

Prática em Laboratório UNITY

Máquina de Estado Finito (The Adventurer)

Murilo Boratto

1 Resumo

Esta prática tem como finalidade remodelar as habilidades de um personagem 2D, o qual denominaremos *The Adventurer*. A estrutura do personagem principal consiste em um herói o qual adicionaremos múltiplos comportamentos a partir dos conceitos de **Máquinas de Estado Finito (MEF)**. Para esta cena o *player* inicializa com alguns comportamentos, já pré-programados, como por exemplo: Parado (*IDLE*), correr (*RUN*), pular (*JUMP*) e cair (*GLIDE*). Sendo assim, a idéia básica desta prática é adicionar o maior número de comportamentos possíveis ao personagem. Este roteiro é baseado no tutorial do Livro *UNITY AI Programming* [1].

Parte 1 - Instruções para prática do jogo *The Adventurer*

1. O ponto inicial será fazer o download do projeto no *google classroom* da disciplina. Após descompactá-lo, haverá uma pasta chamada *TheAdventurer*.
2. Abrimos o UNITY HUB. Adicionamos o projeto *TheAdventurer*, o qual consta os assets básicos do nosso jogo. Abriremos o projeto 2D no motor de jogo UNITY.
3. Com o UNITY aberto na aba SCENES, abra uma cena chamada: *StateMachine*.
4. Na hierarquia da cena *StateMachine*, há múltiplos objetos que a compõem e o nosso *player*. Podemos explicá-los a seguir:
 - ◇ Main Camera (Câmera de perspectiva da cena);
 - ◇ Ground (Objeto rígido que serve como chão);
 - ◇ LeftWall (Objeto rígido muro esquerdo);
 - ◇ RightWall (Objeto rígido muro direito);
 - ◇ Platform (Objeto rígido plataforma);
 - ◇ PlayerHeroPrefab (Prefab do *player* no estado inicial (*IDLE*)).

5. Na estrutura de diretórios o conteúdo está estruturada da seguinte forma:

Animations Diretório contendo as animações e o elemento *PlayerAnimatorControler*.

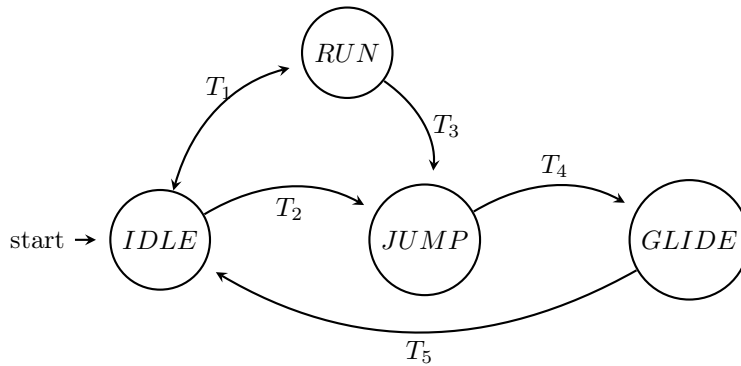
Prefabs Diretório contendo o prefab do *player* com o comportamento inicial de *Idle*.

Scenes Diretório contendo a cena inicial do jogo.

Scripts Diretório contendo o script com a MEF do personagem denominado *PlayerStateMachine.cs*.

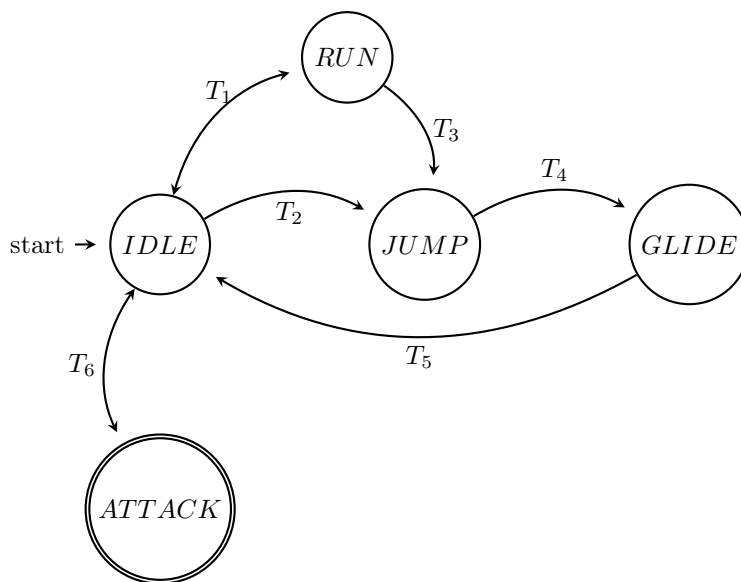
Sprites Diretório contendo os sprites com os múltiplos comportamentos do *player*.

6. Tendo explicado a estrutura do jogo, basta apertamos o *play* e assim completamos, a etapa básica do projeto de testar os 4 estados básicos do personagem. O diagrama a seguir representa os estados e as transições da MEF do *player*:



Parte 2 - Inserção do Comportamento (*ATTACK*) na Máquina de Estado

1. Uma vez testado e entendido o script fonte dos comportamentos do personagem principal, percebe-se que o (*player*) não possui o evento *ATTACK*, sendo assim se propõe a adição deste comportamento à MEF. Dando um duplo click no script *PlayerStateMachine.cs*, o editaremos para logarmos esta adição. O novo diagrama da máquina de estado ficará assim:



2. Primeiro adicionaremos o comportamento privado `attackDuration` de tipo *float*, o qual será exposto na aba do script, através da atributo *SerializeField*.
3. Adicionaremos também o novo estado dentro da lista de enumerados (*enum*) o comportamento *Attack* que estará associado a animação *Attack.anim*.
4. Setamos a variável booleana `isAttack` como falsa. Ela caracteriza a inicialização ou não do ataque pelo jogador.
5. Inicializaremos este comportamento através do teclado utilizando a letra **E**:

```
using UnityEngine;

[RequireComponent(typeof(Animator), typeof(Rigidbody2D), typeof(SpriteRenderer))]
public class PlayerStateMachine : MonoBehaviour
{
    [Header("Settings")]
    [SerializeField] float jumpYVelocity = 8f;
    [SerializeField] float runXVelocity = 4f;
    [SerializeField] float raycastDistance = 0.7f;
    [SerializeField] LayerMask collisionMask;
    [SerializeField] float attackDuration = 1f;

    Animator animator;
    Rigidbody2D physics;
    SpriteRenderer sprite;

    enum State { Idle, Run, Jump, Glide, Attack }

    State state = State.Idle;
    bool isGrounded = false;
    bool jumpInput = false;
    bool isAttack = false;

    float horizontalInput = 0f;

    void FixedUpdate()
    {
        // get player input
        isGrounded = Physics2D.Raycast(transform.position,
                                       Vector2.down, raycastDistance, collisionMask).collider != null;
        jumpInput = Input.GetKey(KeyCode.Space);
        horizontalInput = Input.GetAxisRaw("Horizontal");
        isAttack = Input.GetKey(KeyCode.E);
    }
    ...
}
```

6. A estrutura condicional *switch* também deverá conter um evento para o comportamento *attack*. Adicionaremos a linha “`case State.Attack: AttackState(); break;`” ao script:

```
...  
    if (horizontalInput > 0f)  
        sprite.flipX = false;  
  
    if (horizontalInput < 0f)  
        sprite.flipX = true;  
  
    switch (state)  
    {  
        case State.Idle:    IdleState(); break;  
        case State.Run:     RunState();  break;  
        case State.Jump:    JumpState(); break;  
        case State.Glide:   GlideState(); break;  
        case State.Attack:  AttackState(); break;  
    }  
...
```

7. No diagrama de estados sugerido o personagem somente poderá atacar quando estiver parado. Sendo assim, no módulo *IdleState*, adicionaremos a transição da ação, supondo que o personagem somente poderá atacar quando estiver no chão e a tecla E for acionada.

```
...  
void IdleState()  
{  
    // actions  
    animator.Play("Idle");  
  
    // transitions  
    if (isAttack && horizontalInput == 0f)  
        state = State.Attack;  
  
    else if (isGrounded)  
    {  
        if (jumpInput)  
            state = State.Jump;  
        else if (horizontalInput != 0f)  
            state = State.Run;  
    }  
}  
...
```

8. Para finalizar, será necessário adicionar o método correspondente ao comportamento de ataque.

```
...
float currentAttack;
void AttackState()
{
    // actions
    animator.Play("Attack");
    currentAttack += Time.fixedDeltaTime;

    // transitions
    if (currentAttack > attackDuration)
    {
        state = State.Idle;
        currentAttack = 0f;
    }
}
...
```

Parte 3 - Proposição de Novos Comportamentos à Máquina de Estado

1. A proposta da continuação desta prática é a partir dos assets contidos na pasta Sprites, inserir o maior número possível de estados para o *player*, como por exemplo:
 - ◇ *CLIMB*
 - ◇ *CROUCH*
 - ◇ *FALL*
 - ◇ *STAND*
2. Estas adições de novos comportamentos serão traduzidos em um novo diagrama de estado e deverão ser traduzidos via codificação UNITY.
3. As transições dos comportamentos ficam livres de uma ordem pré-definida, estando à cargo do programador.

Referências

- [1] Aversa, David and Kyaw, August Sithu and Peters Clifford. *Unity AI Programming*. 2018.