INDU - Tre i rad

Programmet är en spelversion av Tre i rad, från engelska Tic-tac-toe. Förutom den normala versionen av spelet, finns det även en tredimensionell version som liknar det engelska spelet “Qubic”.

Hur programmet används

Programmet startas genom att köra klassen “Game” med funktionen “run”. Efter detta så frågas användaren efter inmatning, som scenario, planstorlek (boardsize) och antal simulerade runder. Efter programmet har kört klart öppnas ett fönster med ett stapeldiagram över antal vunna rundor för varje spelare, samt antal rundor som blev oavgjorda.
Förutom detta utmatas även resultatet i kommandotolken från varifrån man startar programmet.

Användaren kan testa att ändra sleep parametern ifall resultaten för den tredimensionella versionen ger orimliga resultat. Detta är på grund av att random.seed() funktionen för koordinat generatorn inte hinner ändra förra datumet (i ms/dag), vilket gör att random.seed() slumpas till samma hash varje gång och det staplas tecken på varandra.

Externa bibliotek

I programmet används endast ett externt bibliotek som kräver installation (utanför pythons standardbibliotek), matplotlib. Vilket läsaren finner på deras officiella hemsida https://matplotlib.org/. Det kommer också med i många python baserade plattformar som exempelvis, Anaconda.

Programstruktur

Programmet bygget på objektorienterad struktur, vilket innebär att själva programmet är uppdelade i olika klasser.
ttt_class.py som innehåller klasserna TicTacToe och Qubic. TicTacToe klassen gör precis som det låter, hanterar spelet även kallat tre i rad på svenska. Qubic är klassen som hanterar den tredimensionella versionen av tre i rad. Qubic klassen bygger på TicTacToe klassen då man kan se den tredimensionella versionen som staplade vanliga tvådimensionella tre i rad spel. Därför ärver klassen Qubic vissa av funktionerna, men vissa av klass metoderna behövs överskridas då dom inte fungerar korrekt på tredimensionella klassen.

Över dessa två klasser finns det en Game klass i game_class.py. Denna fil är mer av en “UI” för användaren. Först så frågar den om användar input för att bestämma vilken av de två klasserna TicTacToe eller Qubic som ska köras och med vilka inställningar. Det är även här som de generella klasserna som att spara resultatet sker.

Game klassen importeras från vilken fil man än vill, så länge man importerar klassen korrekt. Samt måste game_class.py och ttt_class.py vara i samma mapp för att de ska kunna hitta varandra. Med projektet så skickas en main.py med koden som krävs för att köra programmet.

I programmet är bokstaven “O” första spelarens tecken, och “X” är andra spelarens tecken. För att representera en tillgänglig lucka för spelarna används siffran noll. Detta valet är mest på grund av att försöka representera en “korrekt” visualisation av spelet. Det hade förmodligen varit bättre för prestanda skäl att använda siffror istället för O och X.

Koddesign

Jag valde att gå med ett objektorienterat projekt, istället för en “enklare” design med endast funktioner. Detta är mycket på grund av att jag finner det enklare att arbeta i en miljö där klasser används. Det blir ett mer simpelt flow i programmet och mera klart vilka program funktioner som kräver vad från varandra. Även hur kod funktionerna förbereds innan de skickas vidare.
Klasserna sköter även alla klassglobala funktioner och variabler. Vilket gör att man slipper definiera globala funktioner och metoder som ligger kvar i minnet under hela körningen, vilket förhindrar att sakta ner programmet vid jobbigare parametrar.

Jag valde även att använda mig utav random.seed() funktionen i Python för att slumpa vilket ruta som spelaren valt. Detta är på grund av att förhindra liknande mönster i slumpningen av nummer när funktionen körs flera gånger under samma runda, och under samma programkörning. Till funktionen så skickas dagens datum, med ms för att få en slumpad sträng (nästan) varje gång funktionen körs, beroende på användarens dators prestanda kan detta variera.

Referenser

För att förstå mer djupgående om vilket tillvägagångssätt som passade för algoritmerna för programmet, läste jag igenom två dokument som är skrivna på ämnet. Dessa två är “Regularity and positional games”, skriven av A. W. Hales och R. I. Jewett. I dokumentet beskriver de med matematiska termer olika sätt att tänka kring att lösa liknande problem, samt bevisar algoritmerna med matematiska bevis.
Det andra dokumentet “Hypercube Tic-Tac-Toe”, av Solomon W. Golomb och A.W. Hales är en mer precis förklaring av hur man kan skriva algoritmerna för ett tredimensionellt tic-tac-toe, även kallat “Qubic”.
Jag har dock inte byggt programmets algoritmer som “kopior” eller egna direkta versioner på dessa dokument. Läsningen var mer för att förstå hur man skulle kunna ta sig an ett liknande problem som jag skulle lösa.
Värt att lägga märke till är att de båda dokumenten är skrivna av samma författare A.W. Hales, och kan därför ha en viss bias. Det kan finnas antagelser som är felaktiga i dokumenten som bekräftas av vardera. Dock så är dokumenten publicerade utav universiteten där författarna utför sitt arbete, och bör därför vara verifierade av universitet.

Samt är de båda dokumenten äldre än 18 år från då detta dokument är skrivet. Därför kan det ha skett framgångar och finnas nya “state-of-art” metoder. Men igen så bör detta inte vara ett problem i vårt fall, då vi inte kräver det i det skrivna programmet. Det hade möjligtvis varit en viktig sak att kolla upp om vi ville köra mer än 10,000 slumpade matcher, om inte mer.

https://www.ams.org/journals/tran/1963-106-02/S0002-9947-1963-0143712-1/S0002-9947-1963-0143712-1.pdf
http://library.msri.org/books/Book42/files/golomb.pdf