3-EN-RATLLA I ALGUNES VARIANTS

PENGCHENG CHEN
ZHIHAO CHEN

20/01/2023

IL·LUSTRACIÓ1: TIC TAC TOE

ÍNDEX

1. INTRODUCCIÓ

Aquesta pràctica en grup consisteix a completar la programació d'un tauler abstracte a partir d'un programa incomplet donat pels professors i dos programes conductors ("drivers"), que permetin jugar al 3-en-ratlla i algunes variants, tant al terminal com amb el mòdul "pygame".

La nostra idea principal és crear un programa que ens permeti jugar al 3-en-ratlla més bàsic i amb la possibilitat d'empatar. I amb un objectiu final de realitzar un joc funcional amb diverses variants, amb la possibilitat de canviar entre elles, i òbviament aprendre el màxim d'aquesta pràctica, i sintetitzar i consolidar el que ja hem fet durant el curs.

Un cop entès com hauria d'anar el programa aquest, començarem a entendre el programa incomplet donat i els dos programes conductors, i crearem un programa que sigui sòlida del 3-en-ratlla més bàsic dins del programa incomplet donat pels professors.

I finalment, a partir de la base sòlida que hem creat, volem anar ampliant el programa creant així les variants suggerides pels professors i intentar crear noves variants, tant inventades per nosaltres com buscades per internet.

El mètode per arribar a la nostra finalitat, serà primerament, cercar per internet algun vídeo creant un codi pel joc del 3-en-ratlla i inspirar-nos a partir d'això. I un cop visualitzat algun vídeo, anirem a entendre els codis i crear la versió més senzilla, després, cadascú intentarà fer variants a partir de la versió més clàssica("Default"). I ens anirem ajudant mútuament per millorar-ho i intentar fer més eficient.

2. EXPLICACIÓ DE CODIS

Un cop creat la versió més bàsica ("versió *Default*"), hem anat creant variants, en fitxers diferents.

A més, hem modificat els programes conductors ("main_gui.py", "main_txt.py" i "constants.py") proporcionats per tal que amb un únic fitxer del programa es puguin jugar totes les variants creades per nosaltres, amb opcions per seleccionar la variant abans de començar el joc. Ho hem aconseguit amb diversos "input" i segons la resposta, donada pels jugadors, el programa farà un "import" de la variant escollida.

A continuació explicarem per què els nostres codis són una solució correcta del problema plantejat, donant detall d'algunes línies de codi.

2.1 Versió "Default"

El '3-en-ratlla' de la versió "Default" és el joc estàndard: el primer a fer tres en ratlla guanya.

El codi consisteix a interpretar les pedres com tuples amb tres elements: els dos primers com les coordenades de les pedres, i el tercer element com el codi RGB que representa cada jugador ("BLUISH" o "REDDISH").

Aquestes tuples seran el contingut d'una llista L, que serà el que retorna la funció "stones", per poder dibuixar la pedra amb les corresponents coordenades i el corresponent color al tauler.

Cada vegada que un jugador seleccioni un espai buit de la taula, s'afegirà a la llista "L" una tupla que correspon a la pedra. Després d'afegir la tupla, es canvia del jugador, és a dir, es canvia el valor de "*curr_player*" que varia entre 0 i 1, de manera que quan "curr_player == 0" és el torn del jugador de pedra blava, i si "*curr_player* == 1", és el torn del de la pedra vermella.

Aquests fets es poden dur a terme gràcies a la funció "move_st", que la cridem repetidament sempre que no s'acabi la partida. Té arguments "i" i "j", que són les coordenades de l'espai seleccionat per col·locar la pedra.

A més, la "move_st" prohibeix ficar a la llista "L" una tupla amb les coordenades fora del rang. Per exemple, si la mida de la taula ("BSIZ") és 3*3, aleshores no és legal posar una pedra amb coordenades (3,0), ja que el rang de les coordenades és del (0,0) fins a (2,2). Altrament, també prohibeix posar pedres sobre les altres, dit d'una altra manera, si la tupla que es vol afegir a la "L" ja hi és, aleshores no es realitzarà l'acció i escriurà un missatge d'avís al jugador dient que no és permesa l'operació.

La funció retorna un iterable amb 3 elements: el primer element és la funció “select_st”, una funció per indicar si s’ha seleccionat o no la pedra per moure (en la versió “Default” sempre retorna True); el segon argument és el “curr_player”, que ha de retornar 0 o 1; i el tercer la funció “end()”, que també retorna un valor booleà que indica si s’ha acabat la partida o no (en la versió “Default”, si guanya algú o si s’empata).

La funció “end” no té arguments, verifica horitzontal, vertical i diagonalment si s’alineen *BSIZs* pedres, on “BSIZ” és la mida de la taula.

La verificació es realitza agafant una pedra de la llista i veure si en la mateixa línia horitzontal, vertical o diagonal hi són les pedres del mateix color. Si es verifica, aleshores retorna *True* i s’acaba la partida.

Tot l’anterior és obligatori perquè el joc funcioni a nivell abstracte, no s’ha de plantejar la taula perquè ja teníem un programa “conductor” i un fitxer “constants” per construir el tauler abstracte. No obstant això, a nivell textual l’hem de construir nosaltres mateixos, mitjançant la funció “draw_txt”.

La “draw_txt” dibuixa el tauler imprimint les línies, que són elements de les tres llistes que hem definit: “Llista1”, “Llista2” i “taula”.

La “Llista1” està formada per *BSIZs* espai buit (“ “), on es fiquen els símbols ‘X’ (jugador BLUISH) o ‘O’ (jugador BLUISH) i (BSIZ -1) símbols “|” per construir el tauler.

La “Llista2” conté (BSIZ*2-1) símbols “-”, que representen una part de l’estructura del tauler.

La “taula” és una llista amb subllistes “Llista1” i “Llista2”, i els seus elements són els que s’imprimeixen en el terminal mitjançant la funció “board”.

Sempre que no s’acabi la partida, la “draw_txt” revisa el contingut de la llista “L” de l’entorn pare i substitueix l’espai buit de la “taula” pels símbols corresponents als jugadors (indicats pel tercer element de la tupla), amb les mateixes coordenades que les de les tuples de la “L”.

2.2 Versió “Counter”

En la versió “Counter”, la manera per decidir el guanyador és la quantitat de pedres alineades en tres, que ho té més que el perdedor.

Com que la mida del tauler és un senar, aleshores perquè sigui just, el joc acaba quan el jugador ocupa el penúltim espai buit, és a dir, els jugadors tenen la mateixa quantitat de pedres per col·locar al tauler.

L’única diferència que presenta aquesta versió amb la versió “Default” és la implementació de la funció “counter” que compta la quantitat de pedres alineades en

tres que té cada jugador, perquè pugui identificar el guanyador després de posar totes les pedres.

La forma de comptar és triar una pedra qualsevol de la llista “L” i veure si horitzontal, vertical i diagonalment formen tres pedres alineades del mateix color. Si ho formen, aleshores segons el color que té, se suma 1 al “counter1” (del jugador BLUISH) o “counter2” (del jugador REDDISH).

Per poder colorear el fons amb el color corresponent (el color del guanyador o el color lila si s’empata), s’ha de saber quin jugador té més número de pedres alineades en tres, és a dir, el valor del “counter”. Tanmateix, això ens ho proporciona la funció “counter”, que no és retornada pel “set_board_up”, aleshores l’hem d’afegir a l’iterable que retorna.

2.3 Versió “Moveable”

Aquesta versió té la capacitat de moure les pedres de posició un cop ja hem acabat de col·locar-les totes(vuit pedres, 4 per jugador, o segons la mida del tauler).

A diferència de les dues versions anteriors, hem afegit una còpia a la llista de tuples(“Ls”) on es guarden les pedres, anomenada “L” que són les pedres que es dibuixen al tauler. I a la funció select hem afegit una condició, quan totes les pedres ja estan jugades, aleshores podem seleccionar una pedra que del color del jugador i que es pugui jugar. La pedra seleccionada (la tupla “sel_PLAYER”), s’elimina de la llista “Ls”, però no desapareix del tauler, ja que com hem mencionat abans, la llista “L” és la que es dibuixa. Per cada acció realitzada, sortirà un missatge al terminal, tant si és d’error, com si no ho és.

No hi ha grans canvis dins de la funció “move_st”, es pot apreciar que cada vegada que es mou una pedra, s’afegeix a la llista “Ls” i després la llista “L” en fa una copia, aquí és quan les pedres dibuixades canvien. En acabat, fa un canvi de jugador amb el “curr_player” i finalment si la longitud de la llista “Ls” és igual a totes les pedres jugables i encara ningú ha guanyat, aleshores la funció “select_st” pren per valor *False* i, per tant, durà a terme aquesta funció, explicada abans.

Una altra diferència que presenta aquesta versió de codi és la desaparició de la regla d’empatar, ja que segurament hi ha un guanyador entre els dos jugadors.

2.3.1 Versió “Middle”

Aquesta versió és una variant del ”moveable”, la diferència és que en aquesta variant només es pot començar des del mig del tauler.

Això ho hem fet aplicant una condició, de manera que si la longitud de la llista de les pedres és 0, és a dir, que no hi ha cap pedra jugada, aleshores si la pedra que volem

col·locar és diferent del centre, aleshores al terminal s'enviarà un missatge indicant que no és possible aquella acció. Per la resta, no hi ha cap diferència.

2.3.2 Versió "Adjacent"

Aquesta versió és una versió més avançada del "Moveable". En aquest, un cop col·locat totes les pedres al tauler, es poden seleccionar les pedres que tenen una posició adjacent lliure i moure a posicions adjacents, i guanya el que té fa x-en-ratlla o si el contrari no té cap posició adjacent.

Això ho hem aconseguit primer, creant una llista "Ls" on hi ha pedres grises (del color del tauler), després, a la funció "select_st" hem afegit una condició que, un cop estan totes les pedres al tauler, i volem seleccionar una pedra per moure-la, aquesta condició mira la resta de pedres al voltant, i si al voltant hi ha una pedra grisa, aleshores s'executa aquella condició, sinó al terminal s'envia un missatge d'error. Quan la pedra és seleccionada, s'elimina la pedra del jugador de la llista "Ls" i s'afegeix una pedra grisa a aquell lloc.

Com hem explicat abans, la funció "stones" retorna la llista "L" i, per tant, fins no arribi a "L=Ls[:]" la llista del tauler ("L") no canvia.

Finalment, la funció "move_st", hem afegit una variable "nonlocal", anomenada "stones_per_player", que indica el si el joc està en la primera part o en la segona part, és a dir, si estan col·locant les pedres o estan en la part on ja poden moure-les. Quan aquesta variable arriba zero, vol dir que estan en la segona part, i en aquesta part, hem afegit una condició per cada vegada que mouen la pedra. Un cop han mogut una pedra, aleshores hi ha un bucle que mira les adjacències de totes les pedres. Si hi ha una posició lliure adjacent a la pedra i aquella pedra és del color del jugador actual, aleshores la funció "select_st" és *False* i, per tant, podrà seleccionar una pedra per moure. Si no hi ha cap posició lliure adjacent a les pedres del jugador actual, aleshores automàticament la funció "end" serà *True* i terminarà el joc.

2.4 Versió "Misery" dels codis

En la versió "Misery" dels codis, els jugadors han d'intentar "perdre" per guanyar, és a dir, la condició per guanyar que és al contrari que el joc en versió "Normal". Per exemple, en la versió "Default_Misery" el que alinea primer tres pedres del mateix color perd.

Per plantejar-lo, sigui quin sigui el codi, l'únic que es canvia és la frase que s'imprimeix al terminal per indicar quin jugador és el guanyador.

En el programa "conductor" també s'ha d'aclarir si estem jugant la versió "Normal" o la "Misery" per acolorir correctament el fons del color del guanyador. Això es pot dur a terme afegint diverses instruccions condicionals 'if"s.

3. CONCLUSIONS

Aquest treball va començar amb una recerca d'un vídeo que ens ajudés a comprendre i a saber realitzar com un joc. A partir d'aquesta base vam començar el nostre programa que només funcionava dins del terminal, i arran d'aquest programa, van sorgir problemes amb la implementació del mòdul "*pygame*". Per tant, vam contactar amb un dels tutors i vam optar per utilitzar els esquemes ja proporcionats.

Un cop acabat el programa que només funciona a nivell textual, quan tornàvem a veure els dos programes "conductors" que ens havia donat els tutors, ens resultava molt més senzill entendre-ho comparat amb l'inici del treball.

Posteriorment, després de cercar per Internet el funcionament de les funcions o dels termes que no enteníem, vam intentar plantejar el "tres-en-ratlles" més clàssic, que també és el més senzill, i anar evolucionant a partir d'aquí.

La principal dificultat que vam trobar a l'inici va ser el plantejament de la funció "stones", que retorna un iterable amb les pedres jugades, perquè aquesta funció és relacionada amb la funció "move_st", ja que els elements de les tuples afegides a l'iterable se sap després de cridar la "move_st". No obstant això, no sabíem com modificar el contingut d'una variable d'una funció des d'una altra. Així doncs, vam decidir utilitzar la llista com l'iterable, perquè és mutable i ens resulta fàcil d'utilitzar, i posar-la a l'entorn pare de les funcions "stones" i "moveable". I gràcies a l'eina que ens proporciona Python, nonlocal, podem mutar els elements de la llista des de la funció "move_st".

Altres reptes de la pràctica han sigut l'arreglament els "bugs" sorgits durant la creació del joc i les seves variant, amb les seves dificultats respectives. Per exemple, al principi la implementació de la funció "end" no era correcta, aleshores el joc no s'acabava encara que s'alineen tres pedres del mateix color, o bé en la versió de "Moveable", les pedres no seleccionades desapareixen... L'ús del "print" entre les instruccions ens ha resultat força útil per resoldre alguns "bugs", ja que té un funcionament semblant a un "Debugger".

Per concloure, podem dir que hem resolt els objectius plantejats, ja que creiem que hem creat un 3-en-ratlla funcional amb algunes variants dels quals ens podem moure entre elles amb un mateix programa conductor, també hem après bastant sobretot l'ús de les declaracions "nonlocal", els bucles "for", les funcions locals, el funcionament del mòdul "pygame", etc. I d'altres que ens ha pogut resultar útil de cara a l'examen final.

4. REFERÈNCIES

Vídeo buscat: Knowledge Shack. *Easy 2 Player Tic Tac Toe Program with Python | Part 1: Functions | Knowledge Shack.* URL: [https://youtu.be/Al917szWQXA]

Il·lustració1: Tic Tac Toe. Gamesever Team. *Tic-Tac-Toe.* [https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.gamesver.com%2Fdisadvantages-downsides-of-tic-tac-toe-predictability-solved-game%2F&psig=AOvVaw1f99AUokNW1XkNot4Sxxz1&ust=1674328427539000&source=images&cd=vfe&ved=0CBEQjhxqFwoTCPDa2M3t1vwCFQAAAAAdAAAAABAI]