
Como se ataca
neste jogo?

Inicio dia 08 de abril – Fim TBA

Criado por: redofatiav

Duração aproximada até ao momento: 19h20

abril 2025 – maio 2025

Índice

Produção

08 de abril de 2025

Tempo Dedicado: Aproximadamente 1h30min a 2h

O que Consegui Fazer:

- Consegui encontrar um asset que possa começar a desenvolver um jogo plataformas 2D.

Problemas Encontrados:

- Muitos assets com scripts no mesmo que não eram permitidos para o pedido pelos professores.

Notas para o Próximo Dia:

- Tentar perceber como fazer um nível que dure cerca de 1 minuto.

14 de abril de 2025

Tempo dedicado: Aproximadamente 1h30min a 2h

Objetivo do Dia:

Integrar mecânicas básicas para início do protótipo do jogo de plataformas 2D.

O que Consegui Fazer:

- Montagem de tilemap grande (meio minuto de travessia).
- Implementação de movimento horizontal e salto do personagem.
- Correção de problemas de física no tilemap (não ficar preso ao chão).
- Criação de limites de mapa laterais.
- Inserção de fundo com efeito de paralaxe.

Problemas Encontrados:

- Física inicial fazia o personagem travar no tilemap.
- Tentei adicionar animações básicas (correr/parar) mas não consegui fazer funcionar corretamente.

Soluções Aplicadas:

- Ajustes nas colisões do tilemap + física.
- Sobre as animações: registado para estudar melhor na próxima sessão.

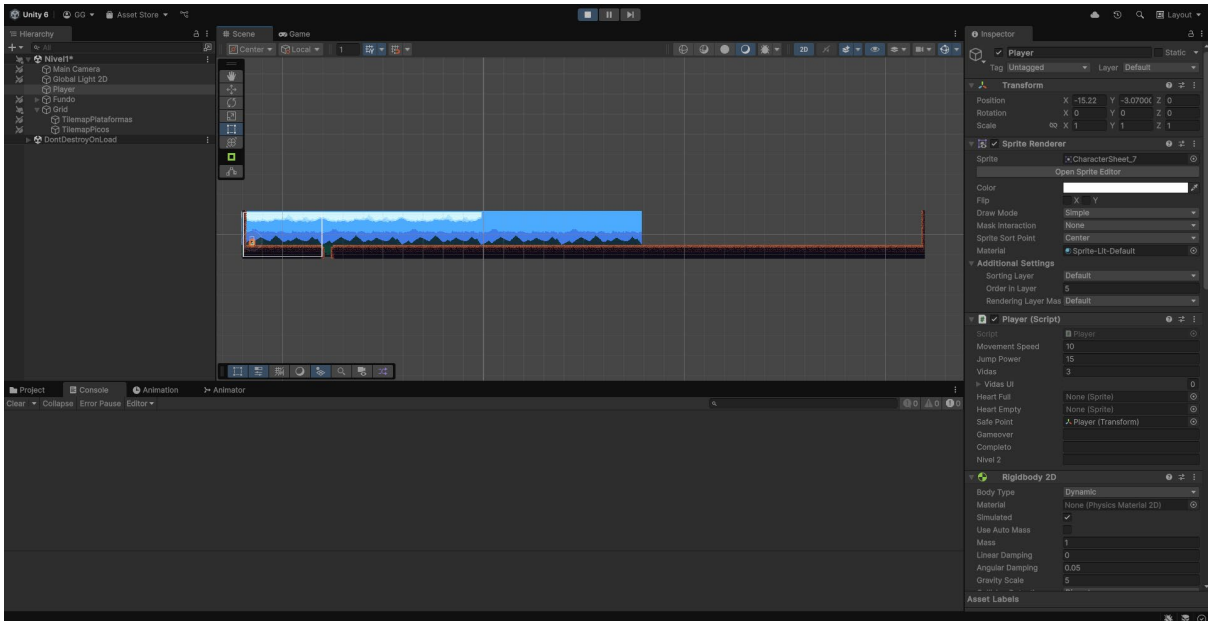
Reflexão Pessoal:

- Apesar de já ter vários elementos prontos de outro projeto, integrar e adaptar tudo foi desafiante e muito satisfatório.
- A dificuldade nas animações mostrou que preciso rever melhor como estruturo os states ou triggers.
- Ver o personagem correr e saltar num cenário montado motiva muito para continuar a evolução do projeto.

Notas para o Próximo Dia:

- Criar primeiras plataformas móveis.
- Resolver implementação das animações do personagem.
- Trabalhar em detalhes visuais (layers de fundo adicionais para dar mais profundidade).

Foto(s) para memoria visual:



27 de abril de 2025 Sessão da Manhã

Tempo estimado de trabalho: Aproximadamente 1h30min a 2h.

Sistema de Animações do Jogador concluído:

- Idle, correr, salto e queda funcionais no player.
- Animações sincronizadas com a física do jogador.
- Double jump implementado com nova animação de salto.
- Ajustes de gravidade para salto e queda mais naturais (Gravity Scale = 5).

Planeamento de inimigos:

- Decidido que o primeiro inimigo só irá aparecer no final do Nível 1.
- Nível 1 focado principalmente em ensinar movimentação e plataformas.

Implementação de Plataforma Móvel:

- Plataforma criada como GameObject separado.
- Movimento automático entre dois pontos configurado.
- Jogador move-se corretamente junto com a plataforma através de parenting dinâmico.

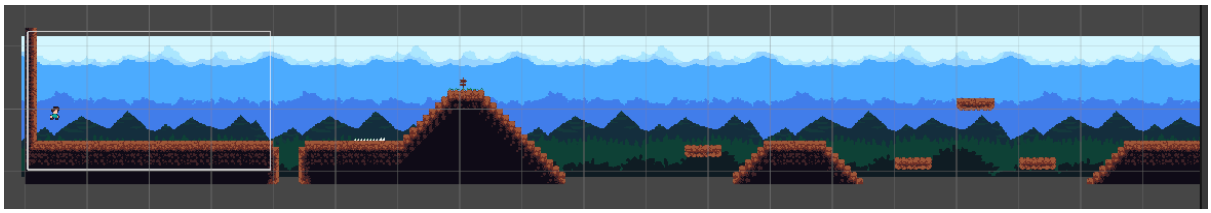
Notas para futuras melhorias:

- Atualizar o sistema de Ground Check para uso de OverlapCircle, melhorando a detecção de chão em rampas e colisões laterais.
- Explorar sistemas de Wall Slide e Wall Jump no futuro

Resumo geral:

Esta sessão focou-se em finalizar o sistema básico de movimentação e animações do jogador, resolver colisões e introduzir a primeira plataforma móvel de forma funcional e realista. O progresso foi rápido e organizado, criando uma base sólida para continuar o desenvolvimento do Nível 1.

Foto(s) para memoria visual:



27 de abril de 2025 Sessão da Tarde

Tempo estimado de trabalho: Aproximadamente 01h30

Sistema de Animações Finalizado:

- Idle, Correr, Salto e Queda do jogador totalmente funcionais tanto do inimigo como do player.
- Double Jump implementado e animação a funcionar também no segundo salto.
- Correções de animação ao aterrar em plataformas móveis ou normais.

Movimentação e Interação com Plataformas Móveis:

- Jogador move-se corretamente com plataformas horizontais.
- Corrigido o deslizar nas plataformas móveis.
- Parent dinâmico e unparent seguro para evitar erros de ativação/desativação.

Inimigos Implementados:

- Inimigos com movimentação esquerda/direita entre dois pontos.
- Inimigos saltam automaticamente ao mudar de direção ou ao tocar no chão.
- Sistema de animações de salto e queda para os inimigos (Idle, Jump e Fall).
- Sprite flip automático conforme a direção de movimento.

Sistema de Limites do Nível:

- Implementados limites horizontais que fazem o jogador perder vida e respawnar (ainda não está funcional por não ter o HUD com os corações para trocar (código aproveitado de outro trabalho)).

Checkpoints Iniciais Criados:

- Checkpoint no local inicial.
- Checkpoint no primeiro pico.
- Nenhum funcional ainda.

Organização do Projeto:

- Scripts separados por função (PlataformaLR ou UD, EnemyLR, Player (LR é Left Right e UD é Up Down)).
- Estrutura limpa e modular para facilitar expansão do projeto.

A fazer na próxima sessão:

- O jogador fica preso na animação de salto ao colidir com rampas ou degraus.
- O jogador não perde dano nem dá respawn ainda.
- Implementar o checkpoint do final do jogo.
- Pensar em adicionar coletáveis.

Prefabs até ao momento:

- Plataforma movel (direita para a esquerda, esquerda para a direita, baixo e cima e cima e baixo).
- Checkpoint.
- Fundo para usar para paralaxe.
- Inimigo.

Notas pessoais:

Consegui cumprir aqui o que ficou para fazer da última vez que toquei neste projeto. Sei que ainda está um pouco cru, mas já tem muito por onde pegar agora. Sinto que o nível 1 está um tutorial para perceber as mecânicas que utilizei no jogo.

Estou a pensar se haverá sessão da noite ou não. Apetece me jogar já que ontem organizei os outros diários todos.

28 de abril de 2025

(Não comecei mais cedo porque houve um apagão)

Tempo estimado de trabalho: Aproximadamente 1h

Objetivos do dia:

- Iniciar o desenvolvimento do HUD (vidas, pontuação).
- Criar sistema básico de score.
- Implementar sistema de vidas.
- Começar a preparar a estrutura de menus.

Tarefas realizadas:

- HUD criado no Canvas (vidas visuais com corações).
- Sistema de gestão de vidas implementado e testado.
- Implementação inicial do ScoreManager para controlar a pontuação.
- Criação de prefabs e organização da cena.
- Configuração do GameManager para gerir vidas e pausar o jogo.
- Análise e correção de erros relacionados ao HUD e score.
- Planeamento de estrutura de cenas (Menus, GameOver, Créditos).

Problemas encontrados:

- NullReferenceException relacionado ao ScoreManager quando mudava de cena sem TMP_Text.

Soluções aplicadas:

- Planeamento das tarefas para os próximos dias sem testes.

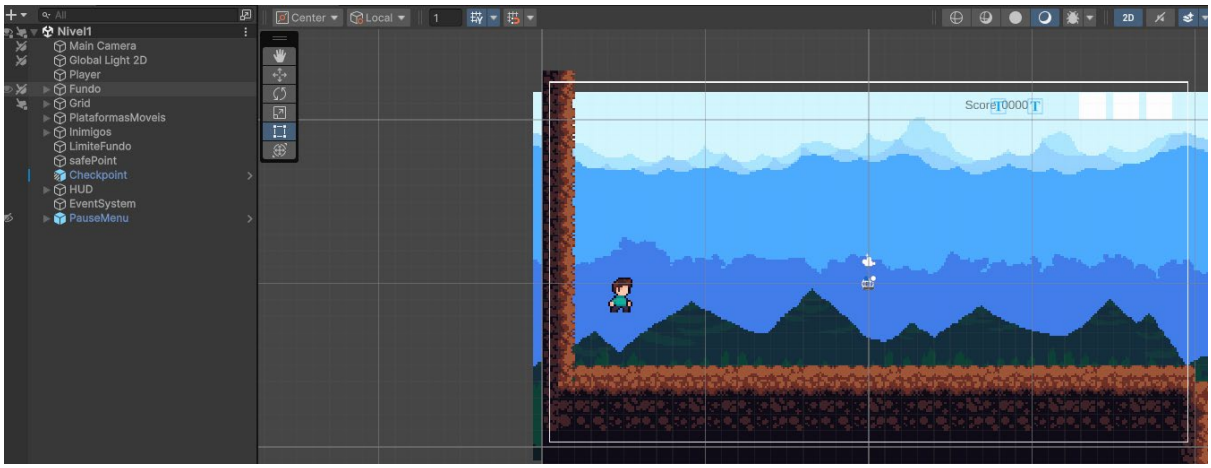
Plano para os próximos dias:

- Terça-feira: Implementar menus principais (Menu Inicial, Créditos, GameOver, Nível Completo).
- Quarta-feira: Desenvolver colecionáveis e integração no HUD.
- Quinta-feira: Construção do Nível 2 (plataformas, inimigos, colecionáveis).
- Fim de semana: Polimento e ajustes finais.
- Criar prefab do Player quando não houver erros no nível 1 para conseguir o exportar para o nível 2.

Reflexões pessoais:

Como maior parte do código é aproveitado de outro trabalho pratico da cadeira estou em dúvida se tenho de começar tudo do 0 ou se sou capaz de aproveitar o que tenho feito. Aos pouco vou dando Debug e vendo o que está a dar erro. Sinto que tenho tempo de apagar tudo e fazer do 0 com a informação que já tenho.

Foto(s) para memoria visual:



29 de abril de 2025

Tempo estimado de trabalho: Aproximadamente 1h40min

Objetivos da Sessão:

- Implementar e estabilizar todos os menus (Inicial, Pausa, Créditos, Game Over, Nível Completo).
- Corrigir bugs relacionados com a persistência do GameManager e sistema de vidas.
- Testar a transição entre cenas e criação de base sólida para o Nível 2.

Tarefas Realizadas:

Menus funcionais criados:

- Menu Inicial com botão "Novo Jogo" a resetar score e vidas, com destruição do GameManager antigo.
- Menu de Pausa ativado com tecla Esc, incluindo botões para retomar jogo e voltar ao menu.
- Menu de Game Over e Nível Completo totalmente operacionais com navegação entre cenas.
- Créditos com botão de retorno ao menu principal.

Correções e otimizações implementadas:

- Corrigido erro grave onde o jogador iniciava com 0 vidas após voltar ao menu:
    - Causa: GameManager persistente (DontDestroyOnLoad) não era destruído.
    - Solução: adição de Destroy(GameManager.instance.gameObject) ao carregar novo jogo.
- Corrigido bug onde o jogador perdia vidas automaticamente ao nascer:
    - Causa: safePoint estava mal atribuído com tag LimiteEcra que é responsável por dar dano.
    - Solução: criação de GameObject separado como ponto de início (SafePoint) e ligação manual ao Player e sem tag atribuida.

HUD de vidas estabilizado:

- Ligação correta de imagens de corações (vidasUI[]) ao Player no nível 2.
- Verificação manual de ligação de heartFull e heartEmpty.

- Debug com Debug.LogError() adicionado para diagnosticar valores null.
- Código ajustado para prevenir NullReferenceException mesmo em situações de falha de ligação.

Outros avanços:

- Criação do prefab do Player para reutilização em múltiplos níveis.
- Inserção do Player no Nível 2 com prefab funcional.
- Teste inicial da transição entre Nível 1 e Nível 2.
- Preparação da estrutura para colecionáveis no próximo dia.

Problemas Encontrados:

- Erro de referência nula ao mudar de cena com Player instanciado sem UI ligada.
- SafePoint com tag mal atribuída causava loop de morte logo ao iniciar.

Soluções Aplicadas:

- Verificação e ligação manual dos elementos necessários ao Player (UI, sprites).
- Atribuição manual de SafePoint nas cenas.
- Testes com debug em consola para identificar o ponto exato de falha.

Reflexões Pessoais:

Esta sessão foi fundamental para estabilizar o núcleo do jogo. A correção do GameManager e do sistema de vidas era crítica, e as soluções aplicadas tornaram o projeto mais robusto. Sinto que agora o fluxo completo do jogo está finalmente funcional, permitindo avançar com confiança para elementos como colecionáveis e construção de níveis adicionais. Apesar da frustração em pequenos bugs, a sensação de os resolver de forma limpa é muito gratificante. Ao tirar os prints para colocar no diário reparei que o erro que tinha nas vidas e onde perdi muito tempo é porque não tenho o GameManager ativo então não tenho como ter corações no HUD a iniciar logo no segundo nível.

Plano para o Próximo Dia:

- Implementar sistema de colecionáveis.
- Refinar transição entre cenas e preparar elementos visuais adicionais.
- Iniciar montagem do Nível 2 com layout definitivo.
- A tirar os prints para colocar no diário reparei em bugs como os picos que não dão dano, quando a personagem salta e bate lateralmente fica preso nessa

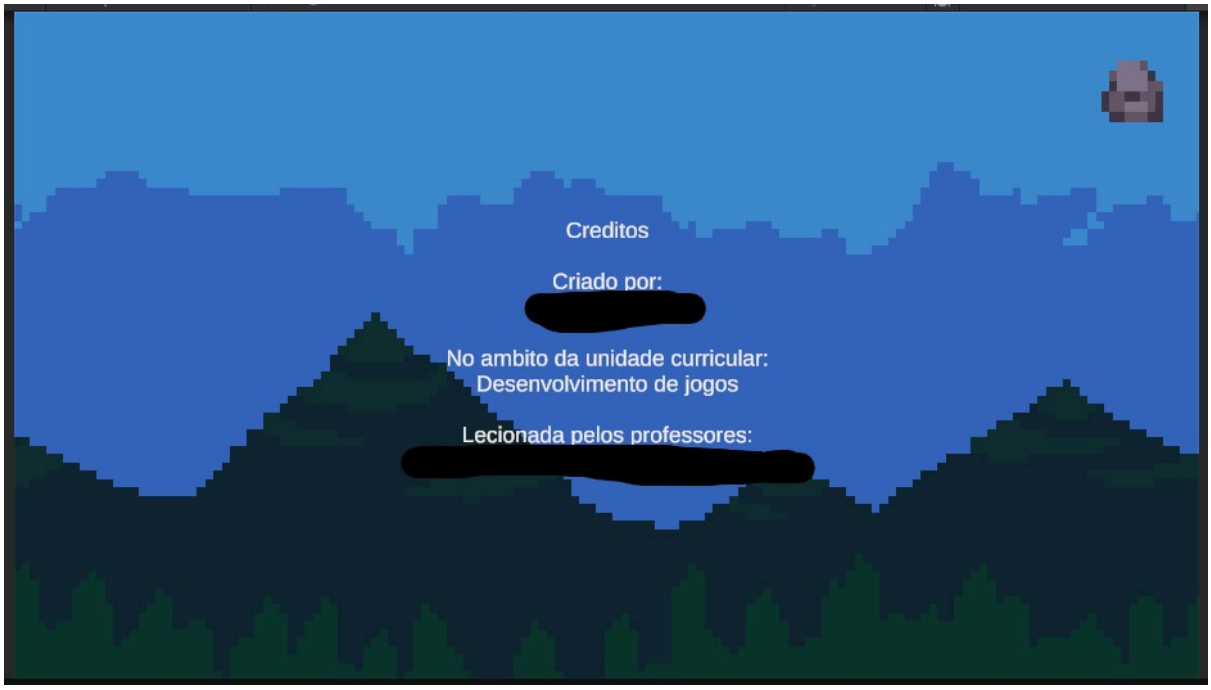
animação e reparei que nas plataformas que sobem e descem o player fica preso na mesma animação e não dá para saltar.

- Testar checkpoint.
- Possivel ativação de um gamemanager dentro dos níveis caso não exista para debug facilitado.
- Verificar se o limite do ecrã em baixo funciona (esqueci com os outros erros).
- Verificar se o checkpoint está funcional e como novo safepoint após passar por ele.

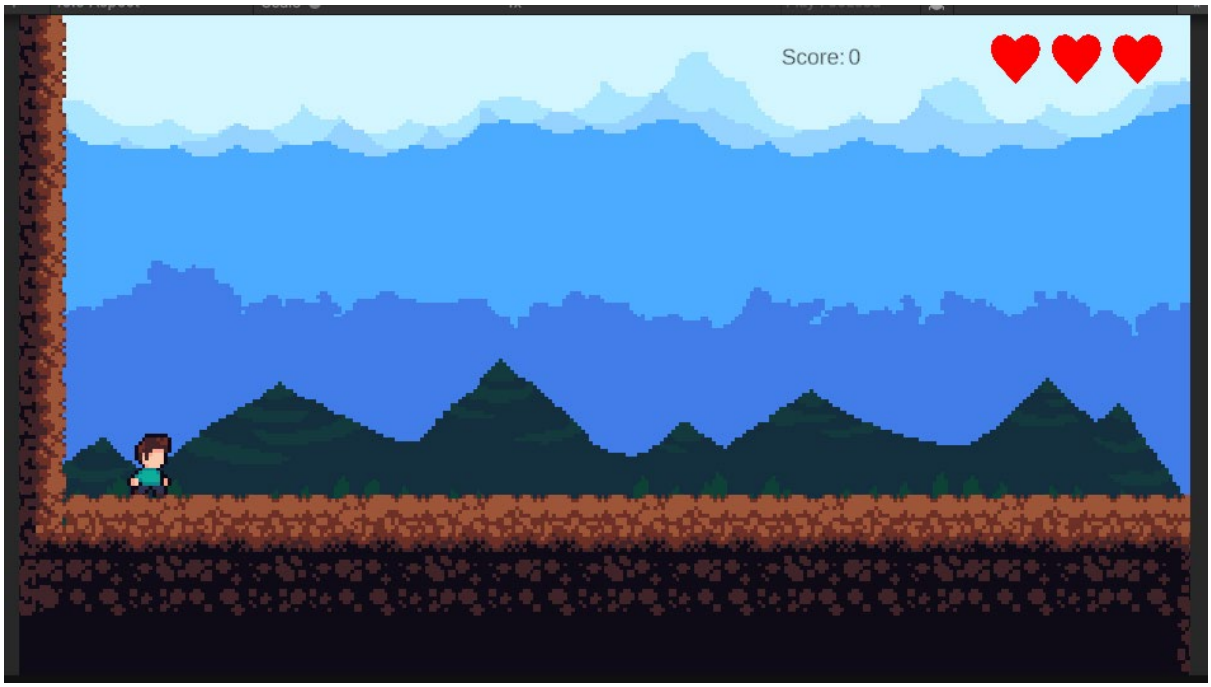
Foto(s) para memória visual:

A partir deste dia vou colocar as scenes que mexo com nome em baixa da scene que é para comparar a evolução.

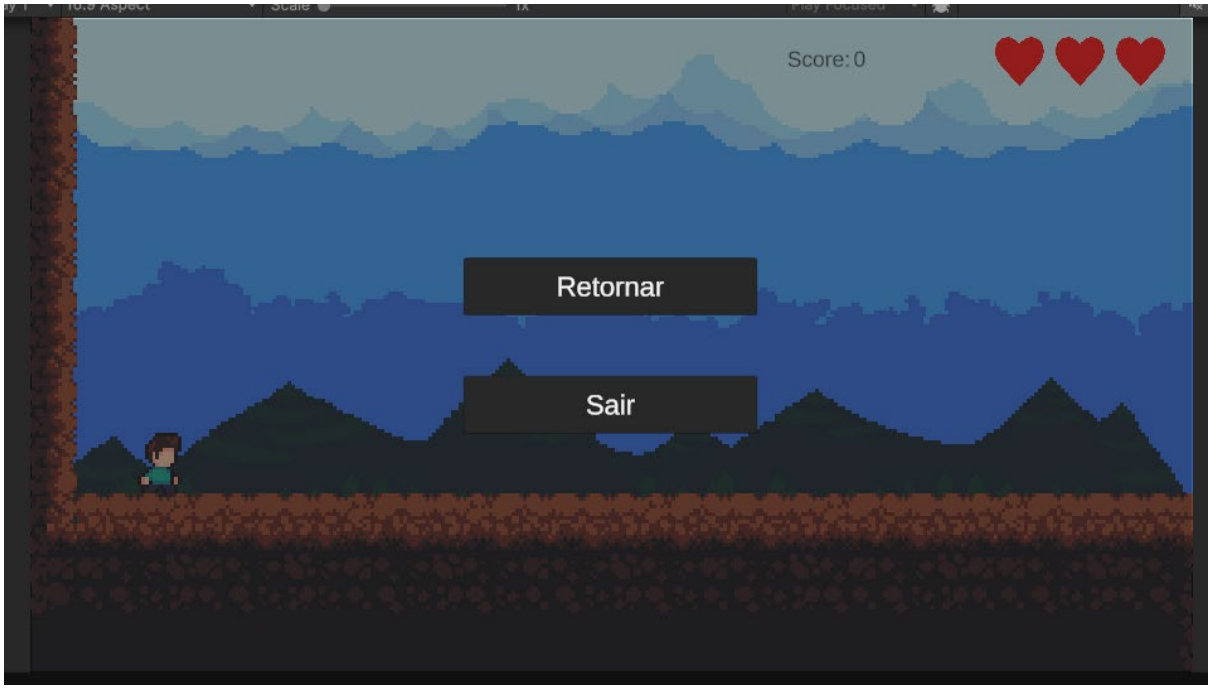
(Menu inicial)



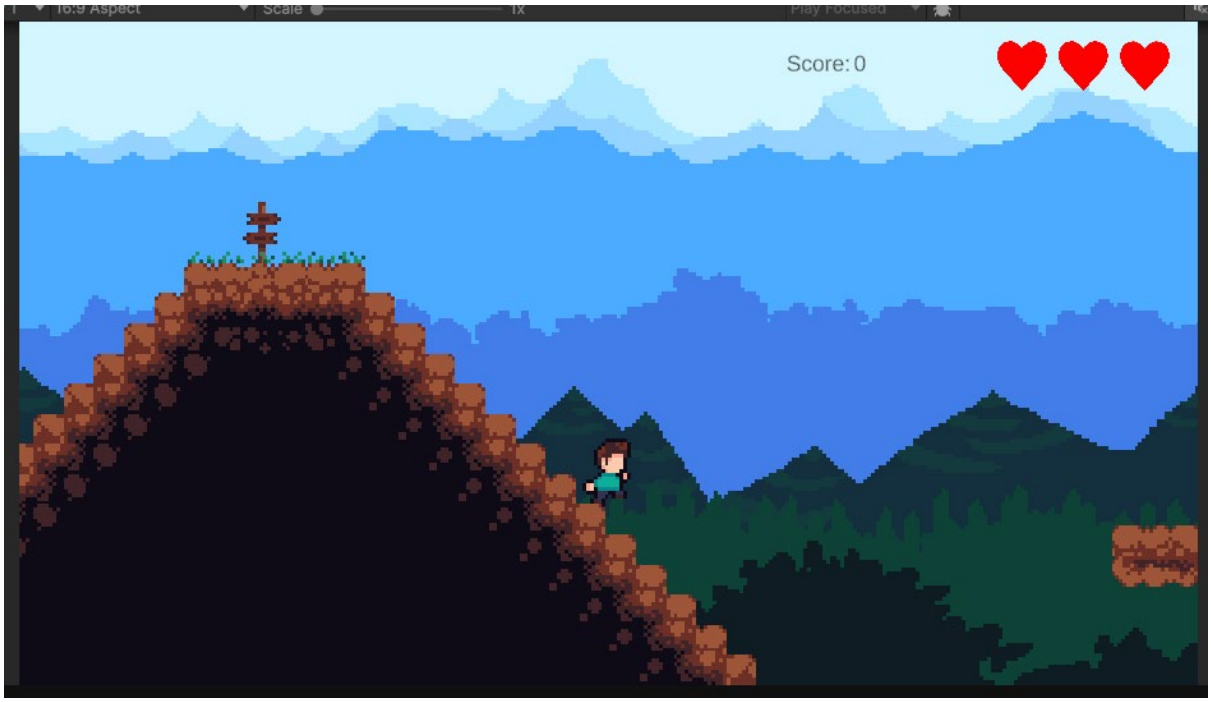
(Créditos)



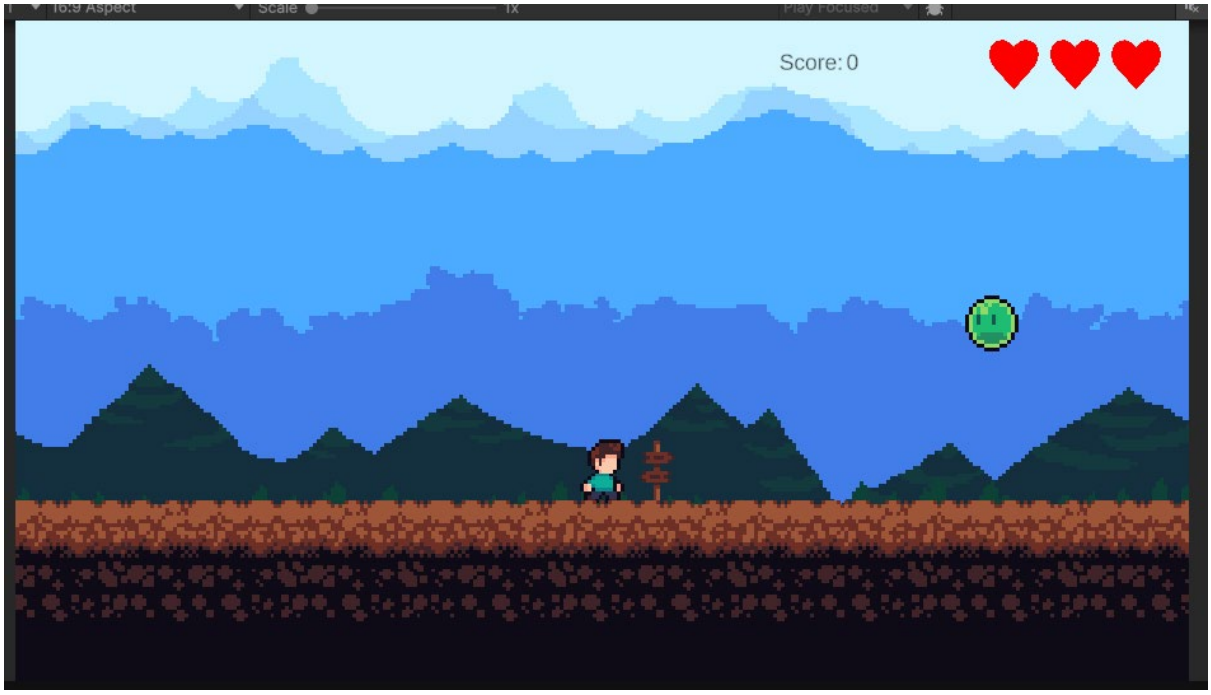
(Nível 1 início)



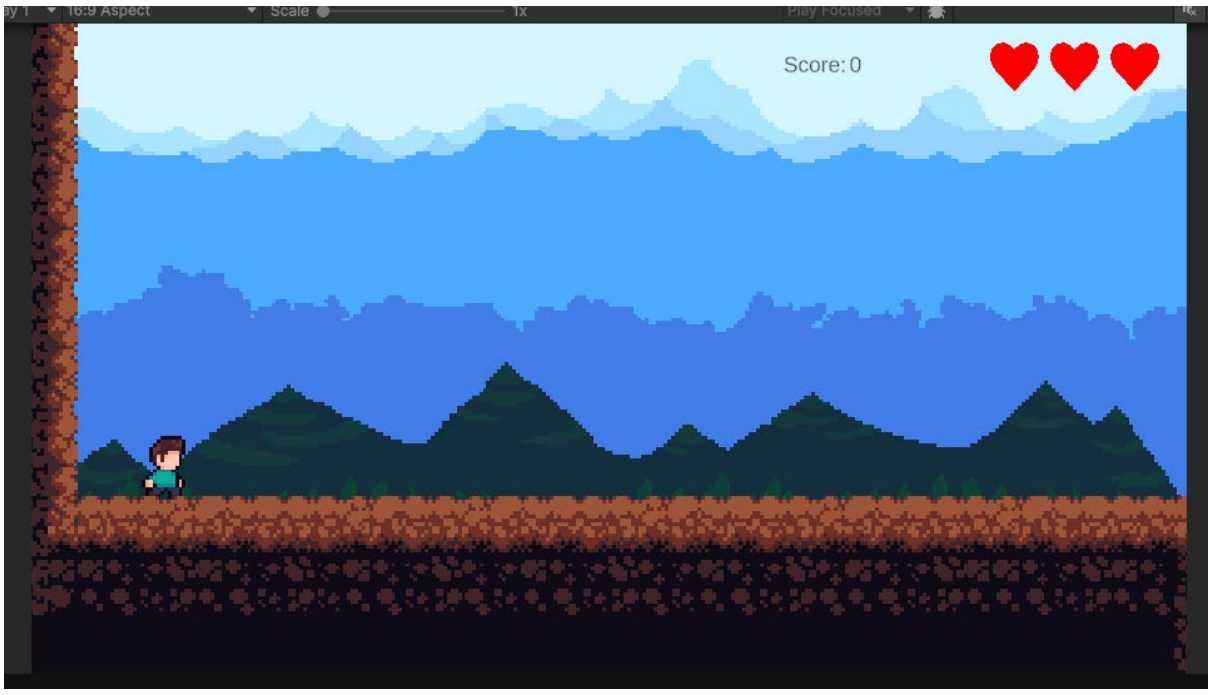
(Menu pausa)



(Nível 1 meio)



(Final nível 1)



(Nível 2 início)

02 de maio de 2025

Tempo estimado de trabalho: Aproximadamente 3h

Objetivos da Sessão:

- Finalizar sistema de colecionáveis.
- Tornar os checkpoints funcionais com sistema de respawn.
- Resolver bugs relacionados com inimigos e dano.
- Testar Game Over e instanciar GameManager em cenas isoladas.
- Corrigir problemas de salto em plataformas móveis.

Tarefas Realizadas:

Colecionáveis e HUD:

- Colecionáveis agora funcionam corretamente (som (não final), pontuação, destruição).
- Prefab funcional criado com sistema de pontuação integrado no HUD.
- ScoreManager adaptado para guardar pontos com PlayerPrefs.

Checkpoints:

- Criado sistema funcional de checkpoint com respawn.
- Mensagem visual "Checkpoint alcançado!" criada com UI temporária (toast).
- Toast aparece apenas na primeira ativação de cada checkpoint.
- Organizado o sistema de UI num ToastManager central.

Limites e dano:

- Adicionado LimiteFundo para dano ao cair do ecrã.
- Aplicado Tilemap Collider aos picos para funcionar o trigger de receber dano.
- Jogador perde vida corretamente ao tocar nos picos.

Inimigos:

- Corrigido bug onde inimigos não causavam dano (uso de IsTrigger).
- Inimigos agora são destruídos ao serem pisados (verificação por Y).
- Bounce inicial ao pisar inimigo testado, mas não funcional (versão refinada adiada para próxima sessão).

GameManager:

- Criado GameManagerIndependente que instancia automaticamente o GameManager quando não está presente para facilitar o debug em nível mais avançados.
- Verificado que GameManager funciona ao iniciar cenas diretamente (ex: Nível 2 isolado).

Organização:

- Estruturada a hierarquia da cena com Managers, Colliders, Checkpoints, HUD, etc.
- Agrupados os objetos de interface e lógica global em pastas visuais.
- Corrigida sobreposição de plataformas com borda do ecrã.

Problemas Encontrados:

- O jogador continua a ficar preso na animação de salto ao colidir lateralmente com rampas ou paredes.
- Tentativa de implementar GroundCheck com OverlapCircle falhou — deixado para futura sessão.
- Após implementar o novo GroundCheck, o double jump deixou de funcionar — revertido para a versão anterior.

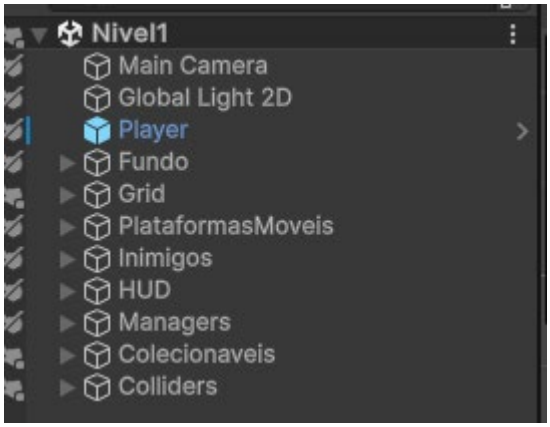
Plano para a Próxima Sessão:

- Corrigir bug do jogador preso na animação de salto (ajuste do sistema de grounded).
- Refinar o bounce ao matar inimigos (uso de velocity.y = valor).
- Adicionar animação ou delay antes de inimigos desaparecerem.
- Iniciar a construção do Nível 2 (layout, inimigos, lógica).
- [Após tudo funcional] Mudar o visual dos checkpoints quando ativados.
- [Após tudo funcional] Polimento visual geral (sprites, paralaxe, efeitos).

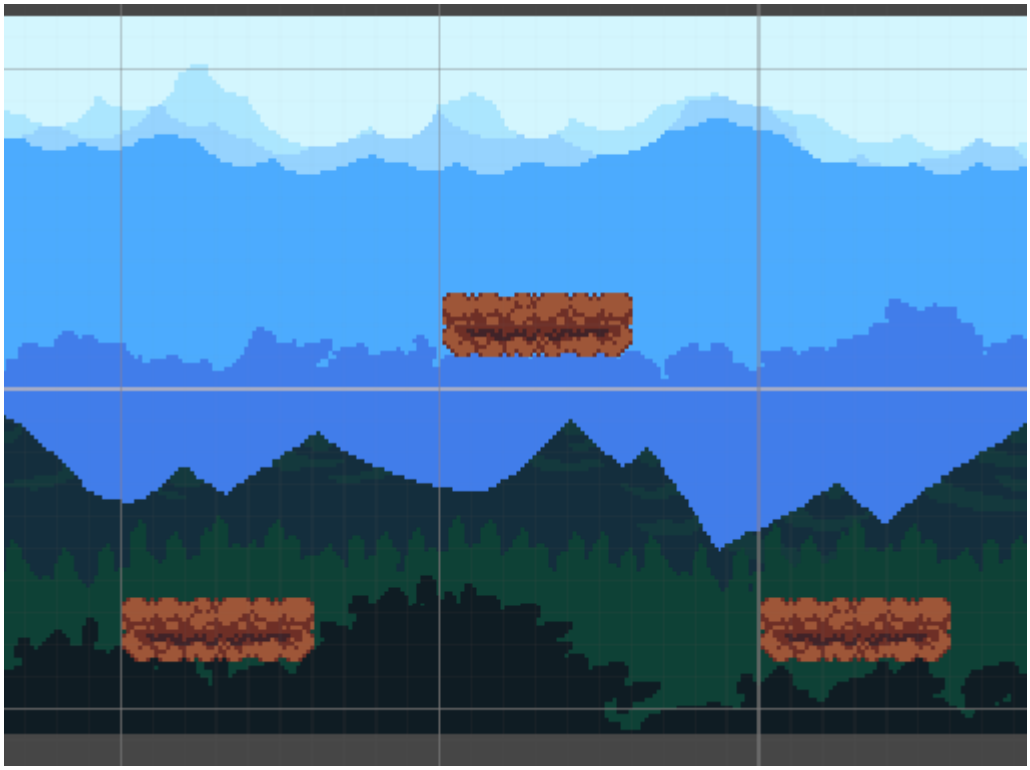
Reflexão Pessoal:

Esta foi uma das sessões mais produtivas do projeto até agora. O sistema de checkpoints e colecionáveis está estável, o dano do jogador foi implementado corretamente, e a lógica geral do jogo está cada vez mais robusta. A estrutura do projeto está muito mais organizada, o que vai facilitar a criação do Nível 2. Apesar do bug do salto não estar resolvido, ficou claro o caminho a seguir. Estou motivado a terminar a base funcional do jogo e deixar o polimento visual para o fim, como planeado.

Foto(s) para memória visual:



(Organização realizada nesta sessão)



(Plataformas mais subidas para não ficarem tangentes no ecrã)

03 de maio de 2025 Sessão da Tarde

Tempo estimado de trabalho: Aproximadamente 30m

Objetivos da Sessão:

- Refinar o sistema de bounce ao matar inimigos.
- Adicionar animação ou delay antes dos inimigos desaparecerem.
- Corrigir bug do salto preso (especialmente a animação que não transitava corretamente).
- Preparar o terreno para a construção do Nível 2 (a ser feito à noite).

Tarefas Realizadas:

- Substituído AddForce por alteração direta da velocity.y ao pisar inimigos para um bounce mais imediato.
- Criado um sistema de "morte" nos inimigos que ativa a animação de queda e aguarda 1 segundo após tocar numa plataforma antes de desaparecer.
- Corrigido bug do salto em rampas/paredes: o problema era de transição no Animator. Bastou ligar corretamente a transição de "Jump" para "Idle" com condição de isGrounded == true.
- Adicionados SetBool para isJumping com base em velocity.y no FixedUpdate para controlo robusto das animações.
- Corrigido também o flip indesejado na sprite das plataformas horizontais.

Problemas Encontrados:

- O trigger "jump" nem sempre funcionava no primeiro salto, por ser ativado antes do Rigidbody aplicar a força.
- Foi substituído por uso de bools baseadas na velocity.y, o que tornou as transições mais fiáveis e consistentes.

Reflexão Pessoal:

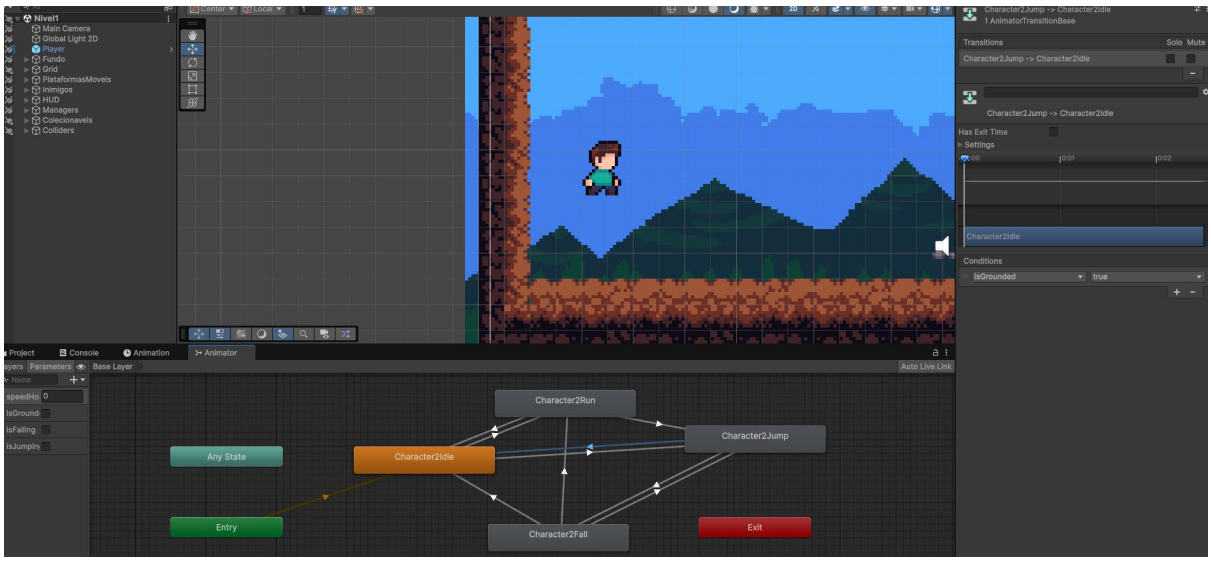
Apesar da sessão ter sido curta, foi extremamente produtiva. Resolvi três problemas que vinham a acumular e estabilizei de vez o comportamento do salto. Fiquei também muito satisfeito com a solução simples de usar a animação de queda como substituta de uma animação de morte. Agora sinto que a base está pronta para começar a montar o Nível 2 com confiança.

Notas para a próxima sessão (noite):

- Iniciar construção do Nível 2 (tilemap, inimigos, checkpoints, colecionáveis).
- Testar transição entre Nível 1 e Nível 2.
- Verificar se todos os componentes do prefab do Player estão a funcionar corretamente no novo nível.

- Verificar se ao receber nado do inimigo recebo 2x ou mais.

Foto(s) para memória visual:



03 de maio de 2025 Sessão da Noite

Tempo estimado de trabalho: Cerca de 1h25min

Objetivo da Sessão:

- Rever e documentar todos os scripts do projeto.
- Avaliar se há funções que deveriam ser movidas ou reorganizadas.

Tarefas Realizadas:

- Revistos e comentados todos os scripts do projeto: Player, Inimigo, GameManager, ParalaxBackground, ScoreManager, MainMenu, CoinPickup, Toast, PlataformaMovimento, GameManagerIndependente e CamaraPrincipal.
- Comentários adicionados linha a linha, explicando de forma clara o que cada bloco faz, mantendo consistência com o estilo do diário.
- Verificada a organização de responsabilidades em cada script: os scripts das plataformas juntaram-se num apenas, foi retirado o script Creditos que retornava ao menu inicial foi removido utilizando o presente no script MainMenu e foi retirado o JogarNovamente() do script MainMenu por fazer o mesmo que o NewGame().

Reflexão Pessoal:

Esta sessão foi um ponto importante no projeto: a documentação completa dos scripts não só vai ajudar-me no futuro, como também torna o jogo mais acessível para quem o quiser estudar ou aprender com ele. Foi uma sessão longa, mas essencial para dar um ar profissional ao código e garantir que a estrutura está limpa e coesa. Agora sinto que o projeto está bem preparado para continuar a evoluir e finalmente realizar o nível 2. Assim que completar o nível 2 e acabar de atualizar todos os elementos visuais sinto me confiante para adicionar algo como uma pequena historia para inicializar o jogo e outros elementos. A partir do dia de hoje serão adicionados os scripts das funções criadas em cada sessão.

Notas para a próxima sessão:

- Iniciar montagem do Nível 2 (como estou com vontade de trabalhar não sei se realizo noutra sessão de hoje se amanhã pego no projeto).
- Começar pelo layout do tilemap, seguida da colocação de plataformas e inimigos.

“Código dos outros é intragável”, dizia o meu primo (nunca mexeu em game dev, só web dev).
Por isso, decidi por os scripts aqui. Para os mais fortes recomendo que leia com café forte. Eles acabam na página 40, dou-vos 18/19 páginas de código para quem gosta de massacre.

CamaraPrincipal.cs

```
// Script responsável por seguir o jogador horizontalmente com limites definidos (scroll lateral)
using UnityEngine;
public class CamaraPrincipal : MonoBehaviour
{
  // Referência ao transform do jogador
  private Transform player;
  // Deslocamento fixo da câmara (z = -10 para garantir visibilidade 2D)
  private Vector3 offset;
  // Limite esquerdo da câmara (posição mínima em X)
  [SerializeField] public float limiteEsquerdo;
  // Limite direito da câmara (posição máxima em X)
  [SerializeField] public float limiteDireito;
  void Start()
  {
    // Encontrar o jogador na cena pelo nome e obter o transform
    player = GameObject.Find("Player").GetComponent<Transform>();
    // Define o offset para manter a câmara na posição correta em Z
    offset = new Vector3(0, 0, -10);
  }
  void LateUpdate()
  {
    // Calcula a posição X da câmara com base na posição do jogador
    float posX = player.position.x + offset.x;
    // Garante que a câmara não ultrapassa os limites definidos
    posX = Mathf.Clamp(posX, limiteEsquerdo, limiteDireito);
    // Atualiza a posição da câmara
    transform.position = new Vector3(posX, offset.y, offset.z);
  }
}
```

CoinPickup.cs

```
// Script responsável por dar pontos ao jogador ao apanhar moedas e reproduzir som
using UnityEngine;
public class CoinPickup : MonoBehaviour
{
  // Quantidade de pontos que esta moeda adiciona
  public int pointToAdd;
  // Som a ser tocado ao apanhar a moeda
```

```
  private AudioSource CoinPickupEffect;
  void Start()
  {
    // Referência ao AudioSource ligado à moeda
    CoinPickupEffect = GetComponent<AudioSource>();
  }
  void OnTriggerEnter2D(Collider2D other)
  {
    // Garante que só o jogador pode apanhar a moeda
    if (other.GetComponent<Player>() == null)
      return;
    // Adiciona os pontos ao ScoreManager
    ScoreManager.AddPoints(pointToAdd);
    // Toca o som de apanhar moeda
    CoinPickupEffect.Play();
    // Esconde o sprite da moeda (opcional — dá sensação de recolhida antes de
desaparecer)
    GetComponentInChildren<SpriteRenderer>().enabled = false;
    // Destroi o objeto após 1 segundo (dá tempo para o som tocar)
    Destroy(gameObject, 1.0f);
  }
}
```

GameManager.cs

```
// Script responsável por gerir o estado geral do jogo (pausa, vidas, e ligação ao menu)
// É persistente entre cenas graças ao DontDestroyOnLoad
using UnityEngine;
using UnityEngine.SceneManagement;
public class GameManager : MonoBehaviour
{
  // Singleton: acesso global ao GameManager
  public static GameManager instance;
   // Referência ao painel de pausa na UI
  public GameObject menuPausa;
  // Estado atual do jogo (em pausa ou não)
  private bool gameIsPaused = false;
  // Nome da cena do menu principal
  public string menu;
  // Número de vidas do jogador
  public int vidas = 3;
  void Awake()
```

```
{
    // Garante que apenas um GameManager existe (padrão Singleton)
    if (instance != null && instance != this)
    {
        Destroy(gameObject);
        return;
    }
    instance = this;
    // Persiste entre cenas
    DontDestroyOnLoad(gameObject);
    // Reinicia as vidas ao criar o GameManager
    vidas = 3;
}
void Start()
{
    // Garante que o menu de pausa começa desativado
    menuPausa.SetActive(false);
}
void Update()
{
    // Atalho para pausar ou retomar o jogo com a tecla ESC
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (gameIsPaused)
            RetomarJogo();
        else
            PausarJogo();
    }
}
// Ativa a pausa: congela o tempo e mostra o menu de pausa
public void PausarJogo()
{
    Time.timeScale = 0f;
    menuPausa.SetActive(true);
    gameIsPaused = true;
}
// Retoma o jogo: reativa o tempo e esconde o menu de pausa
public void RetomarJogo()
{
    Time.timeScale = 1f;
    menuPausa.SetActive(false);
```

```
        gameIsPaused = false;
    }
    // Sai do jogo atual para o menu principal
    public void SairParaMenu()
    {
        Time.timeScale = 1f;
        Destroy(GameManager.instance.gameObject);
        SceneManager.LoadScene(menu);
    }
}
```

GameManagerIndependente.cs

```
// Script auxiliar que garante que o GameManager é instanciado automaticamente em cenas onde ele não existe
using UnityEngine;
public class GameManagerIndependente : MonoBehaviour
{
    // Prefab do GameManager a instanciar se necessário
    public GameObject gameManagerIndependente1;
    void Awake()
    {
        // Verifica se o GameManager (singleton) já existe na cena
        if (GameManager.instance == null)
        {
            // Instancia um novo GameManager a partir do prefab
            GameObject go = Instantiate(gameManagerIndependente1);
            // Dá-lhe o nome correto para organização na Hierarquia
            go.name = "GameManager";
        }
    }
}
```

Inimigo.cs

```
// Script que controla o comportamento de um inimigo que se move lateralmente entre dois pontos e salta ao inverter direcção.
// Também lida com animações e morte do inimigo.
using System.Collections;
using UnityEngine;

public class InimigoLR : MonoBehaviour
{
```

```
  // Ponto inicial (limite esquerdo) do movimento
  [SerializeField] private Transform checkpoint1;
  // Ponto final (limite direito) do movimento
  [SerializeField] private Transform checkpoint2;
  // Velocidade de deslocamento
  [SerializeField] private float speed;
  // Força do salto ao mudar de direcção
  [SerializeField] private float jumpForce;
  // Estado atual da direcção (true = esquerda)
  [SerializeField] private bool isMovingLeft;
  // Indica se o inimigo foi morto
  private bool isDead = false;
  // Referência ao Rigidbody do inimigo
  private Rigidbody2D rb;
  // Referência ao Animator do inimigo
  private Animator animator;
  // Indica se está em contacto com o chão (para permitir salto)
  private bool isGrounded;

  void Start()
  {
    rb = GetComponent<Rigidbody2D>();
    animator = GetComponent<Animator>();
    // Garante que checkpoint1 seja o da esquerda, e checkpoint2 o da direita
    if (checkpoint1.position.x > checkpoint2.position.x)
    {
      var aux = checkpoint2;
      checkpoint2 = checkpoint1;
      checkpoint1 = aux;
    }
  }
  void FixedUpdate()
  {
    Move();
    UpdateAnimations();
  }
  // Move o inimigo para a esquerda ou direita, mudando de direcção ao atingir os
limites
  private void Move()
  {
    int direction = isMovingLeft ? -1 : 1;
```

```
    rb.linearVelocity = new Vector2(direction * speed, rb.linearVelocity.y);
    // Vira o sprite de acordo com a direcção
    GetComponent<SpriteRenderer>().flipX = isMovingLeft;
    // Altera a direcção ao atingir os checkpoints
    if (isMovingLeft && transform.position.x < checkpoint1.position.x)
    {
      isMovingLeft = false;
      Jump(); // Salta ao inverter direcção
    }
    else if (!isMovingLeft && transform.position.x > checkpoint2.position.x)
    {
      isMovingLeft = true;
      Jump();
    }
  }
  // Faz o inimigo saltar se estiver no chão
  private void Jump()
  {
    if (isGrounded)
    {
      rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
    }
  }
  // Atualiza os estados de animação com base na velocidade vertical
  private void UpdateAnimations()
  {
    animator.SetBool("isJumping", rb.linearVelocity.y > 0.1f);
    animator.SetBool("isFalling", rb.linearVelocity.y < -0.1f);
  }
  // Detecta contacto com plataformas e executa lógica de grounded ou destruição
  private void OnCollisionEnter2D(Collision2D collision)
  {
    if (isDead && (collision.gameObject.CompareTag("Plataforma") ||
collision.gameObject.CompareTag("movingPlataforma")))
    {
      // Se estiver morto e tocar no chão, inicia a contagem para desaparecer
      StartCoroutine(DestruirAposTempo());
    }

    if (!isDead && (collision.gameObject.CompareTag("Plataforma") ||
collision.gameObject.CompareTag("movingPlataforma")))
```

```
        {
            isGrounded = true;
            Jump(); // Permite novo salto se necessário
        }
    }
    // Quando sai do chão, deixa de estar grounded
    private void OnCollisionExit2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Plataforma") ||
collision.gameObject.CompareTag("movingPlataforma"))
        {
            isGrounded = false;
        }
    }
    // Função chamada quando o inimigo é pisado pelo jogador
    public void MatarInimigo()
    {
        isDead = true;
        rb.linearVelocity = Vector2.zero;
        // Reutiliza a animação de queda como "morte"
        GetComponent<Animator>().SetBool("isFalling", true);
        // Mantém o collider ativo para cair no chão
        GetComponent<Collider2D>().enabled = true;
        this.enabled = false; // Desativa o script (para impedir movimento)
    }
    // Aguarda 1 segundo e depois destrói o inimigo
    private IEnumerator DestruirAposTempo()
    {
        yield return new WaitForSeconds(1f);
        Destroy(gameObject);
    }
}
```

MainMenu.cs

```
// Script responsável por gerir os botões de navegação entre menus, níveis e sair do
jogo
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
public class MainMenu : MonoBehaviour
{
```

```
  // Nome da cena do primeiro nível (jogo novo ou recomeço)
  public string nivelACarregar;
  // Nome da cena dos créditos
  public string credito;
  // Nome da cena do menu principal
  public string menuprincipal;
  // Inicia um novo jogo, resetando pontuação e GameManager
  public void NewGame()
  {
    PlayerPrefs.SetInt("score", 0);
    if (GameManager.instance != null)
    {
      Destroy(GameManager.instance.gameObject);
    }
    SceneManager.LoadScene(nivelACarregar);
  }
  // Vai para a cena dos Créditos
  public void Creditos()
  {
    SceneManager.LoadScene(credito);
  }
  // Volta ao menu principal a partir de qualquer cena
  public void MenuPrincipal()
  {
    SceneManager.LoadScene(menuprincipal);
  }
  // Sai do jogo (funciona só na build, não no editor)
  public void QuitGame()
  {
    Application.Quit();
  }
}
```

ParalaxBackground.cs

```
// Script responsável por criar o efeito de paralaxe, fazendo o fundo mover-se mais lentamente que a câmara
// Dá sensação de profundidade ao cenário 2D
using UnityEngine;
public class ParallaxBackground : MonoBehaviour
{
  // Referência à câmara (pode ser atribuída manualmente ou pega automaticamente)
```

```
    public Transform cameraTransform;
    // Multiplicador que define a intensidade do efeito de paralaxe
    public float parallaxMultiplier = 0.5f;
    // Guarda a última posição da câmara para calcular o movimento
    private Vector3 lastCameraPosition;

    void Start()
    {
        // Se nenhuma câmara for atribuída, usa a principal automaticamente
        if (cameraTransform == null)
            cameraTransform = Camera.main.transform;
        lastCameraPosition = cameraTransform.position;
    }
    void LateUpdate()
    {
        // Calcula o quanto a câmara se moveu desde o último frame
        Vector3 deltaMovement = cameraTransform.position - lastCameraPosition;
        // Move o fundo proporcionalmente ao movimento da câmara
        transform.position += deltaMovement * parallaxMultiplier;
        // Atualiza a posição da câmara para o próximo frame
        lastCameraPosition = cameraTransform.position;
    }
}
```

PlataformaMovimento.cs

```
// Script reutilizável para mover plataformas entre dois pontos, tanto na horizontal
como na vertical
using UnityEngine;
public class PlataformaMovimento : MonoBehaviour
{
    // Primeiro ponto de movimento da plataforma
    [SerializeField] private Transform checkpoint1;
    // Segundo ponto de movimento da plataforma
    [SerializeField] private Transform checkpoint2;
    // Velocidade do movimento
    [SerializeField] private float speed = 2f;
    // Define se o movimento é vertical (true) ou horizontal (false)
    [SerializeField] private bool movimentoVertical = false;
    // Indica se a plataforma está a mover-se na direção inversa
    private bool direcaoInversa = false;
    void Start()
```

```
  {
    // Garante que checkpoint1 é sempre o mais "baixo" (Y) ou mais "à esquerda" (X),
para facilitar a lógica
    if (movimentoVertical)
    {
      if (checkpoint1.position.y < checkpoint2.position.y)
        SwapCheckpoints();
    }
    else
    {
      if (checkpoint1.position.x > checkpoint2.position.x)
        SwapCheckpoints();
    }
  }
  void FixedUpdate()
  {
    Vector3 pos = transform.position;

    if (movimentoVertical)
    {
      // Verifica se chegou aos limites verticais para inverter direção
      if (direcaoInversa && pos.y <= checkpoint2.position.y) direcaoInversa = false;
      else if (!direcaoInversa && pos.y >= checkpoint1.position.y) direcaoInversa = true;
      float direction = direcaoInversa ? -1f : 1f;
      transform.position += new Vector3(0, direction * speed * Time.deltaTime, 0);
    }
    else
    {
      // Verifica se chegou aos limites horizontais para inverter direção
      if (direcaoInversa && pos.x <= checkpoint1.position.x) direcaoInversa = false;
      else if (!direcaoInversa && pos.x >= checkpoint2.position.x) direcaoInversa = true;
      float direction = direcaoInversa ? -1f : 1f;
      transform.position += new Vector3(direction * speed * Time.deltaTime, 0, 0);
    }
  }
  // Troca os checkpoints caso estejam fora de ordem
  void SwapCheckpoints()
  {
    var temp = checkpoint1;
    checkpoint1 = checkpoint2;
    checkpoint2 = temp;
```

```
    }
}
```

Player.cs

```
// Script principal do jogador. Controla movimento, salto, interações com plataformas e inimigos, animações e respawn.
using System.Collections;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class Player : MonoBehaviour
{
    //Movimento e física
    // Velocidade de movimento horizontal
    [SerializeField] private float movementSpeed;
    // Força aplicada ao salto
    [SerializeField] private float jumpPower;
    // Referência ao Rigidbody do jogador
    private Rigidbody2D playerRb2d;
    // Input horizontal
    private float horizontal;
    // Indica se o jogador está no chão
    private bool grounded;
    // Permite salto duplo
    private bool canDoubleJump;
    // Indica direção em que o jogador está virado
    private bool facingRight = true;

    //Vidas e UI
    // Vidas iniciais do jogador
    public int vidas;
    // Ícones de corações na interface
    [SerializeField] private Image[] vidasUI;
    // Sprite do coração cheio
    public Sprite heartFull;
    // Sprite do coração vazio
    public Sprite heartEmpty;

    //Checkpoints e cenas
    // Ponto de respawn atual
    [SerializeField] private Transform safePoint;
```

```
// Nome da cena de Game Over
public string gameover;
// Nome da cena final do jogo
public string completo;
// Nome da cena do Nível 2
public string nivel2;

//Animação e estado
// Referência ao Animator
private Animator myAnimation;
// Indica se está a saltar
private bool jump;
// Indica se o jogador morreu
private bool isDead;
// Proteção contra dano repetido
private bool invulneravel = false;

//Plataformas móveis
// Rigidbody da plataforma móvel atual (se existir)
private Rigidbody2D platformRb;
void Start()
{
   // Inicializações no início da cena
   playerRb2d = GetComponent<Rigidbody2D>();
   myAnimation = GetComponent<Animator>();
   grounded = true;
   canDoubleJump = false;
   isDead = false;
   // Atualiza a interface com vidas
   AtualizarVidasUI();

   if (safePoint == null)
      // Se não houver checkpoint, começa onde nasceu
      safePoint = transform;
}

void Update()
{
   // Ignora input se estiver morto
   if (isDead) return;
   // Captura o input horizontal
```

```
    horizontal = Input.GetAxis("Horizontal");
    // Salto
    if (Input.GetButtonDown("Jump"))
    {
      if (grounded)
      {
        jump = true;
        playerRb2d.AddForce(Vector2.up * jumpPower, ForceMode2D.Impulse);
        canDoubleJump = true;
        grounded = false;
      }
      else if (canDoubleJump)
      {
        jump = true;
        canDoubleJump = false;
        // Anula velocidade vertical
        playerRb2d.linearVelocity = new Vector2(playerRb2d.linearVelocity.x, 0);
        playerRb2d.AddForce(Vector2.up * jumpPower, ForceMode2D.Impulse);
      }
    }
  }
  void FixedUpdate()
  {
    if (isDead) return;
    // Aplica movimento horizontal
    HandleMovement(horizontal);
    // Atualiza animações de movimento
    myAnimation.SetFloat("speedHorizontal", Mathf.Abs(horizontal));
    myAnimation.SetBool("isGrounded", grounded);
    myAnimation.SetBool("isJumping", playerRb2d.linearVelocity.y > 0.1f &&
!grounded);
    myAnimation.SetBool("isFalling", playerRb2d.linearVelocity.y < -0.1f && !grounded);
    // Se estiver sobre uma plataforma móvel, mover com ela
    if (platformRb != null)
    {
      playerRb2d.position += (Vector2)(platformRb.linearVelocity *
Time.fixedDeltaTime);
    }
  }
  private void HandleMovement(float horizontal)
  {
```

```
    // Aplica velocidade horizontal ao jogador
    playerRb2d.linearVelocity = new Vector2(horizontal * movementSpeed,
playerRb2d.linearVelocity.y);

    // Atualiza direção visual do jogador
    if (horizontal > 0 && !facingRight) Flip();
    else if (horizontal < 0 && facingRight) Flip();
  }
  private void Flip()
  {
    // Inverte estado
    facingRight = !facingRight;
    Vector3 scale = transform.localScale;
    scale.x *= -1;
    // Vira sprite horizontalmente
    transform.localScale = scale;
  }
  private void OnCollisionEnter2D(Collision2D collision)
  {
    if (collision.gameObject.CompareTag("Plataforma") ||
collision.gameObject.CompareTag("movingPlataforma"))
    {
      grounded = true;
      canDoubleJump = true;
      // Desativa trigger de salto
      myAnimation.SetBool("jump", false);

      if (collision.gameObject.CompareTag("movingPlataforma"))
      {
        // Fica “colado” à plataforma
        transform.parent = collision.transform;
        platformRb = collision.gameObject.GetComponent<Rigidbody2D>();
      }
    }
  }
  private void OnCollisionExit2D(Collision2D collision)
  {
    if (collision.gameObject.CompareTag("movingPlataforma"))
    {
      // Liberta do parent após sair da plataforma
      StartCoroutine(RemoveParentNextFrame());
```

```
    }
}
private IEnumerator RemoveParentNextFrame()
{
    // Espera 1 frame
    yield return null;
    transform.parent = null;
    platformRb = null;
}
public void AtivarCheckpoint(Transform novoCheckpoint)
{
    // Define novo ponto de respawn
    safePoint = novoCheckpoint;
}
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("Inimigo"))
    {
        if (invulneravel) return;
        float playerY = transform.position.y;
        float enemyY = collision.transform.position.y;
        if (playerY > enemyY + 0.5f && playerRb2d.linearVelocity.y < 0)
        {
            InimigoLR inimigoScript = collision.GetComponent<InimigoLR>();
            if (inimigoScript != null)
            {
                // Pisa o inimigo
                inimigoScript.MatarInimigo();
            }
            // Bounce
            playerRb2d.linearVelocity = new Vector2(playerRb2d.linearVelocity.x, 10f);
        }
        else
        {
            StartCoroutine(Invulnerabilidade(1f)); // Proteção temporária
            PerdeVida(); // Leva dano
        }
    }
    else if (collision.CompareTag("LimiteEcra"))
    {
        // Caiu do nível
```

```
            PerdeVida();
        }
        else if (collision.CompareTag("nivel2collider"))
        {
            // Transita para o Nível 2
            Nivel2();
        }
        else if (collision.CompareTag("finaljogo"))
        {
            // Vai para o final do jogo
            CompletarJogo();
        }
    }
    private void PerdeVida()
    {
        GameManager.instance.vidas--;

        if (GameManager.instance.vidas < 0)
            GameManager.instance.vidas = 0;

        AtualizarVidasUI();

        if (GameManager.instance.vidas == 0)
        {
            // Game Over
            SceneManager.LoadScene(gameover);
        }
        else
        {
            // Volta ao checkpoint
            StartCoroutine(DeathAndRespawn());
        }
    }
    private IEnumerator DeathAndRespawn()
    {
        isDead = true;
        playerRb2d.linearVelocity = Vector2.zero;

        yield return new WaitForSeconds(1f);

        isDead = false;
```

```
        transform.position = safePoint.position;
        playerRb2d.linearVelocity = Vector2.zero;
    }
    public void AtualizarVidasUI()
    {
        for (int i = 0; i < vidasUI.Length; i++)
        {
            if (i < GameManager.instance.vidas)
                vidasUI[i].sprite = heartFull;
            else
                vidasUI[i].sprite = heartEmpty;
        }
    }
    private void Nivel2()
    {
        // Transição para Nível 2
        SceneManager.LoadScene(nivel2);
    }
    private void CompletarJogo()
    {
        // Transição para cena final
        SceneManager.LoadScene(completo);
    }
    private IEnumerator Invulnerabilidade(float tempo)
    {
        invulneravel = true;
        yield return new WaitForSeconds(tempo);
        invulneravel = false;
    }
}
```

ScoreManager.cs

```
// Script responsável por gerir a pontuação do jogador e atualizá-la visualmente na UI
using UnityEngine;
using TMPro;
public class ScoreManager : MonoBehaviour
{
    // Pontuação atual (armazenada estaticamente para acesso global)
    private static int score;
    // Referência ao ScoreManager da cena (usado para atualizar UI)
    private static ScoreManager instance;
```

```
  // Texto da UI onde a pontuação é mostrada
  public TMP_Text textoPontuacao;
  void Awake()
  {
    // Guarda esta instância para uso nos métodos estáticos
    instance = this;
  }
  void Start()
  {
    // Recupera a pontuação guardada anteriormente (se existir)
    score = PlayerPrefs.GetInt("score");
    if (score < 0) { score = 0; }
    // Atualiza o texto da UI com a pontuação inicial
    instance.textoPontuacao.text = "" + score;
  }
  // Método público e estático para adicionar pontos
  public static void AddPoints(int points)
  {
    score += points;
    // Guarda nos PlayerPrefs
    PlayerPrefs.SetInt("score", score);
    // Atualiza a UI, se o ScoreManager e o texto estiverem ativos
    if (instance != null && instance.textoPontuacao != null)
      instance.textoPontuacao.text = "" + score;
  }
  // Devolve a pontuação atual
  public static int GetPoints()
  {
    return score;
  }
  // Reseta a pontuação para zero
  public static void Reset()
  {
    score = 0;
    PlayerPrefs.SetInt("score", score);
    if (instance != null && instance.textoPontuacao != null)
      instance.textoPontuacao.text = "" + score;
  }
}
```

Toast.cs

```
// Script responsável por mostrar uma pequena mensagem (toast) ao ativar um checkpoint
// Também atualiza o ponto de retorno (safePoint) do jogador
using UnityEngine;
using System.Collections;
public class CheckpointToast : MonoBehaviour
{
  public GameObject toastUI; // Elemento visual (UI) a ser mostrado como "toast"
  private bool activated = false; // Impede que o toast seja ativado mais do que uma vez
  void Start()
  {
    // Garante que o toast comece invisível
    if (toastUI != null)
      toastUI.SetActive(false);
  }
  // Detecta entrada do jogador no checkpoint
  private void OnTriggerEnter2D(Collider2D other)
  {
    if (activated) return; // Ignora se já tiver sido ativado
    if (other.CompareTag("Player"))
    {
      activated = true;
      Player player = other.GetComponent<Player>(); // Atualiza o safePoint do jogador (ponto de respawn)
      if (player != null)
      {
        player.AtivarCheckpoint(transform); // Define este checkpoint como novo ponto de retorno
      }
      // Ativa a mensagem temporária na tela
      if (toastUI != null)
        StartCoroutine(ShowMessage(2f)); // Mostra o toast por 2 segundos
    }
  }
  // Coroutine que mostra e depois esconde o toast após determinado tempo
  private IEnumerator ShowMessage(float duration)
  {
    toastUI.SetActive(true);
    yield return new WaitForSeconds(duration);
```

```
        toastUI.SetActive(false);
    }
}
```

04 de maio de 2025 Sessão da Tarde

Tempo estimado de trabalho: Aproximadamente 2h30min

Objetivo da Sessão:

Finalizar e organizar o **Nível 2** com foco em aumentar a complexidade, reforçar a narrativa ambiental e testar os sistemas em cenas múltiplas.

Tarefas Realizadas:

Organização:

- Estrutura da cena do Nível 2 replicada com base no Nível 1:
    - Fundo, Grid, PlataformasMoveis, Inimigos, HUD, Managers, Checkpoints, Colecionaveis e Colliders.
    - Agrupamento consistente de objetos na Hierarquia para facilitar manutenção futura.

Design de Nível:

Nível 1:

- Criadas **4 zonas distintas**:
    - **Zona 1** um pequeno salto e picos para o jogador saber se cair e se tocar nos picos morre.
    - **Zona 2** plataforma que se move horizontalmente para o jogador perceber que tem de esperar e que ele anda nela.
    - **Zona 3** plataformas verticais que ensina o jogador a saltar entre plataformas de modo inteligente.
    - **Zona 4** apresentação dos inimigos, se na zona 3 temos 3 plataformas porque não meter 4 inimigos na zona 4? Quem sabe no futuro meter 2 plataformas horizontais.
- Colocados **17 colecionáveis** (pedras) no nível com foco em exploração. As pedras seguem o caminho normal do nível, facilmente encontrando todas as pedras.

Nível 2:

- Criadas **3 zonas distintas**:
    - **Zona 1** com plataformas móveis vertical e picos no topo onde temos de saltar para uma plataforma movel horizontal para saltar calculadamente para não acertar no inimigo (início mais exigente).

    - **Zona 2** vertical com plataformas móveis em sequência, checkpoint e colecionável escondido fora da visão direta do jogador. Picos entre plataformas para castigar caso falhe as plataformas. Inimigo no final para dar espaço de escolher o caminho de cima (caso percebas que a pedra grande é uma maior recompensa) ou o caminho de baixo (sem recompensa e mais rápido).
    - **Zona 3** com 2 inimigos na zona rápida e fácil (para quem não quer se chatear muito). Zona subterrânea para colecionar o resto das pedras. Quem não teria medo de um monstro e virar para trás e encontrar uma passagem secreta?
- Colocados **22 colecionáveis** (pedras) no nível com foco em exploração e segredo.

Funcionalidades:

- NivelTracker totalmente funcional com toast no final ao apanhar todas as pedras.
- CheckpointToast corrigido (problema era a Tag "Player" ausente no prefab).
- Toasts reorganizados para iniciarem desativados, evitando problemas visuais ao mudar de cena.
- Testes com respawn após morte confirmaram o funcionamento correto dos checkpoints.

Problemas Encontrados:

- Toast de fim de pedras estava a aparecer no início do Nível 2 → resolvido ao desativar o objeto PedrasToast manualmente na Hierarquia.
- Checkpoints não funcionavam inicialmente → o Player estava sem a tag correta (“Player”).

Reflexão Pessoal:

Este segundo nível serviu para consolidar o que aprendi no Nível 1 e para aumentar o desafio de forma natural. Introduzi elementos de exploração mais elaborados, caminhos alternativos e reforço da narrativa visual. O processo de organização da cena e validação das funcionalidades ficou muito mais fluido, e sinto que o projeto está bem preparado para escalar para o Nível 3 (não pedido no trabalho, mas quem sabe um dia para testar mais coisas). Foi adicionada relva nos sítios onde o jogador não recebe dano pois ou não tem como cair ou levar dano de inimigos.

Notas para a próxima sessão:

- Melhorar o que ainda está cru como Menu Inicial, Creditos (pronto para testar lá meter este PDF), Completo e Game Over. Também terei de melhorar o HUD e Toast. Sendo isto o mais importante no momento para acabar
- Terei de procurar sons para meter como fundo e meter ao apanhar pedras e ao completar todas as pedras. Também importante para o jogo de momento.
- Se tiver tempo adicionar uma história inicial com 2 amigos a jogar às apanhadas e um para não se perder meter pedras no chão para dar contexto às pedras no chão.
- Embelezar os níveis. Tenho materiais que não utilizei à minha espera.

Outra vez a parte chata. O código que se criou o que melhorei. Acaba na página 47. Hoje sou mais amigo.

TimerUI.cs

```
// Script responsável por mostrar o tempo decorrido do nível no HUD
using UnityEngine;
using TMPro;
public class TimerUI : MonoBehaviour
{
  // Referência ao texto na UI onde será exibido o tempo
  [SerializeField] private TMP_Text timerText;
  // Variável que acumula o tempo passado desde o início do nível
  private float tempoDecorrido = 0f;
  // Controla se o temporizador está a contar ou não
  private bool emContagem = true;
  void Update()
  {
    // Se o temporizador estiver parado, não faz nada
    if (!emContagem) return;
    // Soma o tempo passado desde o último frame
    tempoDecorrido += Time.deltaTime;
    // Calcula os minutos e segundos separadamente
    int minutos = Mathf.FloorToInt(tempoDecorrido / 60f);
    int segundos = Mathf.FloorToInt(tempoDecorrido % 60f);

    // Atualiza o texto da UI com o tempo formatado (ex: 01:23)
    timerText.text = string.Format("{0:00}:{1:00}", minutos, segundos);
  }
  // Função pública que pode ser chamada para parar o temporizador
```

```
  public void PararTemporizador()
  {
    emContagem = false;
  }
  // Função pública para obter o tempo final (útil para guardar recordes)
  public float ObterTempoFinal()
  {
    return tempoDecorrido;
  }
}
```

Adicionei um timer pois queria saber se os meus níveis duravam 1m no mínimo. Agora estamos prontos para speedrunner fazer isto em menos de 1m. Futuramente quem sabe adicionar algo que guarda o tempo que demorei, estilo Time Attack do Tekken.

ScoreManager.cs

Funções alteradas e explicação:

```
  void Start()
  {
    // Recupera a pontuação guardada anteriormente (se existir)
    score = PlayerPrefs.GetInt("score");
    if (score < 0) { score = 0; }
    // Atualiza o texto da UI com a pontuação inicial
    instance.textoPontuacao.text = score.ToString("D3");
  }

  public static void AddPoints(int points)
  {
    score += points;
    // Guarda nos PlayerPrefs
    PlayerPrefs.SetInt("score", score);
    // Atualiza a UI, se o ScoreManager e o texto estiverem ativos
    if (instance != null && instance.textoPontuacao != null)
      instance.textoPontuacao.text = score.ToString("D3");
  }


  public static void Reset()
  {
    score = 0;
    PlayerPrefs.SetInt("score", score);
```

```
    // Atualiza a UI, se o ScoreManager e o texto estiverem ativos
    if (instance != null && instance.textoPontuacao != null)
      instance.textoPontuacao.text = score.ToString("D3");
  }
```

Todas as linhas com o código: “instance.textoPontuacao.text = "" + score;” foram mudadas para: “instance.textoPontuacao.text = score.ToString("D3");”. Isto foi implementado para condizer com o timer que é predefinido como 00:00, deixando então o score com 3 dígitos aparecendo como 000 sempre e somando para 001.

NivelTracker.cs

```
// Script responsável por acompanhar o número de colecionáveis apanhados num nível específico
// Mostra uma mensagem (toast) quando o jogador apanha todos os colecionáveis do nível
using UnityEngine;
using System.Collections;

public class NivelTracker : MonoBehaviour
{
  // Número total de colecionáveis disponíveis neste nível
  [SerializeField] public int colecionaveisNoNivel;
  // Número de colecionáveis já apanhados pelo jogador neste nível
  private int apanhados = 0;
  // Referência ao objeto UI (toast) que será mostrado ao apanhar todos os colecionáveis
  public GameObject toastFinal;
  void Start()
  {
    // Inicializa o contador de colecionáveis apanhados
    apanhados = 0;
    // Garante que o toast começa invisível
    if (toastFinal != null)
      toastFinal.SetActive(false);
  }

  // Função pública chamada sempre que o jogador apanha uma pedra
  public void AdicionarPedra()
  {
    // Soma 1 ao número de pedras apanhadas
    apanhados++;
    Debug.Log("Pedra apanhada! Total: " + apanhados + "/" + colecionaveisNoNivel);
```

```
        // Se o jogador já apanhou todas as pedras, mostra o toast
        if (apanhados == colecionaveisNoNivel && toastFinal != null)
        {
            // Ativa a mensagem na tela
            toastFinal.SetActive(true);
            // Inicia a rotina para esconder após 3 segundos
            StartCoroutine(EsconderToastFinal());
        }
    }
    // Coroutine que esconde o toast após 3 segundos
    private IEnumerator EsconderToastFinal()
    {
        // Espera 3 segundos
        yield return new WaitForSeconds(3f);
        // Esconde a mensagem
        toastFinal.SetActive(false);
    }
}
```

CoinPickup.cs

```
    void OnTriggerEnter2D(Collider2D other)
    {
        // Garante que só o jogador pode apanhar a moeda
        if (other.GetComponent<Player>() == null)
            return;
        // Adiciona os pontos ao ScoreManager
        ScoreManager.AddPoints(pointToAdd);
        // Toca o som de apanhar moeda
        CoinPickupEffect.Play();
        // Esconde o sprite da moeda (opcional — dá sensação de recolhida antes de
desaparecer)
        GetComponentInChildren<SpriteRenderer>().enabled = false;
        // Avisa o NivelTracker local
        FindFirstObjectByType<NivelTracker>()?.AdicionarPedra();
        // Destroi o objeto após 1 segundo (dá tempo para o som tocar)
        Destroy(gameObject, 1.0f);
    }
```

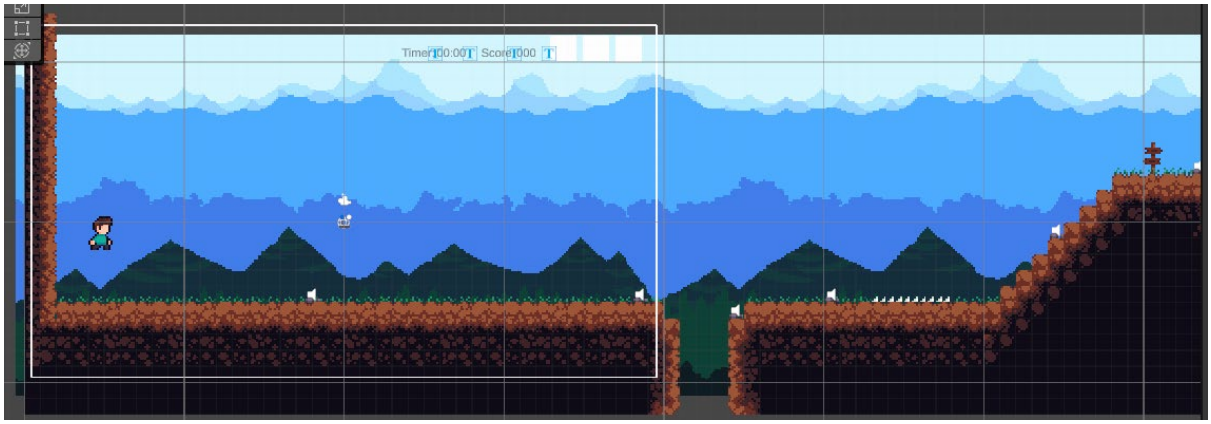
Adicionadas as linhas:

```
// Avisa o NivelTracker local
```

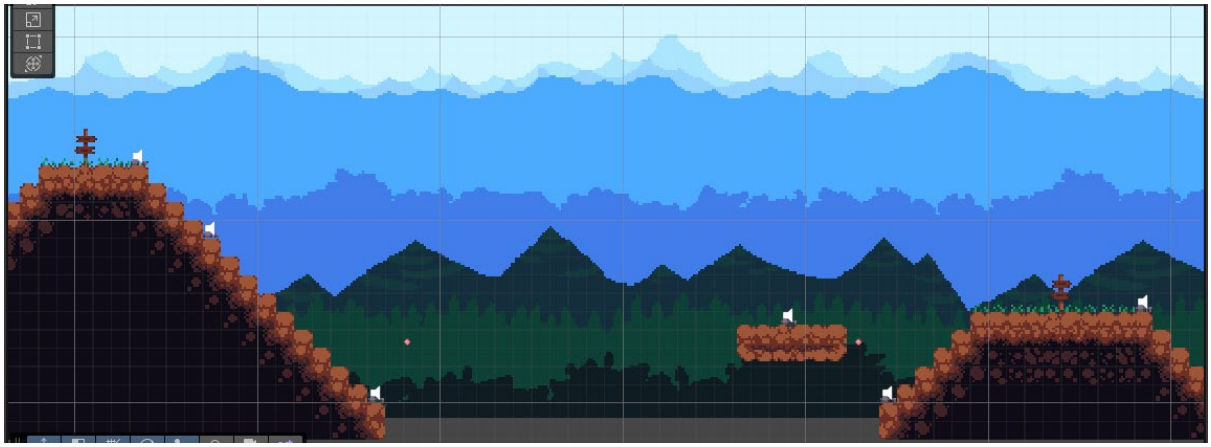
FindFirstObjectByType<NivelTracker>()?.AdicionarPedra();

Para adicionar pedra quando apanhada de modo ao NivelTracker funcione.

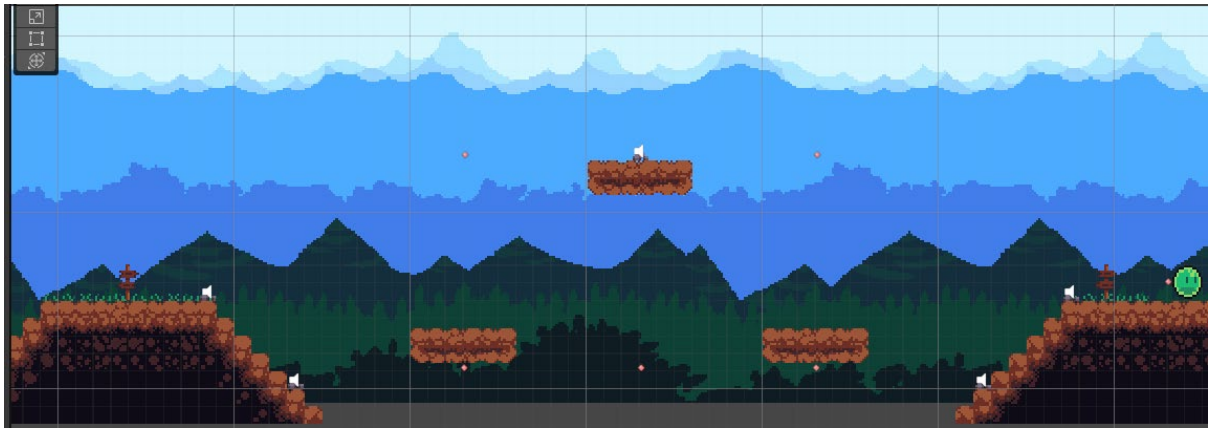
Foto(s) para memória visual:



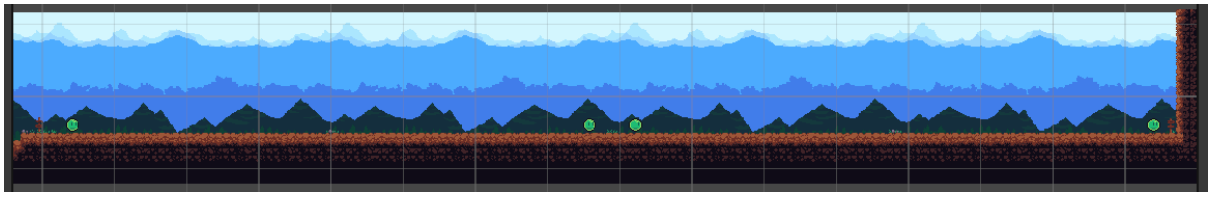
Nível 1 zona 1



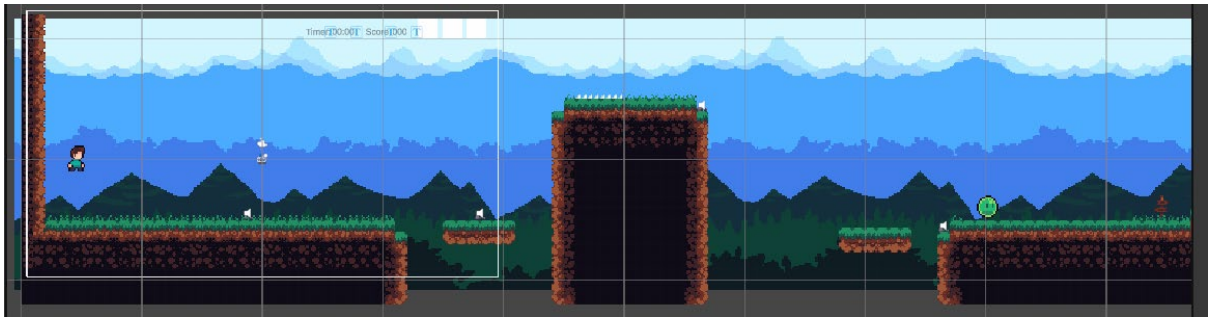
Nível 1 zona 2



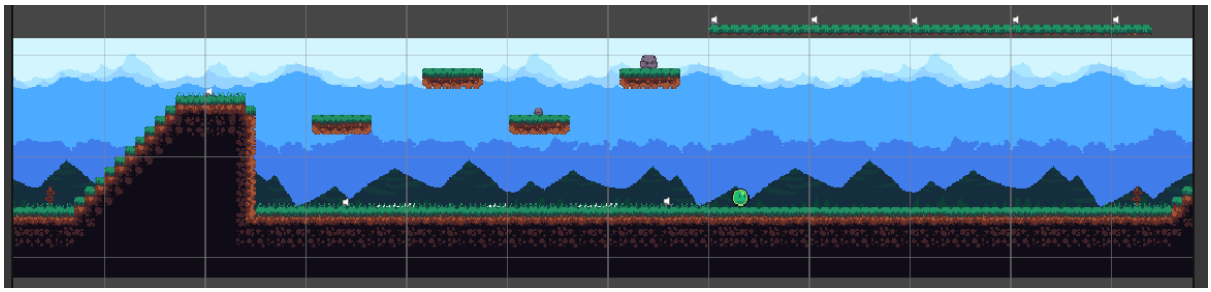
Nível 1 zona 3



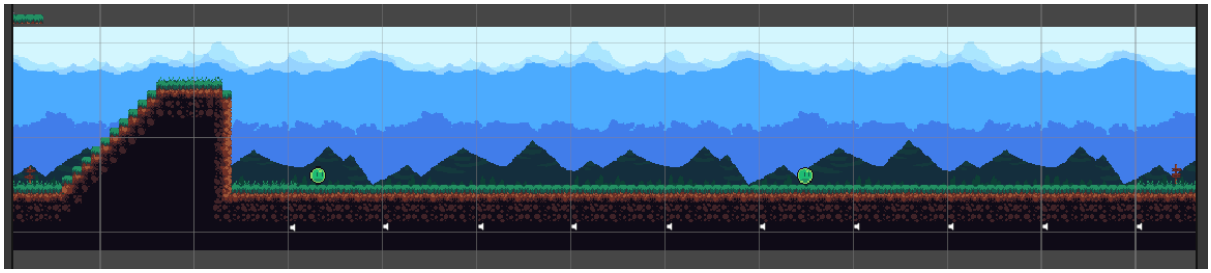
Nível 1 zona 4



Nível 2 zona 1



Nível 2 zona 2



Nível 2 zona 3

Posso dizer que o trabalho está quase quase pronto para entrega. Por favor, lê isto só no final de jogares o jogo. Senão o nível 2 não mete piada.

04 de maio de 2025 Sessão da Noite

Tempo estimado de trabalho: Aproximadamente 1h45min

Objetivo da Sessão:

Finalizar os detalhes visuais do jogo, incluindo HUD, feedback visual de interação e fundo do Nível 2.

Tarefas Realizadas:

Interface e feedback visual:

- Atualização e integração de **toasts pixelizados** para feedback de:
  - Checkpoint alcançado
  - Todas as pedras colecionadas
- Adicionados ao sistema de UI com estrutura consistente e ativação temporária automática.

HUD:

- Criados e aplicados **corações pixelizados** para representar a vida do jogador:
  - HeartFull, HeartEmpty (coração negro para dano)
  - Colocados no canto superior direito com alinhamento e tamanho consistentes.
- Confirmada compatibilidade visual com o timer e score usando a fonte *Jura*.

Fonte:

- Fonte escolhida: **Jura**
- Aplicada à UI principal (títulos, botões, score e timer)
- Ajustado espaçamento e hierarquia visual para reforçar clareza e estilo.

Menus:

- Criado botão de fechar com "X" pixelizado desenhado no Illustrator.
- Implementado no menu de créditos com sprite visível, funcional e responsivo.

Ecrãs de fim de jogo:

- Tela de "**Game Over**" criada com destaque em vermelho e botões de reinício/voltar ao menu.
- Tela de "**Completo**" implementada com score apresentado e botão de continuar/menu.
- Ambas utilizam UI consistente com fundo, fonte, cor e layout iguais.

Design de nível:

- Foi adicionado o **limite de fundo ao Nível 2**, que ainda não tinha elemento para dar dano caso o player caia para o abismo (nem sei que nome dar a essa parte de baixo).
- Fundo colocado corretamente atrás dos elementos de Grid e PlataformasMoveis na hierarquia.

Menu pausa:

- Atualização de botões para ficarem idênticos ao presente no Menu Inicial, Completo e Game Over.

Reflexão Pessoal:

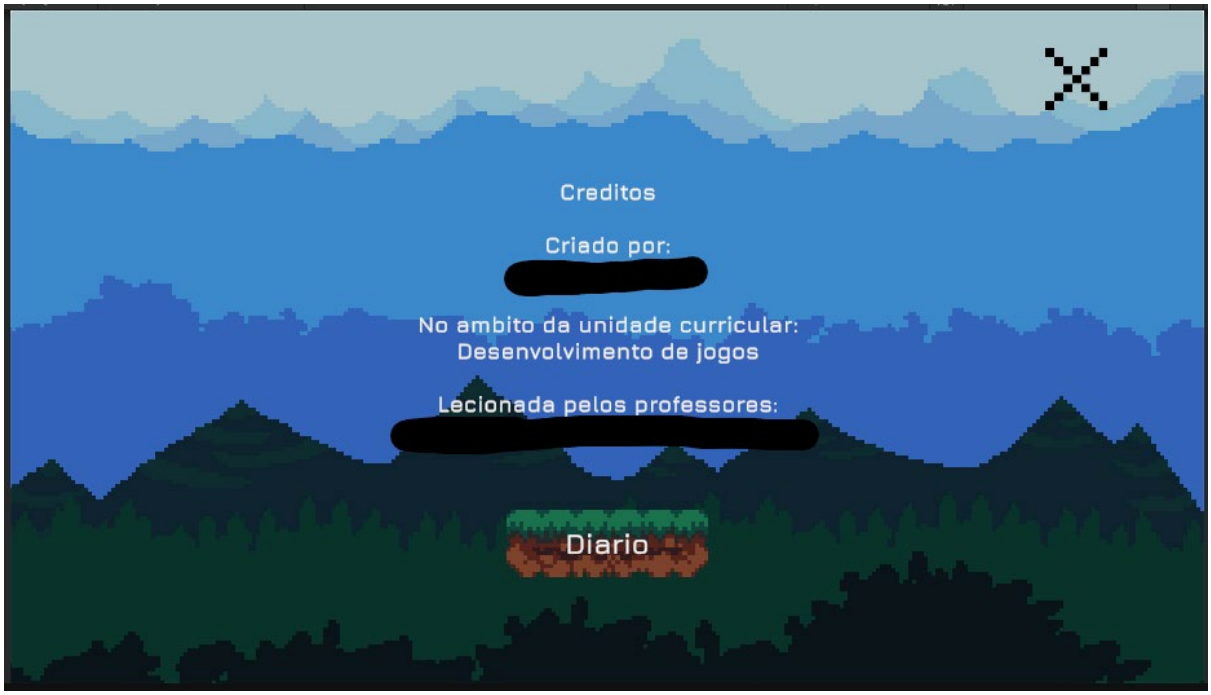
Esta sessão foi focada em dar **personalidade e clareza visual** ao jogo. Agora a HUD está não só funcional, mas com estilo próprio. Os toasts, corações, menus e feedbacks visuais estão todos alinhados com o espírito calmo e misterioso do jogo. Com o fundo do Nível 2 adicionado, o visual geral está completo e coeso.

Nota para a próxima sessão:

- Procurar e integrar **sons** — música de fundo, som ao apanhar pedras e ao terminar o nível. Esta será a próxima etapa essencial para a conclusão.
- Meter os botões do menu pausa a funcionar.

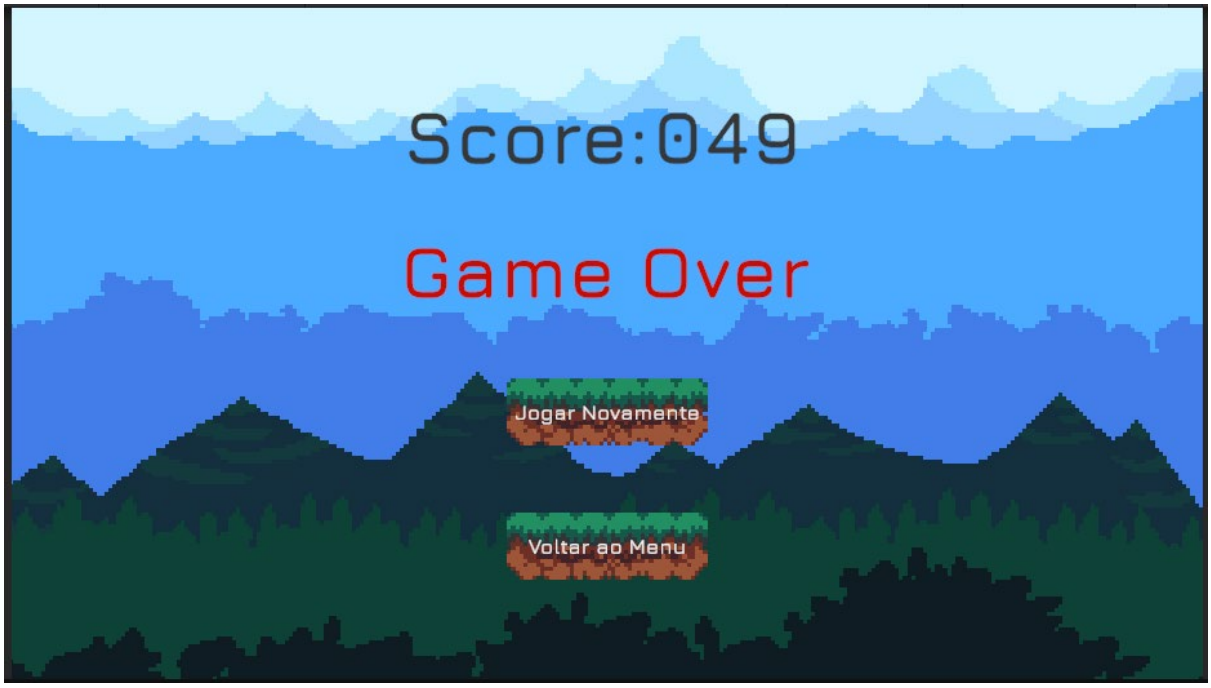
Foto(s) para memória visual:

Menu Inicial

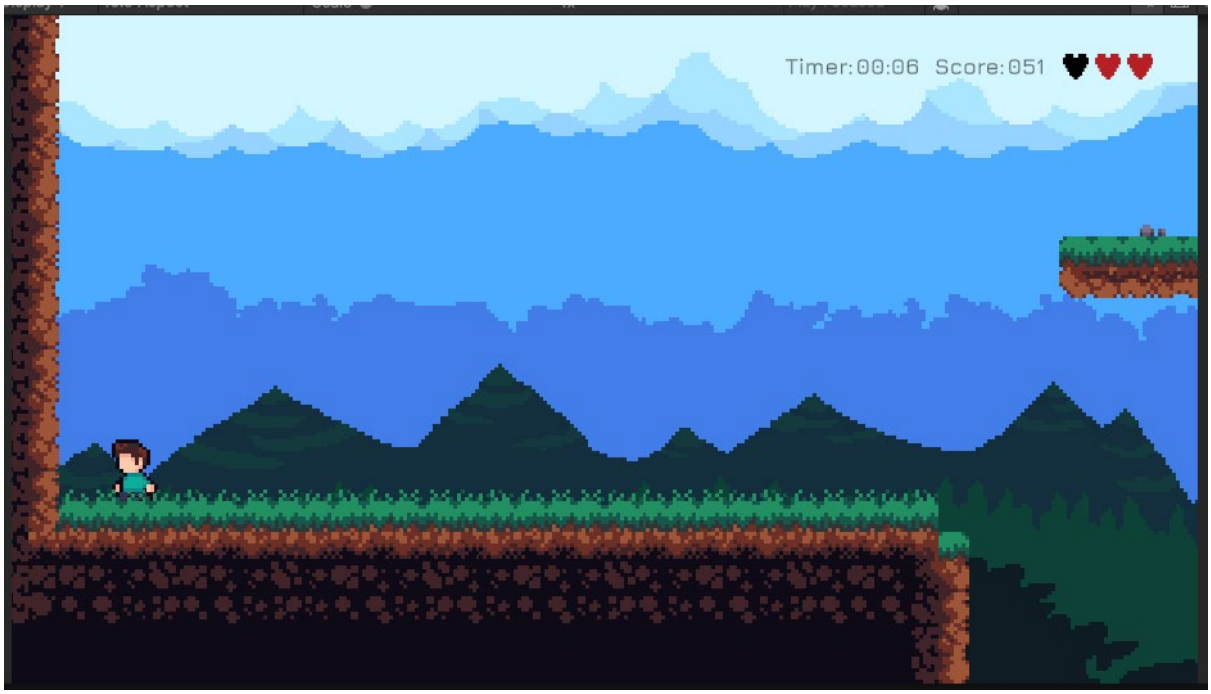


Créditos

Completo

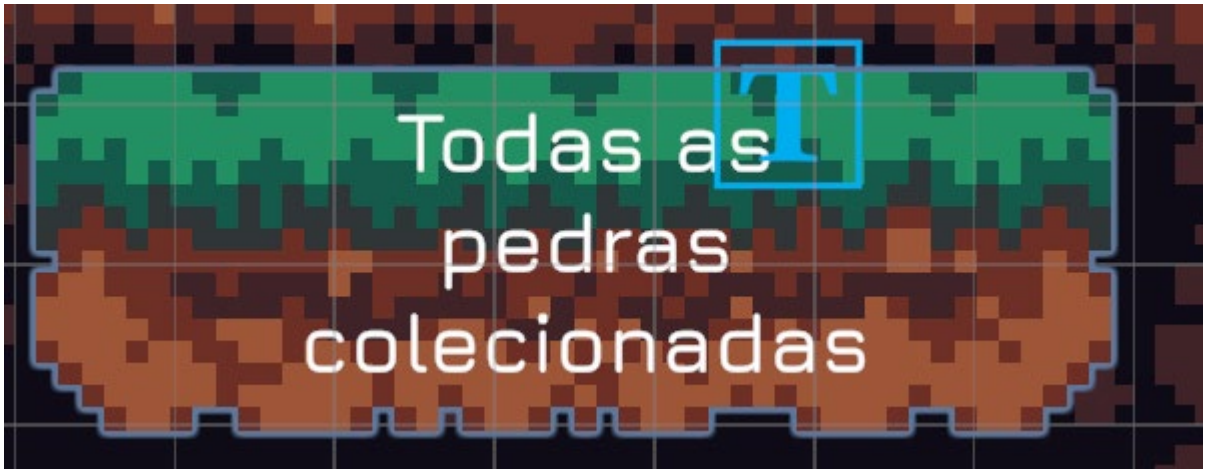


Game Over



HUD atualizado com corações pixelizados

Menu pausa

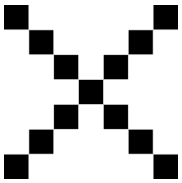


Toast Todas as pedras colecionadas!

Toast Checkpoint alcançado!

Coração Cheio criado no Illustrator por mim para usar no HUD

Coração Vazio criado no Illustrator por mim para usar no HUD



X criado no Illustrator por mim para usar nos Créditos

Admite que sem código fica mais fácil ler este diário. Eu sei.

Notas finais:

- As imagens da pessoa e do gato foram geradas por AI (ChatGPT a simples e a capa, as restantes Sora);
- Todos os screenshots foram tirados de gameplays guardadas pela Steam.
- Texto gerado por AI (ChatGPT) e melhorado por mim após ler tudo.