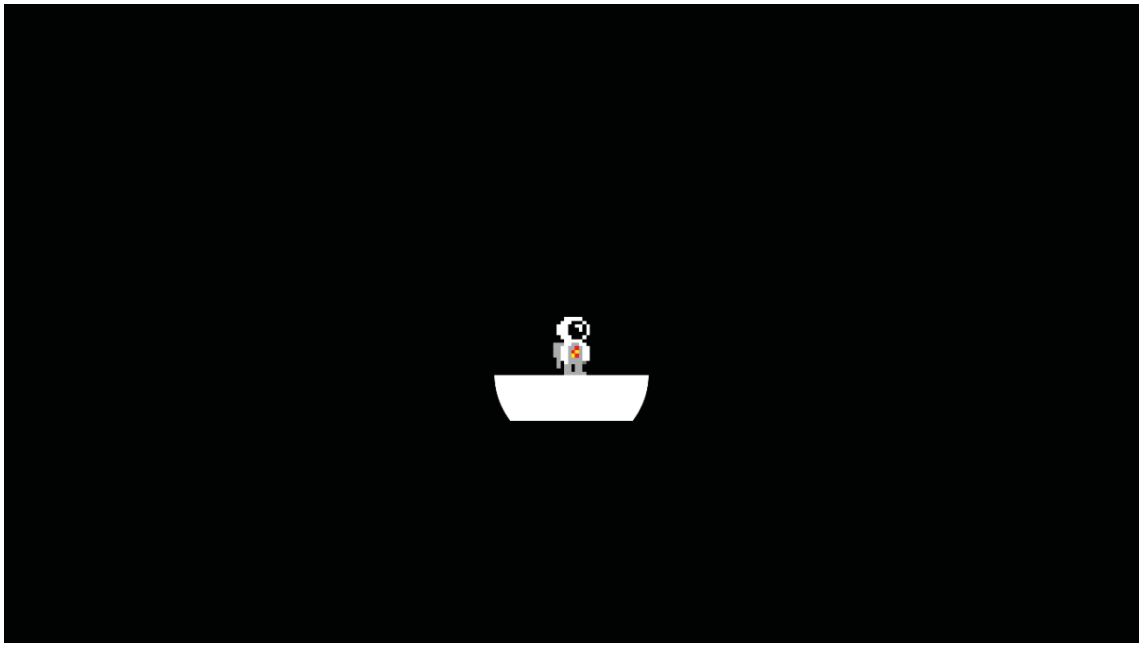

Figura 6. Demonstração da luz com o personagem parado.

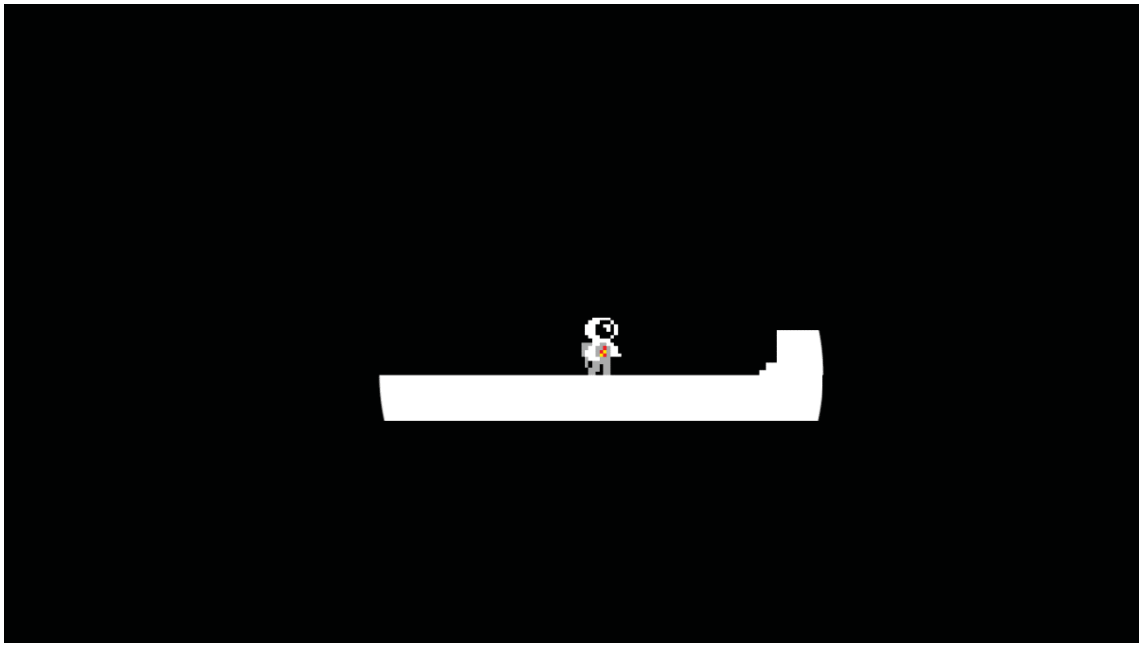

Figura 7. Demonstração da luz com o personagem em movimento.

4.4. Cenário

O mapa do jogo foi construído a partir de blocos tridimensionais, *sprites*, fontes de iluminação do tipo Spotlight e triggers (Figura 8 e Figura 9), distribuídos dessa forma: blocos utilizados como chão, paredes e teto do cenário; *sprites* para o próprio personagem controlado pelo jogador, o inimigo, objetos decorativos e textos; as fontes de luz Spotlight no próprio personagem, para *sprites* decorativos, e em certas áreas do jogo; e triggers em áreas do jogo (percursos que o jogador pode navegar).

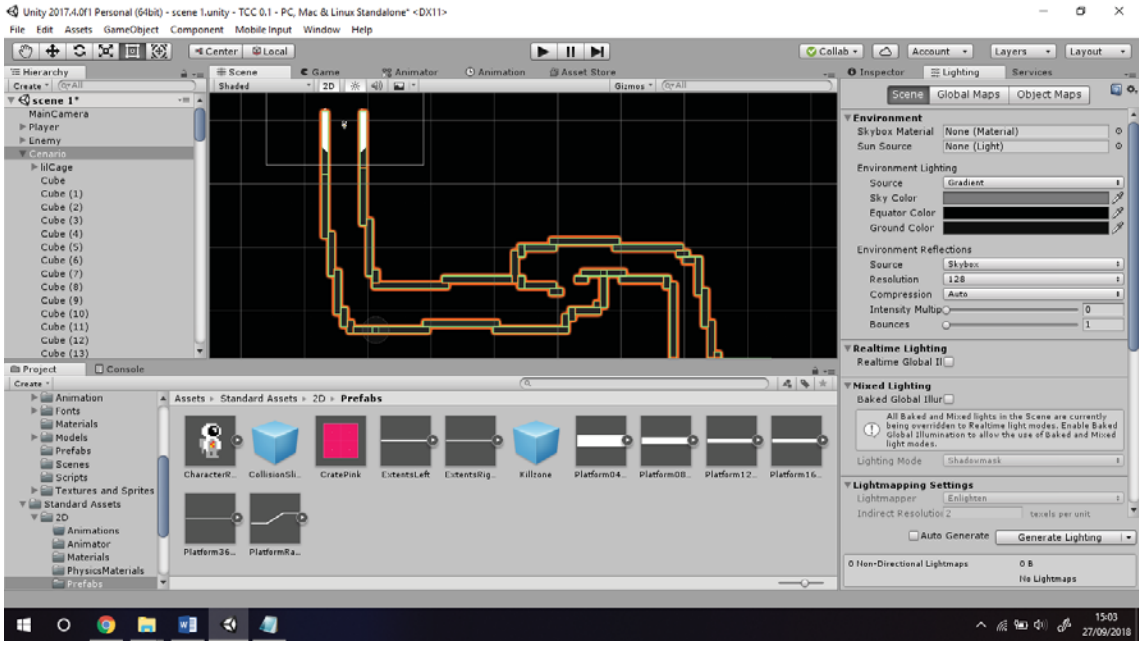

Figura 8. Construção do cenário.

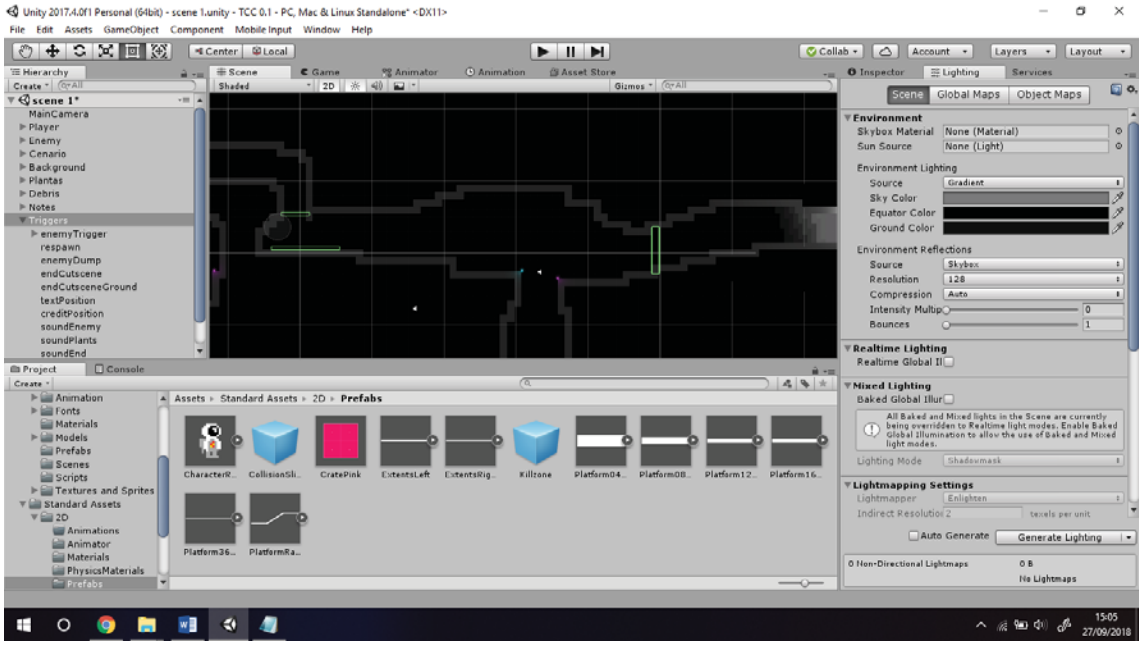

Figura 9. Duas áreas com Triggers.

4.5. Inimigo

O inimigo (Figura 10) aparece por um trigger, que está representado à esquerda da Figura 9. Utiliza-se inteligência artificial simples para o inimigo seguir a posição onde o personagem se encontra. Ao encostar no jogador, o personagem é transportado para a área anterior ao encontro, como uma espécie de checkpoint, e o inimigo é enviado para uma área externa do mapa. A cada acionamento da trigger, um novo inimigo é criado.

4.6. Final do Jogo

Ao finalizar o cenário (Figura 11), existe uma trigger que inicia um diálogo com o jogador, travando o personagem no local. O jogador pode usar a tecla ESC para retornar ao menu.

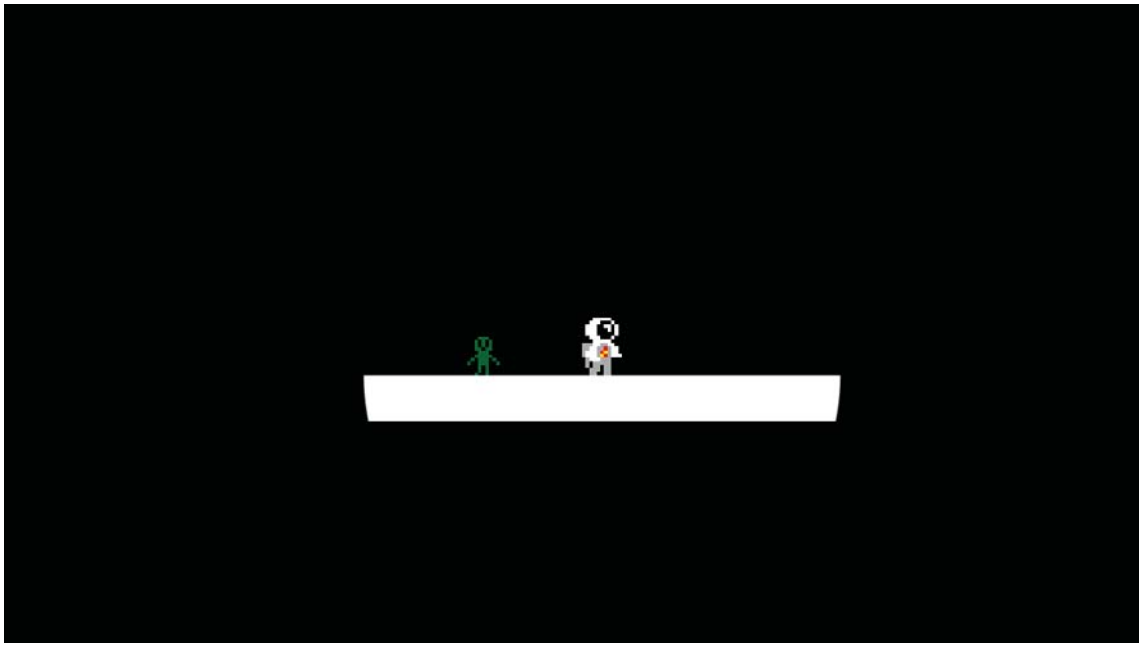

Figura 10. Captura de tela do inimigo seguindo o jogador.

Figura 11. Captura de tela da área final do jogo.

4.7. Scripts Utilizados

Foram utilizados 12 scripts, sendo quatro nativos do Asset Pack que é providenciado gratuitamente pela Unity. Dois destes scripts estão relacionados a movimentação do personagem, e foram modificados para esta abordagem. Incluíram-se variáveis para sons,

luzes e valores utilizadas no código, e funções que atualizam o raio da luz presente no pé do jogador e que reproduzem os sons de andar, pousar e pular. Os códigos apresentados nas Figura 12 e Figura 13 apresentam essas funcionalidades.

```
50          private void Update()
51          {
52              // land light
53              if (counterj == 1)
54              {
55
56                  soundjump = true;
57                  lightJump.range = lightJump.range - (float)0.2;//7;
58                  if (lightJump.range < 1)
59                  {
60                      counterj = 2;
61                  }
62              }
63              if (m_Grounded && counterj ==0)
64              {
65                  landing.Play();
66                  lightJump.range = (float)7;
67                  landJump = true;
68                  counterj=1;
69              }
70              if (!m_Grounded)
71                  counterj = 0;
72              //------------------------------------------------------
```

Figura 12. Código que atualiza o raio da luz no pouso.

```
73              // jump light
74              if (!test && counter == 1)
75              {
76                  soundjump = false;
77                  lightJump.range = lightJump.range - (float)0.2;//7;
78                  if (lightJump.range < 1)
79                  {
80                      counter = 0;
81                  }
82              }
83              if (test)
84              {
85                  counter = 1;
86                  lightJump.range = (float)8;//7;
87                  test = false;
88              }
89
90              if(soundjumpground)
91                  footsteps.UnPause();
92              else
93                  footsteps.Pause();
94
95          }
```

Figura 13. Código que atualiza o raio da luz quando o personagem pula.

Foram utilizados dois scripts para o travamento do personagem no final do jogo em conjunto com a exibição de um texto. O primeiro desabilita os dois scripts de controle do personagem (Figura 14), e o segundo interrompe o som de movimento. Eles marcam o personagem como estando no chão e com velocidade zero, e transportam o texto que posteriormente estava em uma área não alcançável pelo jogador para uma parte visível da tela (Figura 15).

Existem três scripts utilizados pelo inimigo, um para sua inicialização no cenário (Figura 16), outro para seguir o jogador (Figura 17) e outro executado quando o inimigo encosta no jogador (Figura 18). A inicialização é feita a partir de uma trigger, que instancia o inimigo e inicia o efeito sonoro de fundo. Para seguir o jogador, o script pega a posição do personagem e ordena o inimigo a ir até a posição do mesmo. Como o script é atualizado a cada frame, o inimigo sempre se desloca até a posição atual do jogador. Ao encostar no jogador, aciona-se o script que transporta o jogador para uma área anterior ao encontro, e para ativar os efeitos sonoros de fundo. Encontra-se também todos os objetos com a tag “enemy”, transportando-os para uma área fora do alcance e visão do jogador.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityStandardAssets._2D;
5
6  public class EndCutscene : MonoBehaviour {
7
8      public AudioSource footsteps;
9
10     void OnTriggerEnter2D(Collider2D other)
11     {
12         footsteps.Pause();
13         GameObject.Find("Player").GetComponent<Platformer2DUserControl>().enabled = false;
14         GameObject.Find("Player").GetComponent<PlatformerCharacter2D>().enabled = false;
15     }
16 }
17
```

Figura 14. Função trigger que desabilita os scripts que controlam a movimentação.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class End2 : MonoBehaviour {

    public Animator m_Anim;
    public GameObject textPosition;
    public GameObject creditPosition;
    public GameObject text;
    public GameObject credit;
    GameObject gameObjecttext;
    GameObject gameObjectcredit;
    public AudioSource footsteps;

    void OnTriggerEnter2D(Collider2D other)
    {
        footsteps.Pause();
        gameObjecttext = GameObject.FindGameObjectWithTag("text");
        gameObjectcredit = GameObject.FindGameObjectWithTag("credit");
        m_Anim.SetBool("Ground", true);
        m_Anim.SetFloat("Speed", (float) 0.0);
        text.transform.position = textPosition.transform.position;
        credit.transform.position = creditPosition.transform.position;
    }
}
```

Figura 15. Script trigger que finaliza animações e transporta o texto.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TriggerEnemy : MonoBehaviour {

    public Transform spawnPoint;
    public AudioSource plant;
    public GameObject enemy;
    public static int flag = 0;

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player")&&flag==0) {
            Instantiate(enemy, spawnPoint.position, spawnPoint.rotation);
            plant.Play();
            flag = 1;
        }
    }
}

```

Figura 16. Script trigger que instancia o inimigo e desabilita a criação de outros.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyFollow : MonoBehaviour {

    public float speed;
    public float stoppingDistance;

    private Transform target;

    // Use this for initialization
    void Start () {
        target = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
    }

    // Update is called once per frame
    void Update () {
        if (Vector2.Distance(transform.position,target.position)>stoppingDistance)
        {
            transform.position = Vector2.MoveTowards(transform.position, target.position, speed * Time.deltaTime);
        }

    }
}

```

Figura 17. Script de inteligência artificial que segue o jogador.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TriggerGameOver : MonoBehaviour {

    public GameObject respawn;
    public GameObject enemyDump;
    public AudioSource background;
    public AudioSource end;
    public AudioSource plant;
    GameObject[] gameObjects;

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            TriggerEnemy.flag = 0;
            other.transform.position = respawn.transform.position;
            background.Stop();
            end.Stop();
            plant.Stop();
            gameObjects = GameObject.FindGameObjectsWithTag("enemy");

            for (var i = 0; i < gameObjects.Length; i++)
                gameObjects[i].transform.position = enemyDump.transform.position;
        }
    }
}

```

Figura 18. Script que é ativado no contato entre o jogador e o inimigo.

Os outros três scripts utilizados são para o retorno ao menu principal com a tecla ESC, o funcionamento da tela de carregamento e o acionamento dos sons.

4.8. Sprites, Animações e Materiais

Os *sprites* foram criados com a ferramenta web Piskel, utilizada para o desenho de 8-bits ou similares. Após a exportação dos *sprites*, todos foram incorporados ao projeto na Unity (Figura 19) – porém, nem todos foram utilizados na versão final.

As animações também foram desenhadas utilizando o Piskel, e posteriormente animadas na Unity com a ferramenta Animator (Figura 20). Em relação a materiais, criaram-se opções com a própria Unity (Figura 21) – porém, nem todas foram aplicadas na versão final.

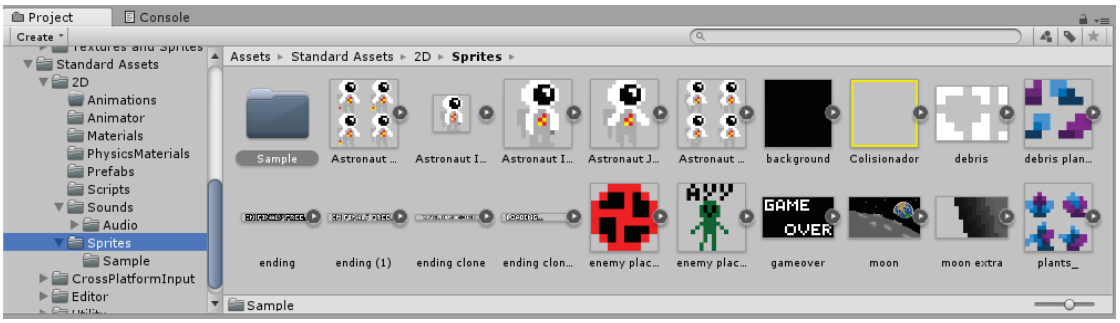

Figura 19. Captura de tela dos sprites utilizados no projeto.

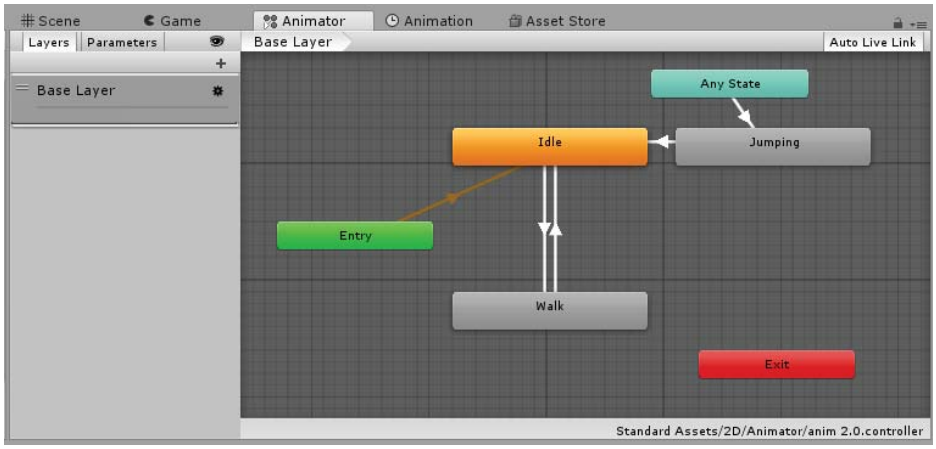

Figura 20. Captura de tela do animator presente na Unity.

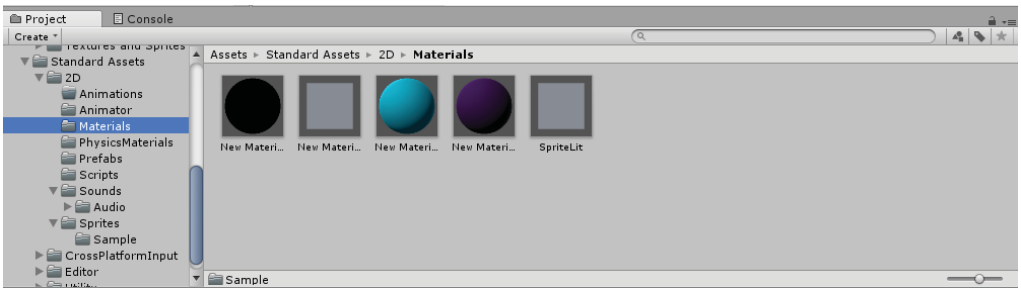

Figura 21. Captura de tela dos materiais criados com a Unity.

4.9. Produto Final

O resultado deste projeto é um jogo experimental, *Side-Scroller*, e que utiliza técnicas de jogos *Audio-Based*, com o foco na interação por sons produzidos no jogo. A Figura 22 demonstra uma área específica do jogo com detalhes do cenário. Pode-se notar a iluminação que irradia do pé do personagem a medida que ele se desloca pelo cenário, além da iluminação ambiente.

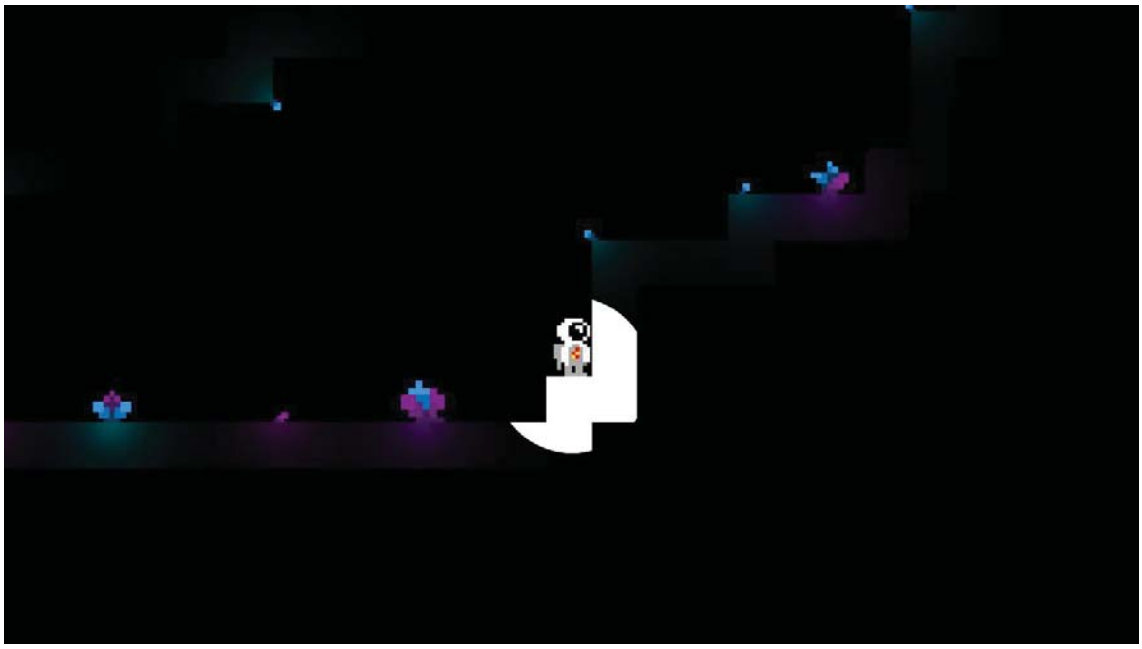

Figura 22. Captura de tela do jogo, destacando a iluminação ambiente.

5. Considerações Finais

Este trabalho teve como objetivo criar um jogo que ofereça uma experiência diferenciada ao jogador, utilizando conceitos de *audio-based game* em um jogo plataforma Side-Scroller.

O projeto foi criado com o uso da Unity e ferramentas on-line para criação de imagens (Piskel), edição de sons (TwitstedWave) e depósitos de sons royalty-free. Na Unity foram usados scripts C# para a criação de funções.

O diferencial deste projeto foi apresentar um jogo que utiliza, em conjunto, as técnicas de *Side-Scrolling* e *Audio-Based* com o objetivo de oferecer uma nova experiência de jogabilidade ao usuário final.

Uma dificuldade encontrada foi o de aprendizado da ferramenta Unity. No caso de uma aplicação, é necessário implementar diversos scripts para o acontecimento de eventos. Muitos materiais disponíveis e interessantes para aprendizado possuem documentação própria, fechadas ao público. A base obtida a partir de pesquisas e documentação da comunidade ainda é fraca para o tipo de aplicação proposto. Outra dificuldade foi encontrar exemplos de jogos e artigos que implementam a mecânica de jogos baseados em áudio que utilizam o recurso de *echo-localization*.

Cabe lembrar que o projeto em questão tem como objetivo ser uma demonstração, e não um jogo completo. Como trabalhos futuros, sugere-se inicialmente a melhoria de dinâmicas como o travamento do personagem caso ocorra uma movimentação em direção a parede quando em contato com ela. Outro ponto a ser melhorado é, futuramente, tornar a aleatoriedade da iluminação mais suave em relação a movimentação do personagem, que poderá deixar o jogo mais atraente.

Referencias

Fernandes, Ruan (2015) “A importância dos jogos independentes para a indústria dos games”, https://www.gameblast.com.br/2015/08/importancia-jogos-independentes-industria-games.html.

Cardoso, Bruna (2017) “A importância dos jogos independentes para a indústria dos games”, http://seugame.com/o-que-e-um-jogo-aaa/.

Lurking (2014) “Lurking Game”, Digipen Institute of Technology Singapore, https://www.lurking-game.com/.

One Hand Clapping (2018) “USC Games – One Hand Capping”, University of Southern California, https://games.usc.edu/one-hand-clapping/

Rovithis, Emmanouel (2012). "A Classification of Audio-Based Games in terms of Sonic Gameplay and the introduction of the Audio-Role-Playing-Game: Kronos". *Proceedings of the 7th Audio Mostly Conference: A Conference on Interaction with Sound*. Corfu, Greece: ACM New York, NY, USA. pp. 160–164. doi:10.1145/2371456.2371483

Oldenburg, Aaron. (2013). Sonic Mechanics: Audio as Gameplay. Game Studies: The International Journal of Computer Game Research. 13:1