OOP - Asteroids

Examinationsform: projektarbete i grupp - skriftlig inlämning

**OBS**: läs igenom hela dokumentet noggrant - det innehåller mycket information om uppgiften.

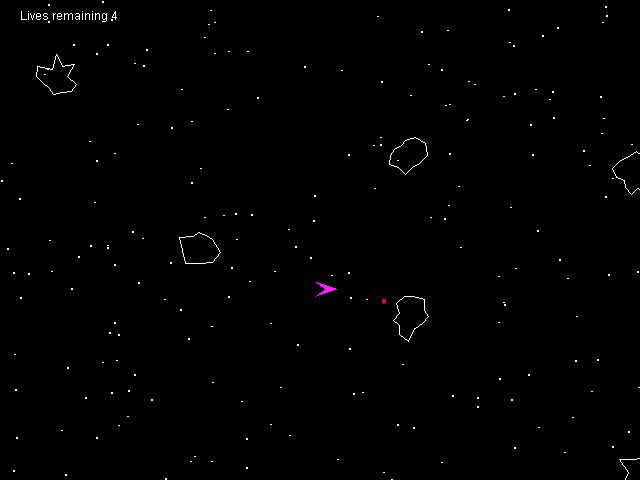
### Changelog

Här kommer jag att dokumentera tydligt alla förändringar som görs till dokumentet:

* 2017-10-30 - första versionen

# Introduktion

Asteroids är en klassisk arkadspel från 1979. Du styr ett rymdskepp - målet är att undvika alla asteroider som flyger omkring samtidigt som du försöker skjuta sönder de.



För lite mer historik om spelet kolla på den här Youtube videon: <https://youtu.be/w60sfReTsRA>

Ni kan också testa att spela själv (med Flash tyvärr) i browsern: <http://www.freeasteroids.org>

Mål

Att arbeta i grupper av 2-3 personer och använda objektorienterad programmering för att bygga en alldeles egen Asteroids spel i Python! Samt dokumentera hur det gick.

**Två leveranser:**

* Källkod för spelet som ZIP fil
* En kort (cirka 1-2 A4) rapport som reflektera över det ni har lärt er:
  + Beskriv hur ni arbetade som grupp
  + Vad fungerade bra, dåligt, hur skulle ni kunna jobba bättre nästa gången?
  + Var Python rätt språk för att bygga ett Asteroids spel?
    - På vilket sätt var Python bra?
    - På vilket sätt var Python dålig?

Det här projektet har ett lågt minimikrav -- men ingen högre gräns. Ni får bygga vidare så mycket ni vill, göra spelet till vad ni vill: bara ni håller er till OOP principer.

### Startpaket

Projektet börjar med följande:

* Det här dokumentet med steg för steg beskrivning av designen och hur man kommer igång samt vilka objekt ska finnas och hur de sitter ihop.
* En ZIP fil med ett PyCharm projekt med färdig stomme för projektet - dvs, ni behöver inte börja från grunden med tom projekt. Jag står för mycket av det komplexa i hur man arbeta med PyGame, hur man ritar på skärmen och lite till. Ni står för OOP implementationen av alla objekt i spelet.
* PyGame - ett fantastiskt Python modul som underlättar att bygga spel.

För att installera PyGame så behöver du följa instruktionerna från den här sidan: <https://www.pygame.org/wiki/GettingStarted>

# Komma igång

Om du inte har redan - kolla på videon på första sidan, och testa spelet själv. Ni kommer att implementera de följande fyra klasser:

* Ship
* Asteroid
* Star
* Bullet

Börja med att öppna upp mitt projekt och titta runt i koden. Det finns en del kod som är kanske svårt att förstå - särskilt allt som rör PyGame - men den koden behöver ni inte kunna skriva själva.

För att få till ett grafiskt spel behöver vi några metoder för att måla upp saker på skärmen. Spelet använder PyGame för det - och det är min kod som styr PyGame. Min kod öppnar ett grafiskt fönster och ger alla kommandon för att rita streck på skärmen. Det blir din kod som styr själva logiken så att alla streck upplevs som ett roligt spel.

### Installera PyGame

Följ instruktioner här: <https://www.pygame.org/wiki/GettingStarted>

Vi kommer också gå igenom det tillsammans under lektionstid.

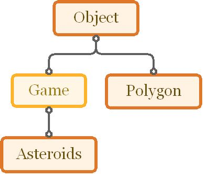
### Att skriva spel

Lite bakgrund till spelutveckling som kanske hjälper under projektet. Alla grafiska datorspel har några saker gemensamt som jag ska försöka förklara här med samma ord som PyGame använder.

* **Window**: vi behöver ha ett fönster där spelet existerar
* **Screen**: i fönstret har vi något som kallas för en screen i PyGame - en yta där vi kan rita precis vad vi vill med hjälp av x och y koordinater samt färg (men just nu är allting byggt för att vara svartvit)
* **Input**: spelaren interagerar med spelet med hjälp av tangentbord - alla tryckningar kommer till oss som input tack vare PyGame
* **Update**: logiken för hur vi uppdaterar spelet ett steg - dvs, hur vi ska flytta på alla objekt och göra de ändringar och koll som vi behöver göra för att få till spelets beteende
* **Draw**: koden som ritar upp alla spelobjekt på screen.
* **Game loop**: en evig loop som körs om och om igen för att sköta spelet. Det är den här loopen som:
  + kollar *input*,
  + kör *update*
  + och till sist anropar *draw*

Det låter kanske komplicerad - men jag har redan skrivit koden för det här i projektet - ni kan fokusera på OOP designen nedan istället.

# OOP Design

Vi kommer att ha två typer av klasser i spelet: klasser som sköter spel logiken, samt klasser som representerar alla typer av objekt (skeppet, asteroider, kulor, mm) som kan finnas i spelet.

Så vår design kanske börjar enligt bilden till höger. Gula klasser är abstrakta, oranga är konkreta. Alla objekt i Python ärver från en basklass som heter *object*. Vi har ett generiskt Game klas som sköter typisk spellogik (för vi kanske kommer att skriva fler spel framöver?), och en specifik klass Asteroids som ärver från den för att implementera all logik som är specifik till just Asteroids spelet.

### Polygon

Polygon klassen representerar alla linjebaserad objekt som syns i spelet - eftersom spelet är så enkelt och “vektorbaserad” så kan en polygon representera de flesta visuella komponenter (trekant, fyrkant mm).

Polygons konstruktor tar en lista, *points*, och en *position*. Position är en objekt av typ Point. Listan *points* är en lista av objekt av typ Point. Points representerar alla koordinater för alla linjer som ska ritas upp för polygonen.

Polygon har en metod - draw() - som tar hand om logiken för att rita den på skärmen. Den ser till att alla linjer definierat i *points* ritas upp på korrekt positionen på skärmen (med hjälp av *position*).

### Ship

Det första pjäs eller “character” i spelet är själva skeppet. Skeppet kommer vi att rita upp på skärmen med hjälp av en massa streck - så den den kan ärva från Polygon. Polygon klassen (som jag har skrivit) vet hur den anropar till PyGame för att rita upp alla streck som tillhör polygonen. Detta betyder att Ship klassen behöver endast bestämma vilka streck som tillhör skeppet - Polygon klassen kommer att ta hand om resten.

### Asteroid

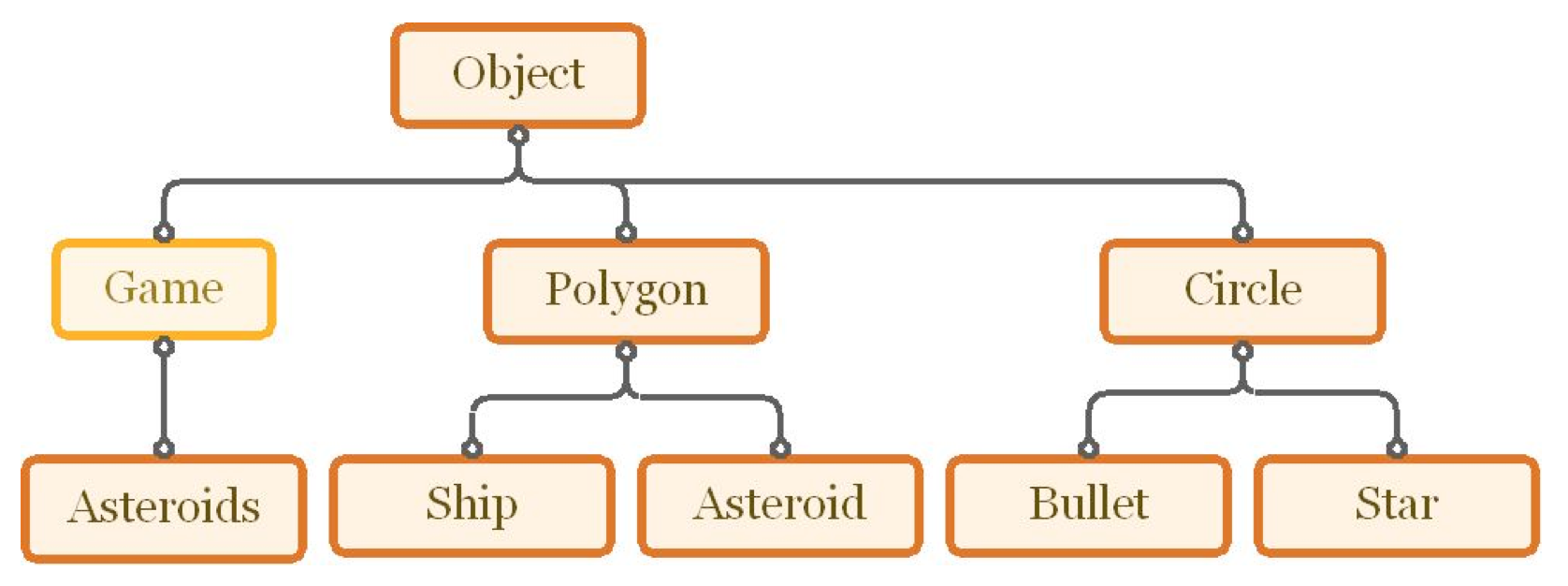
Nästa pjäs är en asteroid. Den är ganska likt skeppet - inte på utseendet förstås men ur ett OOP perspektiv. En asteroid är också en polygon. Den kan också flytta sig runt på skärmen. Den kommer också kunna “dö”, osv. Så vi behöver ytterligare en klass för den: Asteroid, ärver också från Polygon.

### Circle

Två saker kvar: vi vill kunna skjuta kulor, och spelet komma kännas lite tomt utan stjärnor i bakgrunden. Vad har stjärnor och kulor gemensamt? De både två ser ut som små cirklar på skärmen. Så vi har en ny basklass för objekt som ser ut som en cirkel: Circle.

### Bullet och Star

Nu kan vi implementera de sista två pjäserna i spelet: Bullet och Star, men hjälp av Circle. Båda Bullet och Star borde ärva från Circle - men de kommer att fungera lite olika. T.ex en Star är förmodligen statiskt: den flyttar aldrig på sig. En Bullet bör flytta på sig - den har en *velocity*. En Bullet ska förmodligen inte leva för evigt heller - i Asteroids så försvinner den efter några sekunder.



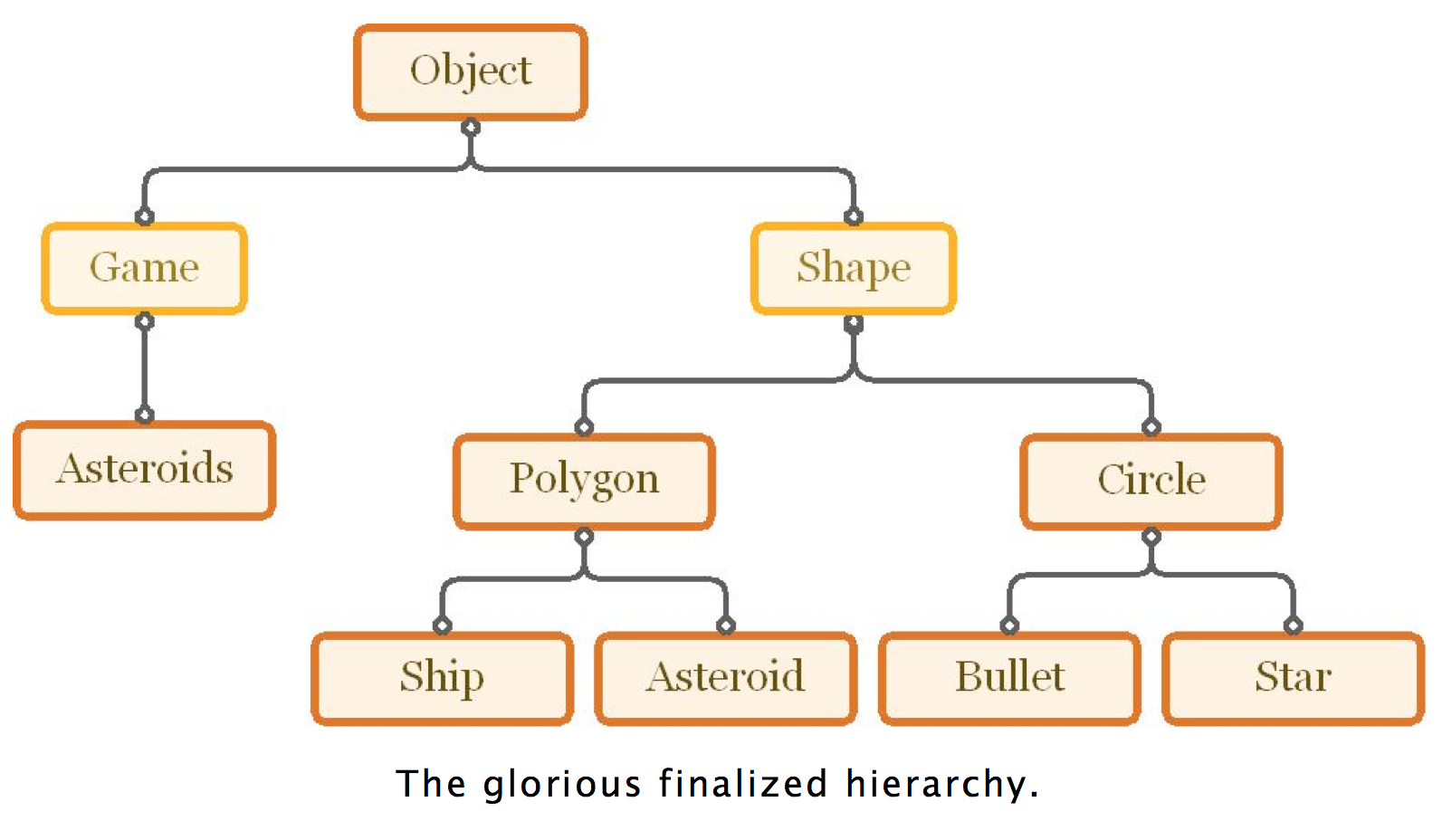
### One last thing…

Circle och Polygon har några saker gemensamt - de både två representerar grafiska objekt på skärmen, som kan röra på sig, och lite till. Det låter som att det som är gemensamt bör flyttas upp till en ny basklass för de: Shape.

Shape kan även hantera att alla pjäs kan röra på sig, samt snurra. För att röra på sig finns ett medlem som heter *pull* - en dragningskraft som gör att positionen ändras varje steg i spelet. Det finns även en metod *accelerate*() som gör att man kan putta på och ge fart till en Shape - alternativt bromsa ner den.

För att snurra på en shape finns ett medlem som heter *rotation* - som styr åt vilken håll en Shape pekar. För att snurra på en Shape så kan man anropa metoden *rotate()* med antal grader som man vill rotera Shapen. Precis som pull ändrar positionen varje steg, så finns det *angular\_velocity* som ändrar rotation varje steg.

Det låter komplicerat - men det är lugnt, jag har redan implementerat all matematik bakom detta. Ni behöver bara anropa allting på rätt sätt.



# Instruktioner

Konkret då - vad ska ni göra? Och i vilken ordning? Vart ska ni börja? Den följande listan ska ses som ett förslag - vill ni köra helt på egen hand och hitta på så går det också bra. Men om ni upplever att det är svårt att komma igång, prova att följa mina instruktioner.

Ni ha redan klasser för Game, Asteroids, Shape, Polygon och Circle. Ni behöver då implementera Ship, Asteroid, Bullet och Star.

1. Skapa klassen **Ship** - den ska ärva från Polygon och ha några punkter (utseendet få ni bestämma över själv om ni vill). Vill ni använda samma skeppdesign som min, så hade jag följande koordinater:
   1. [ Point(0,0), Point(-10,10), Point(15,0), Point(-10,-10) ]
   2. Men det är fritt fram att “designa” eget skepp
2. Uppdatera koden i Asteroids klassen så att den skapar en objekt instans av din Ship i konstruktorn. Resten av Asteroids klassen är redan förbered för att använda den.
3. Testa koden - nu ska du ha en skepp som kan dessutom flyga omkring med hjälp av pilarna på tangentbordet.
4. Implementera **Asteroid** klassen - lite likt Ship men med en annan utseende. Kanske ska alla se likadant ut? Eller så vill du att de ser lite olika - kanske vill du använda Pythons *random* module. Det är upp till er!
5. Uppdatera konstruktorn i Asteroids klassen för att skapa några Asteroid objekt.
6. Testa all kod - nu borde vi ha några asteroider på skärmen. Men de gör inte så mycket! De ska röra på sig.
7. I Shape basklassen finns två attribut - *pull* och *angular\_velocity*. Asteroider bör sätta dessa till något - *pull* är en Point som säger hur mycket (x,y) asteroiden ska flytta på sig varje steg. *angular\_velocity* säger hur mycket den ska rotera varje steg (i grader).
   1. *pull* kan sättas direkt, men den kan också sättas via metoden *accelerate*() som definieras i basklassen Shape.
8. Testa all kod - nu borde de snurra runt!
9. Nu är det dags att tänka lite på atmosfären i spelet - vi behöver stjärnor! Implementera en klass **Star** som ärver från Circle.
10. Uppdatera konstruktorn i Asteroids så att den skapar ett antal Stars.
11. Testkör - vilken förbättring!

**Paus:** Så här långt har vi lite grundelement till spelet - vi kan flytta vår skepp, och vi har en massa asteroider som flyger omkring och en fin bakgrund. Men vi kan inte spela. Nu är det dags att göra det till ett spel!

1. Vad är ett spel om man inte kan dö? Nu ska vi implementera logiken för att dö. I Asteroids klassen, längst ner, hittar ni metoden *handle\_collisions*()
   1. Den bör kolla om vår skepp har krockat med en asteroid - gå igenom listan över alla asteroider och använd Polygon klassens collide() metod för att testa om två objekt som är polymorfiskt Polygons krockar.
   2. Om det blir krock - ja då ska man dö.
2. Och till sist - om vi kan dö så är det bara rättvist om vi kan slå tillbaka. Dags att implementera klassen **Bullet** som också ärver från Circle. Bullet är lite annorlunda än Star - till att börja med så rör den på sig. Men en Bullet också har en begränsat livslängd. Den kanske bara åker så långt, eller kan leva i så många iterationer av loopen, eller så många sekunder. Oavsett så måste den tas bort någon gång.
   1. Skapa själva Bullet klassen - den ska börja med en acceleration - något du kan sätta via accelerate() metoden som även Bullet har tillgång till tack vare att den ärver från Circle som i sin tur ärver från Shape.
   2. Sedan ska vi skapa bullets varje gång spelaren trycker på mellanslag - detta kan man göra i Asteroids klassen som sköter spellogiken (se *handle\_input*() metoden).
3. När en bullet träffar en asteroid så ska asteroiden (och bulleten) försvinner ur spelet - detta göras i *handle\_collisions*() metoden i Asteroids klassen
4. Om en bullet inte träffar en asteroid så ska den tas bort någon gång - dvs den få inte finnas för evigt. Det skulle man kunna lösa i update\_simulation() metoden i Asteroids klassen. Men när är det dags att ta bort en bullet? Det är upp till er att bestämma exakt hur - men kom ihåg att tillämpa OOP!

När ni har kommit så här långt så har ni en enkel Asteroids spel som går att spela. Grattis - klart godkänt!

# Nästa steg

Om ni har tid över - och lust - passa på att göra spelet ännu bättre så kanske det blir lite mer än bara “godkänt”!

Håll er till OOP principer, men spelmässigt är vad som helst tillåtit. Några förslag på nästa steg:

* Det är lite tråkigt att bara ha ett liv - det vore roligare om man hade 3 eller 4 liv!
* Vad är ett spel utan poäng? Poänglöst.
* När jag dör skulle jag helst vilja kunna börja om utan att behöva starta om
* Om vi ändå har poäng och kan börja om - kanske en high score funktion?
* Levels - när man har lyckats skjuta ner alla asteroider.. Vad händer sen? Kanske blir det en ny bana, med nya asteroider, som är kanske svårare på något sätt?
* Originalspelet hade en sköld som man kunde använda då och då för att skydda sig från asteroider (och som sedan tog några sekunder att ladda om)
* Snyggare grafik säljer alltid mer - det går att göra animationer när saker exploderar eller bättre grafik och effekter i PyCharm
* Originalet hade även en liten UFO som dök upp då och då och var generellt taskigt mot spelaren
* Originalet hade även en teleport funktion - tryck på rätt knapp och man direkt försvann bara för att dyka upp någon annanstans på skärmen - kanske direkt framför en asteroid!
* Alla dessa knappar och funktioner - kanske vill man ha en liten hjälp instruktion innan man börjar spela?
* Någon gång efter 1979 kom färg till datorspel…
* Ljud! Om vi bara hade något bra sätt att skapa roliga ljudeffekter (se inlupp 2 när den kommer)
* I ZIP filen hittar ni en liten video som visar upp andra förslag på förbättringar

Om ni vill lära er lite med om hur PyGame fungerar, så finns det mycket dokumentation här: <https://www.pygame.org/docs/>