PA1458 Hemtenta Exempel III

Mikael Svahnberg

May 21, 2021

Contents

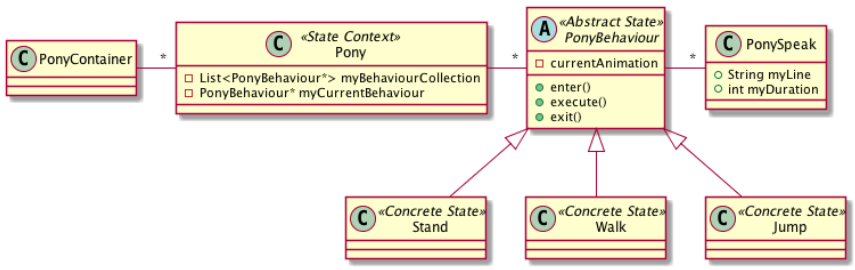
1 Systembeskrivning

Desktop Ponies är ett program som låter MLP ponnys springa runt på skärmen. Varje Ponny har olika beteenden (tillstånd). Till excempel, stå, springa, jhoppa. Dessa olika beteenden implementeras som ett State Pattern.

2 Klassdiagram

```
PonyContainer - "*" Pony
```

```
class Pony <<State Context>> {
 -List<PonyBehaviour*> myBehaviourCollection
 -PonyBehaviour* myCurrentBehaviour
}

Pony - "*" PonyBehaviour
PonyBehaviour - "*" PonySpeak

abstract class PonyBehaviour <<Abstract State>> {
 +enter()
 +execute()
 +exit()
 - currentAnimation
}

class Stand <<Concrete State>>
class Walk <<Concrete State>>
class Jump <<Concrete State>>

PonyBehaviour <|-- Stand
PonyBehaviour <|-- Walk
PonyBehaviour <|-- Jump

PonySpeak : +String myLine
PonySpeak : +int myDuration
```



3 Beskrivning av klasser

Pony är «context» i State Pattern. Den äger en samling med PonyBehaviours, och är ansvarig för att veta vilket beteende som är aktuellt

för stunden. Själva beteendet, vad som skall göras just nu, delegeras till sitt currentBehaviour.

PonyBehaviour är «Abstract State» och beskriver därmed gränssnittet, vilka metoder, som alla konkreta beteenden skall ha.

{Stand, Walk, Jump} är konkreta implementationer av olika tillstånd.

PonyContainer innehåller alla Ponies

PonySpeak är en replik som en Ponny kan säga i ett visst tillstånd.

4 Pseudokod

```
Pony::create() {
 List<String> behaviourNames = PonyInitFile::getBehaviours(myPonyName);
 behaviourNames.forEach( (n) => {
   PonyBehaviour* pb;
   switch(n) {
   case "Stand": pb=new Stand();
   case "Walk" : pb=new Walk();
   case "Jump" : pb=new Jump();
   }

   myBehaviourCollection.append(pb);
  });

 myCurrentBehaviour = myBehaviourCollection[0];
}

Pony::setBehaviour(String newBehaviour) {
  PonyBehaviour* newBehaviour = myBehaviourCollection->find(newBehaviour);

  if(newBehaviour) {
   myCurrentBehaviour->exit();
   myCurrentBehaviour = newBehaviour;
   myCurrentBehaviour->enter();
  }
}

Pony::executeBehaviour() {
```

```
  myCurrentBehaviour->execute();
}

Stand::enter() {
  cout << "Starting to *Stand*" << endl;
}
Stand::execute()  {
  cout << "Still Standing" << endl;
}
Stand::exit()  {
  cout << "No longer standing" << endl;
}
```

5 Designmönstrets användande av GRASP

- **GRASP1** Information Expert
- **GRASP2** Controller

`«Context»`-klassen är information expert på vilka tillstånd som finns, hur man byter mellan dem, och vilket tillstånd som är aktuellt just nu. Den är också en controller genom att den alltid delegerar till det nuvarande tillståndet när något skall utföras.

`«Abstract State»`-klassen är information expert på vilket *gränssnitt* som alla konkreta tillstånd skall erbjuda.

`«Concrete State»`-klasserna ärver från «abstract state» och implementerar alla metoderna. Det gör dem till information expert på hur varje metod skall bete sig i just det tillståndet.

6 Systemets användande av GRASP

- PonyContainer är Information Expert om vilka Ponnys som finns.
- Pony är information expert om vad det innebär att vara en viss ponny. Som «context»-klassen i Abstract State är den information expert på vilka beteenden som en viss ponny har, och vilket beteende den har just nu. PSS är den också controller för att bestämma när och hur beteendet skall användas/anropas.
- PonyBehaviour, som «abstract State» är information expert på vilket gränssnitt, vilka metoder, som varje beteende implementerar.

- PonySpeak är information expert på en enskild replik som en Ponny kan säga.
- {Stand, Walk, Jump} som «concrete states» är de information experts på hur ett visst tillstånd skall bete sig.