Produção de Jogos 4

Design Patterns (Padrões de Projeto)

Criação	Estrutura	Comportamental
Singleton	Component	**Strategy**
Factory Method	Flyweight	**Observer**
Builder	Bridge	Command
Abstract Factory	Decorator	**State**
Prototype	Composite	Iterator
	Proxy	Mediator
	Adapter	Memento
	Facede	Interpreter
		Template Methode
		Visitor
		Chain of responsibility

Produção de Jogos 4 - State

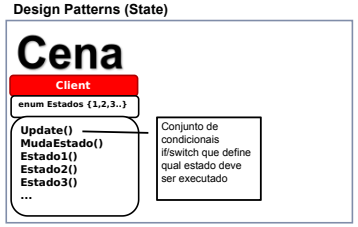
Descrição:
Permite que um objeto altere o seu comportamento quando o seu estado interno muda. O objeto parecerá ter mudado de classe.

Uso:
Semelhante os Strategy, o objeto pode alterar seu estado, porém, no state, um estado faz a transição para outro estado. O Objeto então só pode estar em um de um conjunto de estados (finito) (Finit State Machine)

Prática:
Crie uma interface State com os métodos de transição de estado.
Crie objetos concretos para representar cada estado. Cada estado deve implementar o método de transição que define qual o novo estado.
Crie uma classe concreta (FSM) que recebe um estado e executa os métodos do estado, que resulta na troca do estado atual.

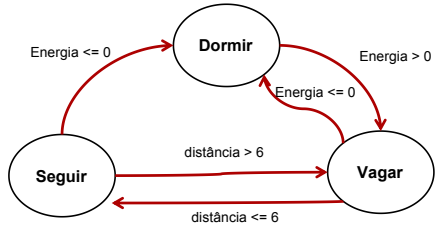
Produção de Jogos 4 - State

Design Patterns (State)



- Inclusão de novos estados com edição do cliente
- Muitos estados podem deixar a classe complexa
- Difícil manutenção e entendimento
- Utilização de muitas condicionais podem tornar complexo o entendimento do fluxo

Produção de Jogos 4 - State



Produção de Jogos 4 - State

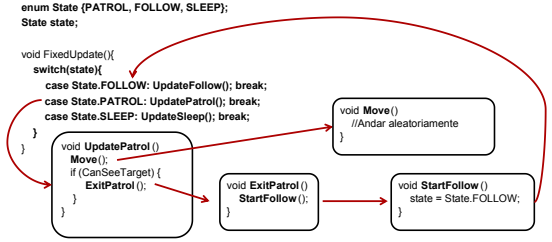
```
//FSM Hardcode
using UnityEngine;
public class FSM : MonoBehaviour {
   public enum State {PATROL, FOLLOW, SLEEP};
   public State state;
   void FixedUpdate(){
      switch(state){
         case State.PATROL: Patrol(); break;
         case State.FOLLOW: Follow(); break;
         case State.SLEEP: Sleep(); break;
      }
   }
   void Patrol(){
      if (CanSeeTarget()) StartFollow();
      Move();
   }
   void Follow(){
      if (!CanSeeTarget()) StartPatrol();
      dir = target.position - transform.position;
      Move();
   }
   void Sleep(){
      energy += Time.fixedDeltaTime;
      if (energy > 5.0f) StartPatrol();
   }
   void StartPatrol(){ state = State.PATROL; ...}
   void StartSleep(){ state = State.SLEEP; }
   void StartFollow(){ state = State.FOLLOW; }
   void Move(){ ...}
   bool CanSeeTarget(){...}
}
```

Produção de Jogos 4 - State

Esquema para Hardcode

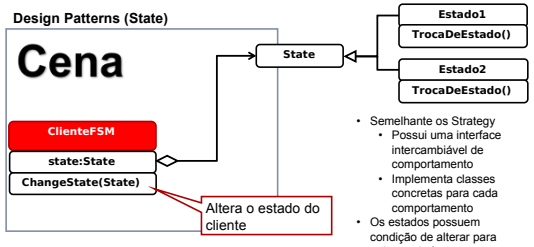
```
//Atributos
enum State {PATROL, FOLLOW, SLEEP};
State state;

void FixedUpdate(){
   switch(state){
      case State.FOLLOW: UpdateFollow(); break;
      case State.PATROL: UpdatePatrol(); break;
      case State.SLEEP: UpdateSleep(); break;
   }
}
```



Produção de Jogos 4 - State

Design Patterns (State)



- Semelhante os Strategy
  - Possui uma interface intercambiável de comportamento
  - Implementa classes concretas para cada comportamento
- Os estados possuem condição de alterar para outros estados

Produção de Jogos 4 - State

```
//FSM Hardcode
using UnityEngine;
public class FSM : MonoBehaviour {
   public enum State {PATROL, FOLLOW, SLEEP};
   public State state;
   void FixedUpdate(){
      switch(state){
         case State.PATROL: UpdatePatrol(); break;
         case State.FOLLOW: UpdateFollow(); break;
         case State.SLEEP: UpdateSleep(); break;
      }
   }
   void UpdatePatrol(){
      if (CanSeeTarget())  StartPatrol();
      Move();
   }
   void UpdateFollow (){
      if (!CanSeeTarget())  StartPatrol();
      dir = target.position - transform.position;
      Move();
   }
   void UpdateSleep (){
      energy += Time.fixedDeltaTime;
      if (energy > 5.0f)  StartPatrol();
   }
   void StartPatrol(){ state = State.PATROL; ...}
   void StartSleep(){  state = State.SLEEP; }
   void StartFollow(){ state = State.FOLLOW; }
   void Move(){ ...}
   bool CanSeeTarget(){...}
}
```

Produção de Jogos 4 - State

Design Patterns (State) MODELO

StateFSM.cs

```
using UnityEngine;

public interface StateFSM{
    public void Enter();
    public void Update();
    public void Exit();
}
```

SleepStateFSM.cs

```
using UnityEngine;
public class SleepStateFSM: StateFSM {
    PlayerMachineFSM player;
    public SleepStateFSM(
        PlayerMachineFSM player){
        this.player = player;
    }
    public void Enter() {}
    public void Exit() {}
    public void Update() {}
}
```

PlayerMachineFSM.cs

```
using UnityEngine;

public class PlayerMachineFSM : MonoBehaviour {
    StateFSM state;

    void Start() {
        SetState(new SleepStateFSM(this));
    }
    void Update() {
        state?.Update();
    }
    public void SetState(StateFSM state){
        state?.Exit();
        this.state = state;
        state?.Enter();
    }
}
```

Produção de Jogos 4 - State

Design Patterns (State) Exemplo

StateFSM.cs

```
using UnityEngine;

public interface StateFSM{
    public void Enter();
    public void Update();
    public void Exit();
}
```

SleepStateFSM.cs

```
using UnityEngine;
public class SleepStateFSM: StateFSM {
    PlayerMachineFSM player;
    float time;
    public SleepStateFSM(
        PlayerMachineFSM player){
        this.player = player;
    }
    public void Enter() {
        time = Time.time + 3;
        Debug.Log("Sleep");
    }
    public void Update() {
        if (Time.time > time){
            player.SetState(new PatrolStateFSM(player));
        }
    }
    public void Exit() {
        player.energy = 3;
    }
}
```

Produção de Jogos 4 - State

Design Patterns (State) Exemplo

StateFSM.cs

```
using UnityEngine;

public interface StateFSM{
    public void Enter();
    public void Update();
    public void Exit();
}
```

FollowStateFSM.cs

```
using UnityEngine;
public class FollowStateFSM: StateFSM {
    PlayerMachineFSM player;
    Vector3 dir;

    public FollowStateFSM(
        PlayerMachineFSM player){
        this.player = player;
    }
    public void Enter() {
        Debug.Log("Follow");
    }
    public void Update() {
        player.Move(player.TargetDir().normalized);
        if (player.energy < 0){
            player.SetState(new SleepStateFSM(player));
        }
        if (!player.IsNearTarget()){
            player.SetState(new SleepStateFSM(player));
        }
    }
    public void Exit() {}
}
```

Produção de Jogos 4 - State

Design Patterns (State) Exemplo

StateFSM.cs

```
Assets > StateFSM.cs > ...
1   using UnityEngine;
2
3   public interface StateFSM{
4       public void Enter();
5       public void Update();
6       public void Exit();
7   }
```

PatrolStateFSM.cs

```
Assets > PatrolStateFSM.cs > ...
1   using UnityEngine;
2   public class PatrolStateFSM: StateFSM {
3       PlayerMachineFSM player;
4       float time;
5       Vector3 dir;
6       public PatrolStateFSM(
7           PlayerMachineFSM player){
8           this.player = player;
9       }
10      public void Enter() {
11          Debug.Log("Patrol");
12          time = Time.time;
13      }
14      public void Update() {
15          if (Time.time > time){
16              dir = Random.onUnitSphere;
17              dir.z = 0;
18              time = Time.time + 1;
19          }
20          player.Move(dir);
21          if (player.energy < 0){
22              player.SetState(new SleepStateFSM(player));
23          }
24          if (player.IsNearTarget()){
25              player.SetState(new FollowStateFSM(player));
26          }
27      }
28      public void Exit() {}
29  }
```

Produção de Jogos 4 - State

Design Patterns (State) Exemplo

StateFSM.cs

```
Assets > StateFSM.cs > ...
1   using UnityEngine;
2
3   public interface StateFSM{
4       public void Enter();
5       public void Update();
6       public void Exit();
7   }
```

PlayerMachineFSM.cs

```
Assets > PlayerMachineFSM.cs > ...
1   using UnityEngine;
2   public class PlayerMachineFSM : MonoBehaviour {
3       StateFSM state;
4       public Transform target;
5       public float energy;
6       public float speed;
7       void Start() {
8           SetState(new PatrolStateFSM(this));
9           energy = 3;
10      }
11      void FixedUpdate() {
12          state?.Update();
13      }
14      public void SetState(StateFSM state){
15          state?.Exit();
16          this.state = state;
17          state?.Enter();
18      }
19      //Metodos auxiliares
20      public Vector3 TargetDir(){ //vetor direção para target
21          Vector3 dir = target.position-transform.position;
22          return dir;
23      }
24      public bool IsNearTarget(){ //se está próximo do target
25          return (TargetDir().magnitude < 6.0f);
26      }
27      public void Move(Vector3 dir){ //mover em direção
28          energy -= Time.fixedDeltaTime;
29          transform.position += dir * speed * Time.fixedDeltaTime;
30      }
31  }
```

Produção de Jogos 4 - State

MonoBehaviour

- Conjunto de estados
- Intercambiável (Component Script)

MonoState.cs

```
Assets > FSMState > MonoState.cs > ...
1
2   using UnityEngine;
3
4   public class MonoState : MonoBehaviour {
5
6       void Awake() {}
7       void OnEnable() {}
8       void Start() {}
9
10      void FixedUpdate(){}
11      void Update() {}
12      void LateUpdate() {}
13
14      void OnDisable(){}
15      void OnDestroy(){}
16  }
```