Projektuppgift

Programmering i C#.NET

Tic-Tac-Toe

Elias Eriksson

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Elias Eriksson, eler2006@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: HT, 2022

Sammanfattning

Ett Tic-Tac-Toe spel för en konsol byggs. Tic-Tac-Toe spelet har stöd för olika stora spelbrädor. Innan spelet startas finns det även en meny där användaren får välja om det ska vara spelare mot spelare, spelare mot AI eller AI mot AI. Varje AI har även 3 olika svårighetsgrader att välja mellan. Svårighetsgraden på AI:n utgår ifrån Den svåra AI:n men ju lättare AI:n blir desto fler funktioner i algoritmen är borttagna för att göra den lättare.

Innehållsförteckning

Introduktion

1.1 Bakgrund och problemmotivering

Ett konsol baserat Tic-Tac-Toe ska skapas där spelbrädan inte nödvändigtvis har den klassiska formen 3x3 och 3 i rad för vinst. Detta Tic-Tac-Toe ska ha stöd för att vara stort som specificeras och kravet för vinst ska också vara justerbart.

För att skapa detta spel måste en strategi för att lätt kunna undersöka om en spelbräda är i ett vinst läge eller inte hittas så inte alla kombinationer måste hårdkodas in med massor utav if-satser.

Förutom att bara spelbrädans storlek och spelregler ska kunna ändras ska det också finnas möjlighet att spela spelare mot spelare, spelare mot en AI samt en AI mot en AI. Det ska även finnas tre olika svårighetsgrader utav AI:n. AI:n ska implementeras genom en algoritm.

1.2 Avgränsningar

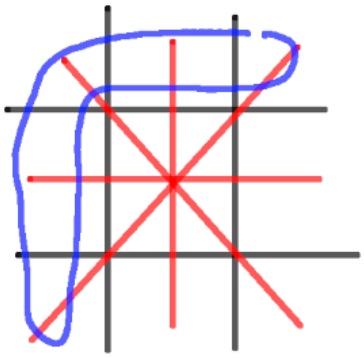
Algoritmen för AI:n går ju hela tiden att förfina så mycket det går men målet är att få den att fungera bra på en klassisk 3x3 bräda men att den även ska hantera andra storlekar.

2 Teori

För att skapa programmet används plattformen .Net med programmeringsspråket C#. .Net är en plattform utvecklad Microsoft [1] som klarar av att kompilera språken C#, F# samt Visual Basic och gör det möjligt att köra den kompilerade koden på massor av olika plattformar. C# som kommer vara det språket denna applikation skrivs i är ett objektorienterat statiskt typat språk som då är ett utav de språken som .Net klarar av att köra. [1] .Net stödjer massor av plattformar och detta projekt kommer att skrivas mot .Nets konsol API för att skapa det klassiska spelet Tic-Tac-Toe.

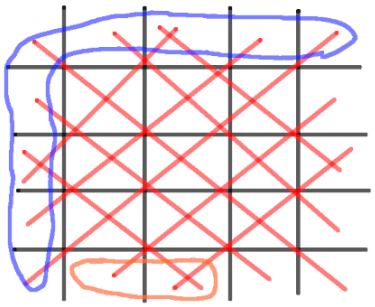
Klassikern Tic-Tac-Toe är ett klassiskt spel som spelas i ett två dimensionellt rutnät typiskt sätt är rutnätet 3x3 med ett krav på att få tre av sin egen markör i rad för att vinna. I denna standard konfiguration utav Tic-Tac-Toe så finns det 8 olika sätt att vinna: 3 i rad över 3 olika kolumner, 3 i rad över 3 olika rader och 3 i rad över 2 olika diagonaler. För att undersöka alla dessa vinstchanser så går det att göra en serie av 8 if-satser som kollar om något av dessa krav är uppfyllda. Om det däremot är önskvärt att kunna göra spelplanen större och ha ett krav på olika antal markörer i rad för att vinna så blir denna strategi med att skapa if-satser snabbt överväldigad och programmet kommer börja likna det ökända programmet ”my_first_calculator.py” [2]. Det är därför mer hållbart programmatiskt att undersöka alla de olika vinstmöjligheterna genom att undersöka alla vinstrader istället för alla koordinater.

För att undersöka om det finns en vinst måste det undersöka som det finns 3 av samma markörer längst något utav raderna, kolumnerna eller diagonalerna. Alla dessa vinstrader går att undersöka genom att börja vid någon utav spelbrädans övre eller högra kant och sedan gå åt en riktning. Detta mönster gäller för en en större spelplan så länge det är kvadratiskt och en hel rad, kolumn eller diagonal är ett krav för vinst.



Figur 1: Röda linjer indikerar ett sätt att vinna. Alla de röda linjerna hittas i det blåa fältet som visar de rutor i kanten som måste undersökas för att se om det finns en vinst på spelbrädan.

Detta ändras dock när en hel rad inte är ett krav för vinst. Om en spelplan som är 5x5 med ett krav på 3 i rad ritas upp så blir det precis som som förut gå att täcka in alla rader och kolumner genom att undersöka en viss riktning från någon utav rutorna längst upp eller till vänster. Diagonalerna kommer dock att vara lite annorlunda. De flesta diagonaler går fortfarande att hitta längst någon utav rutorna längst upp eller till vänster med det kommer vara två diagonaler som inte går att hitta. För att hitta dessa diagonaler krävts det att även två till rutor längst ner på spelplanen undersöks. Hur många extra går att räkna ut genom att subtrahera spelbrädans bredd med hur många markörer som krävs för vinst och subtrahera med ett för att ta bort det hörn som redan täcks utav det blåa fältet. Dessa rutor hamnar en ruta höger om hörnet längst ner till vänster och sträcker sig till höger.



Figur 2: Röda linjer visar alla diagonaler där en vinst kan finnas. Det blåa fältet precis som tidigare ringar in kanten av spelbrädan men missar denna gång två diagonaler. De två diagonalerna som saknas är de diagonaler som är inringade i orange.

Genom att undersöka dessa rutor på spelbrädan och sedan undersöka en rad, kolumn eller diagonal så går det att avgöra om en spelare har vunnit.

3 Metod

Applikationen som skall skapas kommer vara en konsol applikation men UI:n ska försöka göras lite roligare där man istället för att skriva in någon text som läses med Console.ReadLine så kommer menyerna vara navigerbara med piltangenterna och valbara med mellanslag eller enter. Målet är att får det att efterlikna hur spelkonsolen NES ofta hade sina menyer med en liten pil på sidan som styrs och visas vad som är valt och sen en annan knapp för att välja.

Figur 3: Menyn från det gamla NES spelet Duck hunt. För att navigera i menyn styrdes en pil på sidan upp och ner och kunde väljas med en separat knapp.

I konsol appen kommer valen att följa följande format (wireframe):

```
Which option do you want
> option 1
  option 2
  option 3
```

Användaren kommer också att bli ombedda att ange storlek på spelbrädan samt hur många markörer i rad som krävs för att vinna. Dessa kommer lösas med vanligt Console.ReadLine som sedan konverteras till ett heltal.

Spelbrädan som ritas kommer följa följande utseende (wireframe):

```
        v
      |   |
  -   +   +   -
>     |   |
  -   +   +   -
      |   |
```

Där mitten av spelbrädan fylls i med den markör av spelare som äger rutan. Standardvärdet kommer vara ett mellanslag så rutan ser tom ut men spelar markörerna kommer vara stora bokstaven X och stora bokstaven O. Pilarna på sidan kommer vara kontrollerade av piltangenterna. Den övre som väljer kolumn kommer vara styrd av piltangent höger och vänster och den på vänstersidan som väljer rad kommer vara styrd av pil upp och ner.

När användaren startar applikationen kommer användaren bli presenterad av en huvudmeny med 4 val där den får välja att spela mot en annan spelare, en AI eller om den bara vill se en AI spela mot en annan AI samt valet att avsluta applikationen. Om det ingår en AI i spelomgången så efterfrågas användaren

vilken svårighetsgrad AI(erna) ska vara, hur bred spelbrädan ska vara, hur hög den ska vara samt hur många markörer i rad som krävs för att vinna. Sedan startar spelet och spelarna i spelet spelar varsin tur om och om igen tills en av dom vinner eller om det blir oavgjort. Om en av dom vinner kommer den som van att få en poäng. Sen blir användaren presenterat av valet att fortsätta spela igen eller inte. Om användaren väljer att fortsätta kommer en ny spelbräda att skapas med samma inställningar och spelarna kommer återigen att spela efter varandra. Om användaren istället väljer att inte fortsätta kommer användaren tillbaka till huvudmenyn. (1)

För kunna se om en specifik spelare har vunnit och hur AI:n rör sig över spelbrädan för att undersöka hur den ska lägga sitt nästa drag kommer det utnyttjas att alla vinstrader kan hittas längst kanterna. Dessa koordinater kommer genereras och rekursiva metoder kommer att gå längst med den utsedda raden för att så att AI:n kan avgöra vad den ska göra samt att spelbrädan vet om det finns en vinst på den eller inte.

4 Konstruktion

En konsolapplikation skapas i Rider som genererar ett ”hello world” program skelett i en klass Program. Till program klassen skapas några statiska metoder som utför väldigt generella saker och inte platsar bättre i en annan klass. Dessa metoder är: ReadInt, Mod, ClearCurrentLine och ClearN.

ReadInt skapas eftersom att användare bara kan ange strängar via konsol input men det är av intresse att denna sträng lätt kan tolkas till ett heltal. Om användaren inte anger något som kan tolkas som ett heltal så blir användaren kvar i en oändlig loop tills användaren anger ett heltal.

Mod metoden skapas då C# språkets inbyggda återstående operator % är väldigt lik moduls men eftersom % inte ger ett positivt nummer vid användning utav negativa nummer så implementeras en metod för detta på program klassen. Detta används utnyttjas mycket i menyer.

Metoderna ClearCurrentLine och ClearNskapas som alternativ till Console.Clear() då console.Clear() rensar heta displayen så kommer det att hända att saker som inte kan skrivas ut i ett vist scope försvinner. För att radera ett vist antal rader så ritas rader fyllda med mellanrum som ger intrycket av att det inte finns något där längre.

För att representera spelbrädan i Tic-Tac-Toe skapas en generic klass Grid som har en intern tvådimensionell array. När klassen instansieras anges storleken som den tvådimensionella arrayen ska ha samt ett generic startvärde som rutnätet ska fyllas med. Till klassen kopplas ett gäng get och set metoder för att läsa ut information om rutnätet som hur brett och högt rutnätet är, innehållet på en rad och kolumn, möjligheten att läsa ut och ändra vad som finns på en specifik plats samt index till alla de rutor som inte har ändrat värde från sitt initiala värde.

Utöver att skapa en generic rutnätsklass skapas en till generic klass Node som beter sig som en länkad lista. Skillnaden från C# egna länkade lista är att den bara består utav noder istället för att ha en objekt runt noderna. Denna nod klass har stöd för standard metoder som en länkad lista har samt en metod som gör det möjligt att läsa av datan från en viss nod till slutet in i en C# List. Denna länkade lista används utav AI:n.

När programmet startas med Program klassens Main metod som till en början rensar konsolen, stänger av blink funktionen på markören och sen startar programmet så kommer användaren att hamna i Game klassens MainMenu metod.

Game klassen är den klass som innehåller allt som gör det möjligt för användaren att navigera i menyer. I MainMenu metoder så definieras först några strängar som används i menyn och sen en array av tuples med 2 element. Dessa tuples första element är en sträng och kommer motsvara den text som skrivs ut för ett meny

val. Det andra elementet är en funktion som returnerar en boolean. Denna funktion är den funktion som hanterar vad som ska hända om användaren väljer ett specifikt val i menyn. Funktionen returnerar en boolean då det då blir möjligt att avgöra om något ska fortsätta eller inte efter att Chose metoden har används.

Chose metoden är också en metod som finns på Game klassen. Chose metoden tar en array av tuples som är beskrivna i stycket ovan. Chose är en oändlig loop och låter användaren navigera med upp och neråt pilen genom att läsa av Consol.ReadKey() om användaren trycker upp ökas indexerings variabeln för tuple arrayen med ett. Och om användaren trycker ner så minskar indexerings variabeln med ett. Efter att användaren har tryckt på upp eller ner körs Mod funktionen på det nya indexet och arrayens längd. Detta gör att om indexet blir lika med eller större än arrayens längd kommer indexet gå tillbaka till 0 och det kommer se ut som att användaren går från det sista meny alternativet till det första. Detsamma händer när användaren går från index 0 till ett negativt index. Då kommer Mod funktionen se till att indexet blir arrayens längd-1 istället och det ser ut som användaren gick från första alternativet till det sista. När användaren trycker på antingen mellanslag eller enter så kommer funktionen för det valda alternativet att väljas och efter att valet är färdigkört kommer funktionens returvärde att returneras utav Chose metoden.

När Chose metoden körs i MainMenu metoder kommer användaren att bli presenterad av olika val. Det första kommer att vara om den vill spela människa mot människa, människa mot AI eller om den vill se på en match när en AI möter en annan AI. Om ett val där en AI ingår kommer användaren att även få välja vilken svårighetsgrad av AI:n som kommer att användas. Sedan kommer användaren att få ange spelplanens storlek och hur många markörer som krävs för att vinna och så kommer en spelomgång att starta genom att en instans utav Game klasser skapas.

Innan en ny spelrunda startas i Game klassen game-loop så skapas en spelbräda med Board klassen. Board klassen innehåller en emun med tre olika värden. De tre olika värdena är 32, 79 och 88 som är unicode codepoints för tecknena mellanrum. Stora bokstaven O samt stora bokstaven X. Mellanrum skrivs ut på de platser där ingen spelare har lagt en markör än och de andra läggs ut beroende på vilken spelare som har valt att spela på en viss ruta. Mellanrum skickas ner till grid klassens konstruktor som det initiala värdet att fylla hela rutnätet med. Grid klassens två olika Set metoder skrivs också över så att det först kontrolleras om den markören som ligger på en viss plats i rutnätet är ett mellanrum och alltså en tom ruta innan den önskade markören tillåts sättas in. Om det inte är en tom ruta så kommer en error att kastas.

Utöver funktionerna Relevanta till den underliggande Grid klassen innehåller också Board klassen olika interna metoder som används av den publika metoden Draw som skriver ut spelbrädan till konsolen. Samt en metod Traverse och en metod IsWinner för att kontrollera om det finns en vinst på spelbrädan. Traverse metoden innehåller fem olika for-loopar som genererar koordinater längst kanten av spelbrädan för alla positioner där en rad, kolumn eller diagonal kan starta. Varje for-loop ansvarar för en riktning och ser till att generera det koordinater relevanta för en viss riktning. Dessa koordinater ges sedan till en valfri funktion

som Traverse tar som parameter. Om funktionen som Traverse har hittat det den söker efter kan Traverse stoppas genom att funktionen returnerar true. För spelbrädan själv används detta av IsWinner metoden. IsWinner ger Traverse metoden en wrapperfunction av en intern IsWinner metod som petar ner koordinaterna från Traverse metoden på rätt parametrar samt vilken marker som ska kontrolleras för en vinst. Den interna IsWinner metoden kommer på så sätt att kontrollera alla rader, kolumner och diagonaler efter en vinst som den gör genom att räkna antalet markörer den ser på rad. Ser den något annat än den givna markören så börjar den räkna om från noll igen och den gör detta tills den hamnar utanför spelbrädans kanter. Hittar den en vinst returnerar metoden true för att få Traverse metoden att sluta generera mer koordinater och en variabel sätts till true som sedan returneras av den publika IsWinner funktionen som indikerar att en vinnare har hittas. Hittades inte en vinnare så har variabeln ifråga initierats med false och kommer på så sätta säga att det inte fanns en vinnare.

Internt i Game klasen är spelarna sparade som en array. Eftersom spelarna är i en array så är det möjligt att skapa en oändlig while-loop och sedan loopar över spelarna med en foreach-loop. För varje gång som foreach-loopen körs hanteras en utav spelarnas spelomgång så för varje iteration i while loopen har alla spelare spelat en runda.

Eftersom att spelarna i spelet behandlas exakt lika i foreach-loopen så måste båda AI:n och en mänsklig spelare representeras med en gemensam gränssnitt. Den gemensamma gränssnittet kommer i from av en abstrakt klass Player.

Player klassen håller information om vilken marker spelaren är samt hur många rundor den har vunnit. Till båda dessa fält finns det en get metod och poäng metoden har en metod för att lägga till en poäng. Sen finns en abstrakt metod Play som implementeras olika av olika AI:s och en mänsklig spelare.

Den abstrakta Player klassen ärvs utav en annan abstrakt klass AI och en konkret klass Human. Human klassen implementerar Play metoden och har två interna variabler selectX och selecyY som styr vart markörerna som visar rutan som användaren för närvarande har valt skrivs ut runt spelbrädan. När Play metoden körs så kommer användaren i i en oändlig while-loop som gör det möjligt för användaren att styra dessa selectX och selectY variabler genom att trycka på piltangenterna pil vänster, höger, upp och ner. När användaren är nöjd med var markörerna sitter kan användaren trycka på mellanslag eller enter så kommer Play metoden att lägga en markör på den rutan och returnera. Om rutan är upptagen kommer ett error att fångas från board klassens setMarker metod och användaren får försöka igen.

Den abstrakta klassen AI ärver också från Player klassen och agerar som ett mellansteg för de konkreta AI klasserna och implementerar metoder som de olika AI:erna potentiell skulle kunna använda sig av i sina Play metoder. Två utav dessa metoder är FindMoves och IdentifyWin. Båda dessa metoder fungerar exakt som den interna metoden IsWinner som tillhör Board klassen beskrivet ovan då de använder sig utav koordinater och riktningar genererade från board klassens Traverse metod. Det finns även metoder för att slumpa heltal.

IdentifyWin kommer att söka efter ställen där en viss markör är en markör ifrån att vinna. Detta görs genom att söka efter tomma och den specificerade markörens rutor längst den givna riktningen från Traverse. Om en ruta är tom eller av den specificerade markören kommer den koordinaten att läggas till i ett minne i form av ett Node objekt. Om en ruta skulle innehålla motståndarens markör kommer minnet att återställas. Om minnet blir lika långt som antalet markörer som krävs för att vinna kommer minnet att undersökas. Om det bara är en koordinat som pekar på en tom ruta kommer det anses finnas ett sätt att vinna för den specificerade markören.

FindMoves metoden används för att hitta koordinater på spelbrädan som på ett eller annat sätt kan leda till en vinst och är alltså det antal rutor på rad som inte är blockerade av motståndaren. FindMoves kommer att lägga ihop alla koordinater som är hittade i alla riktningar så koordinaterna som hittas kan alltså hittas längst flera olika vinstrader. FindMoves har två olika minnen som representeras i form av Node objekt. Ett minne håller koll på de noder som varit tomma eller egna men för tillfället va för få noder i rad för att kunna säga att det är en vinst. Det andra minnet är tillför de noder där det är bekräftat att det är tillräckligt med noder i rad för att bekräfta en vinst. När FindMoves för sig över en potentiell vinstrad så kommer en nod att läggas till i den ena eller det andra minnet beroende på hur många koordinater som redan hittats. Så fort som Det finns tillräckligt många i rad för att kunna bekräfta en vinst kommer noderna flyttas från det minnet som håller koll på potentiella koordinater till bekräftade koordinater och därefter kommer koordinater läggas till i bekräftade koordinater tills en motståndares markör påträffas.

De olika AI:erna har sedan möjlighet att utnyttja dessa metoder om de behöver. Den simplaste AI:n EasyAI använder sig inte utav någon av dessa mer komplexa metoder utan efterfrågar bara icke ändrade platser från spelbrädan rutnät och väljer en utav de lediga helt slump baserat.

Den mellansvåra AI:n MediumAI är lite mer komplex då den först använder sig utav IdentifyWin för att se om den själv kan vinna genom att lägga på en specifik ruta. Om den hittar en ruta där den kan vinna kommer den att lägga en markör på den rutan annars kommer den att hitta rutor där den potentiellt kan vinna med FindMoves. Alla koordinater som hittas läggs till i en lista och ett index i listan slumpas fram och koordinaten som kommer fram från slumpen kommer vara den ruta som AI:n lägger sin marker på. Eftersom att vissa koordinaterna i listan kan vara samma koordinat kommer det betyda att vissa koordinater som leder till flera olika vinst chanser har högre chans att bli slumpade. Om FindMoves inte hittar någon ruta betyder det att det inte finns något sätt att vinna och MediumAI kommer precis som EastAI att slumpa en av de kvarstående rutorna.

Den svåraste AI:n HardAI kommer precis som MediumAI att börja med att se om det finns något ställe som den själv kan vinna på. Om den själv kan vinna på ett drag kommer den att gå för vinsten. Om den inte själv kan vinna kommer AI:n undersöka om motståndaren kan vinna på ett drag. Om motståndaren kan vinna på ett drag kommer HardAI:n i så fall att blockera detta. Om motståndaren inte kan vinna på ett drag kommer HardAi:n att hitta koordianter med FindMoves. HardAI till skillnad från MediumAI kommer inte att slumpa en koordinat som

hittats utan kommer istället att hitta den koordinat som är den mest frekventa i listan. Om flera koordinater har samma frekvens kommer en utav dom med högst frekvens att slumpas. Om däremot FindMoves inte hittar något kommer HardAI:n precis som EasyAI att slumpa en koordinat från spelbrädan som inte är tagen då det inte längre går att vinna.

Efter att en spelrunda är klarspelad mellan spelarna presenteras användaren med ett val att fortsätta. Om användaren svarar nej kommer användaren tillbaka till MainMenu men om spelaren väljer att fortsätta kommer ett nytt spelbord att genereras och spelarna kan spela en runda till. Högst uppe på skärmen kommer det finnas en ställning mellan spelarna om vem som vunnit mest.

5 Resultat

Ett fungerande Tic-Tac-Toe spel har skapats. Spelet flyter på bra och är lättnavigerat med piltangenterna.

Det är helt möjligt att spela på en större eller mindre spelbräda med ett annat krav på markörer i rad för att vinna. Begränsningen är storleken på konsol fönstret.

AI:n som utvecklats spelar mycket bra på en klassisk 3x3 spelbräda med 3 i rad för att vinna. AI:n spelar Generellt bra på en spelbräda som är kvadratisk och där kravet för vinst är spelbrädans sidlängd. Så AI:n spelar även bra på en 5x5 bräda med 5 i rad för vinst. Om däremot kravet för vinst är mindre än sidlängden får AI:n problem med att blockera vinster i tid.

6 Slutsatser

Spelet fungerar mycket bra. När det kommer till hur spelet ser ut i konsolen finns det inte så mycket som skulle kunna förbättras.

Storleken så finns det inte heller så mycket som skulle kunna förbättras. Det skulle ju såklart kanske vara kul att skriva om programmet att använda något annat än konsolen. Annars finns det bara lite konstigt beteende om spelbrädan görs större än vad konsolfönstret är men det finns inte så mycket mer att göra än att kanske ge användaren en varning om det skulle bli för brett / högt.

När det kommer till AI:n så är det klart att den går att förbättra. Den svåra AI:n är den AI som kanske ska kvantifieras då de andra bara är sämre versioner av den svåra. Just nu så är det problem med att den inte märker av vinster tillräckligt tidigt om det går att vinna från två håll. Detta är något som kanske inte skulle vara överdrivet svårt att implementera. AI:n har inte någon bra koll på vart den själv redan har lagt den markör. Om den har lagt längst en rad kommer FindMoves inte att på något sätt överprioritera en rad som redan påbörjats jämfört med en rad som inte är påbörjad. AI:n ser endast att den är nära att vinna när den är en markör ifrån vilket gör att den kommer att blockera motståndaren när den ser att den är på väg att vinna men ju större och större spelbrädan blir desto långsammare kommer den själv att hitta en vinst. Detta skulle bli lite knepigare att lösa och på något sätt måste riktningarna där det finns markörer också på något sätt sparas så att det går att se vilka rader som AI:n själv är nära att vinna längst vilket skulle vara en betydligt större omskrivning.

Källförteckning

[1] Microsoft ”What is .NET”
https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

[2] Alex Lewis ”my_first_valculator.py”
https://github.com/AceLewis/my_first_calculator.py/blob/master/my_first_calculator.py

Bilagor

(1) programFlow.png