Desenvolvimento de Videojogos

Licenciatura em Engenharia Informática – 2020/2021

Guia de Laboratório nº.5

***Unity* – Som, *prefabs* e animações**

Introdução e Objetivos

No laboratório anterior programou-se um jogo recorrendo-se aos conhecimentos adquiridos anteriormente de forma a consolidar-se a matéria. Neste laboratório serão abordados os *prefabs*, sons e animações de personagens.

Preparação teórico-prática

Prefabs

É comum necessitarmos de criar objetos que fazem parte do cenário e que existem repetidamente ao longo do jogo. Se forem criados individualmente, sempre que é necessário fazer uma alteração (por exemplo, o comportamento, a cor ou a textura), teríamos de os editar um a um. Felizmente, o *Unity* possui um tipo de recurso, chamado *Prefab*, que permite armazenar um *GameObject* completo com componentes e propriedades.

Um *prefab* funciona como um modelo a partir do qual se podem criar novas instâncias do objeto na cena. Quaisquer edições feitas num recurso *prefab* são imediatamente refletidas em todas as ocorrências produzidas a partir dele. De qualquer forma, é sempre possível substituir componentes e configurações em cada instância individualmente.

Um *prefab* é criado selecionando *Asset* → *Create Prefab* e, em seguida, arrastando um *GameObject* para o *prefab* que aparece no painel *Project*. Se arrastar um *GameObject* diferente para o *prefab*, será perguntado se deseja substituir o *GameObject* atual pelo novo. Basta arrastar o recurso pré-fabricado da visualização do projeto para a visualização da cena e, em seguida, criar instâncias do *prefab*. Os objetos criados como instâncias de *prefab* são mostrados na exibição hierárquica em texto azul (os objetos normais são mostrados em texto preto - Figura 1).

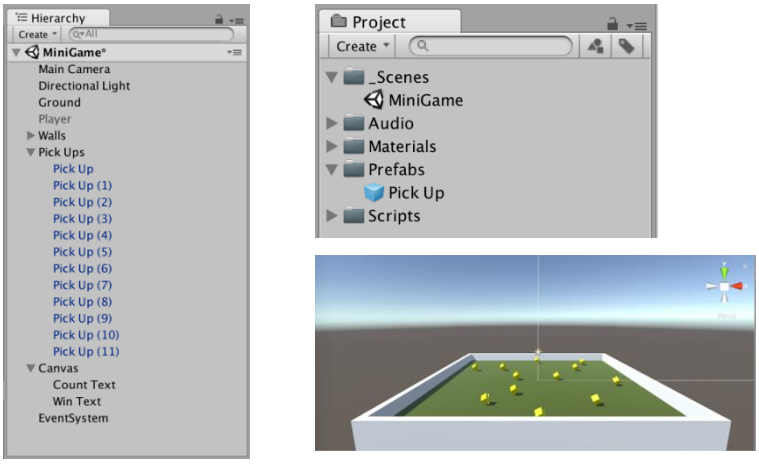

Figura 1 – *Prefabs*: cubos amarelos.

Som (*AudioListener* e *AudioSource*)

O componente *AudioListener* recebe, ou escuta, fontes de áudio (*AudioSource*) que estão no mundo do jogo. O controlo do comportamento do áudio é feito no componente de fonte de áudio e, por isso, o próprio ouvinte não tem configurações. É muitas vezes associado a um personagem e apenas pode haver um ouvinte por cena.

Para testar o áudio pode mover-se a *camera* na cena, ativando-se o botão de áudio da *Scene* (ver Figura 2). Desta forma não é preciso executar o jogo para testar o áudio na cena.

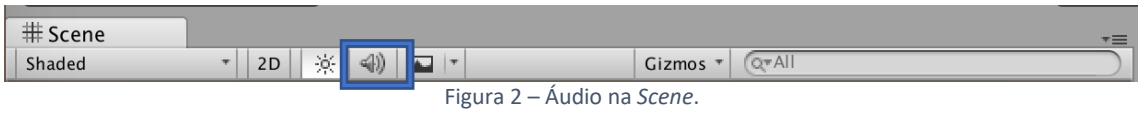

Figura 2 – Áudio na *Scene*.

As *AudioSources* são componentes que reproduzem áudio. Podemos atribuir um som arrastando e soltando ou usando código para selecionar um *clip* para reproduzir (ver Figura 3).

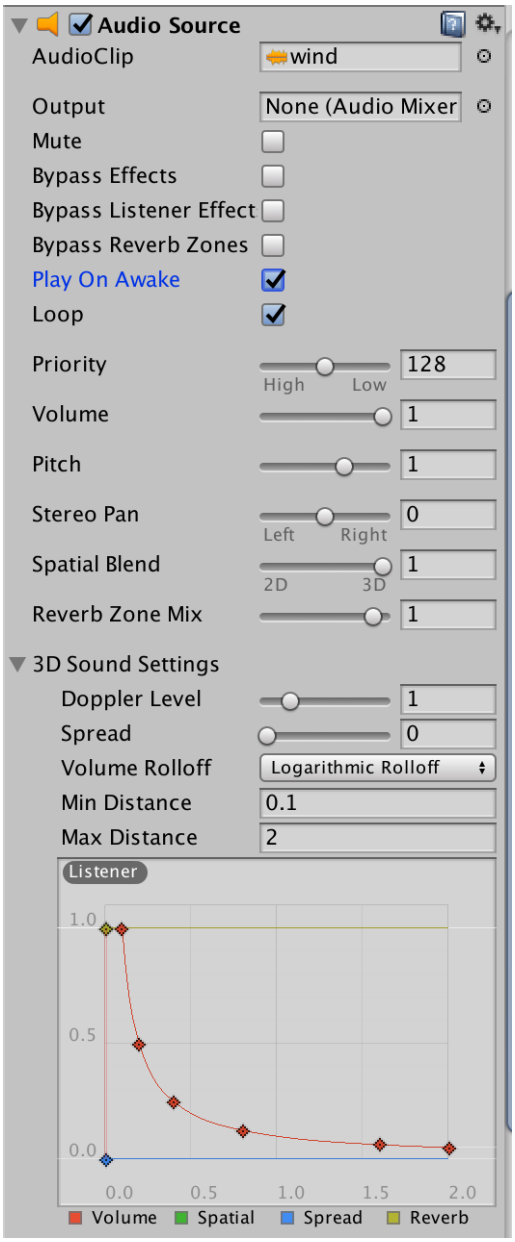

Figura 3 – *AudioSource*: vento.

Podemos silenciar, ignorar qualquer filtro, escolher reproduzir a fonte de áudio quando o jogo começa ou fazer um *loop* do *clip*. A prioridade varia entre 0 e 255, sendo 0 a prioridade mais alta. É aconselhável ter a música definida como a prioridade mais alta para evitar que outros *clips* substituam quando muitos clipes estão sendo reproduzidos de uma vez. O volume determina o volume de reprodução do *clip*. *Pitch* controla o tom do *clip*. É possível trabalhar com sons 3D. O nível Doppler define o quanto o Efeito Doppler será usado. Esta é uma mudança percecionada em termos reais quando o *player* se move para longe ou perto da fonte de som. Parecerá mais elevado aproximando-se e mais baixo enquanto se afasta. O volume *roll-off* determina o tipo de *roll-off* a ser utilizado pelo som como o ouvinte se afasta da fonte. Pode ser logarítmico, linear ou personalizado manualmente usando uma. A distância mínima define a proximidade da reprodução do som no volume total e funciona na conjunção com a definição de distância máxima. O nível de *pan* define como verdadeiramente 3D o som é. O padrão é 1 para som 3D, tornando-o totalmente *panned* quando o ouvinte estéreo é movido passado dele. *Spread* é o controlo do ângulo de propagação do som 3D para som multicanal.

Para a programação da reprodução de um clip usando-se scripts é necessário o seguinte código:

```
public AudioClip pickSound;
private AudioSource audioSource;

void Start(){
   ...
   audioSource = GetComponent<AudioSource>();
   ...
}

void OnTriggerEnter(Collider other){
   if (other.gameObject.CompareTag("Pick Up")) {
      ...
      audioSource.PlayOneShot(pickSound, 1F);
      ...
   }
}
```

Animações

Recentemente, o *Unity* introduziu um novo mecanismo para gestão de animações (*Mecanim*). É aplicável em diversos contextos, embora a introdução aqui apresentada vá incidir essencialmente na utilização deste mecanismo para animação de personagens. Cada vez mais existem personagens disponíveis compatíveis com este sistema, e os conjuntos de animações são variados.

Para o efeito devem ser importados os seguintes Assets:

- *Raw Mocap data for Mecanim*
  https://www.assetstore.unity3d.com/en/#!/content/5330
  Consiste num conjunto de animações diversas em formato *Mecanim*.

- *Space Robot Kyle*
  https://www.assetstore.unity3d.com/en/#!/content/4696
  Modelo de um robot compatível com o sistema *Mecanim*.

No sistema *Mecanim*, as transições entre as diversas animações recorrer um componente *Animation Controller*. Este pode ser criado na opção *Assets* → *Create* → *Animation Controller*. Este controlador é editado na janela *Animator* e consiste numa representação em grafo que representa as diversas animações e as transições entre elas.

As principais funcionalidades a considerar são os seguintes:

- Criar um novo estado: botão direito no fundo da janela → *Create State* → *Empty.*
- Criar uma transição: botão direito num estado → *Make Transition* → Clique no estado destino.
- Mudar o nome de um estado: utilizar o inspetor após selecionar o estado.
- Definir a animação associada a um estado: opção *Motion* no inspetor após selecionar o estado.

Na Figura 4 é apresentado um exemplo de um controlador simples com transições entre um estado de parado e outro em andamento.

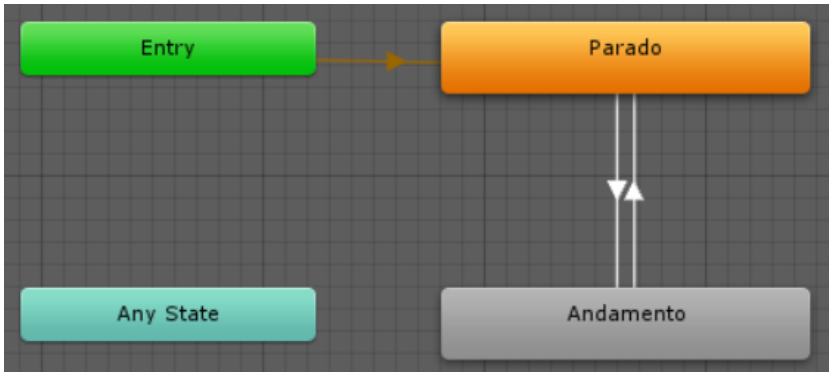

Figura 4 – Exemplo de *Animation Controller*.

Por enquanto, não foram ainda definidas as condições para a transição entre estados. Assim, o controlador arranca em "*Entry*" (o controlador arranca sempre neste estado), transita para "*Parado*" imediatamento, mantendo-se nesse estado pela duração da animação, passando depois para o estado de andamento antes de entrar em ciclo entre estes dois estados.

Para testar o funcionamento do controlador atual podemos fazer os seguintes passos:

- Selecionar o *Prefab* "*Robot Kyle*" nos *Assets* e aceder às suas propriedades, no inspetor.
- Nas propriedades apresentadas no inspetor, aceder à opção *Rig* e selecionar o tipo de animação como sendo humanoide, indicando que deve ser criado um avatar para este modelo. Deste modo, o robot ficará pronto para funcionar com o sistema *Mecanim*.
- Adicionar o "*Robot Kyle*" à cena.
- No objeto adicionado, aceder ao componente *Animator* e selecionar o controlador de animação criado anteriormente.
- Executar a cena e verificar que o robot realiza os movimentos.

As transições entre estados podem ser feitas de diversas formas: podem estar associadas à gama de valores de certas variáveis, realizarem-se automaticamente após um intervalo de tempo, ou serem despoletadas por um *trigger*.

Para este exemplo, iremos despoletar as transições com *triggers*. Estes devem ser criados na janela *Animator*, no separador relativo a parâmetros, conforme apresentado na Figura 5.

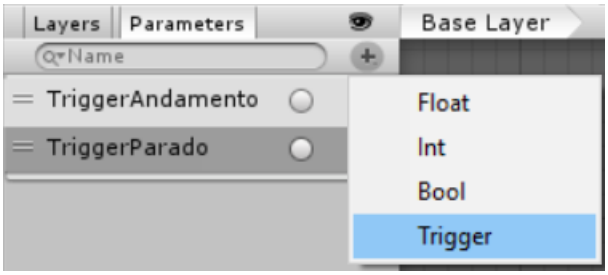

Figura 5 – Criação de *Triggers*.

A transição é configurada no inspetor após selecionar-se a seta que representa essa transição na janela *Animator*. Neste caso há a destacar as seguintes configurações apresentadas na Figura 6:

- Foi desmarcada a opção "*Has Exit Time*" para indicar que a animação não termina automaticamente.

- Foi adicionada uma condição na lista *Conditions* (botão +), tendo sido selecionado o *TriggerAndamento*. Assim, a transição entre os estados ficará associada a este *trigger*.

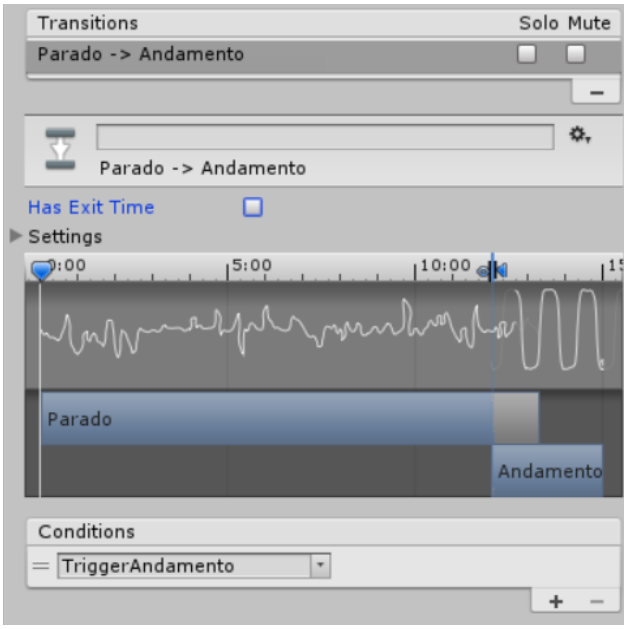

Figura 6 – Configuração da transição entre estados.

De forma semelhante, pode ser configurada uma transição de "*Andamento*" para "*Parado*", associando-se o "*TriggerParado*".

Estes *triggers* podem então ser despoletados a partir de código. Em seguida é apresentado um script simples que ativa a animação de andamento com a barra de espaços:

```
void Update ()
{

    if (Input.GetKeyDown(KeyCode.Space))
    {
        GetComponent<Animator>().SetTrigger("TriggerAndamento");
    }

    if (Input.GetKeyUp(KeyCode.Space))
    {
        GetComponent<Animator>().SetTrigger("TriggerParado");
    }

}
```

Para finalizar esta introdução sobre animação, devem ainda ser tidas em conta as seguintes considerações:

- A opção "*Apply Root Motion*" do componente *Animator* permite indicar se se pretende que a animação aplique os movimentos associados na transformação do objeto ou não. Quando esta opção é desseleccionada, a animação é feita "no lugar" e o movimento da personagem deve ser programado.

- É possível configurar o quão abrupta é a transição entre os movimentos associados à animação de cada estado. Isto pode ser feito na janela *Settings* apresentada anteriormente na Figura 6. Esta possui um conjunto variado de opções e configurações, as quais devem ser estudadas complementarmente a este trabalho.

Exercício

Neste exercício deverá programar uma combinação do tutorial "*Roll a Ball*" do *Unity* (https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial) e do jogo *Snake* (https://en.wikipedia.org/wiki/Snake_(video_game)).

Pretende-se programar uma arena (ver Figura 7) onde aparecem objetos que podem ser apanhados pelo *player* (numa primeira aproximação o *player* poderá ser uma esfera - ver **Melhoramento** em baixo). Os objetos a apanhar têm todos as mesmas características pelo que se pretende usar *prefabs* para os objetos. Fica ao critério dos alunos decidir qual o aspeto do objeto a usar.

Apenas estará um objeto visível a cada momento do tempo. De cada vez que esse objeto é apanhado, é destruído (utilize o método *GameObject.Destroy*) e outro aparecerá (tal como no jogo *Snake*) noutra localização na arena. O novo objeto poderá ser criado recorrendo-se à instanciação de um *prefab* (método *Instantiate*), usando um *GameObject* fornecido como propriedade do script, ou carregado através do método *Load* da Classe *Resources*.
Cada objeto possui um som contínuo que se torna mais audível à medida que o *player* se aproxima do objeto (ver Figura 8).

Quando um objeto é apanhado também deverá ouvir-se um som pontual que indicará o sucesso dessa ação.

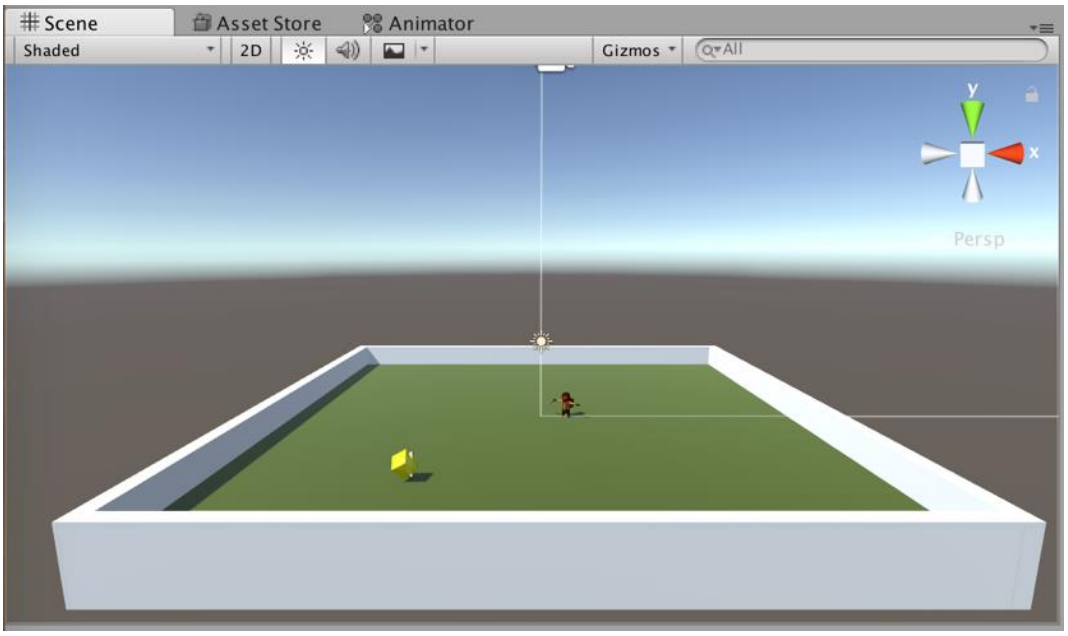

Figura 7 – Arena do jogo.

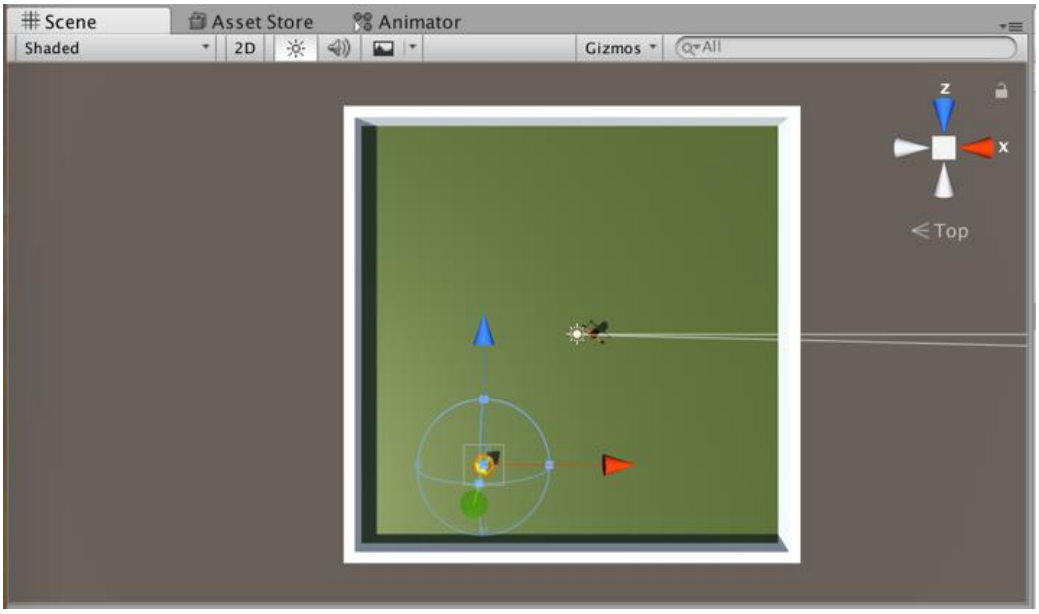

Figura 8 – Semiesfera azul: Região onde o som é audível.

Deverá ser visível um contador textual do número de objetos apanhados.
Uma forma de terminar o jogo é impor um tempo limite. O final do jogo poderá acontecer, por exemplo, ao fim de 30 segundos de jogo e neste caso deverá ser mostrada a contagem do tempo no ecrã e a mensagem de fim de jogo no final da contagem.

Melhoramento: os alunos deverão importar da *Asset Store* um *player* humanoide com pelo menos as animações *walk* e *idle* e utilizá-las para animar o *player*.

Variantes possíveis: a velocidade do boneco aumenta de cada vez que apanha um objeto ou o número de objetos a aparecer em simultâneo pode aumentar em patamares de tempo (por exemplo, aos 10 segundos passam a ser 2 e aos 20 segundos passam a ser 3).

Tutoriais de Apoio – Complementares para consolidação

Prefabs
https://docs.unity3d.com/Manual/Prefabs.html

Som
https://unity3d.com/pt/learn/tutorials/topics/audio/audio-listeners-sources
https://unity3d.com/pt/learn/tutorials/topics/audio/sound-effects-scripting

Animar um personagem
https://unity3d.com/pt/learn/tutorials/topics/animation/animator-controller?playlist=17099
https://unity3d.com/pt/learn/tutorials/topics/animation/animator-scripting?playlist=17099

Documentação Oficial

Instanciar *prefabs*
https://docs.unity3d.com/Manual/InstantiatingPrefabs.html

Método *Load* da Classe *Resources*
https://docs.unity3d.com/ScriptReference/Resources.Load.html