Curso de Pós-Graduação em Cloud Computing e Mobile
Disciplina DM 117 – Introdução a Desenvolvimento de Jogos com Unity
Professor: Phyllipe Lima

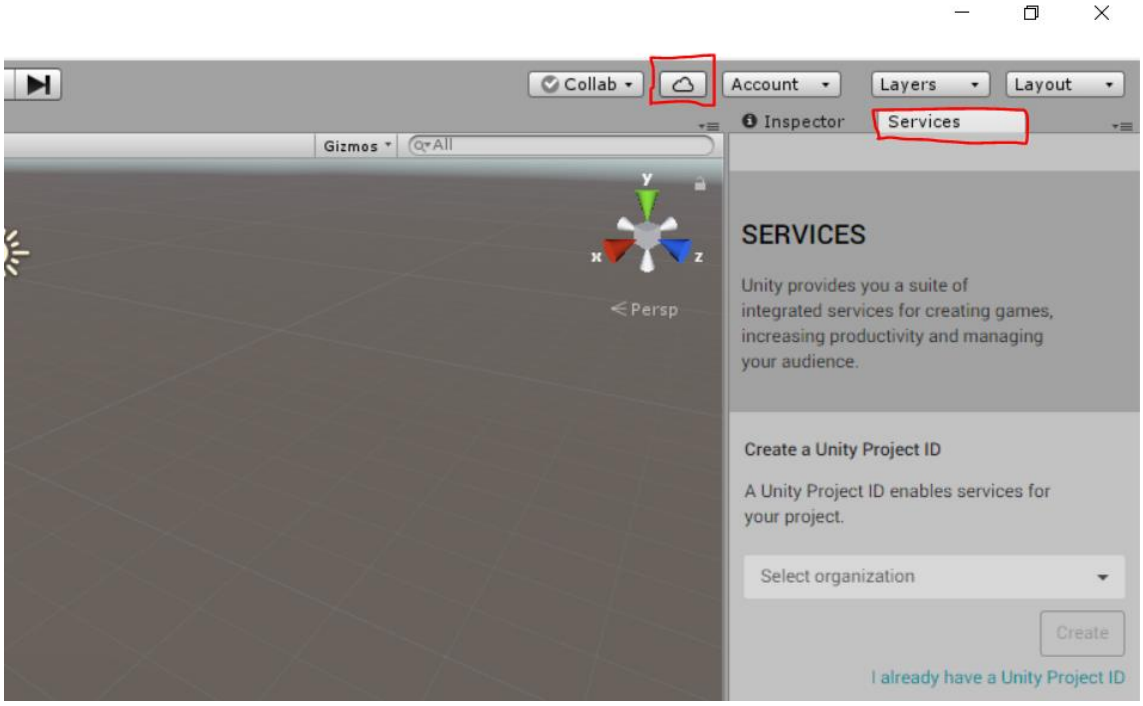
Aula 4

Ao final desse relatório você terá:

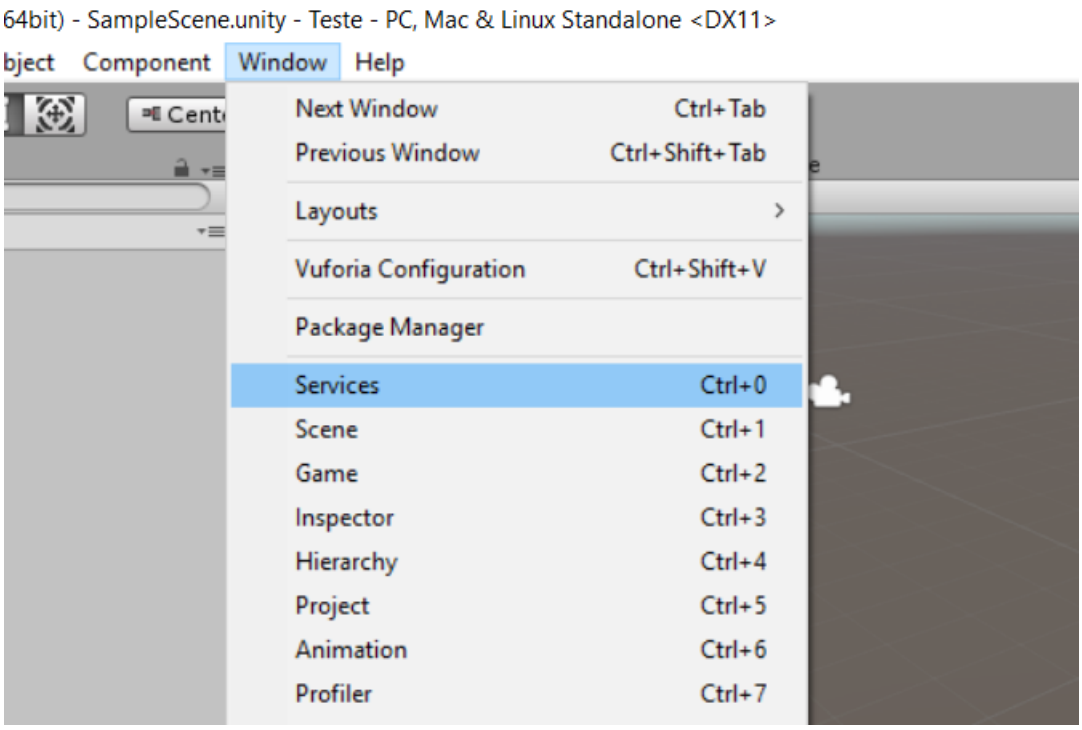
- Adicionar ads (anúncios) ao seu jogo

1 – Configurações

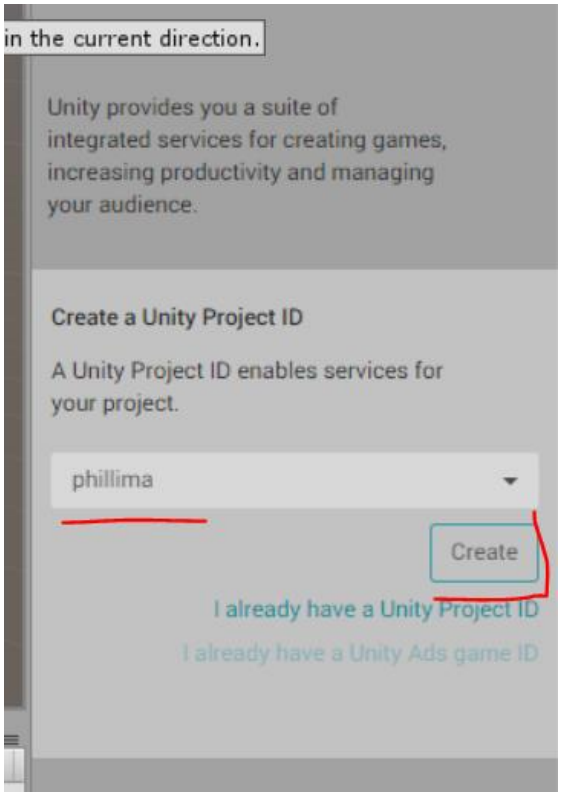
O primeiro passo é habilitar o **Unity Services** no seu projeto. Existem duas formas. A mais simples é clicando na nuvem que aparece na barra de ferramentas.



Ou você pode clicar em Windows -> Services



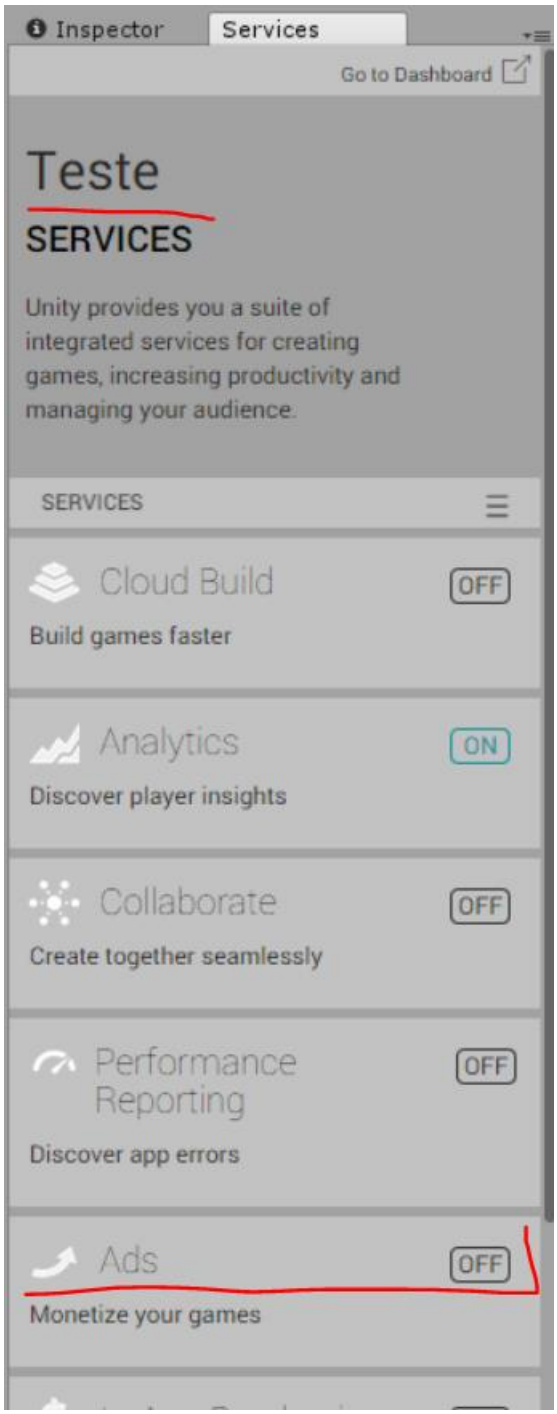
Se for a primeira utilizando o Unity Services, talvez você precise cadastrar uma organização e um projeto. Observe como está a sua aba **Services**.



O nome do projeto é criado automaticamente baseado no nome que você usou. E a organização o Unity cria uma baseada no seu *username*. Você pode personalizar a sua organização pelo link https://id.unity.com/organizations

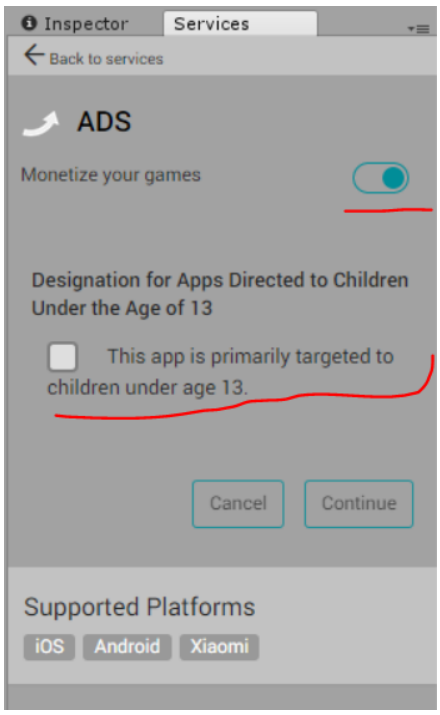
Escolha uma organização (*phillima* no meu caso) e clique em *create*, como mostrado na Figura acima.

Agora o seu jogo á possui um ***ID.*** Agora ligue a opção ***ADS*** clicando no botão de *toogle*.

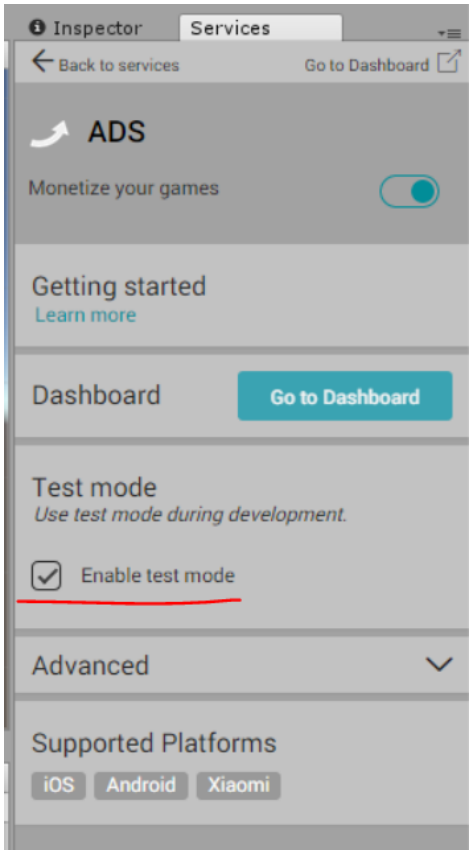


Até este ponto utilizei imagens de um projeto genérico, Teste, apenas para ilustrar as configurações sobre Ads. As próximas imagens são feitas novamente com o projeto apresentado em sala de aula.

Agora no topo da aba ***Services*** você pode de fato ligar o ads e deverá responder algumas perguntas sobre o seu jogo.



Neste caso deixei a opção em branco e cliquei em ***continue*****.** Agora os ***ads*** estão habilitados. Dê sequência habilitando o modo de teste ***Enable Teste Mode***.



Com isso a configuração está pronta. Podemos adicionar um anúncio ao nosso jogo.

2 – Adicionando um anúncio simples.

O *framework* ***Unity Ads*** possui dois tipos de *ads* – Simples ou Recompensa. O ***simples*** permite ao desenvolvedor apresentar um anúncio de tela inteira. Bastante útil quando o jogador reiniciar ou trocar de fase.

Comece criando um ***script*** que será responsável por gerenciar todos os ***ads*** deste jogo. Chame de **UnityAdControle.**

```
5   #if UNITY_ADS
6   using UnityEngine.Advertisements;
7   #endif
8
9   public class UnityAdControle : MonoBehaviour {
10      public static bool showAds = true;
11
12      public static void ShowAd() {
13
14      #if UNITY_ADS
15              if (Advertisement.IsReady()) {
16                  Advertisement.Show();
17              }
18      #endif
19          }
20
```

Primeiro observe as diretivas sobre ADS. A ideia é apenas compilar código para plataformas que suportam ads, como Android e iOS. Caso o código esteja cinza para você é por que você está fazendo a ***build*** para plataformas que não suportam ***ads***. Mude a ***build*** para Android, por exemplo.

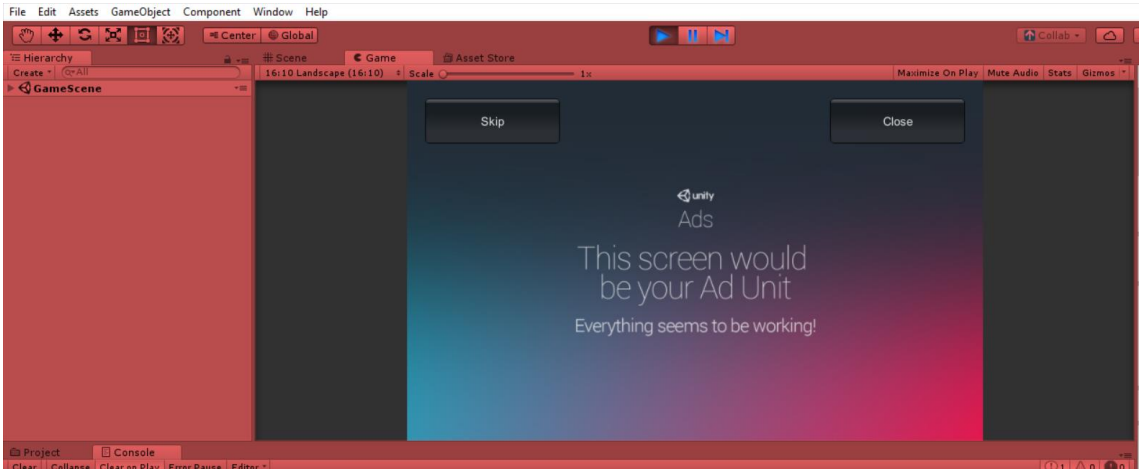
Na linha 12 criamos um método responsável por mostrar o ***ad.*** Se ele estiver pronto, invocamos o método ***Show()*** na linha 16. Na linha 10 temos um atributo estático para controlar se devemos ou não apresentar o ***ad***.

Vá no ***script*** MenuPrincipal e faça as alterações abaixo:

```
8       public void CarregaScene(string nomeScene) {
9           SceneManager.LoadScene(nomeScene);
10
11      #if UNITY_ADS
12
13          if (UnityAdControle.showAds) {
14              //Mostra um anuncio
15              UnityAdControle.ShowAd();
16          }
17
18      #endif
19      }
```

O que fizemos foi verificar se o ***ad*** está pronto. Se estiver invocamos antes que a tela do jogo seja carregada. Agora vamos testar

De ***play*** no jogo, depois **pause** e vá para o Menu Principal. Pressione o botão ***Jogar***. Um anúncio de teste deverá aparecer.



3 – Usando call-backs

Você deve ter percebido que quando o ad aparece, o jogo não pausa. Se estiver em um ambiente mobile real, o próprio Unity irá pausar o jogo. No nosso caso, que estamos no ***Unity Editor*** podemos fazer uma adaptação para que o jogo pause. Faça as modificações na classe ***UnityAdControle*** como mostra a Figura abaixo.

Na linha 26 e 27 criamos um ***ShowOptions***, que permite configurar opções de ads. E logo em seguida atribuímos o método ***Unpause*** para o ***resultCallback.***

Na linha 35 e 36 acessamos o ***MenuPauseComp*** para pausarmos o jogo. E na linha 42 temos o método ***Unpause***. Observe que na linha 31 passamos o ***opções*** como parâmetro.

```
21      public static void ShowAd() {
22
23  #if UNITY_ADS
24
25          //Opcoes para o ad
26          ShowOptions opcoes = new ShowOptions();
27          opcoes.resultCallback = Unpause;
28
29          if (Advertisement.IsReady()) {
30              //Mostra o anuncio
31              Advertisement.Show(opcoes);
32          }
33          //Pausar o jogo enquanto
34          //o ad esta sendo mostrad
35          MenuPauseComp.pausado = true;
36          Time.timeScale = 0;
37  #endif
38      }
39
40  #if UNITY_ADS
41      public static void Unpause(ShowResult result) {
42          //Quando o anuncio acabar
43          //sai do modo pausado
44          MenuPauseComp.pausado = false;
45          Time.timeScale = 1f;
46      }
47  #endif
```

Para garantir que o ***MenuPauseComp*** não sobrescreva esse comportamento, faça uma alteração no método ***Start()***.

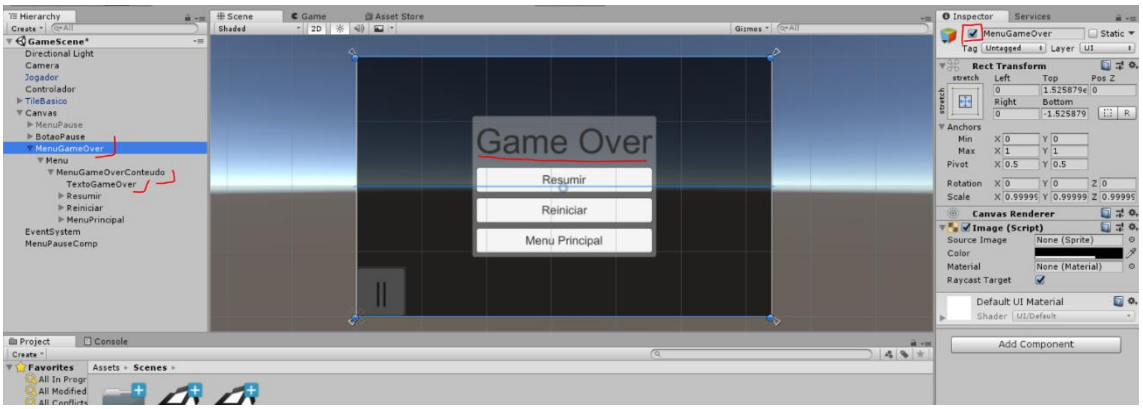
```
40      private void Start() {
41          //pausado = false;
42  #if !UNITY_ADS
43          SetPauseMenu(false);
44  #endif
45
46      }
```

Neste exemplo, se estivermos compilando para uma plataforma que não usa suporta Ads (PC, por exemplo) então não precisamos nos preocupar com Ads e por padrão o jogo irá começar com o ***MenuPause*** desligado.

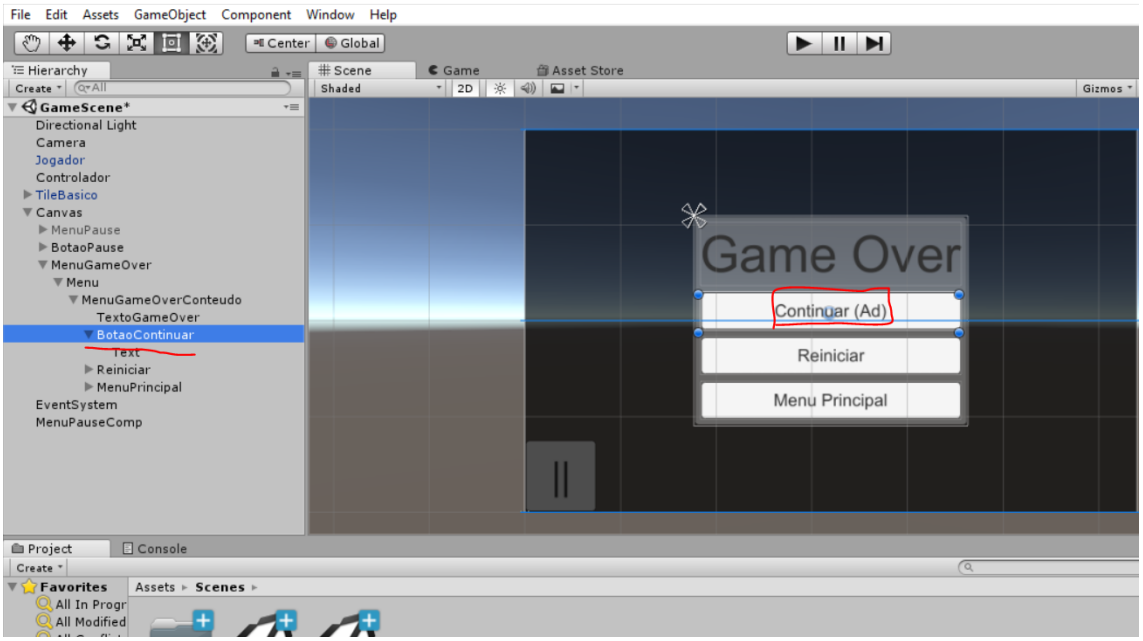
4 – Usando ads do tipo Recompensa

A ideia destes ***ads*** é oferecer ao jogador alguma recompensa se ele assistir o anúncio.

Vamos iniciar criando um menu ***GameOver***. Comece duplicando o ***MenuPause*** e renomeie para ***MenuGameOver***. Deixe este ***panel*** visível (clique no botão na aba ***Inspector*** ao lado do nome do ***GameObject***). Mude o texto para ***Game Over*.** Altere o nome dos componentes conforme figura abaixo.



No botão ***Resumir***, mude o texto para ***Continuar (Ad)***. Mude o nome do componente conforme figura abaixo.



Vamos agora até o ***script*** que controla o comportamento do obstáculo, ***ObsComp.*** Faremos algumas modificações para suportar essa nova funcionalidade.

O primeiro passo é modificar o método que trata a colisão.

```
25      private void OnCollisionEnter(Collision collision) {
26          if (collision.gameObject.GetComponent<JogadorComp>()) {
27              //Vamos esconder o jogador
28              //ao inves de destruir
29              collision.gameObject.SetActive(false);
30
31              jogador = collision.gameObject;
32
33              //Antes de implementar o sistema de recompensa
34              //Destruiamos o jogador diretamente
35              //Destroy(collision.gameObject);
36
37              //Chame a funcao Reset depois de um tempo
38              Invoke("Reset", tempoDestruir);
39          }
40      }
```

Na linha 29 nós escondemos o jogador, e não destruímos. Na linha 31 atribuímos o objeto de colisão, que no caso é o jogador a nossa variável jogador. É necessário declarar essa variável. Veja a Figura abaixo

```
7   /// <summary>
8   /// Classe para definir o comportamento do obstaculo
9   /// </summary>
10  public class ObsComp : MonoBehaviour {
11
12      /// <summary>
13      /// Variavel referencia para o jogador
14      /// </summary>
15      private GameObject jogador;
16
17      [SerializeField]
18      [Tooltip("Tempo para reiniciar o jogo")]
19      private float tempoDestruir = 2.0f;
20
```

Continuando na classe ***ObsComp***, veja como está agora o método ***Reset()***.

Nas linhas 45 e 46 buscamos o ***MenuGameOver*** e o deixamos visível.
Nas linha 49 buscamos todos os botões do ***MenuGameOver*** e na linha 50 criamos uma variável temporária que irá ser a referência para o BotaoContinuar.

Na linha 53 fazemos um ***foreach*** procurando especificamente o ***BotãoContinuar***. Assim que achar, o salvamos na variável ***botaoContinue***.

Na linha 60 verificamos o ***botaoContinue*** realmente foi encontrado. Na linha 63, associamos o evento de clicar no **BotaoContinuar** ao método ***ShowAdReward*** que se encontra na classe ***UnityAdControle***. Na linha 60 nós atribuímos o obstáculo (this)

dentro da classe ***UnityAdControle***. Essa informação é importante pois caso o jogador assista ao Ad, ele será recompensando podendo continuar o jogo. Logo, nós devemos destruir o obstáculo que causou a colisão.

Observe nas linhas 65 – 68 que caso a plataforma não suporte Ads, não mostramos o **BotãoContinuar**.

```
42      private void Reset() {
43
44          //Faz o MenuGameOver aparecer
45          var gameOverMenu = GetGameOverMenu();
46          gameOverMenu.SetActive(true);
47
48          //Busca os botoes do MenuGameOver
49          var botoes = gameOverMenu.transform.GetComponentsInChildren<Button>();
50          Button botaoContinue = null;
51
52          //Varre todos os botoes, em busca do botao continue.
53          foreach (var botao in botoes) {
54              if (botao.gameObject.name.Equals("BotaoContinuar")) {
55                  botaoContinue = botao;//Salva uma referencia para o botao continue
56                  break;
57              }
58          }
59          //Verifica se o botaoContinue eh diferente de null
60          if (botaoContinue) {
61  #if UNITY_ADS
62              //Se o botao continue for clicado, invoca o metodo ShowRewardAd
63              botaoContinue.onClick.AddListener(UnityAdControle.ShowRewardAd);
64              UnityAdControle.obstaculo = this;
65  #else
66              //Se nao existe add, nao precisa mostrar o botao Continue
67              botaoContinue.gameObject.SetActive(false);
68  #endif
69          }
70          //Nao eh necessario mais recarregar o jogo por aqui.
71          //SceneManager.LoadScene(SceneManager.GetActiveScene().name);
```

Ainda na classe ***ObsComp***. Adicionamos mais dois métodos. Na linha 77 temos o método ***Continue***. Ele é responsável por desligar o ***MenuGameOver***, religar o jogador e chamar a explosão para esse obstáculo (que causou a interrupção do jogo). E na linha 89 temos o método que busca na ***Scene*** pelo ***MenuGameOver***.

```
74      /// <summary>
75      /// Faz o reset do jogo
76      /// </summary>
77      public void Continue() {
78          var go = GetGameOverMenu();
79          go.SetActive(false);
80          jogador.SetActive(true);
81          //Exploda essa obstaculo, caso o jogador resolvar apertar o Continue.
82          ObjetoTocado();
83      }
84
85      /// <summary>
86      /// Busca o MenuGameOver
87      /// </summary>
88      /// <returns>O GameObject MenuGameOver</returns>
89      GameObject GetGameOverMenu() {
90          return GameObject.Find("Canvas").transform.
91              Find("MenuGameOver").gameObject;
92      }
93
```

Agora precisamos fazer algumas alterações na classe ***UnityAdControle***.
Na linha 21 adicionamos uma referência para o obstáculo. Isso é importante pois precisamos da referência para o obstáculo que causou a interrupção.

```
9   /// <summary>
10  /// Classe que controla ads
11  /// </summary>
12  public class UnityAdControle : MonoBehaviour {
13      /// <summary>
14      /// Variavel de controle se devemos ou nao mostrar ads
15      /// </summary>
16      public static bool showAds = true;
17
18
19      // Referencia para o obstaculo.
20      public static ObsComp obstaculo;
21
```

Agora adicione o método ***ShowRewardAd***. Este método que mostra o ad baseado na recompensa. A única diferença para o ad Simples é que passamos um **callback** diferente na linha 55. Este ***callback*** é criado na linha 75. Caso o jogador assista ao anúncio, na linha 80 chamamos o método ***Continue*** da classe ***ObsComp*** e damos sequencia ao jogo normalmente.

```
43      /// <summary>
44      /// Metodo para mostrar ad com recompensa
45      /// </summary>
46      public static void ShowRewardAd() {
47  #if UNITY_ADS
48          if (Advertisement.IsReady()) {
49              // Pausar o jogo
50              MenuPauseComp.pausado = true;
51              Time.timeScale = 0f;
52              //Outra forma de criar a
53              //instancia do ShowOptions e setar o callback
54              var opcoes = new ShowOptions {
55                  resultCallback = TratarMostrarResultado
56              };
57              Advertisement.Show(opcoes);
58          }
59  #endif
60      }
61
```

```
71      /// <summary>
72      /// Metodo para tratar o resultado com reward/recompensa
73      /// </summary>
74      /// <param name="result"></param>
75  #if UNITY_ADS
76      public static void TratarMostrarResultado(ShowResult result) {
77
78          switch (result) {
79              case ShowResult.Finished:
80                  // Anuncio mostrado. Continue o jogo
81                  obstaculo.Continue();
82                  break;
83              case ShowResult.Skipped:
84                  Debug.Log("Ad pulado. Faz nada");
85                  break;
86              case ShowResult.Failed:
87                  Debug.LogError("Erro no ad. Faz nada");
88                  break;
89          }
90
91      // Saia do modo pausado
92      MenuPauseComp.pausado = false;
93          Time.timeScale = 1f;
94      }
95  #endif
96
```

5 – Usando um sistema de *cooldown*

De acordo com os ***Unity Monetization*** cada usuário só pode ver até 25 ads por dia. Então pode ser interessante tomar algumas medidas para que o jogador não veja ads o tempo todo. Um sistema de ***cooldown*** pode ser usado.

A primeira coisa é criarmos uma variável do tipo ***DateTime*** para controlar esse tempo. Vá na classe ***UnityControleAd*,** e faça as modificações.

```
10  /// <summary>
11  /// Classe que controla ads
12  /// </summary>
13  public class UnityAdControle : MonoBehaviour {
14      /// <summary>
15      /// Variavel de controle se devemos ou nao mostrar ads
16      /// </summary>
17      public static bool showAds = true;
18
19      //Tipo que pode ser null
20      public static DateTime? proxTempoReward = null;
```

Na linha 20 observe que a variável é do tipo ***nullable***, ou seja ela pode ser ***null.*** Agora no método ***ShowRewardAd***, atribua um valor para essa variável. A classe ***DateTime*** que estamos usando pertence ao próprio ***.Net***.

```
46      /// <summary>
47      /// Metodo para mostrar ad com recompensa
48      /// </summary>
49      public static void ShowRewardAd() {
50  #if UNITY_ADS
51
52          proxTempoReward = DateTime.Now.AddSeconds(15);
53          if (Advertisement.IsReady()) {
54              // Pausar o jogo
55              MenuPauseComp.pausado = true;
56              Time.timeScale = 0f;
57              //Outra forma de criar a
58              //instancia do ShowOptions e setar o callback
59              print("Teste ad show");
60              var opcoes = new ShowOptions {
61                  resultCallback = TratarMostrarResultado
62              };
63
64              Advertisement.Show(opcoes);
65          }
66  #endif
```

Agora vamos fazer a modificação na classe ***ObsComp***.

Primeiro devemos mudar a forma como chamamos o método ***Reset.*** Dentro deste método, onde verificamos se o botão ***continue*** foi encontrado faça a seguinte modificação.

```
60          //Verifica se o botaoContinue eh diferente de null
61          if (botaoContinue) {
62  #if UNITY_ADS
63              //Se o botao continue for clicado, iremos tocar o anúncio
64              StartCoroutine(ShowContinue(botaoContinue));
65  #else
66              //Se nao existe add, nao precisa mostrar o botao Continue
67              botaoContinue.gameObject.SetActive(false);
68  #endif
69          }
70          //Nao eh necessario mais recarregar o jogo por aqui.
71          //SceneManager.LoadScene(SceneManager.GetActiveScene().name);
72      }
```

Na versão passada, nesse ponto, nós já atribuíamos qual a função seria chamada caso o botão ***continue*** fosse apertado. Agora usar o conceito da um co-rotina. A ideia é verificarmos se passou um tempo suficiente para que o usuário possa ver usar outro ad e ter uma recompensa. A co-rotina é semelhante a uma função que tem a habilidade de pausar, e voltar depois de um tempo no ponto onde parou.

Usamos o método ***StartCoroutine***() e passamos qual será a nossa co-rotina, nesse caso é a ***ShowContinue***. Veja abaixo.

```
107     public IEnumerator ShowContinue(Button botaoContinue) {
108         var btnText = botaoContinue.GetComponentInChildren<Text>();
109         while (true) {
110             if (UnityAdControle.proxTempoReward.HasValue &&
111             (DateTime.Now < UnityAdControle.proxTempoReward.Value)) {
112                 botaoContinue.interactable = false;
113
114                 TimeSpan restante = UnityAdControle.proxTempoReward.Value - DateTime.Now;
115
116                 var contagemRegressiva = string.Format("{0:D2}:{1:D2}",
117                                                        restante.Minutes,
118                                                        restante.Seconds);
119                 btnText.text = contagemRegressiva;
120                 yield return new WaitForSeconds(1f);
121             } else {
122                 botaoContinue.interactable = true;
123                 botaoContinue.onClick.AddListener(UnityAdControle.ShowRewardAd);
124                 UnityAdControle.obstaculo = this;
125                 btnText.text = "Continue (Ver Ad)";
126                 break;
127             }
128         }
129     }
```

Observe que ela retorna um ***IEnumerator***. Não precisa entrar em detalhes. Mas toda co-rotina deve ter esse tipo de retorno. Por padrão, assim que executarmos o ***yield return*** a função será chamada no próximo frame. Mas podemos adicionar um ***delay*** através da chamada ***WaitForSeconds()***.

Na linha 110 verificamos se o tempo da ***proxTempoReward*** existe e se é o tempo atual é menos que este tempo. Se não for, impedimos o jogador de apertar o ***botãoContinue*** e mostramos o tempo restante mudando o componente ***Text*** do botão. Caso contrário, dentro do ***else*****,** deixamos o jogador apertar o botão ***continue***.