

سوال تئوری:

روش اول model checking :

در این روش برای درستی یک گزاره با توجه به KB های ما، تمام حالت های ممکن را بررسی میکنیم. برای مثال در بازی Wumpus world اگر بخواهیم بدانیم در یک خانه ی خاص آیا Wumpus وجود دارد یا نه. برای این کار باید تمام حالات را بررسی کرد و در صورت وجود conflict این جمله غلط و در غیر این صورت گزاره ما درست است.

روش دوم theorem-proving:

در این روش بجای بررسی تمام حالات برای درستی یک گزاره، با استفاده از KB ها و قوانین استنباط، نتیجه گیری های جدید انجام میدهیم. مثلاً با استفاده از modus ponens میدانیم اگر در یک خانه نسیم بیاید آنگاه در خانه های همسایه آن چاه داریم. حالا اگر در خانه ای که هستیم نسیم نیاید پس در خانه های همسایه آن چاه وجود ندارد.

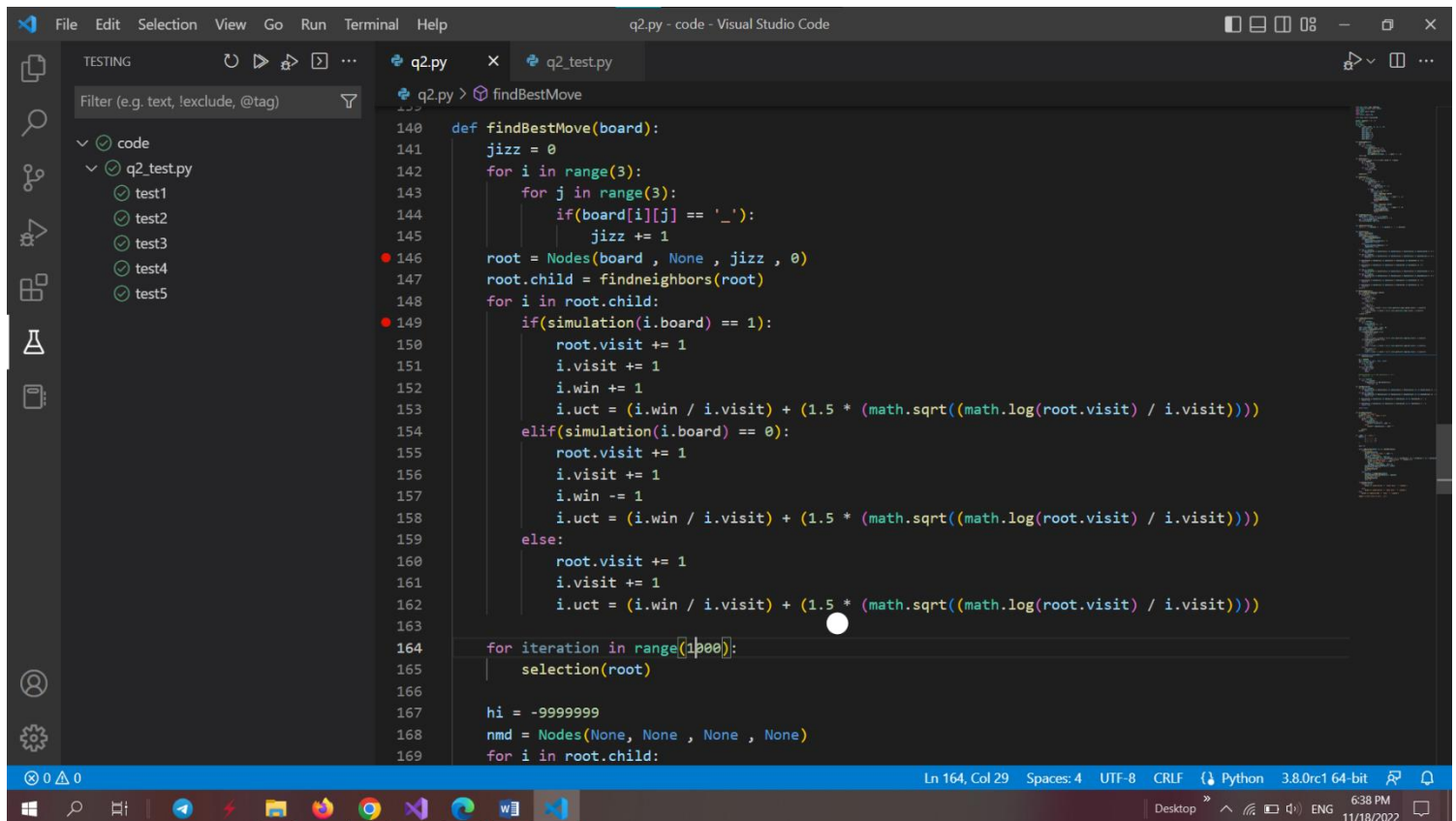
سوال عملی:

برای بازی در ایکس او پس از انتخاب حرکت توسط player حرکت بعدی باید توسط تابع `findBestMove` که مشخصات خانه ای که O در آن قرار میگیرد به عنوان خروجی داده شود.

برای این کار باید ابتدا یک ریشه که وضعیت فعلی جدول است را بسازیم. سپس هر بار 4 مرحله را انجام دهیم:

ابتدا با استفاده از selection نودی که بیشترین ucb را دارد و همچنین تمام نود های چالد آن ساخته شده و leaf نیست، انتخاب میکنیم. در مرحله بعد نود انتخاب شده را باید expand کنیم. به این شکل که یکی از نود های چایلد آن را بسازیم. حال باید با استفاده از simulation یک بار بازی را کامل انجام دهیم به شکلی که حالت اولیه ی جدول حالتی که این نود است، باشد. سپس با استفاده از Backpropagation مقدار ucb , visit , win , تمام نود ها را تغییر بدهیم به این شکل که اگر نوبت player بود، و ما در simulation امان opponent برد، از win در این نود 1 کم شده اگر نوبت opponent بود، به مقدار win نود 1 اضافه میشود و در صورتی مساوی شدن مقدار win تغییر نکرده و visit یکی اضافه میشود. این مراحل 1000 بار انجام داده میشود.

در آخر تمام چالد های ریشه را چک کرده و نودی که بیشترین ucb دارد را به عنوان حرکت بعدی انتخاب میکنیم.



```
140 def findBestMove(board):
141     jizz = 0
142     for i in range(3):
143         for j in range(3):
144             if board[i][j] == '_':
145                 jizz += 1
146     root = Nodes(board, None, jizz, 0)
147     root.child = findneighbors(root)
148     for i in root.child:
149         if(simulation(i.board) == 1):
150             root.visit += 1
151             i.visit += 1
152             i.win += 1
153             i.uct = (i.win / i.visit) + (1.5 * (math.sqrt((math.log(root.visit) / i.visit))))
154         elif(simulation(i.board) == 0):
155             root.visit += 1
156             i.visit += 1
157             i.win -= 1
158             i.uct = (i.win / i.visit) + (1.5 * (math.sqrt((math.log(root.visit) / i.visit))))
159         else:
160             root.visit += 1
161             i.visit += 1
162             i.uct = (i.win / i.visit) + (1.5 * (math.sqrt((math.log(root.visit) / i.visit))))
163
164     for iteration in range(1000):
165         selection(root)
166
167     hi = -9999999
168     nmd = Nodes(None, None, None, None)
169     for i in root.child:
```