Necesidades

- Control de los comportamientos
  - Si ocurre A, B y C quiero que haga D

- No siempre
  - Si no que haga algo que tenga sentido

Necesidades

- La IA tiene que tener objetivos, dirección

- La IA tiene que reaccionar a imprevistos, eventos

Soluciones

Planificación

- Generación de planes
- Lista de acciones que consiguen un objetivo
- No describo cómo hacer las cosas
  - Máquina de estados
  - Script
- Defino acciones posibles
- Defino estado inicial
- Defino estado objetivo
- Obtengo qué acciones aplicar para llegar al estado objetivo
- Muy semejante a pathfinding pero para comportamientos

Planificación: Ejemplo

- Mundo de los bloques
- Acciones
  - Levantar bloque de la mesa
  - Coger un bloque de encima de otro
  - Dejar un bloque en la mesa
  - Dejar un bloque encima de otro
- Cada acción tiene precondiciones
  - Definen cuándo se puede aplicar la acción
- Precondiciones LevantarBloque
  - El bloque está en la mesa
  - Brazo de robot libre
  - El bloque no tiene ningún bloque encima

Planificación: Ejemplo

- Cada acción tiene una serie de efectos
  - Afectan al estado
  - Qué cosas cambian en el mundo al aplicar la acción
- Efectos LevantarBloque
  - Brazo robot ocupado con bloque
  - Bloque ya no está encima de la mesa

Planificación: Ejemplo

Planificación: Ejemplo

Coger(B, A), Dejar(B), Levantar(C), Colocar(C, B)

Planificación

- Mundo de los bloques en un juego
- ¿Script?
- ¿Máquina de estados jerárquica?
- ¿Máquina de estados?
- ¿Muchos ifs encadenados?

Planificación: Ventajas

- Flexible
- Robusto
- Adaptable
- Procedural

Planificación: Desventajas

- Dificultad en el control

Behavior Trees

- Una forma de mezclar lo mejor de varios mundos

Behavior Trees

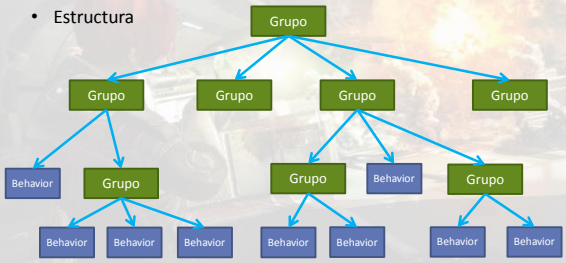
- Objetivos
  - Facilidad de programación
  - Alto nivel de control
  - Adaptabilidad a situaciones imprevistas
  - Reutilización de comportamientos
  - Respuesta rápida a eventos del juego

Behavior Trees

- Ideas
  - Usar la búsqueda para la flexibilidad
  - Construir un arbol de comportamientos

Behavior Trees

- Estructura



Behavior Trees

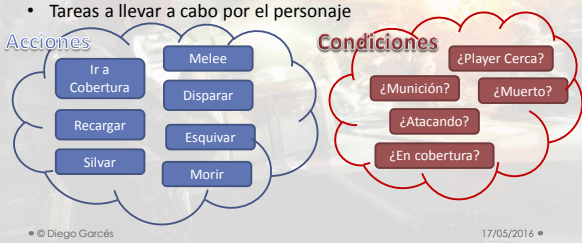
- Se recorre el árbol hasta encontrar una acción a realizar
- Búsqueda en profundidad
- De izquierda a derecha
- La primera acción que se pueda ejecutar se ejecuta

Behavior Trees

- Dos tipos de Nodos
  - Grupos
  - Behaviors
- Grupos
  - Tienen una lista de hijos
  - Distintas variaciones de comportamiento
- Behaviors
  - Hojas del árbol
  - Cosas a ejecutar por el personaje

Behavior Trees: Behaviors

- Son las hojas del árbol
- Tareas a llevar a cabo por el personaje

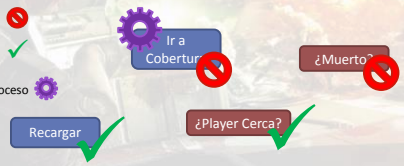


Behavior Trees: Behaviors

- Dos tipos
  - Acciones
  - Condiciones
- Comportamiento homogéneo
- Pueden añadirse en cualquier parte del árbol

Behavior Trees: Behaviors

- Se ejecutan: función update
- Devuelven un estado
  - Fallo
  - Éxito
  - En proceso



Behavior Trees: Condiciones

- Generalmente de ejecución instantánea
- Su update devuelve
  - Fallo → La condición no se cumple
  - Éxito → La condición se cumple

Behavior Trees: Acciones

- Llevan a cabo un comportamiento
- Generalmente visible en el juego
- Reproducción de animaciones
- Reproducción de sonido
- Movimiento

Behavior Trees: Acciones

- Generalmente de ejecución durante varios frames
- Animación → Hasta que termina
- Movimiento → Hasta que llega

Behavior Trees: Acciones

- Generalmente de ejecución durante varios frames
- Animación → Hasta que termina
- Movimiento → Hasta que llega

Behavior Trees: Grupos

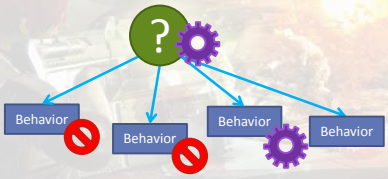
- Agrupan nodos
  - Otros grupos
  - Acciones
- Varios tipos
- Selector
  - Se elije uno de sus hijos para ejecutar
- Secuencia
  - Se van ejecutando sus hijos uno detrás del otro

Behavior Trees: Selectores

- Se van ejecutando sus hijos **hasta** que uno completa con éxito

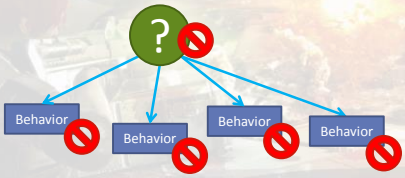
Behavior Trees: Selectores

- Se van ejecutando sus hijos **hasta** que uno completa con éxito



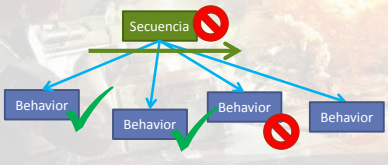
Behavior Trees: Selectores

- Se van ejecutando sus hijos **hasta** que uno completa con éxito



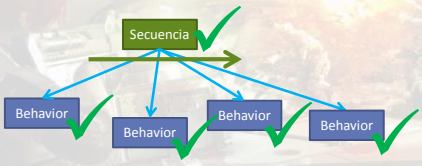
Behavior Trees: Secuencias

- Se van ejecutando sus hijos **mientras** se completen con éxito



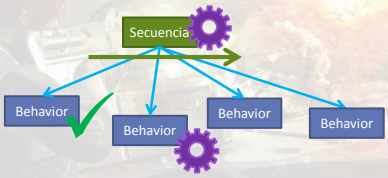
Behavior Trees: Secuencias

- Se van ejecutando sus hijos **mientras** se completen con éxito

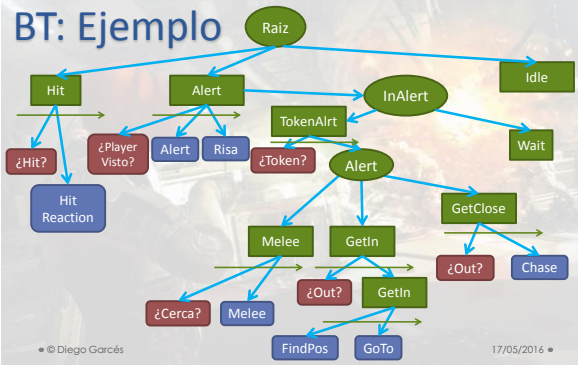


Behavior Trees: Secuencias

- Se van ejecutando sus hijos **mientras** se completen con éxito



BT: Ejemplo



Behavior Trees: Implementación

```
class Behavior
{
protected:
      Status update();
      void onEnter();
      void onExit();
public:
      Behavior();
      Status tick();
private:
      Status m_Status;
};
```

```
enum Status
{
      eInvalid,
      eSuccess,
      eFail,
      eRunning,
};
```


Behavior Trees: Implementación

```
Status Behavior::tick()
{
      if (m_Status == eInvalid)
      {
            onEnter();
      }

      m_Status = update();

      if (mStatus != eRunning)
      {
            onExit();
      }
      return m_Status;
}
```

```
class Behavior
{
protected:
      Status update();
      void onEnter();
      void onExit();
public:
      Behavior();
      Status tick();
private:
      Status m_Status;
};
```


Behavior Trees: Implementación

```
class Group: public Behavior
{
protected:
      typedef std::vector<Behavior*> Behaviors;
      Behaviors m_Children;
};
```

Behavior Trees: Implementación

```
class Sequence: public Group
{
protected:
      void onEnter();
      Status update();

      int m_CurrentChild;
};
```

```
class Group: public Behavior
{
protected:
      typedef std::vector<Behavior*> Behaviors;
      Behaviors m_Children;
};
```


Behavior Trees: Implementación

```
Status Sequence::update()
{
      while (true)
      {
            Status s = m_Children[m_CurrentChild]->tick();

            if (s != eSuccess)
            {
                  return s;
            }
            ++m_CurrentChild

            if (m_CurrentChild == m_Children.size())
            {
                  return eSuccess;
            }
      }
      return eInvalid;
}
```

```
void Sequence::onEnter()
{
      m_CurrentChild = 0;
}
```


Behavior Trees: Implementación

```
Status Selector::update()
{
      while (true)
      {
            Status s = m_Children[m_CurrentChild]->tick();

            if (s != eFail)
            {
                  return s;
            }
            ++m_CurrentChild;
            if (m_CurrentChild == m_Children.size())
            {
                  return eFail;
            }
      }
      return eInvalid;
}
```

Behavior Trees: Implementación

```
class PlayerClose: public Behavior
{
protected:
    Status update();
public:
    PlayerClose(float minDistan
private:
    float m_MinDistance;
};
```

```
Status PlayerClose::update()
{
    float dist = GetDistance(
        GetPlayer()->Position(),
        GetOwner()->Position()
    );
    if (dist < m_MinDistance)
        return eSuccess;
    else
        return eFail;
}
```


Behavior Trees: Implementación

```
class GoTo: public Behavior
{
protected:
    void onEnter();
    Status update();
    void onExit();
public:
    GoTo(Vec3D destination);
private:
    Vec3D m_Destination;
    Path m_Path;
};
```


Behavior Trees: Implementación

```
void GoTo::onEnter()
{
    m_Path = Pathfinding::FindPath(
        GetOwner()->Position(),
        Destination()
    );
}
```

```
void GoTo::onExit()
{
    GetOwner()->PlayAnimation("Stop");
    m_Path.Release();
}
```

Behavior Trees: Implementación

```
Status GoTo::update()
{
	GetOwner()->UpdatePath(m_Path);

	if (CheckArrived())
		return eSuccess;
	else if (CannotMove())
		return eFail;
	else
		return eRunning;
}
```


Behavior Trees: Alternativa

- Las condiciones son un behavior
  - Mismo comportamiento que una acción
  - El árbol es más homogéneo
  - Se soportan directamente condiciones de más de un frame
- A veces resulta un poco farragoso
  - Intercalar grupos de secuencia y Selectores
  - Para hacer que un behavior se haga según una condición

Behavior Trees: Deciders

- Añadir una condición al Behavior → Decider
- Deciders
  - Group decider: Tiene otros behaviors como hijos
  - Behavior decider: Tiene una acción como hijo

Behavior Trees: Deciders

- Añadir una condición al Behavior → Decider
- Deciders
  - Group decider: Tiene otros behaviors como hijos
  - Behavior decider: Tiene una acción como hijo

BT Decider: Implementación

- Muy similar
- No cambia el comportamiento de los grupos
- Hay que chequear la condición del decider antes del tick