Desenvolvimento com Motores de Jogos II

Aula 15 - Jogo Farm Coins - Parte 2

Apresentação

Salva de palmas! Chegamos à última aula da disciplina de Motores de jogos II!

Nesta última etapa, aproveitaremos nosso aprendizado sobre a ferramenta Unity 3D e adicionaremos ao jogo Farm Coins alguns recursos extras, utilizando as técnicas que aprendemos durante o curso.

Além disso, adicionaremos moedas que podem ser coletadas, obstáculos móveis e ainda criaremos o HUD para exibir a pontuação do jogador. Claro que você está totalmente livre para criar também outros desafios no seu jogo.

Vamos lá!

Objetivos

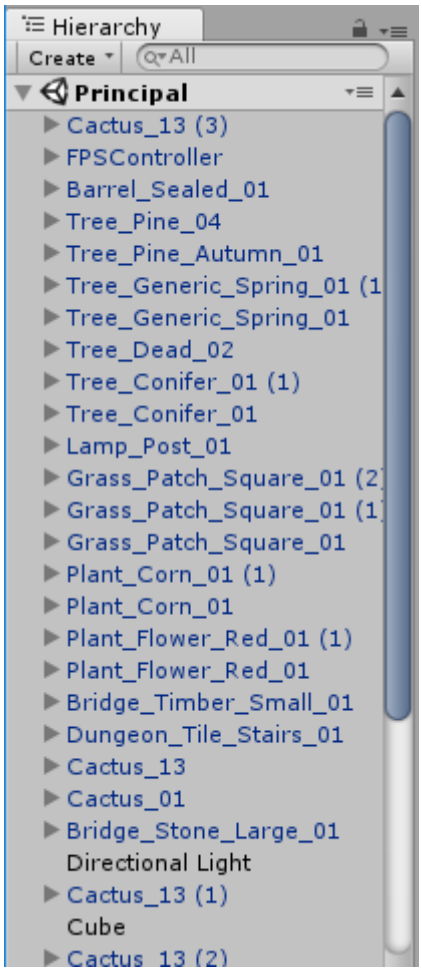
Ao final desta aula, você deverá ser capaz de:

- Adicionar as moedas no jogo Farm Coins
- Criar o HUD do jogo para contabilizar a pontuação.
- Adicionar obstáculos móveis.
- Exportar o seu jogo para a plataforma desejada

Organizando os Objetos do Jogo

Iniciaremos a aula organizando os GameObjects que foram adicionados em nosso jogo na aula passada, na qual você criou o cenário inicial. Se durante a criação do cenário você simplesmente adicionou vários modelos 3D sem se preocupar com sua organização, então a janela Hierarchy deve estar parecida com a mostrada na **Figura 1**.

Figura 01 - Janela Hierarchy com modelos ainda desorganizados.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

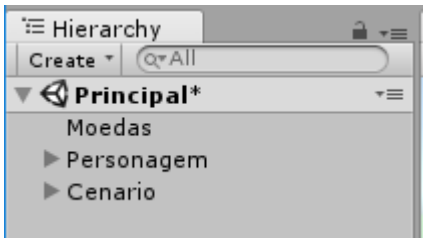
Bem bagunçada, não é mesmo? Não se preocupe, pois arrumaremos isso agora. No Unity, uma maneira fácil de organizar os objetos no Hierarchy é de forma... hierárquica!

Ok, o trocadilho foi ruim, mas a ideia é boa, vamos lá! Siga os passos:

1. Crie inicialmente na raiz do Hierarchy três GameObjects vazios com os nomes “Cenario”, “Personagem” e “Moedas”.
2. Mova todos os modelos 3D do cenário criado para cima do GameObject “Cenario” (pode selecionar vários ao mesmo tempo clicando no primeiro, segurando a tecla SHIFT e clicando no último da lista).
3. Mova o FPSController para cima do GameObject “Personagem”.

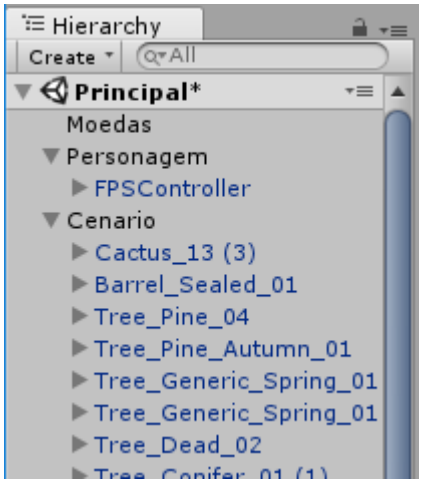
Por enquanto, o GameObject “Moedas” ficará sem nada abaixo dele. Se fez tudo certo, o seu Hierarchy deve estar parecido com o exibido na **Figura 2**. Se expandir os GameObjects da raiz, você verá algo como mostrado na **Figura 3**.

Figura 02 - Hierarchy com objetos separados em categorias.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Figura 03 - Objetos expandidos na raiz do Hierarchy.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Se você desejar, pode organizá-los ainda mais criando uma hierarquia de objetos dentro do Cenario. Você pode, por exemplo, agrupar os objetos por tipo, como cercas, pedras, plantas, ou pode, também, agrupar por regiões, tais como “Casa Principal”, “Primeira Ponte”, “Fazenda”, etc. Tudo depende de como você prefere localizar seus objetos futuramente, além de como está organizada a sua cena.

Atenção!

O Unity oferece completa liberdade para modificar a organização, entretanto é importante lembrar que, quando uma série de objetos está abaixo de um objeto específico, a movimentação, rotação e escalonamento desse objeto específico também mudará a posição, rotação e escala dos seus filhos (os quais estão abaixo dele). Então organize seu projeto de forma que esse comportamento seja útil para você, e não um problema.

Adicionando as Moedas

No jogo Farm Coins o objetivo principal é atravessar os obstáculos coletando as moedas espalhadas pelo cenário o mais rápido possível. Já aprendemos a criar itens coletáveis no jogo Polygonal Rescue, então utilizaremos as mesmas técnicas para as moedas.

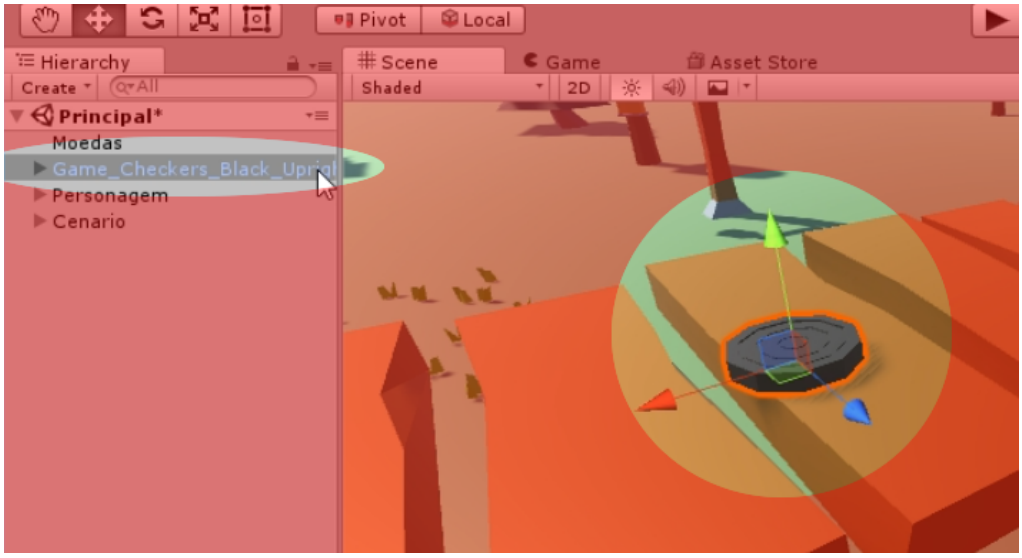
Precisaremos criar um prefab para as moedas com um modelo 3D, com Material e com a tag “Moeda” configurada. A Tag é importante para o personagem poder reconhecê-la quando houver a colisão. Como importamos o pacote de assets chamado “Low Poly v2” no nosso projeto, você pode procurar na sua pasta se existe já pronto um modelo 3D que representa uma moeda, entretanto você pode também utilizar um outro modelo 3D de sua preferência para a moeda. Veja na **Figura 4** um exemplo de um modelo que servirá como a nossa moeda do pacote de assets Low Poly v2.

Figura 04 - Modelo 3D que será utilizado como a nossa moeda.

Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Arraste esse modelo para qualquer região do mapa e veja que ele será exibido na posição escolhida por você. Na **Figura 5** é possível verificar seu posicionamento.

Figura 05 - Moeda ainda não configurada já posicionada na cena.

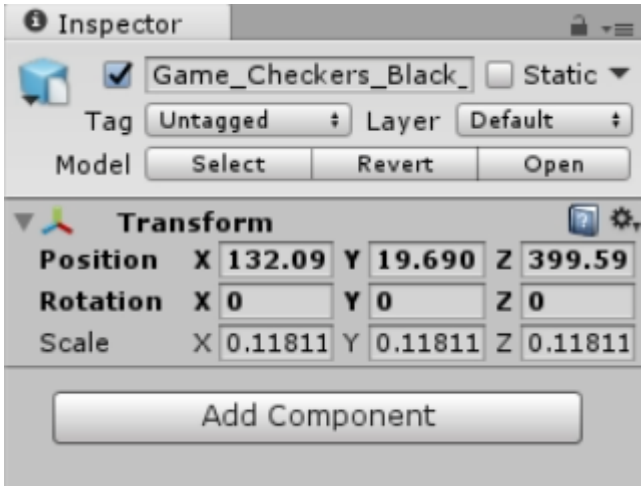


Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Como você observou, a moeda está tanto no Scene View como no Hierarchy. Se você reparar, o nome dela no Hierarchy ficou “Game_Checkers_Black_Upright_01”. Esse foi o nome que o criador desse modelo 3D escolheu. Na verdade, esse modelo

trata-se de uma peça de um jogo de damas, mas servirá para nosso exemplo. Caso clique nesse objeto, verá que no Hierarchy ele é somente um objeto vazio, como mostra a **Figura 6**.

Figura 06 - Inspector da moeda adicionada e ainda não configurada.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Mas como assim vazio?

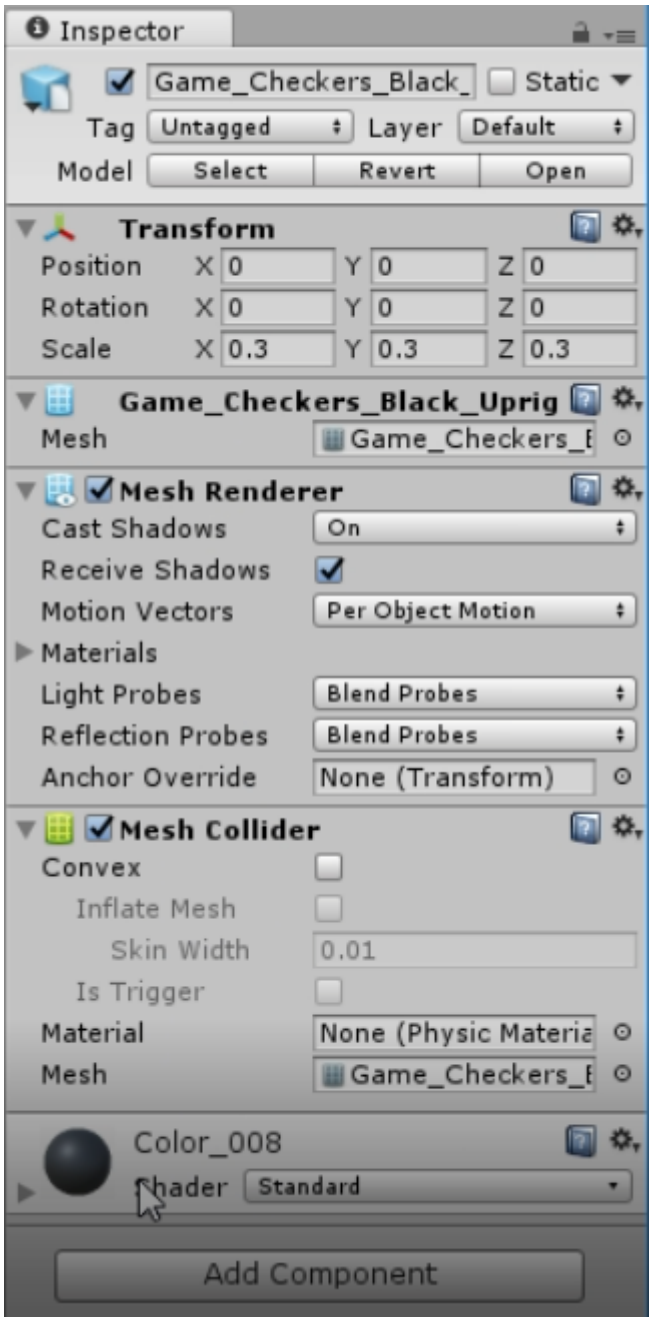
Calma! Acontece, nesse caso, que a peça de damas a ser usada como base para a nossa moeda é na verdade um prefab com outro objeto filho abaixo dele. Expanda a moeda no Hierarchy e verá esse objeto filho, como mostra a **Figura 7**.

Figura 07 - Objeto filho da moeda adicionada.

Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Clicando nesse objeto filho, você verá agora que no Inspector existem todos os componentes esperados, como o Mesh Filter, Mesh Renderer, Mesh Collider e o Material padrão (ver **Figura 8**).

Figura 08 - Inspector do objeto filho da moeda adicionada.

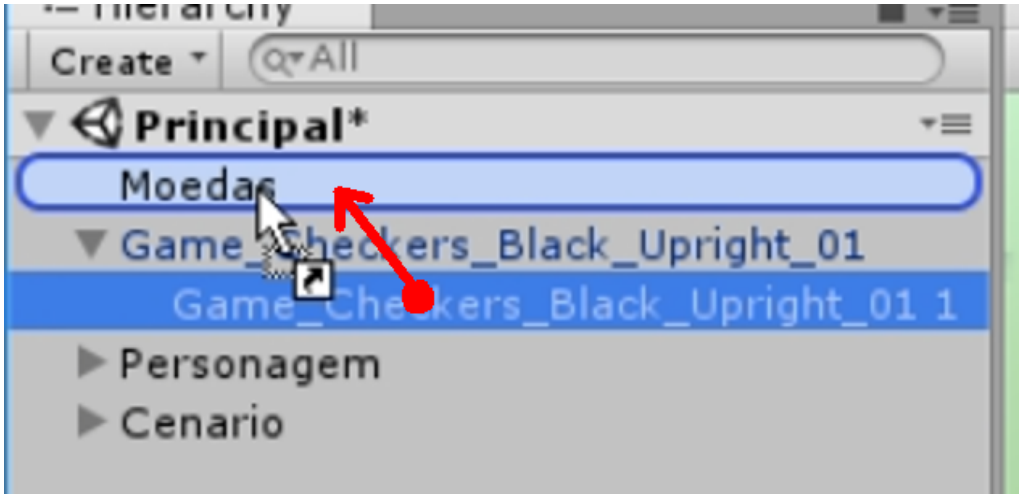


Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Mas por qual motivo esse prefab da moeda veio dessa forma? Isso foi uma decisão do seu criador e você precisa estar atento para a estrutura hierárquica dos objetos adicionada no seu projeto, pois ela nem sempre estará como você deseja. Nesse caso só nos interessa o objeto filho da moeda, então selecione-o (o filho somente) no Inspector e arraste-o para o GameObject vazio que criamos chamado

“Moedas”, como você pode observar na **Figura 9**. Isso fará com que a instância desse prefab original da peça de damas seja quebrada, pois utilizaremos somente o objeto filho.

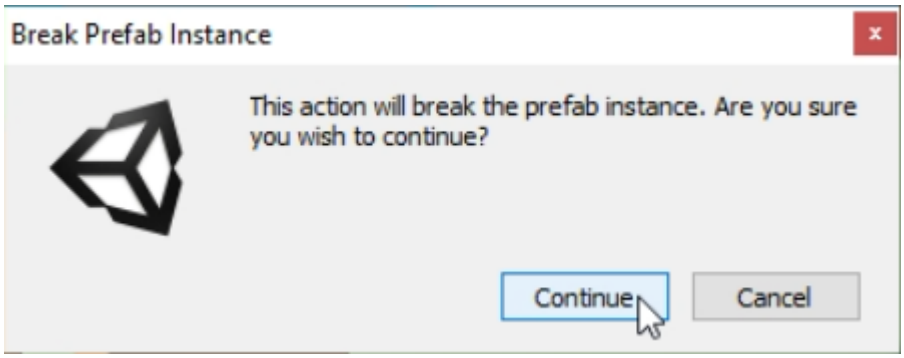
Figura 09 - Arrastando somente o filho do prefab “Game_Checkers_Black_Upright_01” para dentro do objeto “Moedas”.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Uma mensagem informando essa “quebra da instância” será apresentada, como na **Figura 10**. Clique em “Continue”.

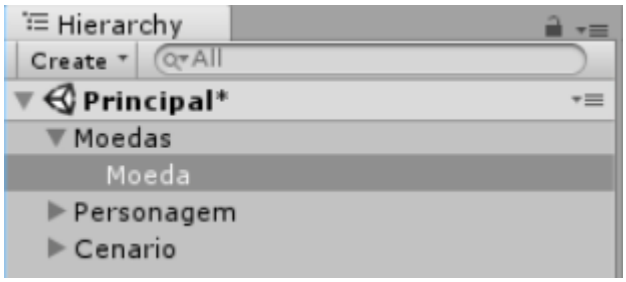
Figura 10 - Mensagem de quebra de instância.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Agora a moeda já está dentro de “Moedas”, mas restou o objeto vazio “Game Checkers_Black_Upright_01” no Hierarchy, portanto você pode removê-lo. Renomeie o objeto que você adicionou em “Moedas” para simplesmente “Moeda” e verá que seu Hierarchy ficará como mostrado na **Figura 11**.

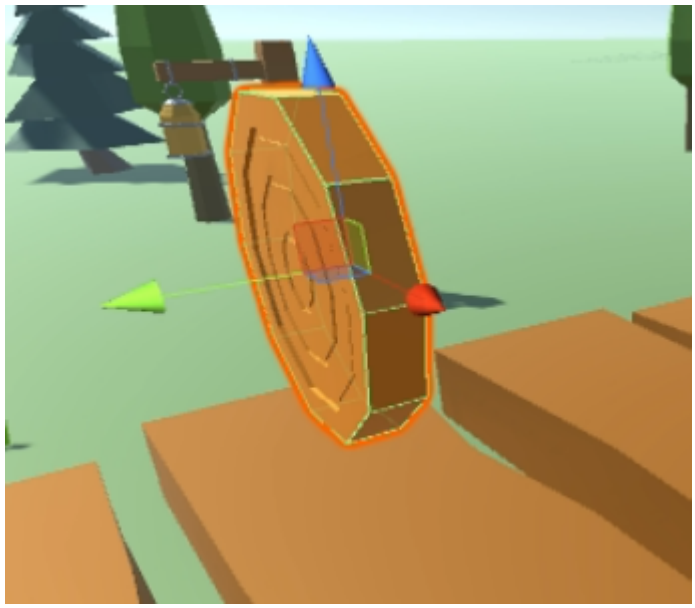
Figura 11 - Hierarchy já com a moeda organizada abaixo de “Moedas” e já renomeada corretamente.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Com a Moeda selecionada, rotacione-a para ela ficar na vertical e troque seu Material para um da cor dourado, arrastando-o para cima desse objeto. Você pode criar um Material ou escolher dentre os disponíveis no pacote de assets Low Poly v2 que você importou, pois esse pacote tem uma pasta chamada Materials com vários disponíveis. O resultado deverá ficar similar ao da **Figura 12**.

Figura 12 - Moeda rotacionada na vertical com um Material dourado.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Agora, configuraremos a nossa moeda para que ela possa ser coletada pelo personagem. Para isso, selecione a moeda, localize o Mesh Collider no Inspector e marque as opções “Convex” e “Is Trigger”, como na **Figura 13**. Isso fará com que a

moeda não se comporte como um obstáculo quando for colidida com o personagem, entretanto o evento OnTriggerEnter será gerado quando detectar a colisão.

Figura 13 - Opção Convex e Is Trigger do Mesh Collider da moeda.

Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Atenção!

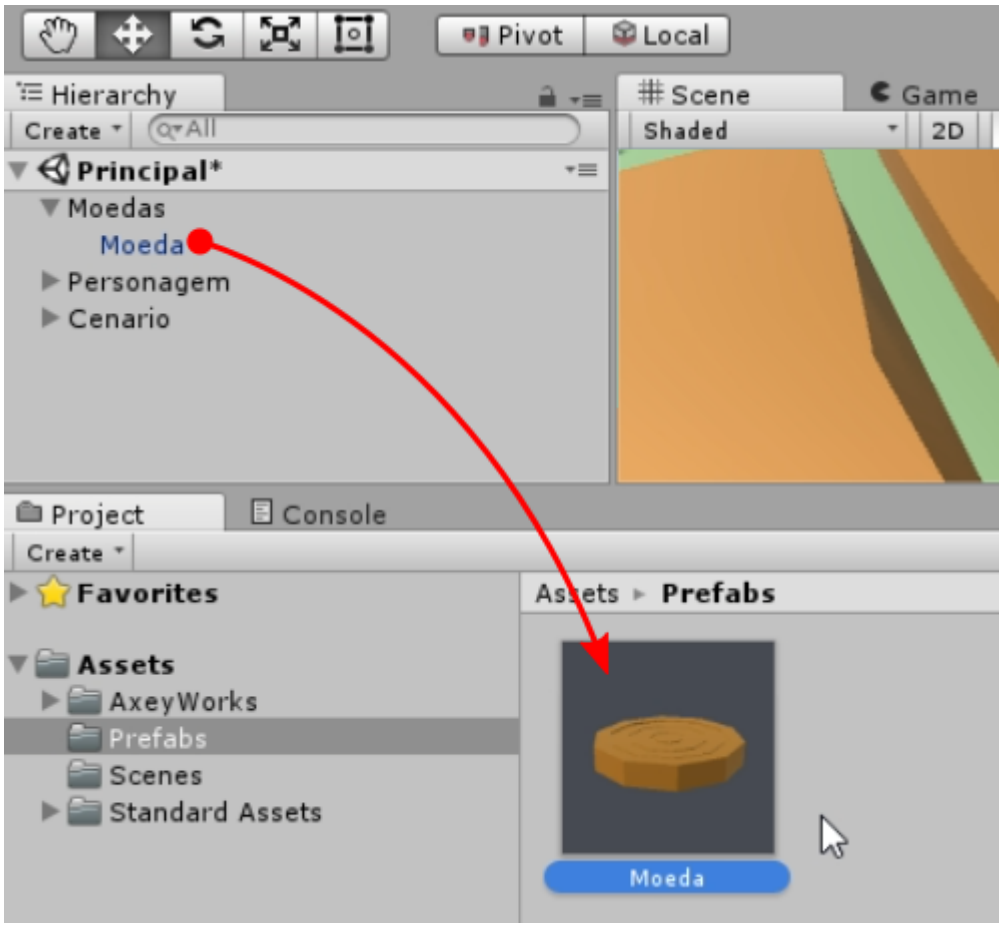
O modelo 3D que estamos utilizando como exemplo nesta aula já tinha um componente Mesh Collider adicionado, por isso só estamos marcando as opções **Convex** e **Is Trigger** para ele poder reagir à colisão com o personagem. É possível que, se você for utilizar um outro modelo 3D para a moeda, ele não tenha um Collider adicionado, então você terá de adicionar utilizando a opção Add Component ->Physics -> Box Collider (ou outro tipo de Collider que achar adequado) e marcando em seguida a opção Is Trigger.

Criando o Prefab “Moeda”

Com nossa moeda configurada, agora é o momento de criar um prefab para ela, de modo a permitir que outras instâncias idênticas a dessa moeda sejam adicionadas no jogo sem você precisar repetir todo o processo de configuração.

Crie uma pasta chamada “Prefabs” na raiz do projeto e arraste o objeto “Moeda” do Hierarchy para essa pasta. Veja a **Figura 14**.

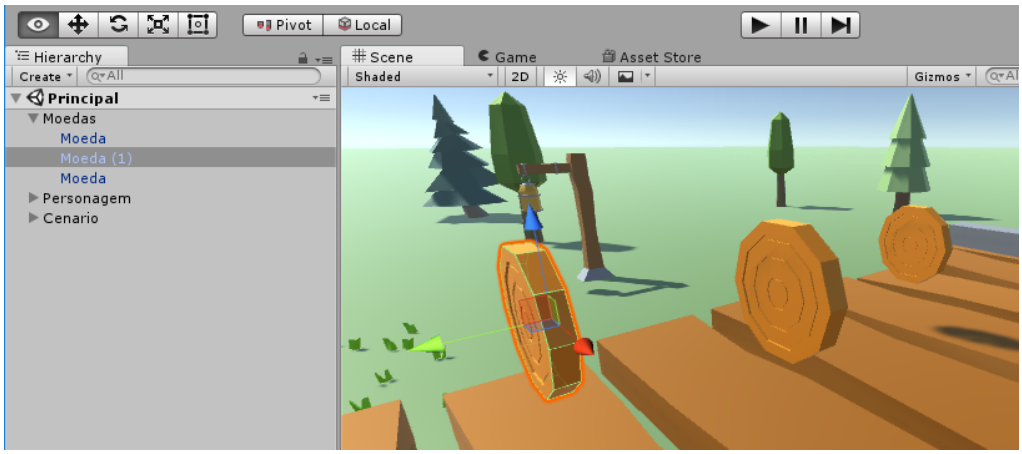
Figura 14 - Criando o prefab para a moeda.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Agora, com o prefab “Moeda” criado, repare que o objeto com o nome “Moeda” no Inspector já ficou em azul, demonstrando ser uma instância desse prefab. Para criar mais instâncias, basta arrastar o prefab “Moeda” que você acabou de criar para qualquer região do cenário. Assim você pode adicionar quantas moedas desejar, todas elas com a mesma configuração. Não esqueça de, quando arrastar uma nova moeda para o cenário, sempre ajustar sua posição e também organizar esses objetos no Hierarchy para que todos fiquem dentro do GameObject “Moedas”. Na **Figura 15** você pode ver algumas moedas criadas no cenário utilizando instâncias desse prefab.

Figura 15 - Algumas moedas adicionadas no cenário como instâncias do prefab criado.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Atividade 01

1. Crie outro prefab chamado MoedaEspecial com uma Tag de mesmo nome e com o Material de uma cor diferente. Esse prefab pode ser utilizado para moedas posicionadas em locais mais difíceis do cenário.

2. Modifique seu cenário posicionando diversas moedas nos locais que desejar.

Coletando as Moedas

Agora que as moedas já estão posicionadas, é hora de criar uma forma de o personagem coletá-las. Para isso, você deverá adicionar um script para o FPSController que está dentro do GameObject Personagem. Veja a **Figura 16**.

Figura 16 - FPSController exibido dentro de Personagem.

Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Selecione o FPSController e adicione um script C# chamado “ColetarMoedas”, o qual terá um método nomeado de OnTriggerEnter(Collider col). Esse método será chamado quando o personagem colidir com uma moeda e inicialmente apenas verificará se o colisor com o qual o personagem entrou em contato é de uma moeda. Caso seja, destruirá o seu GameObject, removendo a moeda da cena.

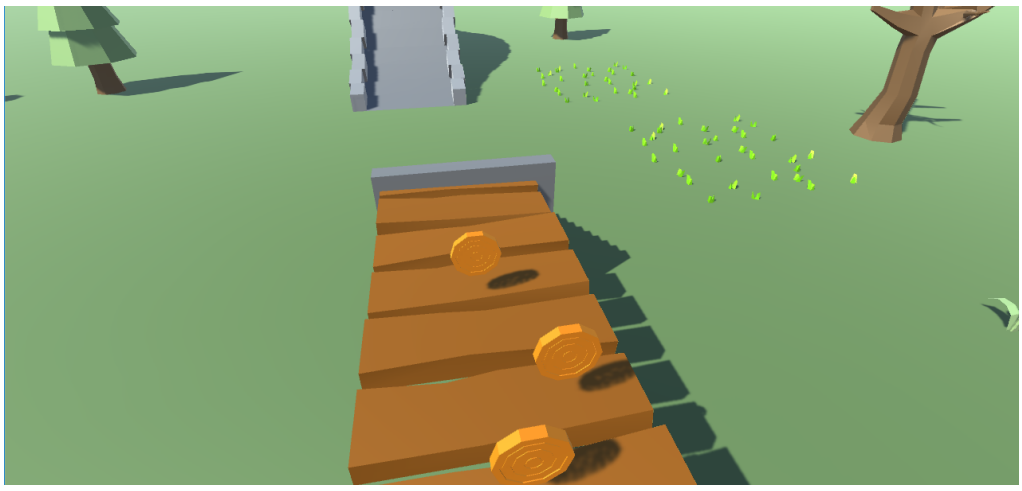
Abaixo segue o código do ColetarMoedas.cs:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ColetarMoedas : MonoBehaviour {
6
7      void OnTriggerEnter(Collider col) {
8          if (col.CompareTag ("Moeda")) {
9              Destroy (col.gameObject);
10         }
11     }
12 }
```

Listagem 1 - ColetarMoedas.cs

Execute agora o jogo e caminhe com o personagem em direção às moedas. Se tudo tiver sido feito corretamente, as moedas sumirão assim que o personagem entrar em contato com elas. Na **Figura 17** você pode ver o jogo sendo executado e a visão do personagem em primeira pessoa.

Figura 17 - Jogo Farm Coins sendo executado e personagem prestes a colidir com uma moeda.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Lembre-se que para controlar o jogador você pode utilizar as teclas W,S,A,D para se mover, espaço para pular e o mouse para olhar para qualquer direção.

Você já iniciou a criação de jogo de plataforma e coleta de moedas em primeira pessoa e ainda há muito a ser adicionado no jogo, só depende de sua criatividade.

Veja agora um vídeo que mostra como adicionar moedas na cena e também como criar um script no personagem para coletá-las quando ele entrar em contato com uma delas.

Vídeo 01 - Moedas

Atividade 02

1. Faça um planejamento de um cenário desafiador e crie-o no Unity utilizando as técnicas estudadas.
2. Crie regiões onde, se o jogador cair, o jogo será reiniciado. Para isso, utilize Colliders nelas e crie um script que verifica se o jogador entrou em contato com a região (o script pode ser um GameObject vazio com um Collider). Caso isso tenha ocorrido, carregue novamente a cena atual da mesma forma que fez com o sistema de mudança de fase do jogo Polygonal Rescue.

Acumulando e Exibindo a Pontuação

Agora, para adicionar o recurso de acumular e exibir a pontuação do jogo no Farm Coins, utilizaremos as técnicas que aprendemos na criação do jogo Polygonal Rescue.

No processo de desenvolvimento de jogos, é comum se aproveitar muitos códigos e assets criados em jogos anteriores, fazendo somente as adaptações necessárias para o seu novo jogo. Isso é algo muito importante para que você não precise sempre reinventar tudo e começar do zero.

Caso lembre-se, no Polygonal Rescue criamos um GameObject vazio chamado “GameManager” que continha um script de mesmo nome. Esse script era responsável por acumular a pontuação do jogo em uma variável chamada inteira “pontuacao”. Além disso, ele verificava se o tempo restante para finalizar a fase já tinha se esgotado e, se tivesse, a variável “finalizado” recebia o valor “true”. Vamos usar o mesmo script, que também terá um tempo limite para finalização, no jogo Farm Coins. Na verdade, o jogo Farm Coins é bem parecido com o Polygonal Rescue, entretanto você pode ficar à vontade para criar vários recursos novos utilizando as técnicas que aprendeu.

Acumulando a Pontuação

Inicialmente, crie um GameObject vazio chamado “GameManager” e adicione nele um novo script também com o nome “GameManager”. Esse script deve ter o seguinte código, igual ao do jogo Polygonal Rescue:

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

public class GameManager : MonoBehaviour {
    public int pontuacao;
    public float tempo_restante = 60.0f;
    public bool finalizado = false;

    //Para acessar: GameManager.instancia
    public static GameManager instancia;

    //Singleton
    void Awake(){
        if (instancia == null) {
            instancia = this;
            DontDestroyOnLoad (gameObject);
        } else {
            Destroy (gameObject);
        }
    }

    void Update() {
        tempo_restante -= Time.deltaTime;
        if (tempo_restante <= 0f) {
            tempo_restante = 0f;
            // finalizar o jogo
            finalizado = true;
        }

    }

}
```

Listagem 2 - Código do GameManager.cs

O script GameManager tem a variável “pontuacao”, que acumula a pontuação atual do jogador; a variável tempo_restante, que guarda quantos segundos restam para finalizar a fase; a variável boelana “finalizado”, que informa se o jogo finalizou ou não; e também a variável “instancia”, que é marcada como static e é do tipo GameManager (o próprio tipo da classe). A variável “instancia” guardará uma referência à única instância do GameManager existente no jogo e capaz de ser acessada diretamente pelo comando GameManager.instancia. O método Awake garante a existência de somente uma instância do GameManager, armazenando sua referência na variável “instancia”, assim como executando o comando DontDestroyOnLoad, o qual indica que o GameManager será mantido na memória mesmo se uma nova cena for carregada (na mudança de fase, por exemplo). O

método Update() é responsável por decrementar a variável tempo_restante e setar a variável “finalizado” para true caso o tempo se esgote. Lembrando: esse padrão de projetos que mantém sempre apenas uma instância de uma classe na memória é chamado de Singleton.

Agora precisamos somente fazer uma adaptação no script ColetaMoedas.cs para ele executar o comando GameManager.instancia.pontuacao++ toda vez que uma moeda for coletada.

O código final do ColetaMoedas.cs segue abaixo:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ColetarMoedas : MonoBehaviour {
6
7      void OnTriggerEnter(Collider col) {
8          if (col.CompareTag ("Moeda")) {
9              Destroy (col.gameObject);
10             GameManager.instancia.pontuacao++;
11         }
12     }
13 }
```

Listagem 3 - Código do ColetaMoedas.cs acumulando a pontuação no GameManager.

Veja agora um vídeo que mostra como adaptar o sistema de acúmulo de pontuação quando o jogador coleta moedas. Esse sistema também foi criado anteriormente no jogo Farm Coins.

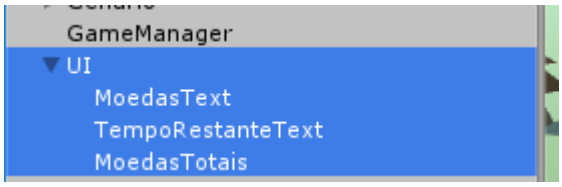
Vídeo 02 - Pontuação

Criação do HUD (Exibindo a pontuação)

Agora, exibiremos a pontuação do jogo em um HUD criado por meio da utilização dos componentes UI do Unity. Usaremos as mesmas técnicas aprendidas no jogo Polygonal Rescue para realizar essa tarefa.

Inicialmente, crie um Canvas (GameObject -> UI -> Canvas) chamado “UI” na sua cena. Depois, adicione pelo menos um Text para as moedas coletadas na fase atual, outro para o tempo restante e um terceiro para as moedas coletadas no total. Veja a **Figura 18**.

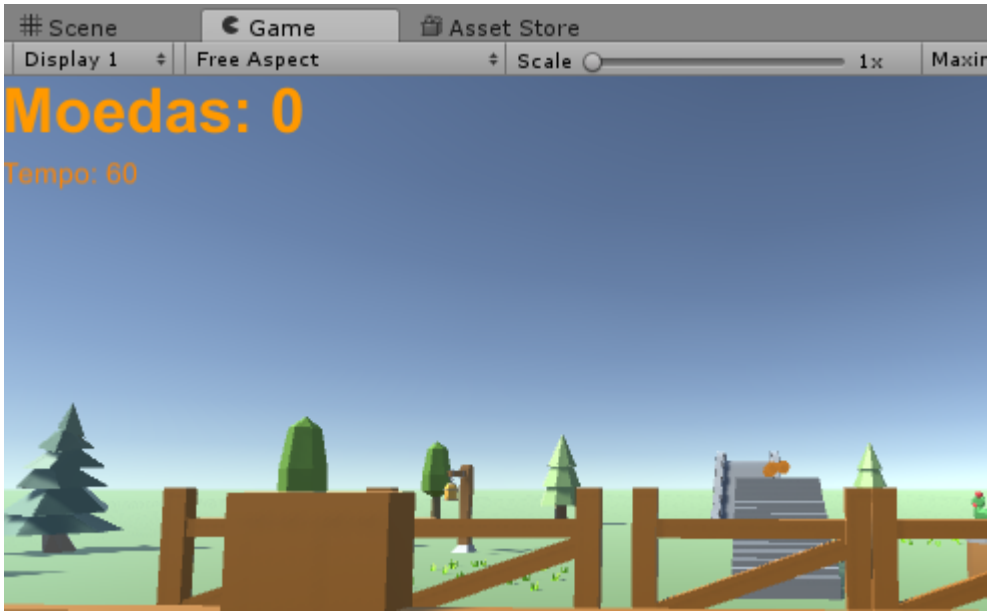
Figura 18 - HUD com os campos de texto necessários.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Clique duas vezes no UI para visualizá-lo no Scene View e utilize as técnicas estudadas para organizar os objetos Text no Canvas da maneira como deseja que sejam exibidos no jogo. A **Figura 19** mostra um exemplo. Lembre-se que o Text exibidor das moedas finais deve ter o componente Text desabilitado (basta desmarcar o checkbox do componente no objeto), pois só é exibido no fim do jogo.

Figura 19 - Elementos da UI durante a execução do jogo.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

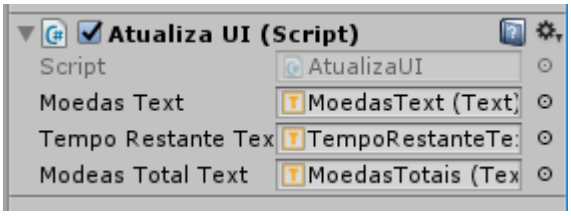
Agora adicione um script chamado AtualizaUI no objeto UI, com o seguinte código:

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

public class AtualizaUI : MonoBehaviour {
    public Text moedasText;
    public Text tempoRestanteText;
    public Text modeasTotalText;

    // Update is called once per frame
    void Update () {
        if (GameManager.instancia.finalizado) {
            moedasText.enabled = false;
            tempoRestanteText.enabled = false;
            modeasTotalText.enabled = true;
            modeasTotalText.text = GameManager.instancia.pontuacao.ToString() + " moedas coletadas
        } else {
            modeasTotalText.text = "Pontos: " + GameManager.instancia.pontuacao.ToString ();
            tempoRestanteText.text = "Tempo restante: " + ((int)(GameManager.instancia.tempo_restant
        }
    }
}
```

Listagem 4 - Código do AtualizaUI.cs

O script AtualizaUI precisa que sejam atribuídos os objetos Text para as suas variáveis moedasText, tempoRestanteText e moedasTotalText. Faça isso arrastando esses objetos para os seus respectivos locais no Inspector do UI, como exibe a **Figura 20**.

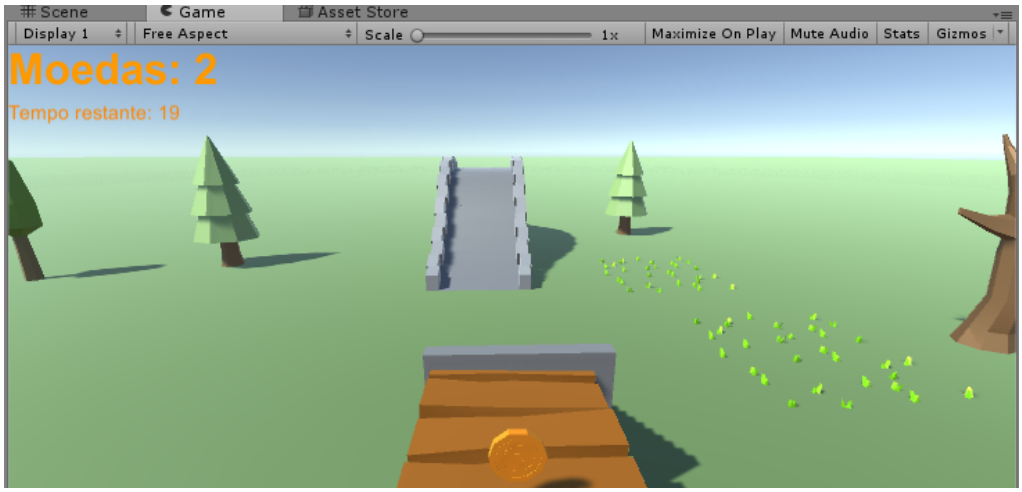
Figura 20 - AtualizaUI com os objetos atribuídos às variáveis.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Salve e execute o jogo. Se tudo estiver adequado, você verá a quantidade de moedas aumentar quando coletá-las, assim como o tempo restante, como mostra a **Figura 21**. Ao final do tempo, você verá o total de moedas no centro da tela.

Figura 21 - Jogo em execução com a UI mostrando as moedas coletadas.



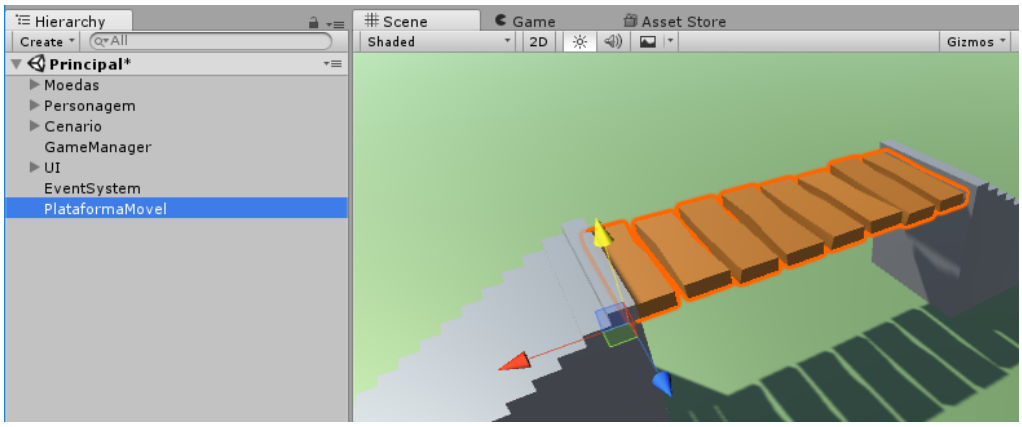
Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Obstáculos Móveis

Até o momento, adicionamos no jogo Farm Coins recursos que já tínhamos aprendido anteriormente, fazendo somente pequenas adaptações. Agora, criaremos um novo recurso, relativo ao de obstáculos móveis. Para isso, criaremos um prefab chamado “PlataformaMovel”, o qual terá um modelo 3D da plataforma e um script que a movimentará.

Utilize as técnicas aprendidas, a fim de escolher um modelo 3D para uma plataforma e arraste-o para a cena. Logo em seguida, verifique se não existe um GameObject vazio como pai desse modelo 3D e, se houver, mantenha no Hierarchy somente o seu filho (que contém o mesh rendere, mesh filter e collider, assim como fizemos com a moeda). Renomeie, em seguida, esse objeto para “PlataformaMovel”. Você terá algo como mostrado na **Figura 22**, na qual a plataforma que será móvel é a ponte entre as duas escadas.

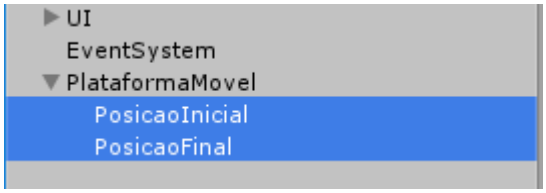
Figura 22 - Plataforma que será movida.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Nossa plataforma deverá se mover alternadamente, ou seja, de um ponto A até um ponto B, depois em direção ao ponto A novamente e repetindo sempre todo o processo. Precisamos, então, definir esses pontos que serão chamados de PosicaoInicial e PosicalFinal. Adicione dois GameObjects vazios como filhos de PlataformaMovel e seus nomes devem ser exatamente "PosicaoInicial" e "PosicaoFinal", como mostra a **Figura 23**.

Figura 23 - GameObjects de posição inicial e final adicionados como filhos de PlataformaMovel.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Atenção!

É muito importante que os dois GameObjects vazios adicionados tenham exatamente os nomes "PosicaoInicial" e "PosicaoFinal", pois vamos obter uma referência aos dois no script que criaremos utilizando seus nomes. Portanto, se desejar utilizar outros nomes, você precisará alterar também o script que criaremos mais adiante.

Como já temos os dois GameObjects “PosicaoInicial” e “PosicaoFinal” criados, selecione cada um deles, mova-os e deixe-os um pouco distantes da PlataformaMovel, o quanto você deseja que ela se mova. Esses são os dois pontos de controle que definirão os limites do movimento da PlataformaMovel. Você pode ajustar depois essas posições, pois não há a necessidade de elas ficarem perfeitas nesse momento.

Selecione o PlataformaMovel e adicione um script chamado MoverPlataforma com o seguinte código:

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

public class MoverPlataforma : MonoBehaviour {
    Vector3 posicaoInicial;
    Vector3 posicaoFinal;
    public float velocidade = 3f;
    Vector3 destinoAtual;
    bool voltando;

    void Start() {
        posicaoInicial = transform.FindChild ("PosicaoInicial").position;
        posicaoFinal = transform.FindChild ("PosicaoFinal").position;
        destinoAtual = posicaoFinal;
        voltando = false;
    }

    void Update() {
        // Se move um pouco em direção ao destino atual
        float movimento = velocidade * Time.deltaTime;
        transform.position = Vector3.MoveTowards (transform.position, destinoAtual, movimento);

        float distanciaAoDestino = Vector3.Distance (transform.position, destinoAtual);
        if (distanciaAoDestino <= 0.1f) {
            // alternar o destino
            if (voltando) {
                voltando = false;
                destinoAtual = posicaoFinal;
            } else {
                voltando = true;
                destinoAtual = posicaoInicial;
            }
        }
    }
}
```

Listagem 5 - Código do script MoverPlataforma.cs

Execute o jogo e se tudo estiver correto você deverá ver a PlataformaMovel se mover do PosicaoInicial ao PosicaoFinal de forma alternada em uma determinada velocidade.

Vamos agora revisar o script que criamos. MoverPlataforma tem cinco variáveis, são elas:

- **Vector3 posicaoInicial:** armazenará a posição do GameObject "PosicaoInicial".
- **Vector3 posicaoFinal:** armazenará a posição do GameObject "PosicalFinal".
- **Public float velocidade = 3f:** determina a velocidade da plataforma (por padrão 3.0).
- **Vector3 destinoAtual:** armazena o destino atual do movimento, ou seja, para onde a plataforma está se movendo naquele momento (deve alternar entre posicaoInicial e posicaoFinal). Todas as vezes que ela mudar de direção mudamos o valor dessa variável.
- **Bool voltando:** um booleano que guarda a informação sobre se a plataforma está indo para a posição final (voltando = false) ou para a posição inicial (voltando = true). Todas as vezes que ela mudar de direção, mudamos o valor dessa variável.

Observe o método Start() abaixo:

```
1  void Start() {
2        posicaoInicial = transform.FindChild ("PosicaoInicial").position;
3        posicaoFinal = transform.FindChild ("PosicaoFinal").position;
4        destinoAtual = posicaoFinal;
5        voltando = false;
6     }
```

Repare que nele definimos o valor de quatro variáveis. Primeiramente, queremos guardar as posições dos GameObjects **PosicalInicial** e **PosicaoFinal** nas variáveis **posicaoInicial** e **posicaoFinal** respectivamente (que são do tipo Vector3). Como não criamos uma referência aos GameObjects filhos no script, utilizaremos uma técnica de pegar essa referência pelo seu nome. Para isso, usaremos o transform.FindChild(), o qual recebe o nome do GameObject filho que está sendo

buscado como parâmetro. O retorno do FindChild() é o transform do GameObject para o qual passamos o nome, e dele queremos recuperar a posição, por isso utilizamos o seguintes comandos para os dois GameObjects filhos do PlataformaMovel:

```
1 posicaoInicial = transform.FindChild ("PosicaoInicial").position;
2 posicaoFinal = transform.FindChild ("PosicaoFinal").position;
```

Ainda no método start, utilizamos o comando **destinoAtual = posicalFinal;** para determinar que a princípio a Plataforma deve se mover em direção à posicaoFinal. Da mesma forma, determinamos que inicialmente **voltando = false**, ou seja, como a Plataforma inicia indo para a posicaoFinal, então ela não está "voltando" ainda.

No começo do método Update() temos os comandos:

```
1 float movimento = velocidade * Time.deltaTime;
2 transform.position = Vector3.MoveTowards (transform.position, destinoAtual, movimento);
```

Primeiramente, dizemos que a variável **"movimento"** corresponderá à velocidade que escolhemos multiplicada pelo tempo passado desde o último frame exibido na tela, sendo esse tempo sempre armazenado em Time.deltaTime. Essa é uma prática muito comum para ter uma física, independentemente da taxa de atualização dos frames do jogo. A variável **"movimento"** é do tipo float e guarda o quanto a plataforma deve se mover nesse frame atual (mas não ainda para onde ela se move).

Em seguida, com o comando **transform.position = Vector3.MoveTowards (transform.position, destinoAtual, movimento);** determinamos que a nova posição de PlataformaMovel (transform.position) será alterada e receberá um novo valor correspondente justamente à posição do ponto que está entre a plataforma e o destino, porém somente a uma distância da plataforma do tamanho da variável **"movimento"**. O Vector3.MoveTowards faz exatamente isto: recebe duas posições (a da plataforma e a do destino), além de uma certa quantidade de unidades a se mover (variável movimento), e retorna um novo Vector3 que representa esse passo em direção ao destino. Isso faz a plataforma efetivamente "andar" em direção ao destino um pouco a cada frame.

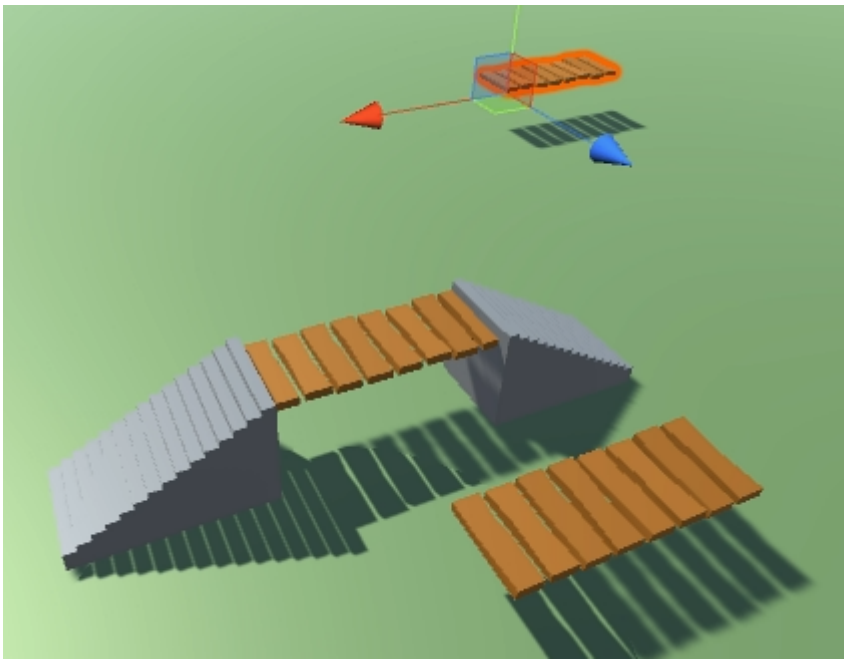
Ainda no método Update() temos o restante dos comandos como a seguir:

```
float distanciaAoDestino = Vector3.Distance (transform.position, destinoAtual);
   if (distanciaAoDestino <= 0.1f) {
      // alternar o destino
      if (voltando) {
         voltando = false;
         destinoAtual = posicaoFinal;
      } else {
         voltando = true;
         destinoAtual = posicaoInicial;
      }
   }
```

O trecho de código acima inicialmente calcula a distância da Plataforma até o destinoAtual utilizando o Vector3.Distance(), o qual recebe dois Vector3 como parâmetro e retorna a distância entre eles. Logo em seguida, é feita uma verificação sobre se distanciaAoDestino é menor que 0.1, ou seja, se o objeto chegou ao destino e é o momento de alternar a direção do movimento. É importante fazer essa verificação considerando uma certa margem de erro (+- 0.1), pois às vezes o objeto pode "passar direto" do seu destino, sem nunca ter ficado exatamente sobre ele, já que a física do jogo é calculada em intervalos frame a frame. Caso seja verdade que o objeto chegou ao destino, então se verifica se ele estava indo em direção ao Início ou ao Final com o comando **if (voltando)**. Em cada caso se realiza a troca das variáveis **voltando** e **destinoAtual**: se ele está voltando, então a variável **voltando** agora é igual a **false** e o **destinoAtual = posicaoFinal**, caso contrário então a variável **voltando** agora é igual a **true** e o **destinoAtual = posicaoInicial**.

O script MoverPlataforma é extremamente genérico, pois permite que você realize a mudança dos pontos de controle (PosicaoInicial e PosicaoFinal) no SceneView e também da velocidade no Inspector. Assim você pode criar um Prefab do PlataformaMovel (movendo-o para uma pasta) e reutilizar em qualquer lugar desejado. Cada uma dessas instâncias pode ter um movimento distinto e com velocidades diferentes. A **Figura 24** mostra três instâncias da PlataformaMovel criadas no mapa.

Figura 24 - Instâncias da PlataformaMovel criadas no mapa.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Veja agora um vídeo que mostra em detalhes o processo de criação da plataforma móvel, desde a escolha do modelo 3D até a criação do script de movimento.

Vídeo 03 - Plataforma Pontuação

Você pode aplicar as técnicas de movimento aprendidas para criar qualquer outro tipo de objeto que se move na cena utilizando pontos de controle. Pode, também, por exemplo, criar prefabs que se comportam como os tradicionais espinhos dos jogos de plataforma, com os quais se o seu jogador entrar em contato ele perde vida, moedas ou até mesmo é obrigado a reiniciar a fase. Você já aprendeu todos os elementos para criar esses novos recursos e muitos outros. Sua imaginação é o limite!

Saindo do Jogo

Antes de prosseguirmos, criaremos uma forma de você sair do jogo. Abra o script GameManager e altere o método Update para inserir o comando Application.Quit(); assim que acabar o tempo. Isso fará com que o jogo seja fechado. O código final do método Update do script GameManager é este:

```
1  void Update() {
2      tempo_restante -= Time.deltaTime;
3      if (tempo_restante <= 0f) {
4          tempo_restante = 0f;
5          // finalizar o jogo
6          finalizado = true;
7          Application.Quit (); // Fecha o jogo
8      }
9    }
```

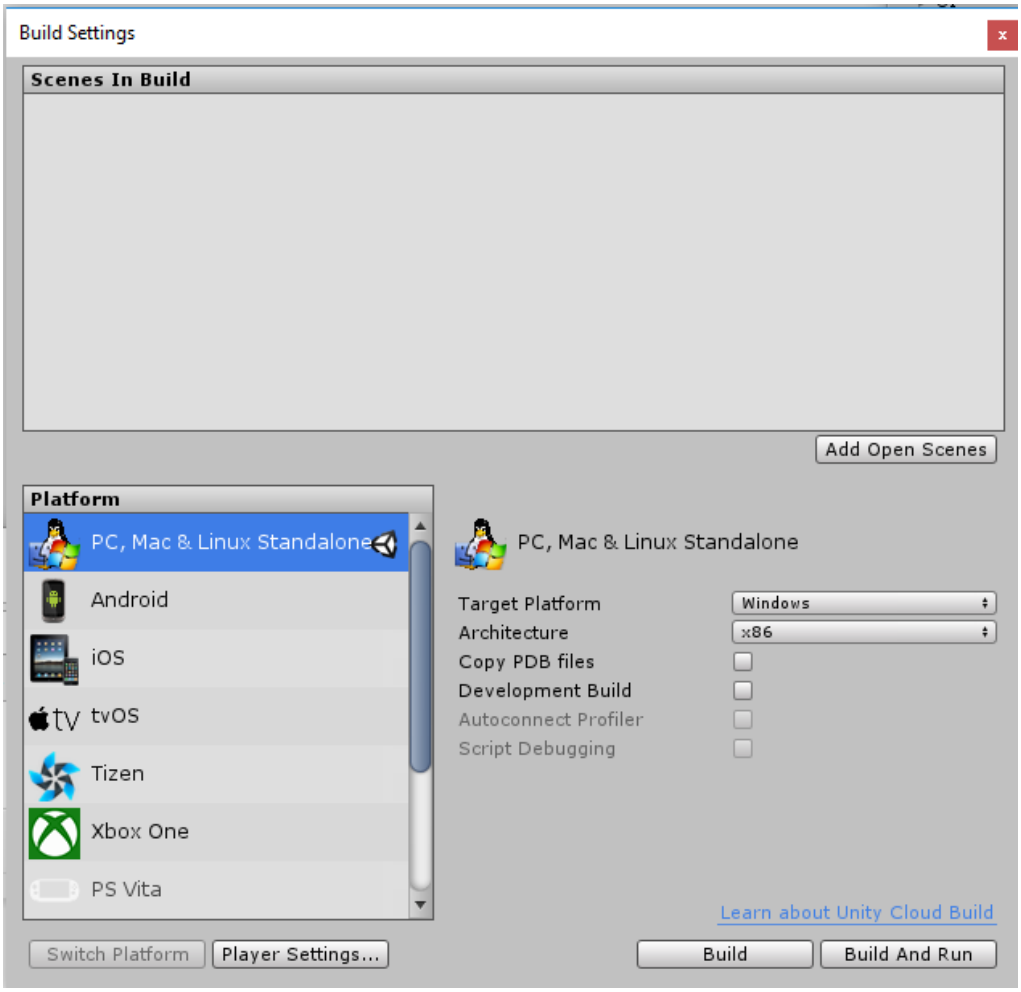
Você pode melhorar esse comportamento de saída do jogo. Assim que o jogo finalizar, você pode exibir, através de elementos de UI, botões por meio dos quais o jogador possa escolher entre “Sair” ou “Reiniciar” a partida, por exemplo. Outra alternativa é carregar uma cena especial contendo somente o menu do jogo com a opção de “Sair” também. Isso fica ao seu critério!

Exportando seu Jogo para a Plataforma Desejada

Até agora você só testou o seu jogo no editor do Unity, entretanto é muito fácil exportá-lo para as diversas plataformas que o Unity suporta.

Com o jogo aberto, escolha a opção no menu File -> Build Settings..., de modo a exibir uma janela como a da **Figura 25**.

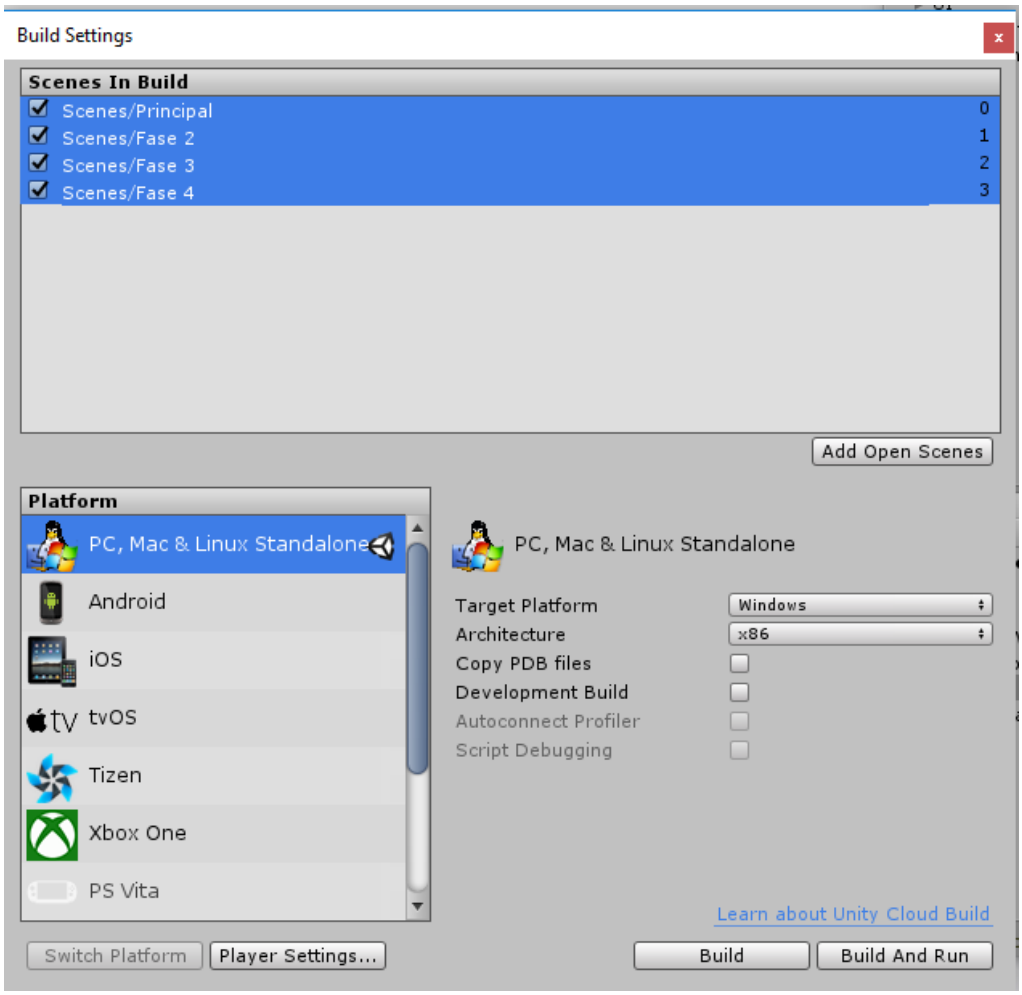
Figura 25 - Build Settings.



Fonte: Captura de tela do Build Settings.

Na parte superior da janela, você tem uma lista das cenas que devem ser inseridas no seu jogo. Por padrão, essa lista vem vazia e você deve inserir as cenas desejadas simplesmente arrastando-as da janela Project para a janela Build Settings que está aberta, ficando com algo semelhante ao mostrado na **Figura 26**.

Figura 26 - Cenas inseridas no Builds Settings.



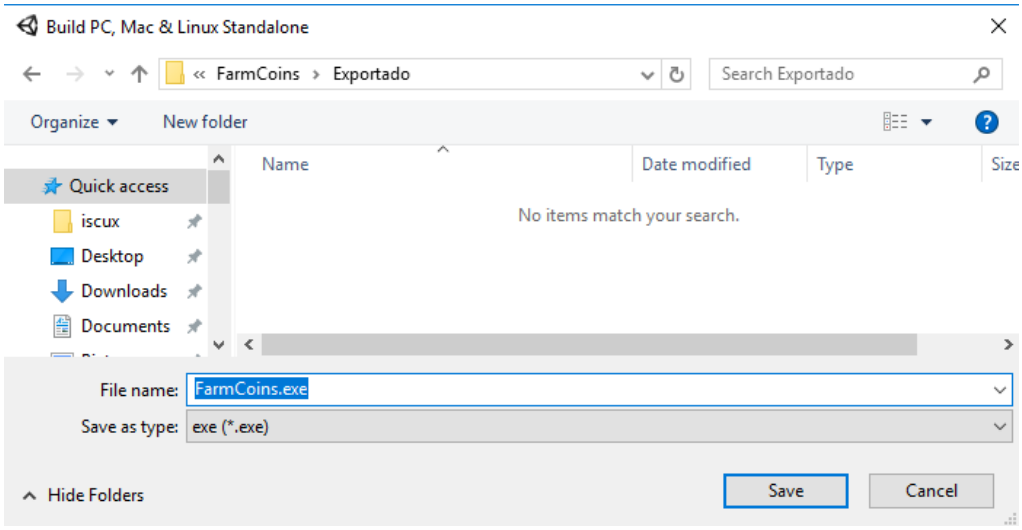
Fonte: Captura de tela do Build Settings.

Você pode reordenar a lista de cenas arrastando-as para cima ou para baixo, porém deixando a primeira sempre como a cena inicial do seu jogo, ou seja, como a que será aberta assim que o jogo iniciar.

Em seguida, ainda na janela Build Settings, escolha uma das plataformas listadas (por padrão é PC, Mac & Linux Standalone). Isso fará o Unity compilar o seu jogo para a plataforma desejada. Entretanto, cada uma das plataformas, fora o PC, exige que você instale outras ferramentas e realize outras configurações no seu computador para que o Unity consiga gerar o jogo nessa plataforma. Veja o site https://docs.unity3d.com/Manual/PlatformSpecific.html para mais informações acerca de como configurar cada uma delas.

Escolha a plataforma PC e clique no botão “Build”, de maneira a abrir uma janela perguntando onde você deseja salvar o jogo. Escolha uma pasta qualquer do seu computador e nomeei-a “FarmCoins.exe”, como na **Figura 27**.

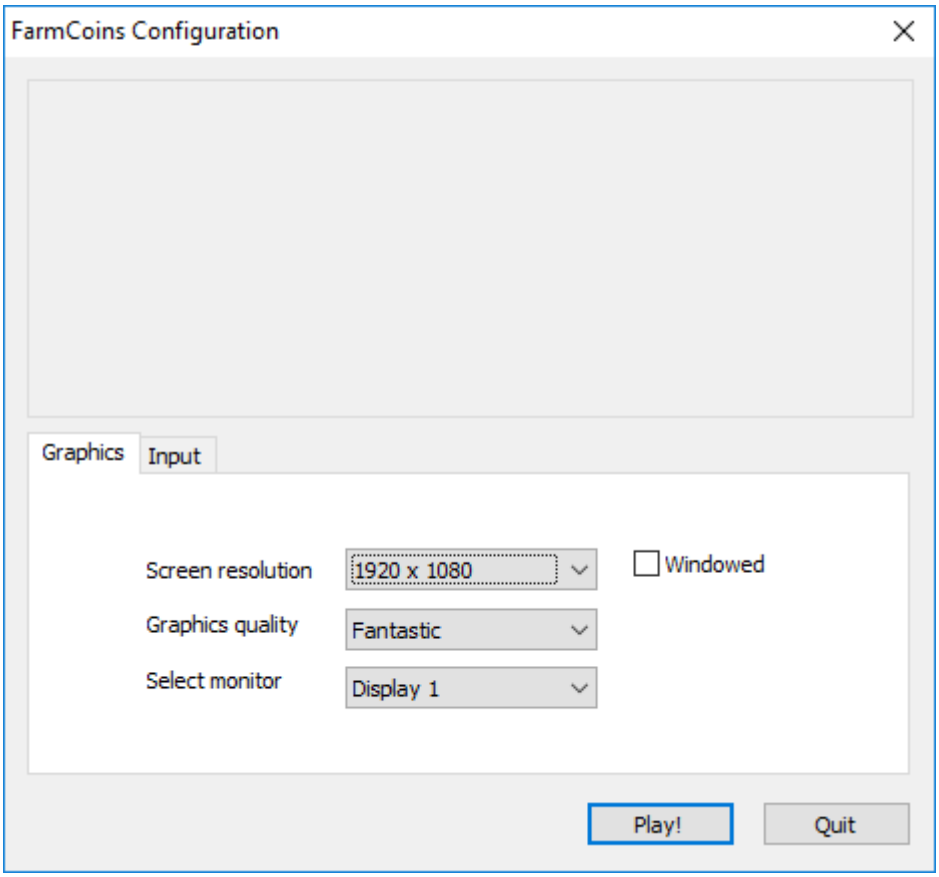
Figura 27 - Exportação do jogo para PC.



Fonte: Captura de tela do Windows

Localize no gerenciador de arquivos o arquivo FarmCoins.exe que você criou e dê um duplo clique para executá-lo. Você verá inicialmente uma tela para escolher a qualidade do jogo, como podemos observar na **Figura 28**.

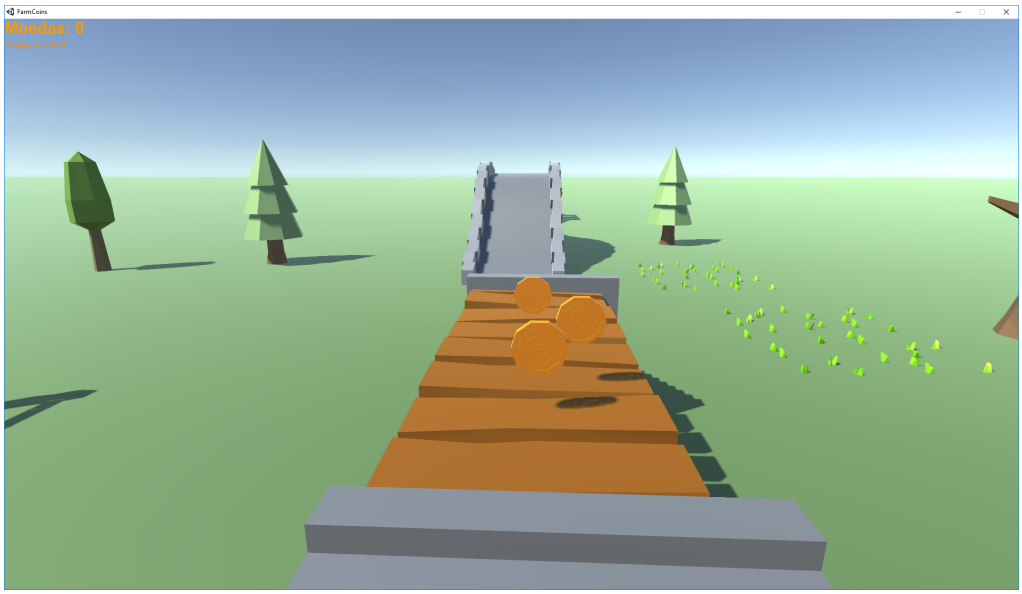
Figura 28 - Tela de configuração de qualidade do jogo.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Escolha a resolução, a qualidade gráfica, determine se o jogo vai rodar em janela (Windowed) ou não e clique em Play!, fazendo com que seu jogo inicie. Veja na **Figura 29** o jogo Farm Coins executando completamente independente do Unity.

Figura 29 - Farm Coins executando em modo “standalone” independente do Unity.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 22 de maio de 2017

Você pode agora distribuir o executável do seu jogo com a pasta FarmCoins_Data, criada junta ao arquivo FarmCoins.exe.

Parabéns por chegar até aqui e finalizar a disciplina! Tomara que tenha aprendido bastante sobre o Unity 3D e suas ferramentas. Evidentemente, ainda há muito o que aprender sobre essa ferramenta, não deixe de procurar sempre mais informações. Até mais!

Resumo

Nesta nossa última aula, incrementamos o jogo Farm Coins, utilizando as técnicas aprendidas durante a disciplina para criarmos o HUD, a fim de realizarmos a contagem de pontos. Também aprendemos a criar elementos mais dinâmicos, utilizando como exemplo uma plataforma que se move por meio de pontos de controle. Além disso, vimos como exportar o seu jogo para que ele possa ser executado sem a necessidade do Unity estar instalado na máquina (modo standalone).

Leitura Complementar

Seus estudos no Unity 3D não podem parar por aqui, pois ele é uma ferramenta muito vasta. Uma boa fonte de aprendizado são os tutoriais oficiais do Unity que estão disponíveis no site unity3d.com. Segue abaixo alguns deles:

- https://unity3d.com/learn/tutorials/projects/space-shooter-tutorial
- https://unity3d.com/learn/tutorials/projects/survival-shooter-tutorial
- https://unity3d.com/learn/tutorials/projects/tanks-tutorial

Você pode criar vários outros desafios no seu jogo. Tente incrementá-lo com novos elementos dinâmicos e desafiantes utilizando todas as técnicas que aprendeu durante esta disciplina e não deixe de estudar o Unity 3D no seu site oficial, ou em qualquer outra fonte, como livros, Youtube, e a Internet em geral.

Autoavaliação

1. Adicione Moedas em cima de plataformas móveis e observe que elas não se movem com a plataforma.

2. Mova no Hierarchy essas moedas adicionadas para serem filhas da plataforma que elas estão acima e veja que agora elas se movem com a plataforma.

Referências

UNITY TECHNOLOGIES. 2016 (C). Unity 3D Online Tutorials [online]. Disponível em: https://unity3d.com/pt/learn/tutorials [Acessado em 16 de novembro de 2016].

UNITY TECHNOLOGIES. 2016 (C). Unity Manual - Prefabs [online]. Disponível em: https://docs.unity3d.com/Manual/Prefabs.html [Acessado em 16 de novembro de 2016].

STOY, C. 2006. Game object component system. In Game Programming Gems 6, Charles River Media, M. Dickheiser, Ed., Páginas 393 a 403.

MARQUES, Paulo; PEDROSO, Hernâni - C# 2.0 . Lisboa: FCA, 2005. ISBN 978-972-722 208-8

UNITY TECHNOLOGIES. 2016 (C). Unity 3D Manual [online]. Disponível em: https://docs.unity3d.com/Manual/index.html [Acessado em 16 de novembro de 2016].