Inlämningsuppgift

Denna uppgift är den obligatoriska inlämningsuppgiften på delkursen PROG3 Programmering i C och C++ HT2017.

Uppgiften ska lösas enskilt eller i grupper om högst tre personer.

Uppgiften presenteras på föreläsning 13 den 2017-11-29.

Lösningen ska lämnas in senast den **2018-01-13** för rättning i samband med kursen. Ett andra inlämningstillfälle kommer att ordnas i samband med omtentan, 2018-03-22.

Inlämning

Inlämningen görs på kursens sida i iLearn2, under rubriken Inlämning.
Det som ska lämnas in ska vara ett ZIP-arkiv innehållande sammanpackade källkodsfiler samt datafiler mm som ingår i lösningen[*], samt en README-fil som text eller PDF, med kortfattad instruktion för spelet och kommentarer, bl.a. ska programmets krav på placering av datafiler (sökvägar) framgå av README-filen. I en inlämningskommentar till inlämningen i iLearn **SKA** gruppmedlemmarnas **namn** och **personnr** stå, samt **vilket betyg** man gör anspråk på.

Om den inlämnade lösningen inte uppfyller kraven för önskat betyg får man betyget Rest och kan komplettera inluppen inom angiven tid. Detta kan upprepas två gånger, om lösningen fortfarande inte uppfyller kraven efter tredje inlämningen blir inluppen underkänd och en ny inlupp ska lösas nästa gång kursen går.

Uppgiften

Uppgiften går ut på att designa och implementera en liten spelmotor[†] för 2D-spel och sedan skapa ett spel (ett tillämpningsprogram) med hjälp av denna spelmotor.
Obs! att det ”viktiga” i uppgiften är den generella spelmotorn, ett API, medan spelet är mest ett testprogram för API:et.

Uppgiften är mycket öppen: du får själv bestämma både designen av spelmotorn och själva spelet. När det gäller designen och implementeringen av spelmotorn är det tänkt att **`cwing`**-exemplet från föreläsning 11-13 kan ge en vägledning, dessutom beskriver jag grovt en möjlig arkitektur nedan.

Beskrivning av tänkt arkitektur

Det är tänkt att spelmotorn består av en klass, säg GameEngine, som implementerar (och döljer från spelprogrammeraren) SDL:s händelseloop, samt av en början till en klasshierarki för "sprite"-klasser. En sprite i ett spelprogram är en tvådimensionell figur som eventuellt kan röra sig över skärmen, av sig själv eller styrd av användaren. Ett Sprite-objekt representeras på skärmen av (åtminstone) en bild och man

[*] Obs alltså: inga projektfiler från era programmeringsmiljöer, DLL:er, SDL-bibliotek osv.
[†] Det har framförts kritik mot att vi kallar det här lilla projektet för en spelmotor eftersom spelmotorer är normalt programsystem av en helt annan magnitud av storlek och komplexitet. Jag hittar dock inget bättre ord för beskrivning av den generella delen av inluppen.

måste kunna rita ut Sprite:ns bild. Dessutom kan det behövas hjälpklasser och -funktioner för att underlätta spelprogrammeringen.

Eventuellt (betyget E) kan Sprite-hierarkin bestå av endast en rot-klass i den generella delen (om du inte hittar lämpliga generella specialiseringar av Sprite-klassen) och överlåta åt den specifika speldelen att skapa lämpliga subklasser.

Hos GameEngine ska man kunna installera sina Sprite-objekt och sedan begära att "händelseloopen" körs igång, varefter Sprite-objekten börjar agera på skärmen.
För betyg högre än E ska man även kunna ange antalet frames per second (FPS) d.v.s. hur många varv i loopen som (högst) ska köras per sekund (det innebär att man eventuellt behöver fördröja loopen om det visar sig att den går för snabbt). Man ska även kunna ta bort Sprite-objekt (som av någon anledning försvunnit ur spelet)[‡].

Till skillnad från föreläsningsexempel ska GameEngine-loopen inte endast vänta på händelser (användarens handlingar som tangentnedtryckningar, musrörelser eller musknapptryckningar) utan ge en takt i vilken Sprite-objekten ska agera. För varje varv i loopen, efter att ha kontrollerat om det finns några användargenererade händelser och i så fall ha tagit hand om dem, ska man gå igenom alla installerade Sprite-objekt och anropa en medlemsfunktion (ofta kallas en sådan medlemsfunktion `tick`) som uppdaterar deras tillstånd (t.ex. ändrar positionen om objektet rör sig o.s.v.). Denna metod bör ta som argument något som ger Sprite-objekten åtkomst till omvärlden, så att de kan interagera med omvärlden (t.ex. samligen av andra Sprite-objekt eller själva GameEngine-objektet). Det är meningen att tillämpningar (de specifika spelen) ska kunna överskugga denna medlemsfunktion i sina egna subklasser till Sprite eller dess subklasser.

I fortsättningen kallar jag loopen för spelloop istf. händelseloop.

I inlämningsuppgiften är det tänkt att spelloopen agerar på följande sätt:
- (för betyget C och högre) om det finns användargenererade händelser så kollas om händelsen är ett kortkommando – i så fall anropas motsvarande funktion
- annars skickas händelsen vidare till Sprite-objekten
- Sprite-objekten gås igenom och deras `tick`-funktion anropas (uppdatering sker)
- Sprite-objekten gås igenom och ritas upp på skärmen (i det uppdaterade tillståndet)
- tiden kontrolleras och eventuell fördröjning framkallas

I sin interaktion med omvärlden bör Sprite-objekt kunna kontrollera om de har kolliderat med något (t.ex. med andra Sprite-objekt) (kollisionsdetektering). Detta kan inte programmeras in i den generella delen (det är alltför spelspecifikt) men det ska förberedas: det ska finnas funktioner (eller medlemsfunktioner hos någon klass) som kan kontrollera om ett Sprite-objekt har kolliderat med ett annat. Hur finmaskig denna kontroll ska göras beror på betygsnivån, för betyg lägre än A räcker det med kontroll av de omgivande rektanglar, för betyget A ska kontrollen göras på pixelnivå.

[‡] Obs alltså att borttagande av Sprites ska vara implementerat i spelmotorn även om denna funktionalitet inte skulle behövas i det specifika spelet där ni testar spelmotorn.

Krav på lösningen och betygskriteria

Det är ett krav att det färdiga programmet ska bestå av en mer generell del (spelmotorn) och en mer specifik del (ditt specifika spel). Spelmotorn ska vara utformad så att man enkelt och utan att behöva modifiera koden för spelmotorn kan skapa andra spel av samma typ.

Nedan följer krav på den generella delen, det finns även krav på det spel som ska skapas för att testa spelmotorn (se längre fram i häftet), men betygsnivåerna gäller endast den generella delen, kraven på spelet måste helt enkelt vara uppfyllda.

För betyget E gäller följande krav:
- programmet ska kodas i C++ och använda grafikbiblioteket SDL (närmare bestämt SDL2)
- objektorienterad programmering ska användas: programmet ska vara uppdelad i klasser med användning av oo-tekniker som inkapsling, arv och polymorfism
- tillämpningsprogrammeraren ska skyddas mot att göra svårupptäckta fel som att använda värdesemantik för objekt av polymorfa klasser
- det ska finnas en gemensam basklass (säg Sprite) för alla figurer, denna basklass ska vara förberedd för att vara en rotklass i en klasshierarki (om tillämpningsprogrammet önskar göra subklasser till den)
- inkapsling: datamedlemmar ska vara privata om inte väldigt speciella skäl föreligger
- det ska inte finnas något minnesläckage, du ska se till att dynamiskt allokerat minne städas bort
- spelmotorn ska kunna ta emot input (tangentbordshändelser, mushändelser) och reagera på dem enligt tillämpningsprogrammets önskemål, eller vidarebefordra dem till tillämpningens objekt
- enkel kollisionsdetektering: man ska kunna kolla om den omgivande rektangeln för en Sprite har kolliderat med den omgivande rektangeln för en annan Sprite
- programmet ska vara kompilerbart och körbart på en dator under både Linux och MS Windows (alltså inga plattformspecifika konstruktioner) med SDL 2 och SDL_ttf, SDL_image och SDL_mixer[§]

För betyget D tillkommer följande krav:
- man ska kunna ange en högsta "frame rate" (antalet frames per sekund)
- det ska finnas olika typer av Sprites i en klasshierarki i den generella delen, det kan t.ex. vara rörliga Sprites (för figurer som flyttar sig själva och där man kan ange hur mycket de ska flytta sig i x- resp. y-led per tick) och orörliga Sprites. Objekt av dessa klasser ska endast kunna skapas dynamiskt och värdesemantik ska vara förbjuden för dem

För betyget C gäller samma krav som för D och även följande krav:
- en av subklasser till Sprite ska kunna vara en animerad figur, representerad internt antingen med flera bilder som växlar efter ett angivet antal tick eller med en sprite sheet
- spelmotorn ska vara förberedd för att tillämpningen vill installera kortkommandon (enkla tangenttryckningar). Tillämpningen ska kunna ange en tangent och en call back-funktion som ska anropas då denna tangent har tryckts ner

För betyget B gäller samma krav som för C och även följande:
- spelmotorn ska vara förberedd för hantering av spelnivåer (*levels*). Tillämpningar ska kunna skapa flera olika "scener" bestående av olika Sprite-objekt och bakgrundsbilder och lätt växla mellan dem
- kortkommandon (tangenttryckningar) ska kunna knytas även till medlemsfunktioner i angivna objekt (förutom till fria funktioner). Det ska vara lätt för tillämpningen att ange antingen en funktionspekare

[§] Om man vill programmera med andra bibliotek kan man få dispens från detta krav men det kräver en överenskommelse med kursansvarige

till en fri funktion eller kombinationen av en objektpekare och en funktionsmedlemspekare tillsammans med tangenten
- det ska finnas en grafisk komponent (Sprite-subklass?) som ska kunna användas som ett inmatningsfält för text, som tillämpningen kan använda för textinmatning (t.ex. namn på spelaren för en high score-lista). Det behöver inte (men får gärna) finnas editeringsmöjligheter för inmatningsfältet.

För betyget A gäller samma krav som för B och dessutom:
- kollisionsdetektering ska göras på pixelnivå, med hänsyn tagen till genomskinliga pixlar
- spelmotorn ska implementera viss funktionalitet hos en fysikmotor, nämligen att rörliga Sprites kan påverkas av en gravitation samt ska kunna studsa mot varandra vid kollision osv.
  Sprites ska ha en viss elasticitet som påverkar deras studs-beteende.

Ett spel

Du ska själv välja vilket spel du vill skapa i din spelmotor. Det ska dock vara av typen arkadspel d.v.s. ett tvådimensionellet spel med rörliga figurer (sprites) av vilka en styrs av användaren. Klassiska spel som SpaceInvaders och PacMan, eller plattformspel som Mario och Sonic är exempel på sådana spel. Här är några länkar som kan ge inspiration:

```
http://en.wikipedia.org/wiki/Space_Invaders
http://en.wikipedia.org/wiki/Breakout_(arcade_game)
http://en.wikipedia.org/wiki/Arkanoid
http://en.wikipedia.org/wiki/Pac_man
http://www.2dplay.com/
```

Spelet skall ha följande egenskaper:

- Spelet ska simulera en värld som innehåller olika typer av objekt. Objekten ska ha olika beteenden och röra sig i världen och agera på olika sätt när de möter andra objekt.
- Det måste finnas minst två olika typer av objekt och det ska finnas flera instanser av minst en av dessa. T.ex ett spelarobjekt och många instanser av fiendeobjekt.
- Ett beteende som måste finnas med är att figurerna ska röra sig över skärmen. Rörelsen kan följa ett mönster och/eller vara slumpmässig.
- Det räcker att grafiken är tvådimensionell (för enkelhetens skull).
- Världen (spelplanen) kan antas vara lika stor som fönstret (du kan göra en större spelplan med panorering, men det blir lite krångligare).
- En figur ska styras av spelaren, med tangentbordet eller med musen. Du kan även göra ett spel där man spelar två stycken genom att dela på tangentbordet (varje spelare använder olika tangenter). Då styr man var sin figur.
- Det ska hända olika saker när objekten möter varandra, de ska påverka varandra på något sätt. T.ex kan ett av objekten tas bort, eller så kan objekten förvandlas på något sätt, eller så kan ett nytt objekt skapas. (Ett exempel på att skapa/ta bort objekt är när man i Space Invaders trycker på skjuta-knappen, t.ex en musknapp, då avfyras ett laserskott och skottet blir då en ny figur som skapas och placeras i världen, på en position vid laserkanonens mynning. Skottet rör sig framåt (uppåt) och om det träffar ett fiendeskepp tas både skottet och skeppet bort, om skottet kommer utanför spelplanen, dvs det missar, tas endast skottet bort.