Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2025-10-28

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. Du ska även skriva ett program som testar din implementation.

Krav

Nedan beskrivs de krav på kvalitet och funktion som ställs på din lösning. Du ska också vissa att du skriver kod enligt de krav som ställdes på tidigare laborationer.

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.

Integrerbarhet

Du ska använda kod skriven av dina kollegor för att lösa problem där det är relevant. Den givna koden 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. Dina spök-klasser ska ligga i en `.h` och en `.cc`-fil, separat från övriga filer. Det är lämpligt att alla spöken samsas i samma `.h`-fil och `.cc`-fil.

Testkrav

Programmodulen ska testas genom ett terminalbaserat gränssnitt som ger utdata enligt *Körexempel 1*. 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 tydliga syften. 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`.

Du ska skiva testprogrammet utan kodupprepning och så det blir så tydligt som möjligt. Det är lätt att hamna i fler if-satser än nödvändigt och med så djup indenteringsnivå att läsbarheten försämras.

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. Koden i `given.h/cc` får inte modifieras.

`Point` används dels för att representera positioner på spelplanen och dels 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
```

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 har här storleken `(7,7)` och därmed är övre högra hörnet positionen `(6,6)`. Spelplanen har här minskats för att spara plats, men din implementation ska självklart rita ut spelplanen i full storlek (se `WIDTH` och `HEIGHT` i `given.cc`).

Det som relaterar till ett spöke ritas ut på det första tecknet av en position och visas som den första bokstaven i spökets färg. Versaler används för att markera ett spökes position och en motsvarande gemen markerar punkten den söker mot.

Spelarfiguren ritas ut på det andra tecknet av en position och representeras med ett `@`.

Efter varje kommando användaren matar in ritas en uppdaterad spelplan ut. Användarens indata markeras här med **fet** stil.

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.

Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2025-10-28

Uppgift

Du ska visa att du förstått hur alla inkluderade koncept används och fungerar i C++ genom att implementera programmodulen så att alla krav uppfylls. Koncept som ingår listas nedan.

Tillvägagång

1. Planera ut vilka klasser du kommer behöva och genomför en enkel objektorienterad analys för att lista ut vilket ansvar varje klass kommer ha.
2. Jobba enligt TDD för att implementera en del i taget. Börja med att skriva en liten del av testprogrammet. Implementera efter det koden i klasserna som behövs för att testet ska fungera.
3. Upprepa samma process för att implementera alla delar av uppgiften.
4. Kontrollera att du uppnått varje krav.
5. Redovisa och lämna in.
   Du ska lämna in en headerfil med tillhörande implementationsfil för spök-klasserna, testprogrammet och de givna filerna.
6. Din assistent utvärderar om du uppnått kraven. Brister ger komplettering.

Grafisk Pacman (frivilligt)

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.

Koncept som ingår

Alla listade koncept ingår i uppgiften. Endast listade koncept ingår i uppgiften. D.v.s. problem som inte är lämpliga att lösa med nedanstående koncept 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`)

Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2025-10-28

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 vill röra 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. I det faktiska spelet byts det mellan dessa två lägen i förbestämda interval. I tabellen nedan följer en beskrivning av varje individuellts spökes 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.

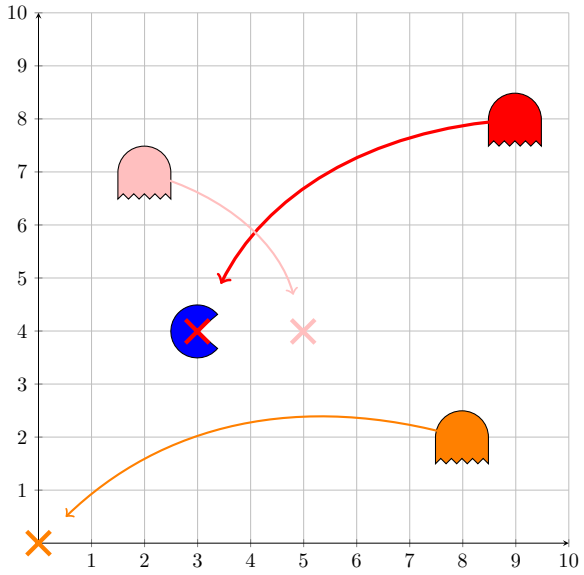

Figure 1: Spelarfiguren är på positionen (3,4) och riktad till höger (1,0). Spökena är i `chase`. Blinky siktar därmed på (3,4). Pinky siktar två steg framför spelarfiguren, på positionen (5,4). Avståndet mellan spelarfiguren och Clyde är mindre än n (vilket i detta fall är 6) och siktar därför på (0,0).

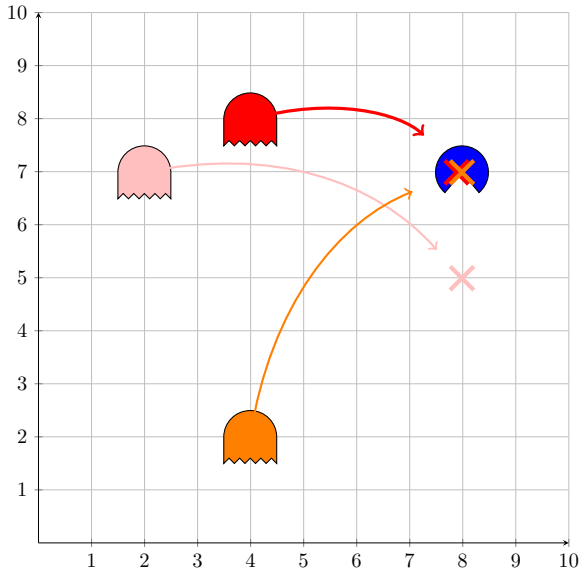

Figure 2: Spelarfiguren är på positionen (8,7) och riktad nedåt (0,-1). Spökena är i `chase`. Blinky siktar därmed på (8,7). Pinky siktar två steg framför spelarfiguren, på positionen (8,5). Avståndet mellan spelarfiguren och Clyde är större än n (vilket i detta fall är 6) och siktar därför på (8,7).