SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Avgörbara och oavgörbara problem - en analys utifrån strategiska spel

av

Yue Su

2025 - No L 15

Avgörbara och oavgörbara problem - en analys utifrån strategiska spel

Yue Su

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Per Alexandersson

2025

Abstract

This paper presents the concepts of decidability and undecidability within the context of computability theory. The main aim is to provide a fundamental understanding of these concepts by analyzing both mathematical problems and strategic games—namely **Magic: The Gathering** and **Dominion**. Central focus is placed on the notion of undecidability: the classical Halting Problem, and Turing machines are introduced. Furthermore, the case where Magic: The Gathering has been proven to be Turing-complete and undecidable is presented to deepen the conceptual understanding of undecidability. Finally, the significance of an undecidable game like **Magic: The Gathering** is discussed, along with how the employed proof technique may inspire further investigation of the undecidability problem in **Dominion**.

Sammanfattning

I denna uppsats undersöks begreppen avgörbarhet och oavgörbarhet inom beräkningsteori. Huvudsyftet är att ge en grundläggande förståelse för dessa begrepp genom att analysera både matematiska problem och de strategiska spelen **Magic: The Gathering** och **Dominion**. Begreppet oavgörbarhet står i centralt fokus och det klassiska oavgörbara stopproblemet samt Turingmaskiner presenteras. Vidare redovisas fallet då spelet **Magic: The Gathering** visats vara Turingkomplett och oavgörbart, vilket fördjupar förståelsen av oavgörbarhet. Avslutningsvis diskuteras betydelsen av oavgörbart i **Magic: The Gathering**, samt hur den använda bevismetoden kan väcka intresse och inspirera till vidare undersökning av oavgörbarhetsproblem i **Dominion**.

Innehåll

1 Introduktion

Strategispel är en del av vardagen för många och kombinerar ofta underhållning med strategiskt tänkande. Det finns studier som visar att spel har djupa kopplingar till matematik och beräkningslogik [HD09]. Strategispelet **Magic: The Gathering** är ett av världens mest populära kortspel och har visats vara så komplext att det kan simulera en universell Turingmaskin. Detta innebär att dessa utgång i vissa fall är oavgörbara.

Syftet med denna uppsats är att ge en grundläggande förståelse för begreppen avgörbarhet och oavgörbarhet inom beräkningsteori. Genom att analysera både matematiska problem och strategispel **Dominion** och **Magic: The Gathering**.

Uppsatsen inleds med en genomgång av vad avgörbarhet respektive oavgörbarhet innebär, illustrerat med klassiska begrepp Stopproblemet och Turingmaskin. Därefter presenteras hur spelet Magic: The Gathering kan konstrueras till en universell Turingmaskin.

Slutligen reflekteras kring möjligheterna att överföra liknande analogi i **Magic: The Gathering** till spelet **Dominion**. Med dessa bakgrund ställs följande frågeställning: Vilka delar av spelet Dominion skulle teoretiskt kunna simulera en universell Turingmaskin utgår från analogi till fallet **Magic: The gathering**?

Notera: Texten har delvis tagit hjälp av ChatGPT för att granska grammatiska avvikelser och omformelera meningar.

Jag har till exempel använt ChatGPT för att fråga vilket ord är mest korrekt och lämpligt, såsom 'kungarikekort eller kungarikeskort' 'cell eller ruta' och 'totalpoäng eller total poäng'.

Ibland har också använt verkteyfet att hjälpa mig omformulera meningar, till exempel 'Hjälp mig formulera om meningen 'Men det faktum att det finns enskilda fall där problem är enkelt att avgöra innebär inte att det alltid går att avgöra dödlighetsproblemet för godtyckliga instanser.'

2 Avgörbara och oavgörbara problem

2.1 Avgörbara problem

Begreppen avgörbarhet och oavgörbarhet används inom matematisk logik och teoretisk datavetenskap för att analysera lösbarhet bland beslutsproblem [Wik25]. Ett beslutsproblem handlar om att undersöka om det finns en algoritm som kan lösa alla möjliga instanser av ett givet problem. Det finns två möjliga utfall: antingen existerar en algoritm som löser beslutproblemet, eller så gör det inte det. I det senare fallet är problemet oavgörbart [Dav82]. Med andra ord handlar det om huruvida det går att avgöra om ett problem är sant eller falskt, alltså att algoritmen kan ge ett korrekt svar för alla möjliga indata, inom en ändlig (begränsad) tid. Ett avgörbart problem utgör en klass av beslutsproblem där det existerar en sådan algoritm som alltid ger ett korrekt och slutligt ja/nej-svar för alla möjliga indata [Gao19].

En klass av avgörbara problem inom matematiken:

”Är ett heltal N delbart med 5?” är ett avgörbart problem. För varje heltal kan vi direkt svara ja/nej med hjälp av algoritmen:

- dela talet med 5.
- Om resten är 0 är svaret ja, annars nej.

På ett mer matematiskt sätt uttrycks detta så här: Givet ett godtyckligt heltal N, om $N \bmod 5 = 0$ är svaret ja, annars nej.

Vi kan även svara på problemet med datorprogram. Här är ett exempel på Python-kod som avgör om ett tal är delbart med 5:

```
1 def divisible_by_five(tal):
2     tal = int(input("Write a number: "))
3     if tal % 5 == 0:
4         return f"{tal} is divisible by 5."
5     else:
6         return f"{tal} is not divisible by 5."
7
8 print(divisible_by_five())
```

Att avgöra om ett heltal N är delbart med 5 utgör ett exempel på ett avgörbart problem. För varje givet heltal kan vi entydigt svara på frågan med ’ja’ eller ’nej’ genom denna algoritm: vid division av N med 5, är N delbart med 5 om och endast om resten är 0; i annat fall är det inte delbart. Här menar vi inte bara att en enskild instans (till exempel konstaterar att $N = 15$ är delbart med 5) är sann utan att hela klassen av beslutproblem: För vilket heltal som helst avgöra om det är delbart med 5. Det är alltså hela klassen av problem som är avgörbar, inte bara ett enstaka tal.

I vardagligt språkbruk används ofta begreppen avgörbart och oavgörbart något slarvigt, där man säger att ett problem är (o)avgörbart när man egentligen avser en hel klass

av beslutsproblem. I den här uppsatsen används ibland (o)avgörbart problem i denna förenklade form.

Vi fortsätter med ytterligare ett exempel på ett avgörbart matematiskt beslutproblem:

Givet en graf, avgör om det finns en stig som besöker alla hörn exakt en gång.

Detta är det så kallade *Hamiltonstigproblemet*: Givet en graf $G = (V, E)$, finns det en stig i G som besöker varje hörn V exakt en gång? Vi vill visa att detta är ett avgörbart problem, det vill säga att det finns en algoritm som för varje given graf alltid kan avgöra om den innehåller en sådan stig.

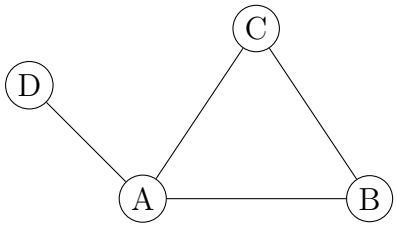
Vi börjar med två konkrekta exempel:

- Graf $G_1 = (V, E)$, där

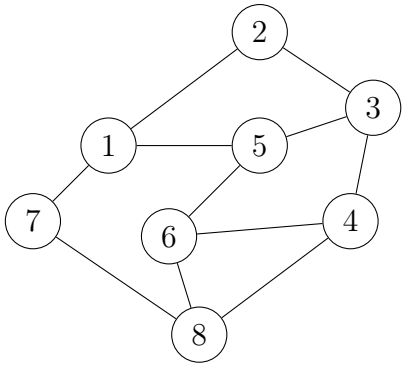
$$V = \{A, B, C, D\} \ , \ E = \{AB, AC, AD, BC\}$$

- Graf $G_2 = (V, E)$, där

$$V = \{1, 2, 3, 4, 5, 6, 7, 8\} \ , \ E = \{12, 15, 17, 23, 34, 35, 46, 48, 56, 68, 78\}$$



Grafen G_1



Grafen G_2

Vi ritar grafer och ser att vi i grafen G_1 kan läsa av svaret direkt i figuren. De uppfyllda stigarna är $D \to A \to C \to B; D \to A \to B \to C; B \to C \to A \to D; C \to B \to A \to D$.

I grafen G_2 blir det betydligt svårare att hitta en stig som besöker alla hörn exakt en gång. Eftersom det för större grafer snabbt blir oöverskådligt att manuellt avgöra om en sådan stig existerar.

För att lösa detta problem använder vi följande algoritm:

Steg 1: Grafen har ett ändligt antal hörn V. Vi kan generera alla möjliga permutationer av hörnen, vilket ger $|V|!$ olika ordningar.

Steg 2: För varje sådan permutation kontrollerar vi om det finns kanter mellan varje par intilliggande hörn i ordning.

Steg 3: Om minst en permutation bildar en sammanhängande stig svarar algoritmen 'ja', annars 'nej'.

Vi använder Python för att implementera algoritmen:

```
from itertools import permutations
vertices={1, 2, 3, 4, 5, 6, 7, 8}
edges= {(1,2), (1,5), (1,7),
(2,3), (3,4), (3,5), (4,6),
(4,8), (5,6), (6,8), (7,8)}

#union since edge can be traversed in both directions
edge_set=edges.union ({(b,a) for (a,b) in edges})

def is_hamilton_path(path):
    for i in range(len(path)-1):
        if (path[i], path[i+1]) not in edge_set:
            return False
    return True

num_solutions = 0
for perm in permutations(vertices):
    if is_hamilton_path(perm):
        print(f"{perm} is a Hamiltonian Path")
        num_solutions += 1
print(f"{num_solutions} Hamiltonian path(s) exists")
```

När vi kör koden ovan får vi ut svaret för grafen G_2. Koden skriver ut vilka stigar som finns och räknar fram att det totalt finns 54 Hamiltonstigar.

Sammanfattningsvis är frågan *'Givet en graf, avgör om det finns en stig som besöker alla hörn exakt en gång'* avgörbar, eftersom det finns en algoritm som alltid kan ge ett korrekt svar för alla instanser.

Avgörbarhet i ett strategispelet Dominion

Dominion är ett populärt strategispel, och den första versionen släpptes 2008. Denna version utgör det ursprungliga grundspelet i **Dominion** [Dom25]. Här diskuterar vi bara grundspelet i **Dominion** när det gäller avgörbarhet.

Grundspelet består av totalt 500 spelkort och fördelade på fyra huvudtyper: 252 kungarikekort, 130 skattkort, 48 poängkort och 30 förbannelsekort. Spelet börjar med tio högar av tio olika kungarikekort, som vanligtvis väljs slumpmässigt inför varje spelomgång. Utöver dessa finns skattkort (koppar, silver och guld), segerkort (värda ett, tre, sex poäng) och förbannelsekort (minus ett poäng). Det innebär att en typisk spelomgång i grundspelet startar med 17 olika korthögar [Dom25].

Enligt spelets regler avslutas en spelomgång antingen när högen med provincekort (värda sex poäng) är slut, eller när tre av de olika korttyperna på spelbordet är slut. När detta sker summerar spelarna sina poäng och den spelaren med högst totalpoäng vinner [Dom25].

Eftersom antalet kort i grundspelet är begränsat till summan 500; det finns inte några kort eller mekanismer som kan förlänga spelet, är antalet spelomgångar ändligt. Även varje spelares möjliga handlingar är begränsade. Detta leder till att varje spelomgång närmar sig sitt slut. Vilket innebär att grundspelet i **Dominion** är avgörbart eftersom det alltid avslutas efter ett ändligt antal drag med en vinnare, förutsatt att alla spelare spelar aktivt.

2.2 Oavgörbara problem

En klass av oavgörbara problem definieras generellt som problem för vilka det inte existerar någon algoritm som kan avgöra alla möjliga fall inom ändlig tid. Det innebär att det inte går att konstruera ett datorprogram, som för varje tänkbar indata alltid kan ge ett korrekt och slutgiltigt svar: 'ja' eller 'nej'. Ett klassiskt exempel är *Stopproblemet*, där man inte kan avgöra om ett givet datorprogram stannar för en viss indata. Denna definition avser den algoritmiska oavgörbarheten. [Pon12]

Enligt [Pon12] används oavgörbarhet även i en annan bemärkelse, nämligen logisk oavgörbarhet. Att ett enskilt påstående är logiskt oavgörbart innebär att det inte kan bevisas eller motbevisas utifrån kraftfulla axiomsystem. Exempel på detta är Kurt Gödels ofullständighetssatser, som visade att i kraftfulla axiomsystem finns även sanna påståenden som inte kan bevisas inom systemet. Det innebär att ett axiomssystem inte kan vara både konsistent och fullständigt [Pon12].

Alonzo Church och Alan Turing formaliserade begreppet algoritm på 1930-talet [Chu36] [T+36], och visade att vissa beslutsproblem inte kan lösas med någon algoritm, sådana problem är oavgörbara i algoritmisk mening. I ett matematiskt sammanhang innebär ett oavgörbarhetsproblem att det inte kan avgöras genom logisk härledning, från givna axiom eller genom databeräkning [Pon12].

2.3 Stopproblemet

Stopproblemet är ett klassiskt exempel på ett oavgörbart problem inom matematisk logik och teoretisk datalogi. Det formuleras genom följande fråga: *Givet ett datorprogram och godtyckligt indata, kan man avgöra om programmet kommer att stanna eller fortsätta köra för alltid* [Pon12].

I sin artikel förklarar Poonen att Alan Turing år 1936 bevisade att detta problem är oavgörbart. Det innebär att det inte finns någon algoritm som i alla tänkbara fall kan avgöra om ett givet program kommer att avslutas eller köra för evigt. Med andra ord existerar inget universellt datorprogram som fullt korrekt kan avgöra om varje möjlig indata stannar eller inte.

Innan vi tittar närmare på Turings bevislogik är det bra att förstå begreppet **självrefererande paradox**. En självrefererande paradox uppstår när ett påstående eller ett program refererar till sig själv och det leder till en motsägelse. Sådana typer av självreferens skapar ofta logiska problem eftersom påståendet måste vara både sant och falskt

om villkoren ska uppfyllas.

Ett känt exempel på självrefererande paradox är den så kallade **barberarparadoxen** som formuleras så här [Wik24a]:

En frisör rakar de som inte rakar sig själva. Frågan är: rakar frisören sig själv?

Oavsett vilket svar man väljer kommer det att uppstå en logisk motsägelse:

1. Om frisören rakar sig själv, bryter han mot regeln att han endast rakar de som **inte rakar** sig själva, det blir en motsägelse.
2. Om frisören inte rakar sig själv, bryter han mot regeln att han **borde raka** sig själv, vilket också är en motsägelse.

Vi ser alltså att båda svaren leder till en logisk paradox. Detta visar effekten av självreferens, där ett uttryck/påstående som handlar om sig själv inte kan besvaras utan att skapa motsägelser [Wik24b].

Nu återvänder vi till hur Turing bevisade att Stopproblemet är oavgörbart. Turings bevis bygger på paradoxen ovan. Han antog att ett sådant program finns och konstruerade därefter programmet som tar sig själv som indata, vilket skapar en självrefererande paradox och därmed en motsägelse. Genom detta visar han att Stopproblemet måste vara oavgörbart [Pon12].

Formulering av Stopproblemet och bevisidé:

Vi antar att det finns ett superprogram P som för varje godtycklig indata X kan avgöra om X stannar eller kör vidare för evigt.

Vi definierar att:

- P(X) returnerar **Sant** om X stannar.
- P(X) returnerar **Falskt** om X kör för evigt.

Nu antar vi att det finns ett program P_1 som för godtyckligt indata stannar om indatat är falskt och kör för evigt om indatat är sant.

Vi definerar:

- $P_1(X)$ stannar om X är falskt.
- $P_1(X)$ kör för evigt om X är sant.

Men om vi matar in P(X) som indata till P_1 får vi P_1(P(X)) och eftersom X är godtyckligt indata inkluderas även P_1(X) i det. då får vi P_1(P(P_1(X))). Så P_1 stannar om P är falskt, annars kör den för evigt och P är sant om P_1 stannar annars den är falsk. Detta är en

motsägelse. Så vi drar slutsatsen att ett sådant program P inte kan existera. Därmed är Stopproblem oavgörbart. Som ett fullständigt bevis definierade Alan Turing programmet (algoritmen) som nu kallas Turingmaskinen.

2.4 Turingmaskin

2.4.1 Definition av Turingmaskin

Alan Turing formaliserade 1936 vad det innebär att räkna 'mekaniskt' genom sin artikel *On computable numbers, with an application to the Entscheidungsproblem* där han introducerade en abstrakt maskinmodell som senare kom att kallas Turingmaskin [T$^+$36]. En Turingmaskin är ingen fysisk apparat utan en teoretisk konstruktion som beskriver hur en beräkningsprocess kan utföras steg för steg.

Följande beskrivning utgår från denna artikel: En Turingmaskin består av tre centrala komponenter: Styrverk (kontrollenhet), ett oändligt band och ett läs- och skrivhuvud.

Styrverket innehåller ett ändligt antal tillstånd. Beroende på vilket tillstånd maskinen är i och vilken symbol den läser från bandet, avgör styrverket nästa operation. Det vill säga vilken symbol som ska skrivas; i vilken riktning läs- och skrivhuvudet ska förflyttas (vänster eller höger); vilket tillstånd maskinen ska övergå till härnäst.

Det oändliga bandet fungerar som maskinens minne. Det är uppdelat i en följd av celler där varje cell kan innehålla en symbol. Symbolen kan antingen vara en tomsymbol eller övriga symboler. Varje cell i bandet kan skrivas över eller raderas.

Läs- och skrivhuvudet är maskinens verktyg för att läsa och manipulera bandet. Det kan läsa en symbol, skriva en ny symbol och förflytta sig ett steg åt vänster eller höger beroende på instruktionen från styrverket.

I en Turingmaskin börjar beräkningen i ett specifikt starttillstånd och huvudet placeras i en given cell. Maskinen fortskrider stegvis genom att läsa symboler, manipulera bandet och uppdatera sitt tillstånd enligt definierade regler. När maskinen når ett tillstånd där ingen regel finns för vad som ska göras, stannar den; annars fortsätter den att arbeta.

För att ge en fullständig formell beskrivning definieras en Turingmaskin som en sjutupel enligt följande [Ri2, Sat]:

- Q är en ändlig mängd tillstånd, $Q \neq \emptyset$.
- q_0 är starttillstånd, $q_0 \in Q$.
- F är mängden sluttillstånd, $F \subset Q$
- Σ är en ändlig mängd av inmatningssymboler.
- Γ är en ändlig mängd av alla bandets symboler, med $\Sigma \subset \Gamma$ och $\Gamma \neq \emptyset$.
- B betyder en blank cell på bandet och motsvarar en tom symbol, $B \in \Gamma$ och $B \notin \Sigma$.

- δ är övergångsfunktion och definieras som

$$\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\},$$

där L innebär att läs- och skrivhuvudet flyttas ett steg åt vänster, R ett steg åt höger.

Övergångsfunktionen δ är den mest komplicerade delen av en Turingmaskin. Denna motsvarar själva programmet eller algoritmen. Funktionen tar in det aktuella tillståndet och symbolen på bandet och retunerar det nya tillståndet, den symbol som ska skrivas samt riktningen som ska flyttas. Turingmaskinen stannar när det når ett tillstånd där $q \in F$ eller funktionen δ är odefinierad. De övriga tupelkomponenterna fungerar som parametrar i övergångsfunktionen.

2.4.2 Ett konkret exempel: addition av 5+3 med Turingmaskin:

För att illustrera hur en Turingmaskin fungerar i praktiken använder vi ett konkret exempel 5+3. Talet 5 representeras som fem '1'-symboler och talet 3 som tre '1'-symboler, och symbolen 'M' används som avgränsare däremellan. Enligt den formella definitionen är då indata

$$\Sigma = \{1, 1, 1, 1, 1, M, 1, 1, 1\} \ .$$

Symbolen 'B' markerar tomma celler på bandet. I starttillstånd q_0 ser ett utsnitt av bandet ut så här:

Tabell 1: Bandets symboler i starttillstånd q_0

...	B	B	1	1	1	1	1	M	1	1	1	B	B ...

Målet med Turingmaskinen är att ta bort 'M' och föra över varje '1' från högersidan till vänstersidan, så att resultatet $5 + 3 = 8$ slutligen framställs som åtta '1'-symboler i följd. För att utföra beräkningen definieras fyra tillstånd $Q = \{q_0, q_1, q_2, q_3\}$, som tillsammans styr läsning, skrivning och riktning:

q_0: Starttillstånd, Maskinen går åt höger steg för steg för att hitta gränssymbolen 'M' mellan de två talen.

q_1: När maskinen passerar 'M', letar den efter en '1' till höger om 'M' och ersätter den med 'M'. När det inte finns flera '1' kvar till höger om 'M' (utan bara blank cell), byter maskinen till q_3.

q_2: Maskinen rör sig till vänstersidan och skriver över 'M' med '1'. Därefter återvänder den till q_o för att upprepa processen.

q_3: Maskinen läser 'M' och tar bort den när ingen mer '1' behöver flyttas.

Tabell 2: Övergångstabell för Turingmaskinen

Tillstånd (Q)	**Läst symbol (Γ)**	**Nästa tillstånd (Q)**	**Skriven symbol (Γ)**	**Riktning**
q_0	1	q_0	1	höger
q_0	M	q_1	M	höger
q_1	1	q_2	M	vänster
q_1	B	q_3	B	vänster
q_2	1	q_2	1	vänster
q_2	M	q_0	1	höger
q_3	M	halt	B	-

Illustration av Turingmaskinens beräkning steg för steg:

I sitt starttillstånd läser maskinen q_o '1' från vänster och läs-skrivhuvudet flyttas till höger.

Tabell 3: Maskinen startar i tillståndet q_0

1	1	1	1	1	M	1	1	1
↓								
q_0								

När maskinen läser 'M', kommer den att skriva 'M' och byter tillstånd till q_1.

Tabell 4: Maskinen byter tillstånd från $q_0$till q_1

1	1	1	1	1	M	1	1	1
				↓				
				$q_0 \rightarrow q_1$				

När maskinen når sitt sluttillstånd q_3 har alla ettor flyttats till vänster om M:

Tabell 5: Maskinens beräkning i tillstånd q_3

...	B	B	1	1	1	1	1	1	1	1	M	B	...
											↓		
											q_3		

...	B	B	1	1	1	1	1	1	1	1	B	B	...
											↓		
											Halt		

De åtta ettorna motsvarar resultatet 8, vilket betyder att additionen är korrekt genomförd i Turingmaskinen.

2.4.3 Universell Turingmaskin

Som nämns ovan används Stopproblemet i bevis genom att formulera ett självrefererande paradox via en universell maskin(program), vilket visar att Stopproblemet är oavgörbart.

En universell Turingmaskin (UTM) är en Turingmaskin som kan simulera vilken annan Turingmaskin som helst, med godtycklig indata [Ri2]. Det innebär att UTM kan läsa av en Turingmaskins tillstånd Q och symboler Γ, därefter utgöra samma beräkning.

Det finns många olika konstruktioner av UTM. Här presenterar vi ett exempel av Yurii Rogozhin. Han upptäckte en UTM med fyra tillstånd och sex symboler. Vi betecknar tillståndsmängden

$$Q = A, B, C, D$$

och bandets symboler

$$\Gamma = 0, 1, 2, 3, 4, 5.$$

Nedan visar tabellen övergångsfunktionen för denna (4,6)-UTM [Ri2].

Tabell 6: Övergångsfunktionen i (4,6)-UTM

Γ \ Q	A	B	C	D
0	3,L,A	4,R,B	0,R,C	4,R,D
1	2,R,A	2,L,C	3,R,D	5,L,B
2	1,L,A	3,R,B	1,R,C	3,R,D
3	4,R,A	2,L,B	HALT	HALT
4	3,L,A	0,L,B	5,R,A	5,L,B
5	4,R,D	1,R,B	0,R,A	1,R,D

Vi undersöker närmare hur övergångsfunktion i (4,6) UTM fungerar. Till exempel när maskinen befinner sig i tillstånd B och läser symbolen 4, då hittar vi i tabellen under kolumn B och rad 4 övergången `0,L,B`. Det betyder att läs-skrivhuvudet skriver över nuvarande symbolen med'0', sedan flyttar ett steg åt vänster och går över i tillstånd B. Därefter fortsätter beräkning i detta tillstånd.

När (4,6)-UTM befinner sig i tillstånd C och läser symbolen 3 eller i tillstånd 'D', läser symbolen 3 sker övergången `HALT`. Eftersom maskinen inte har definierat övergångar för dessa kombinationer av tillstånd och symboler; stannar den så snart den når ett stående odefinierat läge [Ri2].

2.5 Oavgörbara problem i matematiksammanhang

När vi har bekantat oss med begreppen (o)avgörbarhet, Stopproblem och Turingmaskin, kan vi gå vidare och undersöka andra exempel på oavgörbara problem i matematiska sammanhang. Syftet är att fördjupa förståelsen bättre av dessa begrepp samt skillnaden mellan avgörbara och oavgörbara problem.

2.5.1 Dödlighetsproblemet för matriser

Vi börjar undersöka ett intressant problem som kallas för *Dödlighetsproblemet för matriser*, det formuleras så här:

Givet två 3×3 heltalsmatriser, finns det en ändlig produkt (en sekvens av multiplikationer) av dessa matriser som reulterar i en nollmatris?

I vissa enkla fall är svaret tydligt ja. Exempelvis kan vi hitta två singulära matriser (matriser vars determinant är noll och saknar invers) vilkas produkt ger nollmatrisen:

$$M_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 4 \end{bmatrix}, M_2 = \begin{bmatrix} 3 & 4 & 0 \\ 10 & 9 & 0 \\ 6 & 1 & 0 \end{bmatrix}.$$

Här är det inte svårt att kontrollera att produkten M_2M_1 ger

$$M_2M_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

I detta specifika fall är svaret alltså 'ja'. Men det faktum att det finns enskilda fall där problem är enkelt att avgöra, innebär inte att dödlighetsproblemet alltid går att avgöra för godtyckliga instanser. Som tidigare nämnt handlar begreppet (o)avgörbarhet om huruvida det existerar en algoritm som kan lösa alla instanser inom en hel klass av problem, inte enbart enskilda fall. Därför krävs det ett formellt fullständigt bevis för att avgöra om det finns någon algoritm som alltid kan avgöra dödlighetsproblemet för två godtyckliga 3×3 heltalsmatriser.

2.5.2 Det oavgörbara dödlighetsproblemet för matriser

Att avgöra dödlighetsproblemet för två stycken 3×3 heltalsmatriser är fortfarande ett öppet problem. Däremot har [HHH07] visat att dödlighetsproblemet blir oavgörbart om man istället betraktar semigrupper genererade av sju stycken 3×3 heltalsmatriser. Deras bevis bygger på en reduktion, vilket innebär att dödlighetsproblemet för sju 3×3

heltalsmatriser kopplas till ett redan känt oavgörbart problem. Genom att visa att en lösning på det ena problemet skulle medföra en lösning på det andra kan man etablera att båda problemen är lika svåra. I detta fall reduceras dödlighetsproblemet för sju stycken 3×3 heltalsmatriser till två välkända oavgörbara problem: Post Correspondence Problem (PCP) och Turingmaskinsproblem [HHH07].

Post Correspondence Problem (PCP) definieras av två listor av strängar $\alpha_1 \ldots, \alpha_N$ och $\beta_1, \ldots, \beta_N$ över ett alfabet. Frågan handlar om att avgöra om det finns en sekvens av index $(i_1, \ldots, i_k)$ så att $\alpha_{i_1}, \ldots, \alpha_{i_K} = \beta_{i_1}, \ldots, \beta_{i_K}$. PCP är oavgörbart, vilket innebär att det inte finns någon algoritm som kan avgöra om en sådan sekvens existerar. Bevis för detta görs genom att simulera problemet på en Turingmaskin. På grund av detta används PCP ofta som utgångspunkt vid reduktionsbevis för att visa att andra problem också är oavgörbara [con24].

[HHH07] formulerar dödlighetsproblemet för matriser med hjälp av en fix semigrupp S, genererad av matriser $M_2, M_3 \ldots, M_7$. För en given 3×3 heltalsmatris M_1 är frågan om det existerar en matris $M \in S$ så att elementet $(M_1 M)_{11} = 0$.

För att visa detta använder de en reduktion, vilket innebär att man visar dödlighetsproblemet för matriser är minst lika svårt som PCP. Här används en särskild variant av PCP kallad *Claus' instans*.

PCP kan definieras även som följande: givet två morfismer $h, g : \Sigma^* \to \Gamma^*$, avgör om det finns en icke-tom sträng $w \in \Sigma^+$ så att $h(w) = g(w)$. I Claus' instans har vi $\Sigma = \{b_1, b_2, ..b_7\}, \Gamma = \{a_2, b_3\} = (a, b)$. Man inför dessutom en ny symbol a_1. Låt $\Delta = \{a_1, a_2, a_3\}$. De visar att minimala lösning till denna Claus' instans är av formen $b_1 w b_7$, där w inte innehåller b_1 och b_7, det vill säga $w \in \{b_2, b_3 ... b_6\}^*$.

Låt $M_2, M_3 ... M7$ vara fixerade medan M_1 är den som varierar. En kodningsfunktion γ används för att avbilda par av strängar till en unik 3×3 heltalsmatris enligt:

$$M_1 = \gamma(h(b_1), a_1 g(b_1)), M_i = \gamma(h(b_i), g(b_i)) \quad \text{för } i = 2, \ldots, 7.$$

Genom denna kodning motsvarar matrisen M_1 bokstaven b_1, och varje M_i för $i = 2 \ldots, 7$ motsvarar bokstäverna $b_2, \ldots b_7$.

Produkten av matriserna

$$N = M_{j_1} M_{j_2} \cdots M_{j_n} \in M$$

motsvarar strängen

$$w = b_{j_1} b_{j_2} \ldots b_{j_n}.$$

Enligt konstruktionen gäller att elementet i positionen $1, 1$ i produkten N ges av uttrycket:

$$N_{11} = 3^{|u|} + \sigma(u) - \sigma(v) = \sigma(a_1 u) - \sigma(v),$$

där $u = h(w) \in \Gamma^*$, och $v \in \Delta^*$, och σ är injektiv kodningsfunktion av strängar till tal. Då gäller att $N_{11} = 0$ om och endast om $v = a_1 u$, vilket i sin tur betyder att $h(w) = g(w)$. Eftersom lösningen $h(w) = g(w)$ är ett oavgjort problem i PCP, följer direkt att det är oavgörbart om det finns någon matrisprodukt $M \in S$ sådan att $(M_1 M)_{11} = 0$. Därmed är dödlighetsproblemet för denna klass av sju stycken 3×3 heltalsmatriser oavgörbart.[HHH07]

3 Det oavgörbara strategispelet Magic: The Gathering

När vi nu har bekantat oss med begrepp som Turingmaskiner, Stopproblemet och oavgörbarhet, blir det särskilt intressant att undersöka hur dessa idéer dyker upp i oväntade sammanhang; till exempel strategispel. Ett fascinerande fall är Magic: The Gathering (ofta kallat Magic) som har bevisats vara oavgörbart genom att spelets regler och kortmekanik kan simulera en universell turingmaskin. I följande ska vi titta närmare på hur dessa begrepp tillämpas i det verkliga spelet med utgångspunkt i Churchill, Biderman och Herricks artikel *Magic: The gathering is Turing complete* [CBH19]. Beskrivning i detta kapitel baseras huvudsakligen på deras artikel.

Magic är ett populärt samlarkortspel, till början designat av matematikern Richard Garfield och lanserat 1993. Spelet kombinerar strategi, resurshantering och dueller mellan två eller flera spelare. Det ses som ett nollsummestokastiskt kortspel med ofullständig information [Mag25]. Det unika är att spelare bygger sina egna kortlekar från över 20 000 unika kort. På grund av spelets mångfacetterade strategier, komplexa regler och slumpmässighet används det även som testfall inom forskning om artificiell intelligens [CBH19].

För att visa att spelet Magic är Turingkomplett beskriver [CBH19] hur spelets kort och regler kan simulera en Turingmaskin i ett parti mellan de fiktiva spelarna Alice och Bob. I denna del analyseras hur komponenterna i en Turingmaskin, såsom det oändliga bandet, symboler, läs-skrivhuvud och tillstånd representeras i spelet. Genom detta illustreras hur ett strategispel kan uttrycka lika komplexa beräkningsmodeller som Turingmaskiner.

3.1 Simulering av det oändliga bandet

I Turingmaskinens modell utgörs minnet av ett oändligt band uppdelat i diskreta celler. I **Magic** kan detta band simuleras med hjälp av ett stort antal utvalda varelsetokenkort. Dessa tokens är permanenta och ingår inte i kortleken, vilket gör dem väl lämpade att representera bandets celler.

Varje tokenkort motsvarar en cell i bandet. Tokenkortens styrka och svaghet används för att indikera avståndet från läs- och skrivhuvudet: gröna tokens representerar celler till vänster om läs- och skrivhuvudet, medan vita tokens representerar celler till höger. Exempelvis motsvarar en 2/2-token den cellen som läs- och skrivhuvudet står på. Medan en grön 3/3-token representerar ett steg till vänster. I simuleringen startar huvudet i mitten av bandet. När huvudet flyttas mot en symbol som ligger i bandets ytterkant - vänster eller höger, utlöses skapande av en ny likadan symbol och en tom cell i den

riktningen. På så sätt utvidgas bandet potentiellt oändligt [CBH19].

3.2 Läs-och skrivhuvudet

Läs- och skrivhuvudet 'läser' genom att tvinga Alice använda kortet *Infest* som ger alla varelser -2/-2. Det gör att symbolen med 2/2 dör, vilket i sin tur utlöser kortet *Rotlund Reanimator* att skapa en ny 2/2-token med samma egenskaper som den tidigare symbolen. På så sätt flyttas huvudet vänster eller höger.

För att positionera vilka token som finns till vänster (-1/-1) eller höger (+1/+1) om huvudet används först kortet *Cleansing Beam* (gör 2 skada på alla tokens i en färg) alternativt kortet *Vigor/Soul Snuffers* (hindrar skada och ger token +2/+2) som markörer. Detta är ett sätt att representera bandets struktur.

3.3 Symboler på bandet och deras representation via Magic-kort

Magic kan implementera Rogozhins (2,18) universella Turingmaskin [Rog96], där 2 anger antalet tillstånd $Q = \{q_1, q_2\}$ och 18 anger antalet inmatningssymboler i bandet.

För att representera symbolerna i (2, 18) UTM valde [CBH19] 18 varelsetyper från spelet, där varje varelse motsvarar en symbol i bandet. Urvalet sker alfabetiskt för att underlätta identifiering [CBH19]. De 18 varelsetyperna är Aetherborn, Basilisk, Cephalid, Demon, Elf, Faerie, Giant, Harpy, Illusion, Juggernaut, Kavu, Leviathan, Myr, Noggle, Orc, Pegasus, Rhino och Sliver. Exempelvis representerar kortet Aetherborn symbol 1 och kortet Demon representerar symbol 4. I fallet utgör de 18 varelserna indata i mängden Σ enligt den formella definitionen av Turingmaskin som presenterats tidigare.

Tabell 7: Mappning mellan varelsetyper och symboler på Turingmaskinens band

Aetherborn	Basilisk	Cephalid	Demon	...	Pegasus	Rhino	Sliver
1	2	3	4	...	16	17	18

Notera: På Turingbandet anges endast siffrorna 1-18.Siffror motsvarar respektive varelsetyper i bandet

Om man ser en grön 4/4 Aetherborn-token (se bilden (a) nedan) återfinns på spelbrädet innebär det att symbolen '1' finns i den andra cellen till vänster om läs- och skrivhuvudet. Nedan visas också tre ytterligare kort som är viktiga för att simulera Turingmaskinen.

(a) Svart 4/4 Aetherborn varelse

(b) Markör för tillståndet q_1

(c) Stoppvilkoret

(d) Skapa spelbrädets initiala tillstånd

När (2,18) UTM börjar se spelkort ut så här:

Tabell 8: Spelbrädets tillstånd när (2, 18) UTM börjar

Kort	**Kontrolleras av**	**Noteringar**
29 Rotlung Reanimator	Bob	enligt Tabell 7
7 Xathrid Necromancer	Bob	enligt Tabell 7
36 Cloak of Invisibility	Alice	på alla ovan varelser
Wheel of Sun and Moon	Alice	på Alice
Illusory Gains	Alice	på senast skapade token
Steely Resolve	Alice	*Assembly-Worker*
2 Dread of Night	Alice	*Black*
Fungus Sliver	Alice	*Incarnation*
Rotlung Reanimator	Alice	Lhurgoyf, black, Cephalid
Rotlung Reanimator	Bob	Lhurgoyf, green, Lhurgoyf
Shared Triumph	Alice	Lhurgoyf
Rotlung Reanimator	Alice	Rat, black, Cephalid
Rotlung Reanimator	Bob	Rat, white, Rat
Shared Triumph	Alice	Rat
Wild Evocation	Bob	
Recycle	Bob	
Privileged Position	Bob	
Vigor	Bob	
Vigor	Alice	
Mesmeric Orb	Alice	
Ancient Tomb	Alice	
Prismatic Omen	Alice	
Choke	Alice	
Blazing Archon	Alice	
Blazing Archon	Bob	

3.4 Tillstånd

I den simulerande Turingmaskinen av **Magic** definieras två tillstånd:

q_1: Kortet *Rotlung Reanimator* befinner sig på spelplanen. I detta tillstånd kan Turingmaskinen till exempel läsa symbolen 1 *Aetherborn*, skriva symbol 18 och därefter flytta sig åt vänster.

q_2: Kortet *Xathrid Necromancer* är aktuellt i spelplanen. Kortet *Cloak of Invisibility* används som en mekanism för att växla mellan q_1 och q_2.

Tillståndsbyten styrs enligt regler, vid byte av tillstånd spelas 3 omgångar, när tillstånd inte byts spelar 4 omgångar.

När stannar maskinen? När kortet *Coalition Victory* spelas, i omgång 3 när tillstånd inte byts, vinner Alice om hon har ett land av varje typ och en varelse av varje färg. Hon saknar enbart en blå varelse som skapas av haltteknet under tillstånd q_1. Det innebär att spelets komplexitet är tillräckligt kraftfull för att simulera en unviversella Turingmaskin vilket i sin tur leder till att frågan om vem som vinner i vissa partier är oavgörbar.

Tabell 9: Alice hand och kortlek under UTM.

Kort	**Placering**
Infest	1
Cleansing Beam	2
Coalition Victory	3 (Hoppas över när tillstånd byts)
Soul Suffers	4

[CBH19] beskriver att spelet inleds i en förutbestämd situation som motsvarar Turingmaskinens startläge. Alla symboler kontrolleras av Bob, medan den sista skrivningen av läs- och skrivhuvudet kontrolleras av Alice.

Startläget i Turingmaskinen innebär att när det blir Alices tur och hon har kortet *Infest* på handen. Hon kontrollerar ett land men det går inte att återställa sitt ursprungliga mana efter att ha använts. Varken Alice eller Bob kan attackera, eftersom de båda kontrollerar varsin *Blazing Archon* som hindrar attacker.

Bob har inga kort på handen och hans kortlek är tom men kontrollerar kortet *Recycle*, vilket tvingar honom hoppa över dragsteget. Detta stoppar Bob från att förlora när draghögen är tom, eftersom att en spelare förlorar om den försöker dra ett kort ur en tom kortlek.

Genom denna simulering visar författarna att **Magic** kan simulera en universell Turingmaskin. Det vill säga att spelets resultat är lika med att avgöra om Turingmaskinen kommer att stanna [CBH19].

4 Diskussion om oavgörbarhet i andra strategiska spel

Strategispelet **Magic** var det första populära spelet som visades vara oavgörbart. Beviset utmanar den ledande formella teorin om strategispel, nämligen påståendet att alla strategispel är avgörbara eftersom det obegränsade minnet i en Turingmaskin bryter mot strategispels natur [CBH19].

Inspirerad av denna insikt har jag reflekterat över följande fråga: Kan *Dominion* vara oavgörbart genom att simulera en Turingmaskin?

Som nämnts i Kapitel 2 är grundspelet i *Dominion* avgörbart. Men spelet har sedan dess fått sexton expansioner och omfattar idag över 5000 kort [Dom25]. **Dominion** är ett så kallat 'deck-building-spel' där spelare skapar och förfinar sin kortlek under spelets gång.

Nedan följer några preliminära idéer om hur **Dominion** potentiellt skulle kunna användas för att simulera till en Turingmaskin:

1. **Oändligt bandet:** I expantionen **Adventures** i **Dominion** introduceras tokenkort exempelvis *Coin* och *villager*, vilka kan användas för att markera eller ersätta olika spelelement. Dessa resurser skulle i teorin kunna generaras i oändlig mängd och därmed fungera som representation för ett bands oändliga struktur.
2. **Symboler:** Genom att välja ut ett begränsat urval kort från den totala kortpollen kan varje kort fungera som en symbol, vilket motsvarar en cell i Turingbandet.
3. **Tillstånd:** Varje spelare har tre faser (Händelsefasen, Köpfasen, Upprensning) i en spelomgång. Dessa skulle kunna motsvara olika tillstånd i en Turingmaskin.

Dessa idéer är visserligen mycket preliminära, men de kan utgöra en grund för en mer systematisk studie av *Dominions* potentiella oavgörbarhet. En närmare undersökning av dessa möjligheter lämnar jag till framtida forskning. Jag menar att analysen av oavgörbarhet inom ramen för strategispel inte bara kan fördjupa vår förståelse för begrepp som Turingmaskiner, Stoppproblemet och oavgörbara problem, utan även bredda deras tillämpningsområden i oväntade och kreativa domäner.

Referenser

[CBH19] Alex Churchill, Stella Biderman, and Austin Herrick. Magic: The gathering is Turing complete. *arXiv preprint arXiv:1904.09828*, 2019.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

[con24] Wikipedia contributors. Post correspondence problem, 2024. Online;2025-07-20.

[Dav82] Martin Davis. *Computatibility & unsolvability*. Dover Publ., New York, [new], dover ed. edition, 1982.

[Dom25] DominionStrategy Wiki contributors. Dominion(baseset) Wiki, 2025. [Online; accessed 16-July-2025].

[Gao19] Alice Gao. Introduction to undevidability, 2019. Hämtad: 12 maj 2025.

[HD09] Robert A Hearn and Erik D Demaine. *Games, puzzles, and computation*. CRC Press, 2009.

[HHH07] Vesa Halava, Tero Harju, and Mika Hirvensalo. Undecidability bounds for integer matrices using claus instances. *International Journal of Foundations of Computer Science*, 18(05):931–948, 2007.

[Mag25] Magic: The Gathering Wiki contributors. Magic: The Gathering, 2025. [Online; accessed 2025-07-21].

[Pon12] Bjorn Ponnen. Undecidable problems: A sampler, 2012. Hämtad: 22 mars 2025.

[Ri2] Vlad Rișcuția. Computability part 2: Turing machines, 2022. Accessed: 2025-07-26.

[Rog96] Yurii Rogozhin. Small universal turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996.

[Sat] Giorgio Satta. Turing machines. Lecture slides, Master's Program in Computer Engineering, University of Padua. Accessed: 2025-07-28.

[T+36] Alan Mathison Turing et al. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical society*, 42(2)(230-255):?, 1936.

[Wik24a] Wikipedia contributors. Barber paradox — Wikipedia, the free encyclopedia, 2024. Last updated December 18, 2024.

[Wik24b] Wikipedia – den fria encyklopedin. Paradox, 2024. Accessed: 2025-07-20.

[Wik25] Wikipedia contributors. Undecidable problem, June 2025. Uppdaterad: 2025. Åtkomst: 2025-07-03.