Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2024-02-02

Pacman

Scenario

Du jobbar i en grupp som implementerar en version av det klassiska arkadspelet Pacman. Du har fått i uppgift att ta fram programkod för att representera tre av de spöken som finns i originalspelet. Koden du producerar ska integreras i en existerande kodbas och det är därför viktigt att ta hänsyn till hur övriga delar av systemet är strukturerade. Vidare ska du skriva ett program som testar din implementation.

Kvalitetskrav

Du uppnår korrekthet, användbarhet, ändringsbarhet och integrerbarhet enligt nedan.

Korrekthet

Dina kollegor ska inte kunna använda din modul på fel sätt. Det ska enbart gå att skapa fullständigt korrekta objekt. Felaktig användning av din modul ska leda till kompileringsfel i första hand och körfel i andra hand. Din modul ska alltid generera fullständigt korrekta resultat. Din modul ska aldrig generera oväsentligt eller fel resultat. Fel som uppstår när klasserna används ska rapporteras till anropande kod. Anroparen ansvarar för att filtrera vad som rapporteras till slutanvändaren. Du behöver alltså fundera extra noga på hur din kollega använder din kod och hur hen ska detektera fel som uppstår.

Användbarhet

Programkoden för din modul ska vara samlad i en implementationsfil med en tillhörande headerfil. Det ska vara tydligt vilken kod som är avsedd för dina kollegor att använda och vilken kod de inte ska röra. Användning av ej avsedd kod ska leda till kompileringsfel. Din modul ska följa C++ konventioner där det är möjligt. Din modul ska inte ge upphov till kompileringsvarningar eller kompileringsfel.

Ändringsbarhet

Du ska förutsätta att all din kod som är avsedd att användas av dina kollegor också skulle kunna användas i andra program. Du ska se till att du kan ändra den interna representationen av spökena och att detta inte ska påverka dina kollegors program över huvud taget. Deras program ska inte märka någon skillnad i syntax eller funktion mellan din gamla och din nya modul. Dina kollegor behöver alltså inte ändra något trots att du ändrat massor.
Programmet ska vara enkelt att utöka. Det ska gå att lägga till nya typer av spöken utan ändringar till existerande kod i din modul. Testprogrammet och din kollegas kod ska endast behöva utökas med det som är nytt utan att ändra existerande kod.

Integrerbarhet

Du ska använda kod skriven av dina kollegor för att lösa problem där det är relevant. Den givna koden ska fungera oförändrad och får inte modifieras. Koden som produceras ska följa samma namngivning och kodstil som den givna koden och koden skriven av dina kollegor.

Funktionalitetskrav

Du ska implementera klasserna `Blinky`, `Pinky` och `Clyde` som delar i en polymorf klasshierarki. Spökenas specifika beteende finns beskrivet under rubriken *Pacman i ett nötskal* nedan.
Varje klass implementerar

- `get_chase_point()` som ger punkten spöket söker till under 'chase'.
- `get_scatter_point()` som ger punkten spöket söker till under 'scatter'
- `get_color()` vars returvärde är en sträng som representerar spökets färg på engelska. T.ex. `"red"`, `"green"` eller `"blue"`.
- `set_position(Point)` som ändrar spökets position till en annan inom spelplanen.
- `get_position()` vars returvärde är spökets nuvarande position.

Utöver det ska klassen Blinky implementera

- `is_angry()` vars returvärde är sant om Blinky är arg.
- `set_angry(bool)` som sätter om Blinky är arg eller inte.

Du ska använda objektorienterade principer för att återanvända kod där det är möjligt. Upprepning av klass-struktur och implementation mellan klasserna är inte tillåtet. Se "Koncept som ingår" för mer ledning till vad som kommer behövas. Du behöver börja med att sätta dig in i hela denna instruktion, varje listat koncept och den givna koden i `main.cc`, `given.h` och `given.cc`. Dina spök-klasser ska ligga på sin egen `.h` och `.cc`-fil, lämpligt namngivna. Du kan motivera varför det är ok att alla spöken samsas i samma fil.

Testkrav

Programmodulen ska testas genom ett terminalbaserat gränssnitt enligt nedan. Testprogrammet visualiserar objektens position och ger testaren möjlighet att ändra olika egenskaper. Testprogrammet ska implementeras som en inkapslad klass där funktionalitet är uppdelad i medlemsfunktioner med ett tydligt syfte. Även här ska du följa ovanstående kvalitetskrav och testprogrammet ska kunna hantera ett godtyckligt antal unika spöken. Testprogrammet ska ha stöd för att

- flytta ett spöke genom att ange dess färg och en ny position.
- flytta spelarfiguren genom att ange `pos` och en ny position.
- sätta spelarfigurens riktning genom att ange `dir` och en ny riktning.

- byta mellan 'chase' och 'scatter' med `chase` och `scatter`.
- stänga av programmet med `quit`.
- förarga relevanta spöken genom att ange `anger`.

Att skiva testprogrammet utan kodupprepning och så det blir så tydligt som möjligt är en stor utmaning. Det är lätt att hamna i fler if-satser än nödvändigt och med så djup indenteringsnivå att läsbarheten försämras. Du säger så klart "challenge accepted!" och undviker dessa problem.

Körexempel 1 visar hur testprogrammet är tänkt att fungera. Versaler används för att markera ett spökes position och en motsvarande gemen markerar punkten den söker mot. Efter varje kommando användaren matar in ritas en uppdaterad spelplan ut. Användarens indata markeras här med **fet** stil.

Förklaring av testutdata: Varje position på spelplanen ritas ut som två tecken i bredd. Koordinat `(0,0)` är den första tomma rutan i nedre vänstra hörnet. Spelplanen nedan har storleken `(7,7)` och därmed är det övre högra hörnet på position `(6,6)`. Spelplanen har här minskats för att spara plats, men din implementation ska självklart rita ut spelplanen i full storlek (se WIDTH, HEIGHT i `given.cc`). Ett spöke ritas ut på det första tecknet av en position och representeras av den första bokstaven i sin färg. Spelarfiguren ritas ut på det andra tecknet av en position och representeras med `@`.

Tips: För att hitta rätt tecken att rita ut för varje spöke kan man plocka ut första bokstaven i spökets färg och använda de inbyggda funktionerna `toupper`/`tolower` för att konvertera till versal/gemen. Notera att du kan behöva använda `static_cast` för att konvertera returvärdet till ett tecken.

Given kod

Bifogat finns filer med kod för att du ska komma igång med uppgiften. Filen `main.cc` innehåller grunden till ditt testprogram. Utöka denna med din egen kod för att uppfylla testkraven ovan. `given.h` och `given.cc` innehåller ett urval av kod från det kompletta spelet som du behöver använda dig av för att lösa uppgiften. Representationen av spelarfiguren `Pacman` som en klass får inte modifieras. Aggregatet `Point` kan utökas med egna operatorer vid behov. Filerna behöver modifieras så att programmet kompilerar korrekt.

`Point` används dels för att representera positioner på spelplanen men även för att representera spelarfigurens riktning. Då antar variabeln värdena `(1,0)`, `(0,1)`, `(-1,0)`, `(0,-1)` där siffrorna representerar positiv eller negativ riktning i x- och y-led.

Körexempel 1

```
$ ./a.out
+--------------+
|              |
|  R           |
|      P       |
|              |
|        O     |
|  r@  p       |
|o             |
+--------------+
> red 2 6
+--------------+
|    R         |
|              |
|      P       |
|              |
|        O     |
|  r@  p       |
|o             |
+--------------+
> pos 4 6
+--------------+
|    R   r@  p |
|              |
|      P       |
|              |
|        O     |
|              |
|o             |
+--------------+
> scatter
+--------------+
|p   R    @  r |
|              |
|      P       |
|              |
|        O     |
|              |
|o             |
+--------------+
> quit
```

Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2024-02-02

Uppgift

Du ska visa att du förstått hur alla inkluderade koncept i C++ används och fungerar genom att implementera programmodulen så att alla typer av krav uppfylls. Du redovisar muntligen för din assistent under labtid. Därefter lämnar du in digitalt för bedömning. Koncept som ingår listas nedan.

Koncept som ingår

Alla listade koncept ingår i uppgiften. Endast listade koncept ingår i uppgiften, dvs problem där nedan koncept inte är lämpliga att använda kan lösas med enbart förkunskaper från kursen eller tidigare kurser.

- Klasser
  - Datamedlemsintieringslista
  - Delegerade konstruktor
  - Referensmedlem
- Arv
  - Basklass
  - Härledd klass
  - Publikt arv
  - Överlagring av medlemsfunktioner (`override`)
- Polymorfi
  - Virtuell funktion
  - Virtuell destruktor
  - Basklasspekare
  - Dynamisk typkontroll (`dynamic_cast`)

Bonusuppgift: Ytterligare klass

Som extra uppgift ska du nu implementera det fjärde och sista spöket från orginalspelet i din kod. Du behöver också uppdatera ditt testprogram och det givna fulla spelet så det stödjer ditt nya spöke.

Det fjärde spöket, Inky, baserar sin position på spelarfigurens och Blinkys position. Dra ett sträck mellan blinkys position och punkten två steg framför spelarfiguren (i dess nuvarande riktning). Fortsätt det sträcket lika långt till så har du Inkys chase-punkt.

Redovisa för en assistent och visa hur din kod integrerar snyggt i det givna spelet. Lämna sedan in dina uppdaterade filer för klasserna och testprogrammet.

Grafisk Pacman

När din implementation och ditt testprogram fungerar korrekt kanske du känner att det skulle vara roligt att faktiskt bli jagad av dina spöken. Tur för dig då att dina kollegor är klara med sin implementation och lagt upp all kod i den komprimerade mappen `full_game.zip` på kurshemsidan för dig att testa med. Plocka ner deras kod, lägg in din egen kod enligt instruktionerna och börja spela!

OBS: Du kan behöva ändra namngivning och inparametrar i din egen kod om du avvikit från ovanstående beskrivningar.

Pacman i ett nötskal

I det klassiska arkadspelet Pacman ska spelaren plocka upp alla bitar mat utan att bli tagen av de tre (vanligtvis fyra) spöken som jagar hen. Det som är av vikt för din uppgift är hur de fyra spökena rör sig. Ett spöke i Pacman väljer en position i relation till spelarfiguren och planerar sedan fram en väg att gå. Sättet den väljer position beror på vilket spöke det är och vilket tillstånd spelet är i.

Pacman har två lägen; chase och scatter. Under 'chase' jagar spökena spelarfiguren. Det innebär att de väljer en position i relation till spelarfiguren att söka sig till. Under 'scatter' flyr varje spöke till ett specificerat hörn av spelplanen. Spelet byter mellan dessa två lägen i förbestämda interval. I tabellen nedan följer en beskrivning av varje spökes individuella beteende.

	Färg	**Mål i 'chase'**	**Mål i 'scatter'**
Blinky	Röd	Spelarfigurens position	Övre högra hörnet (eller sitt 'chase'-mål om Blinky är arg)
Pinky	Rosa	Två steg framför spelarefiguren, i spelarfigurens nuvarande riktning	Övre vänstra hörnet
Clyde	Orange	Spelarfigurens position om Clyde är mer än n steg bort från spelarfiguren. Annars sitt 'scatter'-mål.	Nedre vänstra hörnet

OBS: Du ska endast implementera kod som plockar fram vilken position som spöket ska sikta på och inte den faktiska planeringen av en väg att gå.

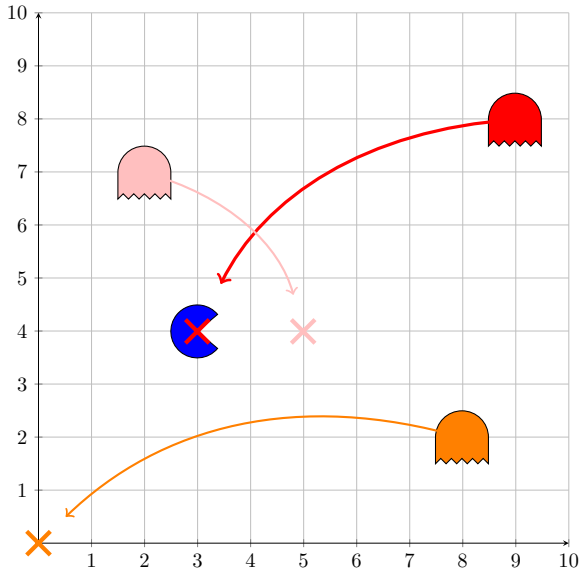

Figure 1: spelarfiguren är på positionen (3,4) och riktad till höger (1,0). Spökena är i 'chase'. Blinky siktar därmed på (3,4). Pinky siktar två steg framför spelarfiguren, på positionen (5,4). Avståndet mellan spelarfiguren och Clyde är mindre än n (vilket i detta fall är 6) och siktar därför på (0,0).

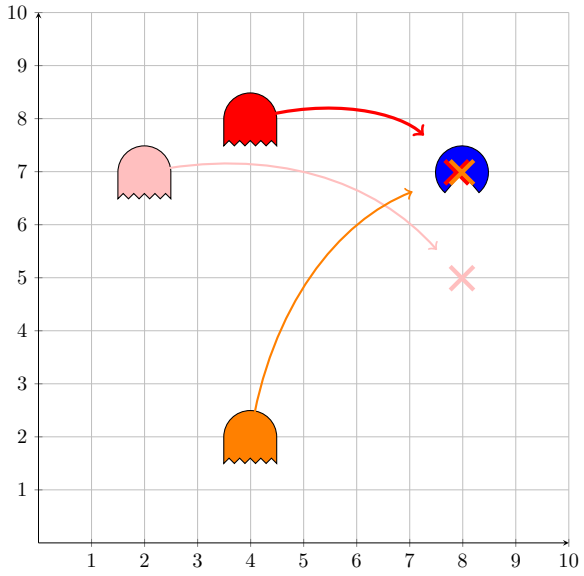

Figure 2: spelarfiguren är på positionen (8,7) och riktad nedåt (0,-1). Spökena är i 'chase'. Blinky siktar därmed på (8,7). Pinky siktar två steg framför spelarfiguren, på positionen (8,5). Avståndet mellan spelarfiguren och Clyde är större än n (vilket i detta fall är 6) och siktar därför på (8,7).