**PRR02 Slutprojekt**

Du ska genomföra ett slutprojekt för kursen Programmering 2. I slutprojektet får du chansen att visa alla kriterier för kursen på alla nivåer, från E till A.

* **Tanken är att projektet ska vara ett spel, men det måste inte vara ett.** Det får gärna bygga vidare på de spelmotorer vi konstruerat gemensamt i klassrummet.
* **Du kommer bara att bedömas på den kod du skrivit själv.** Om du utgår från något färdigt så bedöms du på den kod du själv lagt till.
* **Spelet behöver inte bli helt färdigt.** Det räcker med en tidig prototyp.
* **Du behöver inte göra grafiken till spelet.** Du kan använda grafik från andra håll eller återanvända sådant du gjort hos Martin/Lena.

Det här är vad du förväntas göra:

* Skapa ett **programmeringsprojekt** som helst innehåller:
  + Ett klassdiagram, med flera hierarkier i flera nivåer, och med motivationer.
  + Klasser och instanser.
  + Inkapsling.
  + Klassmetoder med algoritmer, gärna komplexa.
  + Arv och (för A) polymorfi.
  + Generiska klasser (2+ för högre betyg).
  + Ett tydligt gränssnitt – ge instruktioner och tydlig feedback.
  + Stabilitet – vid minst ett tillfälle finns en risk att programmet kraschar pga användaren (t.ex. att hen ska skriva in ett heltal). Din kod hindrar den kraschen.
  + Objektorienterad kod, där du på ett korrekt sätt använder klasser och instanser för att skapa struktur i din kod.
  + Inga syntaxfel, runtime-fel eller uppenbara logiska fel.
  + Kommentarer i koden.
  + Korrekt namngivning
    - Använd C#-standard.
    - Var konsekvent.
    - Var tydlig.
  + Om du kan få till det på ett naturligt sätt: Någon form av nätverkskommunikation. OBS: Detta är nog den svåraste delen att få in på bra sätt. Det är helt OK att skippa den och komplettera den senare.
* Göra en **presentation** inför klassen eller inför enbart mig. I presentationen behöver du kunna visa:
  + Vad som låg till grund för ditt projekt – någon av våra spelmotorer t.ex.
  + Ditt klassdiagram och eventuella motivationer till din design
  + Din kod, som du förklarar och motiverar. Du förväntas här också svara på frågor om koden.
  + Hur bra spelet fungerar i olika situationer, på olika plattformar.

## **Inlämning i Classroom**

* **Din projektmapp.** En git-länk, helst.
* **Ett klassdiagram.** Välj verktyg och format själv. Exportera som **en bild** och lämna in. Lämna in detta separat från ditt projekt.

Både inlämningen och presentationen är obligatoriska och betygsgrundande.

# **Exempel på projekt**

* Unity
  + Bomberman eller annat rutnäts-baserat spel.
  + Rymdshooter.
  + Plattformsspel.
  + Återskapa en del av ett eller flera gamla NES/SNES-spel.
  + En wave shooter i VR eller AR.
  + Återskapa ett brädspel (Fia med knuff, monopol, Mr. Jack Pocket?).
  + Grunden till ett turbaserat strategispel, där förflyttning “kostar” poäng per ruta…
* Konsollen
  + Monstertränarspel a’ pokemon men med något helt annat istället för monster – hockeyspelare, robotar, barn?
  + Ett textbaserat utforskningsspel – kanske går man runt i en dungeon, eller en drömvärld, eller ett kontorslandskap. Klasserna kan vara rum, prylar, folk man träffar på, dialogval… För extra skoj: få det att funka i nätverk så flera kan spela samtidigt.
  + Ett textbaserat, turbaserat strategispel där man ska regera ett kungarike (eller ett megaföretag) – hur ska pengar fördelas, skatter, etc?
  + Om man vill testa att göra något grafiskt som inte är Unity så kan man ju alltid kolla in [SFML](https://sites.google.com/view/csharp-referens/sfml) och göra något med det. Då måste man göra mer själv!
* Inte spel
  + Ett backend-system med användare, kontotyper etc. Kanske med koppling till en databas?
  + En databas för samlare av något. Systemet har användare med lösenord, och använder XML (via serialisering) för lagring. Vad samlar folk på? Frimärken, modelltåg, böcker, tv-spel, hårtofsar från sina offer?

Jag har också samlat en lista [med några gamla enkla spel](https://docs.google.com/document/d/1-nhq1AX72ZtBMU7cedVJpAodfVsJc6UMRqbS6FMK_2I/edit?usp=sharing) som man skulle kunna försöka återskapa – antingen i Unity eller i SFML. Kom ihåg att jobba objektorienterat i så fall!

# **Bedömning**

[Skolverkets kursplan för Programmering 2](https://www.skolverket.se/undervisning/gymnasieskolan/laroplan-program-och-amnen-i-gymnasieskolan/gymnasieprogrammen/amne?url=1530314731%2Fsyllabuscw%2Fjsp%2Fsubject.htm%3FsubjectCode%3DPRR%26tos%3Dgy&sv.url=12.5dfee44715d35a5cdfa92a3)

**VIKTIGT:** Att uppfylla samtliga punkter för ett kriterie och betyg är i princip en garanti för att få det betyget i det kriteriet. Om någon enstaka punkt saknas så kommer bedömningen att bero på hur resten av koden ser ut och hur väl övriga punkter är uppfyllda. Fråga gärna. Du kommer att få möjlighet att komplettera ifall något viktigt saknas.

**Eleven formulerar och planerar (lösningar till) programmeringsuppgifter. Eleven analyserar och modellerar uppgifter. Eleven designar utifrån den analyserade uppgiften en lösning och dokumenterar sin lösning med ett klassdiagram.**

|  |  |
| --- | --- |
| **E** – i samråd med handledare  enklare uppgifter  designar med viss säkerhet  en enkel lösning  enkelt klassdiagram | * Utifrån uppgiften och i samråd med mig kommer du på vilka klasser du ska ha. * Du skapar ett klassdiagram som har:   + 2 eller fler klasser i ett korrekt klassdiagram   + Enkelt arv, variabler och metoder   + Datatyp för variabler och metoder   + Metoder med parametrar |
| **C** – efter samråd  uppgifter  designar med viss säkerhet  en lösning  klassdiagram | * Du kommer fram till vilka klasser du ska ha *efter* samråd med mig. * Klassdiagrammet har:   + Synlighet   + Arv i 2+ steg *eller* två hierarkier. |
| **A** – efter samråd  komplexa uppgifter  designar med säkerhet  en komplex lösning  avancerat klassdiagram | * Klassdiagrammet har:   + Polymorfism (virtual/abstract, override eller interfaces)   + Arv i 2+ steg *och* två olika hierarkier. |

**Eleven väljer ett uttryckssätt och ett gränssnitt som är anpassat för att interagera med den avsedda användaren.**

|  |  |
| --- | --- |
| **E** – Tillfredsställande | * Programmet berättar för användaren vad hen förväntas göra. |
| **C** – Tillfredsställande | * Varje steg i programmet är tydligt för användaren. Hen behöver inte “gissa sig fram”. * Programmet berättar på något ställe för användaren vad hen gjort för fel. |
| **A** – Gott | * Programmet är alltid tydligt både i instruktioner och berättar flera gånger när användare gör fel. Det måste inte ske i textform, så länge användaren förstår vad som är fel. |

**Eleven skapar någon lösning där programmet kommunicerar över internet.**

|  |  |
| --- | --- |
| **E** – enkel lösning | * Du använder någon form av nätverkskommunikation. Till exempel:   + API-anrop (REST eller annat)   + Socket-kommunikation   + Unitys nätverks-komponent (eller Mirror) * Programmet är litet och enkelt, och gör i princip bara ovanstående. Det har någon enstaka klass, någon enstaka interaktion med användaren, någon enstaka algoritm. |
| **C** – någon lösning | * Som ovanstående, men programmet är mer komplett. Flera klasser, flera interaktioner med användaren, flera algoritmer. |
| **A** – genomarbetad lösning | * Som ovanstående, men programmet är mer omfattande och relativt komplett. Se övriga betygskriterier för att få ungefärlig koll på hur komplexa program på A-nivå är. |

**Eleven skapar enklare program med klasser där arv används**

|  |  |
| --- | --- |
| **E** – skapar med viss säkerhet, arv i begränsad utsträckning | * Koden innehåller klasser med arv |
| **C** – skapar med viss säkerhet, arv | * Koden innehåller klasser med arv i 2+ steg *eller* två hierarkier |
| **A** – skapar med säkerhet, omfattande arv | * Koden innehåller flera separata egna klasshierarkier med arv i flera led |

**I sin programmering skapar eleven [...] kommenterad källkod**

|  |  |
| --- | --- |
| **E** – enkelt kommenterad | * Enstaka, korta, korrekta kommentarer |
| **C** – noggrant kommenterad | * All kod förklaras; kanske inte varje rad men varje block |
| **A** – noggrant och utförligt kommenterad | * Kommentarerna förklarar inte bara funktion utan fokuserar på syftet med koden |

**I sin programmering skapar eleven med konsekvent kodningsstil och tydlig namngivning korrekt, strukturerad [...] källkod.**

|  |  |
| --- | --- |
| **C** – tillfredsställande resultat | * Tydlig namngivning * Följer C# standard för namngivning * Abstraktion: varje klass sköter sin egen funktionalitet * Ganska lite kod i Main. |
| **A** – gott resultat | * Tydlig struktur: varje metod gör en sak |

**Eleven använder generiska klasser.**

|  |  |
| --- | --- |
| **E** – Någon, med viss säkerhet | * 1 generisk klass används, t.ex. List, Queue, Dictionary. |
| **C** – Några, med viss säkerhet | * 2 olika generiska klasser används. |
| **A** – Flera, med säkerhet | * 3 olika generiska klasser används, *eller* så har du skapat en egen generisk klass som används i ditt program. |

**Produkten innehåller objektorienterad programmering som är stabil och robust.**

|  |  |
| --- | --- |
| **E** – produkten är av tillfredsställande kvalitet  enkelt program | * Programmet förhindrar vid något tillfälle användaren från att krascha programmet genom att skriva in något felaktigt. * Minst en av programmets klasser innehåller en metod. * Minst en av klassernas metoder innehåller en algoritm, inte bara t.ex. enkel tilldelning. * Grundläggande, enkel funktionalitet – programmet fungerar, men är väldigt begränsat. * Välformulerade klasser används för att lagra information * Klasser och instanser används korrekt – klasser beskriver kategorier, instanser är specifika objekt som tillhör de kategorierna (t.ex. klassen Car, och varje bil i programmet är en egen instans av klassen). |
| **C** – produkten är av tillfredsställande kvalitet  program | * Programmet förhindrar alltid användaren från att krascha programmet genom att skriva in något felaktigt. * Try-catch eller tryparse används för att hindra en krasch baserad på användar-input. * Utbyte av information mellan instanser av en klass eller av olika klasser (t.ex: en metod i en instans anropar en metod i en annan instans och stoppar in någon information som parameter) * Inkapsling används genomgående. * Mer omfattande funktionalitet – programmet är relativt litet men fungerande och komplett. Det innehåller någon lite mer komplex struktur. * En stor del av programmets logik befinner sig i icke-statiska metoder inuti klasser, alltså utanför Program.cs och Main. |
| **A** – produkten är av god kvalitet  komplext program | * Komplex funktionalitet – programmet är relativt stort och har flera olika delar eller innehåller flera komplexa algoritmer och strukturer. * I princip all logik befinner sig i klassmetoder, alltså utanför Program.cs och Main. En övervägande del ligger i icke-statiska metoder. |

**Eleven anpassar sin planering av programutvecklingsprocessen**

|  |  |
| --- | --- |
| **E/C** – viss säkerhet | * Du lämnar in i tid * Du får hjälp att avgränsa dig för att hinna bli klar och kunna lämna in |
| **A** – komplexa stabila program | * Du gör själv en avgränsning för att kunna lämna in i tid |

**Eleven utför felsökning**

|  |  |
| --- | --- |
| **E** – enkla syntaxfel | * Inga röda squiggleylines. |
| **C** – syntaxfel, körtidsfel, logiska fel | * Inga körtidsfel – stabilitet. * Inga uppenbara logiska fel. |

**Eleven anpassar polymorfi**

|  |  |
| --- | --- |
| **A** – polymorfi | * Objektorienterad polymorfi till exempel via abstrakta klasser, virtual/override och/eller interface. * Polymorfin används praktiskt – t.ex. genom att instanser av en subklass lagras i en variabel med basklassen eller ett interface som datatyp. |

**Innan programutvecklingen avslutas utvärderar eleven programmets prestanda och ändamålsenlighet.**

|  |  |
| --- | --- |
| **E** – enkla omdömen, någon situation, något sammanhang | * Berätta om hur spelet/programmet fungerar på en plattform och ett sammanhang. |
| **C** – nyanserade omdömen, några situationer och sammanhang | * Berätta om positiva & negativa sidor hos någon lösning. * Resonera kring hur spelet/programmet skulle fungera på andra plattformar och sammanhang. |
| **A** – Ger förslag på för- bättringar, flera situationer och sammanhang | * Beskriv vilka anpassningar som skulle behöva göras för att spelet/programmet skulle fungera bra på andra plattformar och sammanhang. * ELLER beskriv hur något skulle kunna ändras för att göra koden mer effektiv. |

**Eleven kommunicerar om programutvecklingen och dess utvärdering och använder då datavetenskapliga begrepp.**

|  |  |
| --- | --- |
| **E** – med viss säkerhet | * Använder något enstaka begrepp, någorlunda korrekt. |
| **C** – med viss säkerhet | * Använder flera begrepp, mestadels korrekt. |
| **A** – med säkerhet | * Använder flera begrepp, och alla korrekt. |