GÖTEBORGS UNIVERSITET

Programmering i matematikundervisningen

En litteraturstudie

Charlie Wolfbrandt

Ämneslärarprogrammet
med inriktning mot arbete
i gymnasieskolan

Uppsats/Examensarbete: 15 hp
Kurs: LGMA2G
Nivå: Grundläggande nivå
Termin/år: HT 2018
Handledare: Jakob Björnberg
Examinator: Jan Stevens
Kod: HT18-3001-003-LGMA2G

Nyckelord: Matematik, matematikundervisning, programmering, programmering inom matematik, speldesign.

Abstract

Syftet med denna litteraturstudie är att undersöka hur programmering passar in i matematikundervisningen. Vi tittar på hur programmering har använts som ett inlärningsverktyg för matematik tidigare och vilka matematiska förmågor som påverkas mest av att eleverna får arbeta med programmering istället för traditionell undervisning. Vi tittar på programmering inom speldesign, som ett verktyg för problemlösning och som en möjlighet att utforska matematik.

1 Förord

Jag vill rikta ett stort tack till min handledare Jakob som varit till stor hjälp under arbetets gång och som hela tiden har kommit med konstruktiv feedback.

2 Innehållsförteckning

3 Inledning

I dagens samhälle blir det allt viktigare att ha digital kompetens. Tekniken utvecklas hela tiden och det förändrar samhället på ett sådant sätt att vi som medborgare måste följa med i utvecklingen. Naturligtvis spelar skolan en viktig roll i detta och användningen av digitala verktyg letar sig in mer och mer i styrdokumenten. Den senaste förändringen innebär bland annat att programmering har införts i matematikundervisningen i både grund- och gymnasieskolan.

Detta väcker både frågor och problem. Många lärare saknar kompetens att undervisa i programmering. Mina erfarenheter från VFU-perioder är dessutom att elever ofta tycker det är svårt att arbeta med digitala verktyg i allmänhet, och det går absolut att argumentera för att programmering är ännu svårare än de digitala verktyg som redan används.

Frågor som väcks och som vi ska undersöka i denna text är:
1) Var i matematikundervisningen passar programmeringen in?
2) Hur ska man använda sig av programmering?

I dagsläget ligger programmering under problemlösning i gymnasiet, men i grundskolan ligger det även under algebra. Jag ska försöka ta reda på vad forskningen säger om detta, passar programmeringen bättre in på det ena stället än det andra? Är det bästa att ha med det i så många delar som möjligt, eller ska man försöka hålla sig till ett område där det gör mest (positiv) skillnad? Finns det något sådant område?

Vilket är sen det bästa sättet att använda programmering i undervisningen? Jag kommer att kolla på olika varianter som spelutveckling, problemlösningsverktyg etc. och försöka komma fram till hur programmeringen kan göra som mest nytta i undervisningen.

Jag börjar med att blicka tillbaks och försöker reda ut hur det kommer sig att vi idag har programmering i matematikundervisningen. 2017 beslutade regeringen alltså just detta, men hur hamnade vi där?

Redan 2006 kom Europaparlamentet och Europeiska unionens råd med en rekommendation där digital kompetens angavs som en nyckelkompetens för livslångt lärande. Digital kompetens sågs som viktigt för bl.a. personlig utveckling, aktivt medborgarskap, anställning, innovation, produktivitet och för en mer flexibel arbetskraft som kan anpassa sig efter en föränderlig värld. Senare förde även OECD fram den allt större betydelse som it har för skolan. Skolan har i uppdrag att förbereda eleverna för ett allt mer teknikorienterat arbets- och samhällsliv. Därför måste skolan ge eleverna god digital kompetens och det är viktigt att skolans styrdokument innefattar de kunskaper och förmågor som efterfrågas vid fortsatt utbildning, på arbetsmarknaden och som krävs för ett aktivt deltagande i samhällslivet (Utbildningsdepartementet, 2015).

2008 fick sedan Skolverket i uppdrag av Regeringen att främja användningen av informations- och kommunikationsteknik (IKT). Detta omfattar bland annat att sprida kunskap om användandet och utformningen av IKT i lärprocesser, digitala verktyg och

framtagandet av lärande exempel inom området. I uppdraget ingår även att främja utvecklingen av verksamheternas kommunikation med elever och föräldrar genom IKT och att kontinuerligt följa upp elevers och lärares it-användning och it-kompetens (Utbildningsdepartementet, 2015).

Hösten 2011 beslutade regeringen om en bred och sammanhållen strategi för it-politiken, It i människans tjänst – en digital agenda för Sverige. Regeringens ambitioner inom området presenterades tillsammans med förslag på insatser och åtgärder för att nå målet. Sommaren 2012 tillsatte regeringen en kommitté, Digitaliseringskommisionen, med uppgiften att verka för att målet uppnås och att ambitionerna fullföljs (SOU 2014:13).

Våren 2014 lämnade Digitaliseringskommisionen delbetänkande En digital agenda i människans tjänst – en ljusnande framtid kan bli vår (SOU 2014:13) till regeringen. Där presenterades förslag som syftar till att öka de digitala inslagen i den svenska undervisningen (Utbildningsdepartementet, 2015).

Efter detta fick sedan Skolverket sommaren 2015 i uppdrag att ta fram och genomföra nationella skolutvecklingsprogram. I uppdraget ingår att ta fram förslag till nödvändig kompetensutveckling samt för effektivare administration (Utbildningsdepartementet, 2015).

Den 9 mars 2017 meddelade regeringen i ett pressmeddelande att de beslutat om förtydliganden och förstärkningar i läro-, kurs- och ämnesplaner för grund- och gymnasieskolan. Syftet är att tydliggöra skolans uppdrag att stärka elevernas digitala kompetens.

- Alla elever ska få med sig kunskaper att förstå och kunna påverka världen. Det får inte vara beroende av vilken skola du går på eller vilken lärare du har. Men avsaknaden av ett samlat nationellt ansvar för digitaliseringen av skolan har gjort att skolan vidgat digitala klyftor istället för att sluta dem, sa utbildningsminister Gustav Fridolin.

- Vi ändrar i gymnasieskolans ämnesplaner så att programmering läggs till i flera matematikkurser, sa gymnasie- och kunskapslyftsminister Anna Ekström.

- Skolan måste bli bättre på att ge eleverna förutsättningar för att fungera som medborgare i en tid när digitaliseringen förändrar samhället, arbetsmarknaden och våra sätt att leva och vara, sa bostads- och digitaliseringsminister Peter Eriksson.

Ändringarna avser bland annat att programmering införs som ett tydligt inslag i flera olika ämnen i grundskolan och att eleverna ska kunna lösa problem och omsätta idéer i handling på ett kreativt sätt med användning av digital teknik (Regeringen, 2017).

4 Material och metod

Jag inledde min litteraturstudie genom att söka efter artiklar som innehöll kombinationer av matematik, programmering och undervisning (på svenska och engelska). Då det inte finns så mycket forskning om just programmering i matematikundervisningen så har jag egentligen inte gjort några avgränsningar alls. De flesta texter som studerats i detta arbete handlar om barn som är yngre än elever på gymnasienivå och vissa texter handlar om digitala verktyg i allmänhet snarare än programmering, men jag har gjort vad jag har kunnat för att ändå koppla litteraturen till programmering i matematikundervisningen i gymnasiet. Eftersom litteratur anpassat till gymnasienivå saknas så har jag lagt till ett kapitel i slutet av arbetet där jag ger mina egna tankar och förslag på hur undervisningen skulle kunna se ut i gymnasiet.

5 Teoretiska ramverk

Vi ska titta på två teoretiska ramverk som det ofta skrivs om i samband med forskning om digitala verktyg och matematik. Även om forskningen handlar mer om digitala verktyg i allmänhet än om just programmering så är detta så nära vi kommer, och det finns ingen anledning att tro att ramverken inte kan appliceras på samma sätt när det kommer till programmering. Teoretiska ramverk fungerar som ett stöd för undervisningen. Forskare har tagit fram dessa ramverk utifrån vad de själva anser fungera bäst. Man behöver inte anpassa undervisning helt utifrån ett och samma ramverk, man kan använda de delar som man själv gillar och tycker fungerar, men det är bra att ha forskning som stöd för varför man undervisar på ett visst sätt.

5.1 The instrumental approach

Det första ramverket vi skall titta på kallas för det instrumentella tillvägagångssättet. Vi börjar med en kortare sammanfattning av vad det handlar om, för att sedan titta närmare på de olika begreppen som används. Nedan följer min sammanfattning av vad Misfeldt & Ejsing-Duun (2015) skriver:

Det instrumentella tillvägagångssättet handlar om att hur elever använder teknologi i matematikundervisningen för att lösa problem. Man pratar om artefakter som ett verktyg som eleven använder för att nå sina mål. Eleven och verktyget har en nära relation – elevens aktivitet påverkas av hur verktyget ser ut (instrumentation), men verktyget påverkas också av hur eleven använder det (instrumentalisering). Man skiljer också på att lära sig matematik med hjälp av teknologi och att bara använda teknologi för att lösa uppgifter, och menar att skillnaden ligger i om läraren använder sig av epistemiska eller pragmatiska tillvägagångssätt. Epistemisk betyder att det handlar om användarens inre mål. Det påverkar hans eller hennes uppfattning om, överblick av eller kunskap om någonting. Man kan säga att man använder verktyget för att utforska någonting. Pragatisk betyder istället att det handlar om användarens yttre mål, att ”göra en förändring i världen”. Man använder alltså verktyget för att utföra någonting snarare än att lära sig någonting. Algoritmer är ett exempel på detta.

Låt oss nu titta på ett antal begrepp som nämns ovan.

Instrumentell uppkomst är en process där man bygger ett instrument från en artefakt. Den har två nära sammankopplade komponenter:
- instrumentaliseringsprocessen, riktad mot artefakten.
- instrumentationsprocessen, riktad mot subjektet. (Trouche, 2005)

Innan vi tittar närmare på dessa processer skall vi först reda ut vad en artefakt är, vad ett instrument är och vad skillnaden mellan de båda är.

En artefakt är ett materiellt eller abstrakt objekt som syftar till att upprätthålla mänsklig aktivitet i utförandet av en uppgift (en miniräknare är en artefakt, en algoritm för att lösa kvadratiska ekvationer är en artefakt); den ges till någon. Ett instrument är sedan det som

någon bygger med hjälp av artefakten. Ett subjekt (en elev) bygger ett instrument för att utföra en uppgift; detta instrument är därför sammansatt av både artefakt (egentligen den del av artefakten som används för att utföra dessa uppgifter) och subjektets scheman som tillåter henne/honom att utföra uppgifter och kontrollera hennes/hans aktivitet (Trouche, 2005). Det är alltså en hårfin skillnad mellan artefakt och instrument. Miniräknaren är en artefakt. Ett instrument kan sägas vara sättet du använder miniräknaren på – en kombination av vad du gör med den (exempelvis lösa ekvationer) och hur du använder den, hur ditt schema ser ut.

Vi ska gå tillbaks och titta på processerna, men först några korta ord om de scheman som nämndes ovan. Vergnaud (i Guin et al., 2005) beskriver det såhär: ”Ett schema är en invariant organisering av aktivitet för en given typ av situationer. Det har en intention och ett mål och utgör en funktionell dynamisk enhet.” (Vergnaud, i Guin et al., 2005). Ett schema är alltså en metod som man alltid använder sig av när man angriper en given situation, hur man gör för att lösa en viss typ av problem.

Instrumentaliseringsprocessen, riktad mot subjektet, involverar flera steg: ett steg av upptäckt och val av de relevanta nycklarna, ett steg av personalisering (att ”anpassa verktyget efter handen”) och ett steg av att omvandla verktyget, ibland i riktningar som designern inte planerat: modifiera verktygsfältet, skapa tangentbordsgenvägar, lagra program, automatiskt utförande av vissa uppgifter osv. Instrumentalisering är en process av differentiering rörande artefakterna själva:

- differentiering rörande miniräknarens innehåll: jämför man elevers miniräknare är det möjligt att identifiera skillnader (både kvantitativt och kvalitativt) mellan de olika program som finns lagrade
- differentiering rörande den del av artefakten som mobiliserats av subjektet (för vissa elever, en väldigt liten del av miniräknaren, för andra en stor del).

Instrumentalisering är uttrycket av ett subjekts specifika aktivitet: vad en användare tycker att verktyget designades för och hur det bör användas: utarbetandet av ett instrument tar plats i dess användning. Instrumentalisering kan därför leda till antingen en berikning av en artefakt, eller till dess utarmning. (Trouche, 2005)

Instrumentation är en process där begränsningarna och möjligheterna hos en artefakt formar subjektet. Denna process fortsätter genom uppkomsten och utvecklingen av scheman medan man utför uppgifter av en given typ. Trouche (2005) studerar ett exempel på en sådan process. Medan instrumentaliseringsprocessen fortsätter kan instrumentation genomgå flera olika steg. Defouad (i Guin et al., 2005) analyserar dessa utvecklingsprocesser hos elever som, efter att ha använt grafiska miniräknare, övergår till symboliska miniräknare (TI-92). Han särskiljer två huvudfaser, för det första en explosionsfas och för det andra en reningsfas. I början av instrumentell uppkomst verkar elevernas arbete befinna sig vid ett vägskäl, som om de letar efter en balans mellan deras tidigare tekniker och strategier (kopplade till grafiska miniräknare) och olika nya möjligheter som öppnats upp av den nya miniräknaren och utvecklingen av klassrumskunskap. Han kallar denna fas för explosionsfasen, där nya strategier och tekniker verkar bryta fram. Eleverna går progressivt in i den andra fasen som kallas reningsfasen, när verktygsanvändningen tenderar att nå en balans i den mening att de instrumenterade strategierna och teknikerna stabiliseras. Denna fas genomgås ofta med en

fixering på ett fåtal kommandon (och dessa val kan vara olika för olika elever). (Defouad, i Guin et al., 2005)

Instrumentell uppkomst är en process där ett instrument byggs, från en artefakt, av ett subjekt. Detta instrument byggs från en del av den ursprungliga artefakten (ändrad genom intrumentaliseringsprocessen) och genom scheman som byggts för att utföra en typ av uppgift. En komplex artefakt som en symbolisk miniräknare kommer därför föda, till en given elev, en uppsättning instrument (till exempel ett instrument för att lösa ekvationer, ett instrument för att studera funktionsvariation osv.) (Trouche, 2005)

Instrumentell uppkomst har både individuella och sociala aspekter. Balansen mellan dessa aspekter beror på:

- Materiella faktorer (det är ganska uppenbart att skärmen på en miniräknare passar bättre för individuellt arbete medan datorskärmar tillåter eleverna att jobba i smågrupper istället).
- Artefaktens tillgänglighet (ibland är de bara tillgängliga i skolan, ibland lånas de ut ett helt skolår, ibland är det elevernas egna).
- Vilket sätt läraren använder dessa artefakter på. (Trouche, 2005)

I min mening är skillnaden mellan epistemiska och pragmatiska tillvägagångssätt det viktigaste att ta med sig här ifrån. Förespråkarna för detta ramverk menar att det är viktigt att se till att eleverna använder sig av teknologi för att utforska, det vill säga genom epistemiska tillvägagångssätt. Visst finns det verktyg som man kan använda för att ”göra” något, men då sker ingen inlärning i samband med själva användandet. Att köra en algoritm för att lösa en ekvation gör inte att du lär dig någon matematik. Så när det kommer till själva inlärningen är det viktigt att tänka på att få med en utforskande del.
Man betonar också samspelet mellan användare och verktyg. Dels formas elevens aktivitet av verktyget, men eleven formar också verktyget. Eleven tar sig an uppgifter på olika sätt beroende på vilka verktyg man har tillgång till. Att verktyget formas efter eleven menar man beror på att det spelar ingen roll hur många funktioner ett verktyg har om eleven inte använder dem. Sättet verktygen används på är därför viktigt och det skiljer sig ofta mycket från elev till elev.
Enligt detta ramverket så är det ingen större mening med att eleverna använder programmering för att köra algoritmer. Eleverna skulle snarare lära sig mer av att koda algoritmen än att faktiskt använda den. Programmering som problemlösningsverktyg tror man alltså inte så mycket på. Det är istället bättre att använda det till att utforska matematik och möjligen kan eleverna lära sig matematik genom att själva skriva kod.

5.2 Constructionism

Det andra ramverket vi skall titta på kallas för konstruktionism.

Utanför skolans värld har mästaren och lärlingen ett gemensamt mål, vilket kortsiktigt ofta är att skapa en produkt (skräddaren och hans lärling kanske vill göra en väst), och långsiktigt är det att gå med vinst. Ainley et al. (2006) frågar sig om lärare och elever kan ha sådana gemensamma mål. Lärarens agenda måste vara att fokusera på deras elevers inlärning (och

det vet eleverna om), medan elevernas agenda kommer vara att slutföra uppgiften, förhoppningsvis till lärarens belåtenhet. Även om produktionen av (verkliga eller virtuella) artefakter verkar vara en givande kontext av meningsfull matematisk aktivitet leder detta till att de frågar sig vilka typer av produkter som lärare och elever kan skapa tillsammans. De betraktar tillvägagångssätt som placerar produktskapande i framkant av elevernas aktivitet, fastän de skiljer sig betydligt från hur man gör i skräddarverkstaden (Ainley et al., 2006). ”Vårt fokus har i många år varit på användandet av teknologi för inlärning av matematik, så det är naturligt för oss att titta på litteraturen från det fältet för inspiration.” (Ainley et al., 2006)

Konstruktioniströrelsen (Harel & Papert, 1991) har föreslagit att uppgifter där elever skapar produkter, generellt genom att programmera datorer, är särskilt ledande till inlärning. Därför, på ett sätt, ersätter konstruktionisterna västen med en produkt som programmeras av en elev på datorn. Vår erfarenhet (Pratt & Ainley, 1997, Ainley 2000) säger att sådana programmeringsuppgifter kan generera aktivitet som har några av de egenskaper som vardagsaktiviteter som studerats i forskningen om ”situated cognitionists”. (Ainley et al., 2006) Till exempel blir lärare ofta engagerade i att arbete med eleven för att skapa en virtuell produkt, på ett sätt som påminner om skräddaren och lärlingen som samarbetar för att skapa västen. Uppgiften, snarare än de externt uppsatta målen, tar på sig rollen som domare för vad som räknas som framsteg. Dessutom är all matematisk inlärning som äger rum kontextualiserad inom aktiviteten för produktskapandet, vilket ger mening till matematiken men kanske begränsar dess synbara tillämpningsområde (Ainley et al., 2006). Harel och Papert (1991) känner också igen kopplingen mellan konstruktionism och situated cognition, och betonar några skillnader:

”Vi ser flera trender i nutida utbildningsdiskussioner som situated learning och apprenticeship learning… att vara konvergerande med vårt tillvägagångssätt, men olika i andra avseenden… vår betoning är på att utveckla nya sorters aktiviteter där elever kan träna sitt görande/lärande/tänkande… och på projektsaktivitet som är självstyrd av eleven. (s.42)”

Ainley et al. tror att konstruktionisterna i sitt tillvägagångssätt inser att klassrummet inte är marknaden, och försöker inte sätta fokus på autenticitet. Däremot, genom att betona skapandet av produkter, så sätter det hänsyn till meningsfullhet och motivation högt upp på agendan för designen av uppgifter som är troliga att främja matematisk inlärning. Litteraturen om konstruktionismen främjar användningen av mikrovärldar, virtuella miljöer inom vilka eleverna kan utforska säkert och fritt, designade på ett sådant sätt att de sannolikt kommer stöta på kraftfulla matematiska idéer (Ainley et al., 2006). Inom litteraturen om mikrovärldsdesign har de hittat en andra viktig idé som relaterar till vårt tankesätt kring uppgiftsdesign. Inställningarna i mikrovärlden erbjuder möjligheten att eleverna arbetar med koncept som de ännu inte förstår. Förståelsen uppstår genom aktivitet. Hoyles och Noss (1987) kallar detta för att ”använda innan man vet”. Matematikundervisning tenderar att börja med definitioner och förklaringar av de matematiska idéerna. Mikrovärldar öppnar upp möjligheten att erbjuda matematisk kunskap som färdigbyggda objekt på skärmen. Alltså använder elever som till exempel använder RIGHT/LEFT-kommandon i Logo vinklar för att lära sig om vinklar. Papert (1996) hänvisar till denna idé som kraftprincipen och argumenterar för att den re-inverterar en invertering införd av den konventionella pedagogiken i matematik. Genom att lära sig genom användning får eleverna möjlighet att lära sig matematik på i stort

sätt samma sätt som de lär sig saker naturligt i andra sammanhang: till exempel sätten de lär sig att läsa och skriva på. Detta tillvägagångssätt står i kontrast till det konventionella tillvägagångssättet där eleverna sällan får prova på att använda matematik på meningsfulla sätt (Ainley et al., 2006).

Konstruktionisterna är lite inne på samma spår som det instrumentella tillvägagångssättet. Även här ser man utforskning som en viktig del i inlärningen av matematik. Man menar på att det är naturligare för eleverna att lära sig på detta sätt till skillnad mot traditionell undervisning som ofta inleder med förklaringar och definitioner som eleverna sedan skall försöka förstå. Låt istället eleverna utforska ett nytt område och se vad de hittar, öva på matematik på det sättet. ”Använda innan man vet”, som Hoyles och Noss (1987) uttryckte det.
Konstruktionisterna trycker betydligt hårdare på att skapandet av produkter och verktyg är en central del av inlärningen. Man gör till och med en jämförelse med skräddare – bästa sättet att lära sig att tillverka en väst är genom att försöka tillverka en väst. Eleverna skall försöka skapa sin egen kunskap. De kommer inte att klara av det på egen hand varje gång, men då finns läraren där som vägledare.

6 Praktiska erfarenheter

Hittills har vi bara tittat på teorier om hur programmering passar in i matematikundervisningen. Vi har bekantat oss med det instrumentella tillvägagångssättet och konstruktionismen, men nu ska vi övergå till praktiska undersökningar. I detta avsnitt kommer vi att titta på några undersökningar som forskare har gjort i klassrummet för att se hur programmering faktiskt påverkar inlärningen av matematik. Notera att detta är enskilda undersökningar och inte på något sätt är något facit till hur inlärningen av matematik påverkas genom användning av programmering, men det är likväl seriöst genomförda undersökningar av forskare och kan ge indikationer på olika saker. Resultaten från undersökningarna kan väcka tankar och funderingar, både kring saker som fungerar och saker som inte fungerar. Det kan dessutom ligga till grund för framtida forskning, resultaten kan visa någonting som är värt att utforska ännu mer.

Ett vanligt sätt att få in programmering i matematiken är genom datorspel. Här finns två olika varianter. Man kan antingen lära sig matematik genom att spela dessa spel eller genom att skapa spelen själv. Vi ska titta närmare på båda dessa varianter, men först några ord om Scratch. Scratch är ett dynamiskt, visuellt programmeringsspråk som ofta används för att skapa enklare spel i utbildningssyfte. Att det är dynamiskt gör att man kan ändra koden även medan programmet körs. Man använder sig dessutom av så kallad blockprogrammering, vilket gör att man inte behöver skriva någon kod själv. Scratch är därför väldigt användarvänligt och perfekt för nybörjare. Vi ska titta på två undersökningar som har gjorts där man har använt sig av just Scratch i matematikundervisningen.

6.1 Undersökning 1

Den första undersökningen pågick under en månads tid i två olika skolor.. Totalt 113 elever från femte och sjätte klass deltog i undersökningen, där 40 av dem ingick i en kontrollgrupp som alltså inte tog del av experimentet utan fortsatte sin undervisning som vanligt. Eleverna hade matematikundervisning i en och en halv timme varje dag och utöver det så hölls det fyra 45-minuterslektioner i programmeringsspråket Scratch (för de som ingick i experimentet). Lektionerna i Scratch hölls av forskarna som gjorde undersökningen. Det gjordes ett test innan experimentet startade och ett efteråt för att jämföra utvecklingen hos de elever som fått använda Scratch med de elever som ingick kontrollgruppen. Undersökningen gick ut på att ta reda på om Scratch är ett bra sätt för eleverna att utveckla sina förmågor inom problemlösning. Både testet innan och testet efter experimentet innehöll tre frågor där eleverna fick chansen att visa sina problemlösningsförmågor (Brown, Mongan, Kusic, Garbarine, Fromm, Fontecchio, 2008).

Lektionerna i Scratch är designade så att eleverna skall få möjlighet att tillämpa och utvärdera problemlösningstekniker med hjälp av programmering. Eleverna förväntas sedan kunna använda dessa tekniker till att lösa vardagliga problem. Lektionerna introducerar båda effektiva och inte så effektiva metoder och erbjuder ett visuellt sätt att utvärdera vilka metoder som är bra och vilka som är dåliga när de tillämpas (Brown et al., 2008).

Ett exempel från en lektion är ett spel där karaktären plockar körsbär från ett träd. Varje gång man hoppar med karaktären så plockar man ett fixerat antal bär och lägger i sin korg. Men korgen är inte hur stor som helst, och dessutom så är antalet bär som får plats i korgen inte delbart med antalet bär man plockar vid varje hopp. Hoppar du för många gånger blir korgen överfull och bären hamnar på marken. Denna övning går ut på att eleverna ska inse att multiplikation är en mer effektiv metod än upprepad addition. Man använder multiplikation när man skriver kod för att få karaktären att hoppa av sig själv (och sluta hoppa innan korgen blir överfull). Karaktären illustrerar sedan visuellt att multiplikation ju faktiskt är ”samma sak” som upprepad addition men en mer effektiv metod. Denna poäng visades ännu mer i en annan övning som gick ut på att rita en kvadrat, först genom att dra en linje fyra gånger för att sedan jämföra med att ”loopa” en instruktion fyra gånger (Brown et al., 2008)
Den första övningen är också en övning i att avrunda, något som eleverna hade svårt för i det inledande testet. Många hade svårt att avgöra om de skulle avrunda uppåt eller nedåt i olika situationer. Avrunda till närmsta tal som är delbart med antalet som ryms i korgen var en populär (men inte så bra) metod för att avgöra hur många gånger man skulle hoppa (är detta tal större så blir ju korgen överfull). Även detta visades tydligt visuellt med hjälp av denna övning (Brown et al., 2008).

Forskarna (Brown et al., 2008) satte upp en mall för att rätta testen där det fanns fem olika resultat:
A: Den mest lämpliga tekniken användes för att lösa problemet, och problemet löstes fullständigt.
B: Försök till att använda den mest lämpliga tekniken, men antingen användes den felaktigt eller så löstes inte problemet fullständigt.
C: En olämplig teknik användes (till exempel upprepad addition istället för multiplikation).
D: Oavsett vilken teknik som användes så löstes problemet felaktigt (Forskarna skiljer alltså på felaktigt svar och ej fullständigt svar).
E: Inget eller helt felaktigt svar.
Varje resultat gav sedan en poäng: A=16, B=8, C=4, D=0 och E=0. Eftersom det var tre frågor i varje test så var den maximala poängen alltså 48 (Brown et al., 2008).

De elever som deltog i experimenten var uppdelade i tre grupper. Från första till andra testen ökade två av grupperna sin snittpoäng med ungefär 5 poäng, medan den tredje bara ökade med ungefär 0.5 poäng. Detta ska jämföras med kontrollgruppen, vars snittpoäng inte ändrades mellan testerna.
Överlag så hade experimentgrupperna en ökning med 9% när det gäller att välja en lämplig metod (svaret behöver inte ha blivit rätt, men eleverna försökte lösa uppgifterna på ett lämpligt sätt), att jämföra med kontrollgruppen som var 26% sämre på det andra testet jämfört med det första (Brown et al., 2008).

6.2 Undersökning 2

Denna studie handlar om elever i sjätte klass. Även här har man använt sig av en kontrollgrupp och en experimentgrupp för att se hur programmering påverkar inlärningen av matematik. Totalt medverkar 42 elever, varav 24 är med i experimentgruppen och övriga 18 i kontrollgruppen (Calao, Moreno-León, Correa & Robles, 2015).

Experimentgruppen börjar med flera aktiviteter om sekvenser och processer innan en introduktion till algoritmer och programmering hålls. Eleverna börjar sedan lära sig att använda Scratch. Enkla interaktioner med programmet följs av användning av animerade dialoger. De får sedan lära sig om looper, variabler osv. Slutligen får eleverna programmera sina egna spel och göra egna simuleringar. Detta pågår under tre månaders tid. Under tiden fortsätter kontrollgruppen med sin undervisning på samma sätt som de gjort tidigare (Calao m.fl., 2015).

En modell för att bedöma elevernas förmågor togs fram som bestod av modellering, resonemang, problemlösning och utövning. Varje del innehåller vissa kriterier som ska utvärdera utvecklingen hos eleverna inom varje område. Eleverna får sedan omdömet utmärkt, bra, tillfredsställande eller otillräckligt i varje område. Till exempel, för att nå utmärkt, skall eleven kunna:
- Modellering: Eleven löser alla uppgifter relaterade till modelleringsprocessen, inom vilken upptäckten av variabler och förhållanden mellan dem som fastställer en matematisk modell behövs, såväl som upptäcker mönster som upprepas i vardagliga, vetenskapliga och matematiska situationer, och återuppbygga dem mentalt.
- Resonemang: Eleven använder resonemang för att lösa de problem han/hon ställs inför, uppfattar regelbundenhet och förhållanden, gör förutsägelser och gissningar, eller motiverar argument och resonemang.
- Problemlösning: Eleven löser enkelt problem i situationer som kräver strategier för att tolka givna påståenden, för att hitta resultat och verifiera dessa resultat.
- Utövning: Eleven använder enkla algoritmer, förstår koncepten de bygger på och känner igen när man kan använda en given teknik eller matematisk operation. (Calao m.fl., 2015)

Båda grupperna gör ett test innan experimentet startar, där ovan nämnda förmågor testas. När tre månader sedan har gått så gör båda grupperna ett nytt test där samma förmågor testas igen, för att se inom vilka områden och hur mycket eleverna utvecklats (Calao m.fl., 2015). Testet som görs innan experimentet startar visar att de båda grupperna ligger väldigt nära varandra i dessa förmågor. Tre månader senare, när ett nytt test görs, finner forskarna flera intressanta resultat. För kontrollgruppen ändrades knappt resultaten, vilket tyder på att den vanliga undervisningen inte utvecklade elevernas förmågor särskilt mycket (om ens något). Medelvärdet för experimentgruppen var dock mycket högre efter experimentet än innan, och standardavvikelsen var lägre. Eleverna har alltså dels utvecklats mycket, och det har blivit en jämnare fördelning inom gruppen. (Eleverna med bäst resultat på det första testet har också utvecklats mycket under denna period, men de med sämre resultat innan har utvecklats ännu mer). Om man tittar på förmågorna var för sig så var utövning den förmåga som utvecklades mest, eleverna har blivit mycket bättre på dels använda algoritmer men även förstå dem och inse när de går att använda. Även om alla förmågorna utvecklades betydligt hos experimentgruppen så var skillnaden inte lika stor inom problemlösning (Calao m.fl., 2015).

6.3 Undersökning 3

Vi ska också titta på en undersökning som handlar om speldesign. Både elever och lärare får prova på att designa spel som går ut på att lära sig om bråk. Även om denna del endast

handlar om bråk så finns det många delar som gäller för speldesign i allmänhet och som man kan ta med sig till andra områden inom matematiken.
Forskarna som genomfört undersökningen säger så här:

”Det räcker inte att speldesignsaktiviteter behandlar elevernas problem med att lära sig bråk, de måste också behandla lärares problem med att erbjuda vägledning” (Kafai et al., 1998).

De (Kafai et al.) menar alltså att ett stort ansvar ligger på läraren, att erbjuda vägledning på ett sätt som gör att eleverna kan lära sig på bästa sätt. Forskarna gjorde två undersökningar som de kallar för Students' Game Design och Teachers' Game Design. Det är alltså en undersökning på hur elever designar spel för andra elever och en undersökning på hur lärare designar spel till sina elever. Här skiljer vi på dem genom att helt enkelt kalla dem för ”Eleverna” och ”Lärarna”. I denna text kommer vi titta på båda undersökningarna parallellt, så nedan följer de båda undersökningarna.

6.3.1 Metod eleverna

I den första undersökningen arbetade elever i femte klass i grupper för att designa inlärningsspel med avsikt att lära ut bråk till yngre elever. Målet med undersökningen var att dokumentera de olika aspekterna av inlärningsmiljön och på vilka sätt designen bidrog till eller hämmade elevernas matematiska utforskning. Hela klassen deltog i projektet men undersökningen fokuserade på en grupp med fyra tjejer som hade en särskilt produktiv konversation (Kafai et al., 1998).

Eleverna fick först i uppgift att själva komma på en idé till ett spel. Sedan fick de ungefär en timme på sig att designa spelet i skolan. Först presenterade eleverna sin idé, hur spelet skulle fungera och hur bråk skulle vara inblandat. Sedan ”uppkom” ett designverktyg: använda bråk för att beskriva en verklig situation med flera aktörer i spelet. De (Kafai et al.,) säger att verktyget ”uppkom” eftersom det inte var forskarna som föreslog verktyget, utan det uppkom från elevernas egna idéer och samtal. En av tjejerna hade en idé om ett spel som liknar Jeopardy, där användaren får välja kategorier för att svara på frågor. Introduceringen av kategorier påbörjade idén om att den som spelade spelet skulle få göra olika val. Man skulle kunna välja till exempel ”djur”, ”TV-serier” eller något annat som kategori och sedan få en fråga om bråk som handlade om detta ämne.
En av forskarna gav sedan eleverna en utmaning: kan ni göra ett spel som handlar om bråk, men utan att ställa några frågor? Den sista halvtimmen gick sedan åt till att jobba med detta (Kafai et al., 1998).

6.3.2 Metod lärarna

I denna studie deltog 16 lärare med en snittålder på 24 år där de flesta var nyutbildade. Denna studie pågick under sex veckor, med ett möte ungefär varannan vecka (tre tillfällen). Under det första mötet delades lärarna in i fyra grupper. Varje grupp fick i uppgift att designa ett spel för att lära elever i fjärde klass om bråk. Inga designverktyg presenterades. Efter 20 minuter presenterade varje grupp sin design och varje presentation följdes av en diskussion där lärarna

bedömde varandras designer och ställde frågor om designerna och deras möjliga effektivitet (Kafai et al., 1998).
Vid det andra tillfället fick lärarna samma uppgift, men denna gång med ett designverktyg, samma fråga som eleverna fick: ”kan ni göra ett spel utan att ställa frågor?”. Lärarna fick ungefär en halvtimme på sig, presenterade sina designer följt av diskussion och frågor. Man jämförde även dessa nya speldesigner med de från det första tillfället (Kafai et al., 1998).
Vid det tredje och sista tillfället fick lärarna ytterligare ett verktyg. De fick ta del av elevernas designer där de hade använt dynamiska representationer (att representationerna kan förändras). Under detta tillfälle fick lärarna bygga vidare på sina tidigare idéer genom att försöka inkludera just dynamiska representationer. Återigen fick lärarna arbeta i ungefär 20 minuter, presentera sina förslag och avsluta med en diskussion (Kafai et al., 1998).

6.3.3 Analys eleverna

När forskarna sedan skulle analysera spelen började de med att transkribera hela sessionen. De identifierade de tidpunkter under samtalet där eleverna introducerade och utvecklade sina idéer. Även om det var uppenbart att eleverna inte skulle kunna skapa så många färdiga spel på så kort tid så var det tydligt att idéerna som eleverna föreslog var av olika kvalitet (Kafai et al., 1998).

6.3.4 Analys lärarna

Alla tre tillfällena filmades och dessutom följde forskarna (Kafai et al.) en grupp genom hela processen genom att spela in allt de gjorde och deras diskussioner. Forskarna transkriberade allt material och kommenterade speldesignen med lärarnas egna ord. Därefter utvärderade de lärarnas arbete (Kafai et al., 1998).

6.3.5 Resultat

Analysen delades upp i tre delar: integration av bråk, representation av bråk och hänsyn till användarens tänkande (Kafai et al., 1998).

6.3.6 Integration av bråk i spelen, eleverna

Forskarna definierade tre kategorier för integration av bråk: yttre, inre, och konstruktivistiska.

Yttre integration av bråk beskriver de typer av spel som många av både eleverna och lärarna hade tänkt från början; spel med arkadstil där frågor om bråk ställs ibland men som inte är relaterade till spelets tema eller mål. Till exempel hade en av tjejerna i elevgruppen en idé om ett spel där man sprang i en labyrint och blev jagad av olika saker, och om något hann ikapp en så var man tvungen att svara på en fråga.
Inre integration är istället när bråk är en inneboende del av landskapet, berättelsen eller målet. Ett exempel är ett av elevernas spel med en strand där olika andelar av människorna rör sig ner i och upp ur vattnet (mer om detta spel senare).

Konstruktivistiska idéer låter användaren aktivt skapa sina egna bråk utifrån det som erbjuds. Istället för att hela tiden svara på frågor så tillåts användaren att formulera sina egna. Ett exempel från en av eleverna är att spelaren kunde skapa sina egna bråk genom att färglägga olika delar av ett djur. De inre och konstruktivistiska bråkspelen representerar en sofistikerad avvikelse från både den vanliga övningsrutinen i traditionell undervisning och det icke-matematiska fokuset i de flesta datorspel (Kafai et al., 1998).

Innan eleverna fick några verktyg, i delen där de presenterade sina idéer, var bråken yttre integrerade och dåligt specificerade. Frågor om bråk presenterades antingen som ett nödvändigt hinder för att komma vidare eller som ett straff om du gör något fel. Eftersom eleverna utvecklade dessa idéer hemma, utan hjälp från varken forskare eller andra elever, kan man argumentera för att detta är deras förutfattade meningar om inlärningsspel, bestående av antingen frågor/svar-format eller händelser i spelet som inte är relaterade till bråk. (Kafai et al., 1998)

Efter introduktionen av det första verktyget (det som uppkom) lyckades eleverna med inre integration i alla sina påföljande spelscener. Eleverna använde bråk för att beskriva verkliga situationer med flera aktörer. Men när forskaren bad eleverna att designa spel utan att ställa frågor (det andra verktyget) blev det en utmaning. Eleverna hade svårt att komma på ett inlärningssätt med bråk utan att involvera frågor. Några försökte hitta luckor och formulera sig på sådana sätt att det inte ”lät” som frågor, till exempel ”skriv detta bråk med minsta möjliga nämnare” (Kafai et al., 1998).
Här visas det tydligt hur eleverna såg på inlärningsspel. De har, oavsett om det är på datorn eller inte, frågor och svar som sitt huvudsakliga format. Så småningom skapade eleverna dock tre spel där användaren kunde skapa sina egna bråk. Detta visade sig dock vara väldigt svårt och överlag så var elevernas idéer inre men inte konstruktivistiskt integrerade (Kafai et al., 1998).

6.3.7 Integration av bråk i spelen, lärarna

Lärarnas spel kännetecknas av ett skifte från yttre integration till inre integration efter att designverktygen introducerats. Verktygen förbättrade också speldesignen inom kategorin för inre integration: dessa spel inkluderade bättre berättelser och rikare sammanhang (Kafai et al., 1998).

Vid det första tillfället så skapade lärarna antingen spel med yttre integration eller spel med inre integration som saknade ett meningsfullt sammanhang, till exempel ett spel som bara gick ut på att para ihop ekvivalenta bråk med hjälp av pizzor eller kort. Med andra ord så gick spelen antingen ut på att använda bråk för att komma vidare eller som ett försök att erbjuda en konkret representation – men dessa representationer var ofta förvirrande (Kafai et al., 1998).
Efter att designverktygen introducerats så använde lärarna inre integration i alla sina spel. Varje spel hade ett berättelsetema som inte bara fick spelet att fortsätta utan också använde bråk i meningsfulla situationer. Ett bra exempel var ett spel där användaren var kapten på ett skepp. De matematiska problem var saker som att se till att varje passagerare fick lika mycket mat osv. (Kafai et al., 1998).
Vid det tredje tillfället fortsatte lärarna att utveckla sina spel. Antingen byggde de vidare på sina tidigare spel eller kom på nya sammanhang. I ett spel kunde man välja hur många

kompisar man ville ta med sig när man skulle äta och hur många objekt (kakor, pizzor) man skulle dela på. Spelaren fick alltså göra aktiva val för att ändra bråken (Kafai et al., 1998).

6.3.8 Representation av bråk, eleverna

Representation av bråk är ett kritiskt ämne för utvecklingen av elevernas tänkande och är ofta ett ämne som speglar designerns syn på elevers tänkande. Forskarna (Kafai et al.) kategoriserade nivån för vilken designern erbjöd möjligheter för representation av bråk. På den lägsta nivån erbjöds ingen möjlighet alls. I den symboliska representationskategorin såg designern till att användaren endast fick numeriska representationer, medan man i den fixerade illustrationskategorin fick tillgång till enskilda illustrerade representationer men ingen kontroll över representationen. I den fixerade flerfaldiga kategorin fick spelaren tillgång till flera olika representationer (symboliska och illustrerade), men de var strukturerade i spelet så att man bara hade tillgång till en i taget (den som efterfrågades). Spelaren har inte heller här någon kontroll över representationen eller valet av representation (Kafai et al., 1998).

En av de mest slående utvecklingarna i elevernas tänkande om bråk var i detta område. Förändringarna som ägde rum här påvisar kraften hos designverktyg. Innan eleverna fick några verktyg så hade alla spelen antingen symboliska representationer eller inga alls (Kafai et al., 1998).
Efter att verktygen introducerats så blev representationerna mer komplexa och varierade. När eleverna skulle beskriva vardagliga situationer så var det vanligaste att man hade en illustrerad representation för varje scen, som i exemplet med att färglägga djur. Ett särskilt intressant förskjutning inträffade i och med spelet med stranden som nämndes tidigare (kap 1.1.3.1). Människorna rör sig mellan stranden och vattnet, och användaren frågas hur stor andel som är på varje ställe. Här har vi en dynamisk representation som kan ändras i två riktningar: andelen människor på eller utanför stranden ändrades hela tiden, men även det totala antalet människor ändrades, för människorna kunde komma till stranden eller gå ut ur bild. Sådana animerade scener erbjuder en mer flexibel bild av hur bråk kan användas i verkliga situationer, där varken andelarna eller det totala antalet är fixerade (Kafai et al., 1998).
När eleverna sedan skulle försöka skapa spel utan att ställa frågor utvecklades två olika typer av fria representationer. Den första gick ut på att användaren kunde manipulera de bråk som designern hade valt ut, men inte skapa egna. I den andra varianten fick inte användaren några bråk alls, utan erbjöds istället att skapa egna genom att interagera med miljön.
Representationerna gick alltså från att endast ha symboliska representationer (eller inga alls) till mer komplexa representationer med flera olika varianter (Kafai et al., 1998).

6.3.9 Representation av bråk, lärarna

Vid det första tillfället gjorde varje grupp speldesigner som främst använde symboliska representationer. Efter att verktygen introducerades blev presentationerna mer komplexa och varierade. Speldesignen bestod främst av fixerade, illustrerade representationer med en karaktär som rör sig genom en mer utvecklad berättelse och sammanhang. Ett spel stack ut när det kommer till representation av bråk. Spelet som nämndes tidigare, där man själv fick bestämma hur många kompisar man skulle ha med sig och hur mycket saker man skulle dela

på, går ut på att man ska uppleva så många saker som möjligt på en dag. Här har man möjlighet att själv välja vilka saker man vill göra, med flera olika representationer. Det är inte bara mat som ska delas på, det kan vara platserna i en berg-och-dalbana och mycket annat. Det finns flera representationer som man kan ändra på själv som gör att spelaren känner igen sig i problemen. Spelaren kan när som helst ändra problemen genom att ändra antalet kompisar som är med eller antalet föremål som ska delas. Man kan också lämna ett problem och komma tillbaka senare. Tillsammans skapar detta många olika möjligheter för spelaren. Detta öppnar också upp för att ett problem kan ha många olika lösningar. Överlag blev alla spelen bättre på att släppa de symboliska representationerna och övergå till mer dynamiska och varierade representationer ju längre lärarna arbetade med spelen (Kafai et al., 1998).

6.3.10 Hänsyn till användarens tänkande, eleverna

En av de mest värdefulla (och svåra) funktionerna i design av inlärningsspel för matematisk utveckling är dess fokus på användaren. Speldesign kräver att designern, oavsett om det är en elev eller en lärare, släpper sitt egna perspektiv som spelutvecklare och fokuserar på användaren istället (Kafai, 1995). När lärare designar spel kan man förvänta sig att de tänker över vilka pedagogiska problem som kan finnas. Elever däremot saknar information om matematisk utveckling som kan hjälpa dem att designa sina spel. Utan kunskap om ”hur elever lär sig bråk” måste de tänka igenom sina egna erfarenheter och använda sig av den kunskapen (Kafai et al., 1998).

Forskarna (Kafai et al.) definierar tre olika varianter av hänsyn till användaren för elever: straff, utvärderande och konstruktiva. Straffspel använder frågor om bråk som straff när användaren gör något fel. Om användaren inte lyckas med det han/hon ska så måste han/hon svara på en fråga för att komma vidare. Denna strategi att använda bråk som straff kan ha mer att göra med elevernas negativa inställning till bråk än någon pedagogisk tanke. Hur som helst så kräver det inte mycket hänsyn till användarens tänkande för att designa ett straffspel (Kafai et al., 1998).
Utvärderande spel påminner mycket om de övningsrutiner som elever är vana vid från traditionell undervisning. För att komma vidare i spelet måste du svara på en fråga som egentligen inte har någonting med spelet att göra, det är bara ett hinder för att ta sig vidare. Ofta går det dessutom på tid, det gäller att svara rätt så snabbt som möjligt för att komma vidare, och då handlar det mer om saker man redan kan än om att lära sig något nytt. Sådana spel går bara ut på att testa vad användaren redan kan (Kafai et al., 1998).
Konstruktiva spel skiljer sig på det sättet att deras främsta mål är att få användaren att engagera sig i bråk på ett meningsfullt sätt och erbjuda aktiviteter som ger en möjlighet att bygga vidare på användarens nuvarande kunskap. Denna typ av spel kräver att designern tar användarens perspektiv och överväger vilket som kan vara det mest produktiva sammanhanget för att lära sig bråk. Konstruktiva spel kräver också uppmärksamhet på hur bråk kan integreras till en särskild spelmiljö där användaren kan interagera med dem, vilket gör denna typen av spel svåra att designa (Kafai et al., 1998).
Innan verktygen erbjöds så fanns det inte mycket tanke på användarens tänkande. Fyra av elevernas spel använde bråk som ett straff, och ett använde det som utvärdering av hur bra och snabbt man kunde svara på frågor (Kafai et al., 1998).
Efter det första verktyget bytte eleverna fokus till utvärderande spel. Bråken var mer integrerade i spelscenerna och presentationerna var varierade, men formatet var fortfarande

frågor och svar. När eleverna istället skulle försöka skapa spel utan att ställa frågor skedde det en stor utveckling. När frågor och svar var borta ur bilden så kom eleverna på mer konstruktiva strategier. Ett sätt var att användaren skulle göra sina egna bråk med hjälp av spelets resurser. Det andra var att användaren skulle interagera med miljön i spelet där bråk vad en del av scenerna (Kafai et al., 1998).

6.3.11 Hänsyn till användarens tänkande, lärarna

Från början hade spelen ingen koppling till elevernas tänkande. De innehöll inga uppgifter som gav användaren möjlighet att använda olika strategier och lärarna använde inte sin kunskap om elevers tänkande för att designa spelen. Istället handlade det bara om att öva på och visa kunskaper man redan har. Sammanhanget gav ingen möjlighet för användaren att bygga vidare på sin kunskap, bara bevisa den man redan har (Kafai et al., 1998).
När verktygen började användas förbättrades hänsynen till användarens tänkande. Spelen erbjuder sammanhang som eleverna kan relatera till och som dessutom är matematiskt kraftfulla. Ett av spelen har uppgifter som ”dela upp pizzan så att de tre tjejerna får lika mycket” och ”se till att alla får spela lika mycket” på en fotbollsplan. Dessa sammanhang och problem är mycket enklare för eleverna att relatera till och hjälper till att utveckla idéer. Eleverna får bygga vidare på sina naturliga strategier när de ska dela upp saker. Återigen gör verktygen så att utvecklingen av spelen blir bättre (Kafai et al., 1998).

6.3.12 Forskarnas slutsats

Speldesign om bråk gav både lärare och elever möjlighet att tänka på vilka sammanhang som var matematiskt kraftfulla. Speldesign erbjöd en situation som naturligt kombinerade problem om praktik och teori, och samtidigt erbjöd en möjlighet för diskussion och samarbete i ett meningsfullt sammanhang. ”Vårt fokus på inlärningsspel och användandet av designverktyg tillät oss att se hur speldesign kan vara ett kraftfullt inlärningssammanhang för lärare och elever. Speldesign har inte bara denna potential, utan parallellerna mellan lärares och elevers tänkande föreslår att genom att arbeta tillsammans i klassrummet, där lärare och elever engagerar sig i speldesign, kan man utveckla lärande och skapa klassrum som är inlärningsmiljöer för både lärare och elever” (Kafai et al., 1998).

6.4 Slutsats

I den första undersökningen används Scratch för att spela väldigt enkla spel. Exemplet från undersökningen handlar om att plocka körsbär från ett träd. Här ligger inget fokus på att spelet skall vara särskilt roligt att spela. Man gör det till ett spel för att det ger en visuell bild av vad som händer och även om det inte verkar vara ett särskilt underhållande spel så är det nog ändå roligare än att sitta och titta på en bit kod.
Här har programmeringen fått en stor del av undersökningen. Programmering används som ett verktyg för att lösa problem. Just problemlösning var ju också det som undersökningen fokuserade på. Som vi såg så utvecklade eleverna i experimentet sin problemlösningsförmåga och fick bättre resultat på det andra testet, men det kanske mest intressanta resultatet är att experimentgruppen var betydligt bättre än kontrollgruppen på att välja rätt metod för att lösa

uppgifterna i det andra testet. Även om man inte fick löst uppgifterna fullständigt så var man i alla fall bättre på att inse hur man bör gå tillväga för att ta sig an problemen, vilket visar på en utvecklad förståelse.

Den andra undersökningen om Scratch testade också elevernas utveckling inom problemlösning, men man testade även andra delar av matematiken. Resultaten härifrån visar att eleverna utvecklades mer inom alla de förmågor som testades när de fick arbeta med programmering istället för traditionell undervisning. Intressant var att problemlösning faktiskt var den av förmågorna som utvecklades minst, även om den också utvecklades bättre genom programmering. I gymnasieskolan i Sverige ligger programmering under just problemlösning i ämnesplanen i matematik. Detta är ju också en av frågorna vi ställde oss i inledningen av denna litteraturstudie: var i matematikundervisningen passar programmeringen in? Enligt denna undersökning så gjorde programmeringen förvisso att eleverna utvecklades mer än vad de gör genom traditionell undervisning, men andra förmågor utvecklades ännu mer. Borde programmeringen byta plats i ämnesplanen, eller borde den användas på fler ställen? Här behövs mer forskning för att kunna säga något säkert, men resultaten väcker frågor som kan vara värda att undersöka vidare.

I båda undersökningar får eleverna programmera spel. I den första använder man programmering som ett verktyg för att lösa problem. I den andra används det dessutom för att göra egna simuleringar och utforska matematik. Allt detta passar in i konstruktionismens ramverk som vi tittade på tidigare. Enligt det instrumentella tillvägagångssättet kan det vara problematiskt att använda verktyg som hjälp vid problemlösning om man bara får verktyget och sedan använder det, men eftersom eleverna själva är med och kodar programmet så är det inget problem alls.

Den sista undersökningen handlar främst om speldesign, som är ett populärt sätt att få in programmering eller i alla fall digitala verktyg i matematikundervisningen. I undersökningen nämns inte programmering utan fokus ligger på själva designen av spelen. Men för ”enkla” spel går det naturligtvis att lägga till momentet att man faktiskt programmerar spelet själv. Som vi såg i de tidigare undersökningarna är Scratch ett populärt program för detta.
Studien visar vilka saker det är viktigt att tänka på när man designar ett spel. Som rubrikerna avslöjar så är det flera saker man måste tänka på, både hur man integrerar och presenterar det matematiska innehållet (detta gäller så klart inte bara bråk även om studien handlar om just det) men även att ta hänsyn till att användaren faktiskt ska utveckla sitt tänkande och sin kunskap. Forskarna poängterar dessutom väldigt tydligt vilken skillnad det gör att man faktiskt vet vad man håller på med om man skall designa ett spel. Att eleverna inte tänker så mycket på hur de bygger upp sitt spel är inte så konstigt, men även lärarnas spel blir betydligt bättre bara av att de arbetar med det i ett par timmar tillsammans med forskarna. Jag tror att det är enkelt som lärare att tänka att ”det kan inte vara så svårt att designa ett enklare matematikspel”, men ska man lita på forskningen så verkar det (i alla fall i det här fallet) göra stor skillnad att man faktiskt får extra kunskap om det.
Precis som andra delar i denna litteraturstudie är det bara ett eller möjligtvis några forskningsresultat som studien bygger på. Men även om man inte kan fastslå saker som att ”de flesta lärare designar dåliga spel om de inte får träna på det” eller ”får de träna på det så designar de flesta lärare bra spel” så ger det ändå en indikation på att man förmodligen blir betydligt bättre på det om de får träna på det med de rätta verktygen.

Hur som helst så verkar det som att både lärare och elever kan utvecklas av speldesign. Dessutom så tjänar ju dessutom de yngre eleverna på detta om spelen blir bra. Läraren utvecklar och tränar på sin pedagogiska förmåga, eleverna som designar spelen blir ännu bättre och ännu säkrare på den matematik som spelen handlar om och om man dessutom lägger in momenten att eleverna själva ska programmera spelen så får man med den delen också. De yngre eleverna som sedan ska spela spelen kommer förhoppningsvis lära sig matematiken bättre än vad de hade gjort med traditionell undervisning. Om detta görs på rätt sätt så ser det ut som att alla kan tjäna på denna typ av undervisning.
För att knyta ihop denna undersökning med ramverken vi tidigare har diskuterat så faller detta till viss del in under konstruktionismen. Eleverna som spelar dessa spel lär sig matematik i en sådan miljö som beskrivs i konstruktionismen, även om de i detta fallet inte använder sig av någon programmering själva. Men detta går naturligtvis att bygga vidare på med hjälp av konstruktionismens ramverk – det ställer högre krav på den eller de som designar spelen, men spelen kan designas så att man använder sig av programmering även när man spelar dem. Med tanke på att denna studie är genomförd i en femteklass är det dessutom rimligt att tro att om man skall göra om uppgiften så att den passar på gymnasienivå så är det ett naturligt steg att ta att även den som spelar använder sig av programmering.

7 Egna tankar/förslag

Litteraturen vi har studerat har främst handlat om barn i yngre åldrar än gymnasiet. Dessutom har de exempel vi såg i kapitlet om praktiska erfarenheter främst handlat om så kallad blockprogrammering och inte så mycket om kodning. Vi skall nu titta på hur man kan anpassa undervisningen för gymnasienivå och hur man kan få in att man faktiskt skriver sin egna kod. I ämnesplanen står det inte att man måste lära ut hur man kodar, det står bara att programmering skall användas som ett verktyg i problemlösning, men det är ett sätt att göra det på. Det finns många lärorika sätt att använda sig av kodning på. Jag skall ta upp några exempel här.

När man pratar om programmering som ett verktyg att använda i samband med problemlösning så tänker jag främst att man använder sig av algoritmer. Om eleverna sedan ska koda algoritmerna själva eller om algoritmerna skall delas ut till eleverna så att de kan använda dem direkt går att diskutera, men här kommer vi fokusera på att kodningen skall vara en del av undervisningen. Vi kommer i det här kapitlet titta närmare på Euklides algoritm. Anledningen till att man vill arbeta med programmering i samband med algoritmer är att de kan ta väldigt lång tid att utföra för hand medan en dator kan göra det väldigt snabbt.

Men man kan också tänka sig andra områden än problemlösning där programmering är väldigt användbart. Andra områden där man kan använda datorn till hjälp med saker som tar väldigt lång tid att göra för hand är till exempel sannolikhet och statistik. Om du exempelvis vill kasta en tärning 10000 gånger och anteckna resultaten av kasten så är en dator att rekommendera. Just det är dessutom en sådan sak jag själv minns från skoltiden, då läraren kastade en tärning 10-20 gånger, antecknade resultaten som hade en viss spridning och förklarade sedan att om man kastar tillräckligt många gånger så kommer resultatet ”jämna ut sig”. Det kommer det med största sannolikhet att göra, tärningskast faller in under de stora talens lag som säger att ”det aritmetiska medelvärdet av ett stort antal observationer av en slumpvariabel med stor sannolikhet ligger nära variabelns vänteläge” (”De stora talens lag”, 2018, 10 januari). Men med hjälp av programmering kan man visa det istället för att bara säga att det blir så, vilket jag tror gör stor skillnad.

7.1 Euklides algoritm

Euklides algoritm är en metod för att hitta den största gemensamma delaren hos två tal. Den är döpt efter den grekiske matematikern Euklides som beskrev den i Elementa. Som namnet avslöjar är det en algoritm, en procedur som utförs steg för steg efter väldefinierade regler. Euklides algoritm är en av de äldsta algoritmer som används idag. Även om Euklides beskrev algoritmen redan 300 år f.Kr. så var det troligen inte han själv som kom på den. Elementa är en sammanställning av matematik från tidigare matematiker. Man tror att bland annat Eudoxus kände till algoritmen (375 f.Kr.) och den kan vara ännu äldre än så. Flera århundraden senare upptäcktes algoritmen i både Indien och Kina när man försökte lösa diofantiska ekvationer inom astronomi och när man skulle göra kalendrar. Första gången den beskrevs i Europa var år 1624. I Europa användes den till att lösa diofantiska ekvationer och utveckla kedjebråk. Under åren har Euklides algoritm och versioner av den använts till saker som utveckling av nya talsystem och mycket annat som att förkorta bråk, utföra division i

modulär aritmetik, kryptering, kedjebråk, approximering, samt bevis av satser inom talteori (”Euklides algoritm”, 2018, 2 maj).

Euklides algoritm fungerar på följande sätt:

Låt a=bq+r och hitta ett tal u som delar både a och b (så att a=su och b=tu). Då delar u även r, eftersom r=a-bq=su-qtu=(s-qt)u. Hitta på motsvarande sätt ett tal v som delar både b och r (så att b=mv och r=nv). Då delar v även a eftersom a=bq+r=qmv+nv=(qm+n)v.
Därför är alla gemensamma delare till a och b även gemensamma delare till b och r, och proceduren kan upprepas på följande vis:

$$
\begin{array}{lll}
q_1 = \left\lfloor \frac{a}{b} \right\rfloor & a = b\,q_1 + r_1 & r_1 = a - b\,q_1 \\
q_2 = \left\lfloor \frac{b}{r_1} \right\rfloor & b = q_2\,r_1 + r_2 & r_2 = b - q_2\,r_1 \\
q_3 = \left\lfloor \frac{r_1}{r_2} \right\rfloor & r_1 = q_3\,r_2 + r_3 & r_3 = r_1 - q_3\,r_2 \\
q_4 = \left\lfloor \frac{r_2}{r_3} \right\rfloor & r_2 = q_4\,r_3 + r_4 & r_4 = r_2 - q_4\,r_3 \\
q_n = \left\lfloor \frac{r_{n-2}}{r_{n-1}} \right\rfloor & r_{n-2} = q_n\,r_{n-1} + r_n & r_n = r_{n-2} \\
 & & \quad - q_n\,r_{n-1} \\
q_{n+1} = \left\lfloor \frac{r_{n-1}}{r_n} \right\rfloor & r_{n-1} = q_{n+1}\,r_n + 0 & r_n = r_{n-1}/q_{n+1}
\end{array}
$$

Illustration 1: Euklides algoritm, hämtad från http://mathworld.wolfram.com/EuclideanAlgorithm.html 2018-11-05

När q_{n+1} delar r_{n-1} tar algoritmen slut, och r_n är den största gemensamma delaren till a och b (om man bara använder heltal) (Wolfram, u.å).

Det man gör är alltså att dividera det större talet med det mindre, för att sedan ersätta det större talet med den rest man får från divisionen. Detta upprepas tills divisionen går jämnt ut och resten blir 0. Det är då resten från steget innan som är den största gemensamma delaren till de ursprungliga talen. Vi tar ett exempel för att visa. Vi vill hitta den största gemensamma delaren till 391 och 1196. Vi följer algoritmen steg för steg och får:

1196=391*3+23
391=23*17+0

Här behövde vi bara använda algoritmen i två steg. Ibland behövs det fler, men med det här metoden behövs det aldrig fler steg än antalet siffror i det mindre talet multiplicerat med 5 (Wolfram, u.å). Har vi 391 som det mindre talet kommer det alltså aldrig att krävas mer än 15

steg, oavsett hur stort det andra talet är. Det kan låta mycket men det är fortfarande en mycket effektiv metod, framförallt om man använder en dator till hjälp.

Om man inte använder sig av Euklides algoritm när man vill hitta den största gemensamma delaren till två tal finns det andra metoder. En metod är att hitta alla gemensamma delare och sen se vilken som är störst. Men det tar så klart mycket längre tid att hitta alla än vad det tar att hitta en, så Euklides algoritm är ett bättre alternativ. En annan metod är att dela upp talen i primtal. Om man tar de primtal som är gemensamma för båda talen och multiplicerar dem med varandra så får man den största gemensamma delaren (252=2*2*3*3*7, 105=3*5*7. 3 och 7 gemensamma, 3*7=21). Men för stora tal är primtalsfaktorisering väldigt svårt – modern kryptering bygger på att det är nästintill omöjligt att dela upp stora tal i primtal (”Euklides algoritm”, 2018, 2 maj).
Euklides algoritm är alltså en väldigt effektiv metod för att hitta största gemensamma delare.

Nu skall vi titta på hur man kan få in programmering i det här. Versionen som använder division kan kodas på följande vis:

```
function gcd(a, b)
   while b ≠ 0
      t := b;
      b := a mod b;
      a := t;
   return a;
```

När den här algoritmen har fått arbeta några varv så kommer variabeln b vara den senaste resten man fått efter divisionen och variabeln a är resten från divisionen man gjorde innan. Datorn sparar den nuvarande variabeln b tillfälligt, utför nästa division och ger ett nytt värde till b (resten), tar sedan det tidigare värdet på b och sätter det som värde på a istället. Sedan gör den om samma procedur igen och igen tills det att b=0, det vill säga tills divisionen går jämnt ut och resten blir 0. Då skriver den istället ut värdet på den senaste resten vi fick innan den blev 0, alltså den största gemensamma delaren.
För att klara av att koda detta krävs (förutom grundläggande kunskaper i programmering) att man verkligen förstår hur Euklides algoritm är uppbyggd och hur den fungerar. Det är ett utmärkt sätt att se om eleverna har förstått detta eller om de bara har memorerat hur algoritmen används.

7.1.1 Fördjupning Euklides algoritm

Som vi redan nämnt så kan Euklides algoritm användas till många olika saker inom matematiken. Vi skall titta på några här som förslag på fördjupning inom både matematik och programmering.

Det finns en variant av Euklides algoritm som kallas för den utvidgade versionen (extended Euclidean algorithm). Med den kan man förutom den största gemensamma delaren till två tal även beräkna koefficienterna x och y som uppfyller ax+by=SGD(a,b). Den största

gemensamma delaren till två tal a och b kan alltså skrivas som summan av de båda talen multiplicerade med varsin heltalsfaktor x och y. Metoden för att hitta dessa tal är att använda Euklides algoritm baklänges. Vi ska titta på ett exempel:

Vi har talen 102 och 38. Vi beräknar deras största gemensamma delare med hjälp av Euklides algoritm:

102=2*38+26
38=1*26+12
26=2*12+2
12=6*2+0

Deras största gemensamma delare är 2. Nu vill vi arbeta baklänges tillbaks till början av algoritmen genom att uttrycka talet 2 i termer av de två tal vi använt oss av i raden ovanför. Det ser ut som följer:

2=26-2*12

Vi vill nu ersätta talet 12 med termerna från nästa rad ”uppåt” i algoritmen. Alltså:

12=38-1*26
2=26-2*(38-1*26)
2=3*26-2*38

Vi fortsätter på samma sätt med att ersätta 26 med 102:

26=102-2*38
2=3*(102-2*38)-2*38
2=3*102-8*38

Våra värden på x och y är alltså 3 och -8, och vi kan skriva ax+by=SGD(a,b) som

3*102+(-3)*38=SGD(102,38)=2

(Brilliant, u.å)

Detta kan sedan användas till att lösa så kallade diofantiska ekvationer. Det är en typ av ekvation där man bara är intresserad av lösningar som är heltal. En vanlig linjär diofantisk ekvation ser ut på följande vis:

ax+by=c

där a, b och c är givna heltal. Vi kallar den största gemensamma delaren till a och b för g. Eftersom g delar båda termerna i ax+by så måste g också dela c. Vi kan därför skriva om ekvationen som

sa+tb=g

där g alltså är den största gemensamma delaren till a och b. Vi har nu en ekvation på samma form som vi hade tidigare när vi använde den utvidgade versionen av Euklides algoritm, så vi kan alltså lösa denna ekvation med hjälp av algoritmen.

Detta erbjuder elever som är intresserade av att lära sig mer en möjlighet att utveckla sina kunskaper inom både matematik och programmering. Här används programmeringen som ett problemlösningsverktyg i första hand och inte så mycket för att utforska matematik. Vi skall nu istället titta på hur man kan göra för att använda programmeringen till att just utforska.

7.2 Sannolikhet

En annan intressant fråga är vad man sedan gör med det man har kodat. Jag tror inte att man lär sig särskilt mycket matematik av att skriva koden. Det är snarare så att för att klara av att skriva koden så måste du redan kunna den matematiken som behövs. Det man skulle kunna lära sig av är om man inte får ihop sin kod för att man inte kan matematiken tillräckligt bra och sedan ser en färdig kod och får den förklarad för sig. Då kan koden i sig bli ett verktyg för att lära sig matematiken bättre, men man har inte ”programmerat sig till kunskapen”. Däremot kan man lära sig mycket när man väl har en färdig kod att använda. I fallet med Euklides algoritm finns det väl några intressanta resultat att hitta om man utforskar olika tal men det är framförallt ett verktyg för att lösa olika problem. Vi skall här titta på hur man istället kan fokusera på just utforskning med hjälp av en färdig kod.

I inledningen av kapitlet nämndes exemplet med tärningskast. Det är ett av många utmärkta sätt att utforska med hjälp av programmering. Här används inte programmering som ett verktyg för problemlösning, men det öppnar upp för något annat. Att faktiskt kunna se en simulering av tärningskast tror jag ger en mycket bättre bild än om man bara får det förklarat för sig. Om jag kastar en tärning 6 gånger så ”borde” jag få en sexa, men det är inte särskilt otroligt att jag får ett annat resultat. Om jag kastar 60 gånger borde jag få tio sexor osv. Framförallt om man har en graf som ändras efter varje kast så blir det väldigt tydligt vad som händer när man kastar många gånger. Min egna erfarenhet är att tärningen kastas ett tiotal gånger och sen ritas en graf för hand med resultaten, där det inte är ovanligt att det är betydligt fler av något tal än av något annat. Att faktiskt själv få se hur grafen ”jämnar ut sig” gör det väldigt mycket enklare att förstå att det blir så.

Här blir det inget fokus på själva programmeringen, den används bara som ett verktyg för att utforska. I och med det så behöver simuleringen inte vara enkel att koda, det kan vara en simulering som är skapad av ett proffs. Programmering skall användas som ett verktyg vid problemlösning står det i ämnesplanen, men om man ändå använder sig av programmering där så finns det ingen anledning till att inte använda det även inom andra områden i matematik. Allt möjligt går att simulera bara man vet hur man ska göra och om vi bara ska använda det för att utforska så behöver vi inte bekymra oss om hur svår kodningen är så länge den går att få tag i från någon annan. Det mesta (om inte allt) blir enklare att förstå om man får se en visuell bild av det. Allt som är svårt och/eller tar lång tid att göra för hand går att visa på en dator istället. Det kan vara att rita grafer, kasta tärning, göra långa beräkningar eller i princip vad som helst.

Vill man ändå använda sig av kodning så går det naturligtvis på enklare uppgifter. Simulering av ett tärningskast i programmeringsspråket Python kan se ut såhär:

```
import random

def randomnumber( ):
   return random.randint(1, 6)

print(randomnumber( ))
```

De stora talens lag säger att om vi har ett antal oberoende variabler med samma sannolikhetsfördelning så är sannolikheten för att medelvärdet skall konvergera mot väntevärdet då antalet variabler går mot oändligheten 1. Om vi simulerar tärningskast med en vanlig tärning så är sannolikheten att få en etta 1/6, att få en tvåa 1/6 osv. Väntevärdet beräknas på följande sätt:

$E(x_i)=(1/6)*1+(1/6)*2+(1/6)*3+(1/6)*4+(1/6)*5+(1/6)*6=3,5$

Det innebär, enligt de stora talens lag, att om vi kastar tärningen oändligt många gånger så kommer medelvärdet av tärningskasten att konvergera mot 3,5. Nu är det omöjligt att kasta oändligt många gånger, men kastar du tillräckligt många gånger så kommer du i alla fall så nära att det går att se principen.

Vill man gå vidare med både programmeringen och själva matematiken kan man utveckla exemplet med tärningarna. Ett förslag skulle kunna vara att ändra sannolikheten för de olika utfallen. Man kan dessutom göra aktiviteten roligare genom att byta ut tärningarna men behålla principen. Många ungdomar spelar olika typer av datorspel på fritiden och i många av dessa förekommer det någon form av paket som man kan få eller vinna på olika sätt. Man öppnar sedan paketen för att låsa upp nya karaktärer eller liknande. Karaktärerna är olika bra och sannolikheten att få en av de bästa är mycket mindre än sannolikheten att få en sämre. Detta motsvaras ju precis av att kasta tärningar med olika sannolikhet för utfallen. Man kan dessutom välja hur många olika utfall det ska finnas, så eleverna kan anpassa aktiviteten hur de själva vill. Sedan kan de få testa, hur många paket behöver de öppna innan de får något de vill ha? Om de öppnar 100 paket, hur många saker fick de som de är nöjda med? Här får eleverna dessutom en ny erfarenhet av programmering, att modifiera kod. Det vanligaste i undervisningen är att man antingen skriver eller använder en kod, men att ändra i befintlig kod ger ytterligare ett moment.

För att eleverna skall kunna ändra sannolikheten för de olika utfallen behöver koden se lite annorlunda ut. Man skulle kunna göra på följande sätt:

```
u=rand(0,1)
if u < 1/6, return 1
else if u < 2/6, return 2
else if u < 3/6, return 3
else if u < 4/6, return 4
else if u < 5/6, return 5
```

else return 6

Den här koden är inte särskilt ”snygg” men den är enkel att ändra i. Här slumpas först ett tal mellan 0 och 1. Sedan delar vi in intervallet i de delar vi vill ha. Koden ovan visar en vanlig tärning (alla värden har sannolikheten (1/6)). Vill man ändra till exempelvis att utfallen 1-3 bara har hälften så stor sannolikhet som 4-6 så får man ändra storleken på delintervallen. Då behöver man dessutom tänka till lite hur man skall ändra gränserna. Vi har nu tre utfall som har hälften så stor sannolikhet som tre andra. Sannolikheten att det skall bli något av utfallen är 1. Vi kan göra följande beräkningar:

$3*(p/2)+3p=1$
$9p=2$
$p=2/9$

Sannolikheterna för varje utfall är nu alltså:
1: $p/2=1/9$
2: $p/2=1/9$
3: $p/2=1/9$
4: $p=2/9$
5: $p=2/9$
6: $p=2/9$

Nu vill vi dela in intervallet på rätt sätt för att få denna sannolikhetsfördelning. Koden blir då istället:

u=rand(0,1)
if u < 1/9, return 1
else if u < 2/9, return 2
else if u < 3/9, return 3
else if u < 5/9, return 4
else if u < 7/9, return 5
else return 6

Här får eleverna dels träna på att modifiera kod men de behöver också använda matematik för att dela upp intervallet på rätt sätt. Men de får dessutom möjlighet att använda koden till något som är väldigt svårt att göra med en vanlig tärning, utforska resultaten av att kasta en tärning där det är större sannolikhet att få mindre tal. Eftersom sannolikhetsfördelningen förändrats så kommer även väntevärdet att ändras:

$E(x_1)=(1/9)*1+(1/9)*2+(1/9)*3+(2/9)*4+(2/9)*5+(2/9)*6=4$

”Kastar” vi den nya tärningen tillräckligt många gånger så kommer alltså medelvärdet av resultaten ligga nära 4, istället för 3,5 som vi fick för en vanlig tärning.
Vill man använda en tärning med ett annat antal sidor så får man dela upp intervallet i fler delar istället. Då lägger man bara in fler rader med ”else if” i koden ovan. Här kan eleverna utforska hur mycket som helst. Det går naturligtvis bra att ändra både antalet sidor och

sannolikhetsfördelningen. Om du vill kan du ha en tärning med 50 sidor där det är olika sannolikhet för varje utfall.

8 Diskussion och slutsats

Vi har nu tittat på lite olika förslag om hur man kan använda programmering för att förbättra inlärningen av matematik. I undersökningarna har man testat hur elevernas förmågor inom problemlösning, modellering, resonemang och utövning påverkas av att eleverna får arbeta med programmering. Resultaten från undersökningarna pekar på att samtliga förmågor som testades påverkades positivt av programmeringen. Ett intressant resultat var att problemlösning verkade vara den förmåga som påverkades minst av de som testades. En av frågorna vi ville få besvarade var just inom vilket område i matematiken som programmeringen bör vara med. Som jag redan nämnt så ligger programmering under just problemlösning i den svenska gymnasieskolans ämnesplan, som ett verktyg att använda vid just problemlösning. Det är väl ingen som kan säga emot att det är enklare att lösa vissa typer av problem med hjälp av programmering, men frågan är hur mycket matematik eleverna lär sig av att arbeta på det sättet. Undersökningarna visar, som vi har sett, att eleverna verkar lära sig mer av att använda sig av programmeringen. Men det verkar som att programmering är till mer nytta inom andra områden av matematiken vilket väcker en del frågor. Behövs det mer forskning om hur man kan förbättra undervisningen om problemlösning, eller är det helt enkelt så att programmering faktiskt passar bättre inom andra områden? Borde man i så fall ändra ämnesplanen och byta plats på programmeringen, eller skall man försöka använda programmeringen på fler ställen i undervisningen? Resultaten från undersökningarna som vi har studerat här pekar snarare på det senare. Eleverna verkar lära sig mer i flera olika områden där programmering har testats, så det bästa vore att försöka använda sig av programmering så mycket som möjligt, inte bara som problemlösningsverktyg utan även till annat. Däremot skulle det fortfarande behövas mer forskning om hur undervisningen skall anpassas så att även problemlösningen förbättras lika mycket som de andra förmågorna.

Hur programmeringen skall användas har vi sett olika exempel på. Ett populärt sätt är att använda det till att skapa och spela spel. Eleverna förbättrar sina matematiska kunskapar då de skapar ett spel, men framförallt är det spelandet som verkar vara ett väldigt bra sätt att lära sig matematik på. Men då är det viktigt att spelet är designat på rätt sätt, vilket vi har gått in på i detalj i en av undersökningarna. Som vi såg så finns det mängder med saker att tänka på när man skall designa ett inlärningsspel. Det är ingen idé att låta eleverna spela spel bara för sakens skull, spelen måste vara designade på ett sätt som gör att inlärningen faktiskt förbättras.
Andra sätt att använda programmeringen på är som problemlösningsverktyg och som ett verktyg för att kunna utforska matematik. Programmering som problemlösningsverktyg har vi redan pratat om, men som utforskningsverktyg verkar det som att eleverna har stor nytta av det. Programmeringen öppnar upp möjligheter som helt enkelt inte hade funnits annars. Framförallt inom sannolikhet där en stor del av matematiken bygger på att saker skall göras oändligt många gånger. Som jag redan nämnt tidigare så går det inte att göra något oändligt många gånger ens på en dator, men man kan göra saker extremt många fler gånger än vad som är rimligt att göra för hand. Om man dessutom kombinerar detta med grafer som uppdateras samtidigt som simuleringen körs så får eleverna en visuell representation som inte heller den är rimlig att göra för hand.

Det är inte särskilt många undersökningar vi har tittat på så resultaten går inte att betrakta som någon fakta. Däremot ger de indikationer på hur det faktiskt är i vissa klassrum och hur det skulle kunna vara i andra. Framförallt så väcker resultaten en del frågor som är värda att titta närmare på i framtida forskning, men de frågorna har vi redan diskuterat. En stor del av den forskning som finns tillgänglig handlar dessutom om lägre åldrar än gymnasienivå, så forskning om gymnasieklasser hade varit intressant att se i framtiden. Även om jag tror att mycket av det som det forskats om på yngre barn också är applicerbart i gymnasiet så vill man ha forskning som backar upp det.

9 Referenslista

Ainley, J. (2000) Constructing purposeful mathematical activity in primary classrooms. I C. Tikly & A. Wolf (Red.) *The maths we need now.* London: Institute of Education, Bedford Way Papers.

Ainley, J., Pratt, D., & Hansen, A. (2006). Connecting Engagement and Focus in Pedagogic Task Design. *British Educational Research Journal*, 32(1), 23–38.

Brilliant. (u.å). *Linear diophantine equations*. Hämtad 2018-10-19 från https://brilliant.org/wiki/linear-diophantine-equations-one-equation/

Brown, Q., Mongan, W., Kusic, D., Garbarine, E., Fromm, E., Fontecchio, A. (2008) *Computer aided instruction as a vehicle for problem solving: Scratch programming environment in the middle years classroom*.

Calao, L. A., Moreno-León, J., Correa, H. E., & Robles, G. (2015). Developing Mathematical Thinking with Scratch An Experiment with 6th Grade Students. I G. Conole m.fl. (Red.): *Springer International Publishing* LNCS 9307, pp. 17–27, 2015. DOI: 10.1007/978-3-319-24258-3 2

De stora talens lag. (2018, 10 januari). I *Wikipedia*. Hämtad 2018-10-15 från https://sv.wikipedia.org/wiki/De_stora_talens_lag

Euklides algoritm. (2018, 2 maj). I *Wikipedia*. Hämtad 2018-10-17 från https://sv.wikipedia.org/wiki/Euklides_algoritm

Guin, D., Ruthven, K., & Trouche, L. (2005). *The didactical challenge of symbolic calculators turning a computational device into a mathematical instrument*. New York, NY: Springer.

Harel, I. & Papert, S. (1991). *Constructionism*. Norwood, NJ: Ablex.

Hoyles, C. & Noss, R. (1987) Children working in a structured logo environment: from doing to understanding. I *Recherches en Didactiques des Mathematiques*, 8(12), 131–174.

Kafai, Y. B. (1995). *Minds in Play: Computer Game Design as a Context for Learning*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Kafai, Y.B., Franke, M.L., Ching, C. C., & Shih, J. C. (1998). Game design as an interactive learning environment for fostering students' and teachers' mathematical inquiry. I *International Journal of Computers for Mathematical Learning* 3: 149–184

Misfeldt, M., & Ejsing-Duun, S. (2015). Learning mathematics through programming: An instrumental approach to potentials and pitfalls. I *CERME 9-Ninth Congress of the European Society for Research in Mathematics Education*.

Papert, S. (1996) An exploration in the space of mathematics educations. I *International Journal of Computers for Mathematical Learning*, 1(1), 95–123

Pratt, D. & Ainley, J. (1997) The construction of meanings for geometric construction: two contrasting cases. I *International Journal of Computers for Mathematical Learning*, 1(3), 293–322

Rabardel, P., & Bourmaud, G. (2003). From computer to instrument system: a developmental perspective. *Interacting with Computers*, 15(5), 665–691. doi:10.1016/S0953- 5438(03)00058-4

Regeringen (2017). *Stärkt digital kompetens i läroplaner och kursplaner*. Hämtad 2018-10-04 från https://www.regeringen.se/pressmeddelanden/2017/03/starkt-digital-kompetens-i-laroplaner-och-kursplaner/

SOU 2014:13. *En digital agenda i människans tjänst – en ljusnande framtid kan bli vår*. Stockholm: Fritzes Offentliga Publikationer.

Utbildningsdepartementet (2015). *Uppdrag att föreslå nationella it-strategier för skolväsendet*. Stockholm.

Wolfram. (u.å). *Euclidean algorithm*. Hämtad 2018-10-19 från http://mathworld.wolfram.com/EuclideanAlgorithm.html