سوال 1)

در بازی های مجموع غیر صفر هر بازیکن بدون توجه به بازی حریف خود به دنبال max کردن امتیاز خودش است پس هر وضعیت به صورت یک بردار است که امتیاز های هر شخص به عنوان یک مولفه در آن است که هر مرحله بازیکن max امتیاز خود را انتخاب میکند و این همان اجرای الگوریتم minimax برای بازی چند نفره است.و همانطور که گفته شد چون به بازیکن دیگری توجهی نمیکنند پس ممکن نیست گره ای توسط الفا بتا هرس شود.

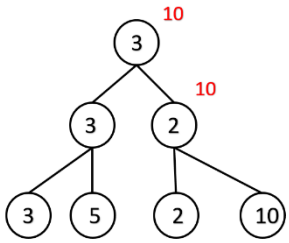
سوال 2)

الف) خیر، هم در درخت max و هم در درخت expectimax گره ای را نمیتوان توسط الفا بتا هرس کرد چرا که در درخت max باید تمامی رئوس بررسی شوند شاید که حالت max در گره های دیگری باشند و در درخت expectimax هم ممکن است حالت maxو یا min در گره های دیگری باشد که بررسی نشده اند.

ب) بله، چون کرانی مشخص داریم برای مثال اگر در درخت max به گره 1 برسیم دیگر نیاز نیست بقیه گره ها را بررسی کنیم و میتوان آن ها را هرس کرد. برای درخت Expectimax هم با حساب کردن احتمال هر گره میایم شاخه ای که احتما پایین تری را دارد هرس میکنیم.

سوال 3)

اگر min به صورت بهینه بازی نکند در اینصورت یا جواب ما همان باقی میماند و یا به نتیجه بهتری میرسیم چرا که به صورت رندوم ممکن است بازی کند که این به نفع ما خواهد بود. پس



الگوریتم minimax ایده خوبی است. اما در شرایطی که بازیکن max در حال باخت است دیگر این الگوریتم خوب نیست چراکه ممکن است با حرکت اشتباه بازیکن min دوباره به شرایط برد برگردد.

سوال 4)

الف) طبق فرض داریم $\alpha \vDash (\beta \wedge \gamma)$ که این به این معناست که :

$\forall\alpha$، $(\beta \wedge \gamma)$ درست است و چون $(\beta \wedge \gamma)$ درست است باید هم β درست باشد و هم γ پس میتوان نتیجه گرف که:

$\alpha \vDash \beta$

$\alpha \vDash \gamma$

پس این عبارت درست است.

ب طبق فرض داریم $\alpha \vDash (\beta \vee \gamma)$ که این به این معناست که :

$\forall\alpha$، $(\beta \vee \gamma)$ درست است که پس یا β درست باشد ویا γ که میتوان این عبارت را با در نظر گرفتن یک مجموعه نقیضش رد کرد پس نا درست است.

سوال 5)

S1: $A \Leftrightarrow (B \vee E)$

S2: $E \Rightarrow D$

S3: $C \wedge F \Rightarrow \neg B$

S4: $E \Rightarrow B$

S5: $B \Rightarrow F$

S6: $B \Rightarrow C$

وقتی میخواهیم $A \sim \wedge B \sim$ را حساب کنیم کافی است ثابت کنیم $A \sim$ و $B \sim$ درست است.

از $B \sim$ شروع میکنیم پس S7: B را اضافه کنیم و میایم به ترتیب ترکیب های دوتایی رو بررسی میکنیم.

S1: A⇔ (B∨ E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: B

با توجه به گزاره های S7 وS6 به گزاره S8: C میرسیم.

S1: A⇔ (B∨ E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: B

S8: C

با توجه به گزاره هایS5 و S7 به گزاره S9: F میرسیم.

S1: A⇔ (B∨ E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: B

S8: C

S9: F

با توجه به گزاره های S3 و S8 به گزاره S10: F ⇒ ¬B می رسیم.

S1: A⇔ (B∨ E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: B

S8: C

S9: F

S10: F ⇒ ¬B

با توجه به گزاره های S9 و S10 به گزاره S11: ¬B می رسیم.

S11 و S7 تناقض هستند.

حال میخواهیم $A \backsim$ را ثابت کنیم.

ابتدا برای $A \backsim$ کافی است S7: A را اضافه کنیم. حال ترکیب‌های دوتایی از این موارد را حساب میکنیم تا بتوانیم به تناقض برسیم.

S1: A⇔ (B∨ E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: A

با ترکیب S7 و S1:

S1: A⇔ (B∨ E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: A

با توجه به گزاره های S1 و S7 به گزاره S8: BVE میرسیم.

S1: A⇔ (B∨ E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: A

S8: BVE

با توجه به گزاره های S8 و S4 به گزاره S9: B می رسیم.

S1: A⇔ (B∨ E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: A

S8: BVE

S9: B

با توجه به گزاره های S9 وS6 به گزاره S10: C میرسیم.

S1: A⇔ (BV E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: A

S8: BVE

S9: B

S10: C

با توجه به گزاره های S5 و S9 به گزاره S11: F میرسیم.

S1: A⇔ (BV E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: A

S8: BVE

S9: B

S10: C

S11: F

با توجه به گزاره های S3 و S10 به گزاره S12: F ⇒ ¬B می رسیم.

S1: A⇔ (BV E)

S2: E ⇒ D

S3: C ∧ F ⇒ ¬B

S4: E ⇒ B

S5: B⇒ F

S6: B⇒ C

S7: A

S8: BVE

S9: B

S10: C

S11: F

S12: F ⇒ ¬B

با توجه به گزاره های S11 و S12 به گزاره S13: ¬B می رسیم.

S13 و S9 تناقض هستند.

پس $B \sim$ و $A \sim$ درست هستند و در نتیجه $A \sim \wedge B \sim$ درست است.

گزارش بخش پیاده سازی)

```
class TicTocToe:
    def __init__(self):···

    def is_valid(self, place):···

    def check(self):···

    def max_alpha_beta(self, alpha, beta):···

    def min_alpha_beta(self, alpha, beta):···

    def play(self):···

    def display(self):  ···
```

کل کد ما هدف این است که کامپیوتر امتیاز خودش را ماکسیموم کند و امتیاز ما را مینیموم. این کار را با کمک حرس alpha-bet و الگوریتم Minimax انجام دادیم. یک کلاس TicTocToe تعریف کردم که توابع مورد نیاز را در آن قرار دادم.

در تابع init مقداردهی اولیه برای ایجاد بازی را انجام میدهم. به این صورت که وضعیت فعلی بازی را به صورت یک آرایه 9 تایی از نقاط تعریف کردم. در این بازی نقطه همان خانه خالی است. هر خانه میتواند یا نقطه یا X یا O باشد.

```
def is_valid(self, place):
    if place<0 or place >8:
        return False
    elif self.current_state[place] != '.':
        return False
    else:
        return True
```

تابع is_valid با قصد بررسی ورودی گرفته شده از کاربر نوشته شده است. هر عددی که از کاربر میگیریم عددی بین 1 تا 9 است. این تابع بررسی می‌کند که آیا عدد گرفته شده در این بازه قرار دارد یا نه؟ همچنین بررسی میکند که آیا این خانه پر است یا نه؟

تابع display برای نمایش صفحه بازی نوشته شده است. یک حلقه است و current_state را چاپ میکند. به طوری که هر 3 مولفه سطر بعد میرود.

```python
def check(self):
    # Vertical
    for i in range(0, 3):
        if (self.current_state[i] != '.' and
            self.current_state[i] == self.current_state[i+3] and
            self.current_state[i+3] == self.current_state[i+6]):
            return (True,self.current_state[i])

    # Horizontal
    if (self.current_state[0] != '.' and
        self.current_state[0] == self.current_state[1] and
        self.current_state[1] == self.current_state[2]):
        return (True,self.current_state[0])
    if (self.current_state[3] != '.' and
        self.current_state[3] == self.current_state[4] and
        self.current_state[4] == self.current_state[5]):
        return (True,self.current_state[3])
    if (self.current_state[6] != '.' and
        self.current_state[6] == self.current_state[7] and
        self.current_state[7] == self.current_state[8]):
        return (True,self.current_state[6])

    # diagonal
    if (self.current_state[0] != '.' and
        self.current_state[0] == self.current_state[4] and
        self.current_state[4] == self.current_state[8]):
        return (True,self.current_state[0])
    if (self.current_state[2] != '.' and
        self.current_state[2] == self.current_state[4] and
        self.current_state[4] == self.current_state[6]):
        return (True,self.current_state[2])

    # Not a Terminal State
    for i in range(0, 9):
            if (self.current_state[i] == '.'):
                return (False,'.')

    # Tie
    return (True,'.')
```

تابع check مشابه خواسته سوال است. بررسی میکند آیا ما در وضعیت پایانی قرار داریم یا نه؟ اگر بودیم یک مولفه دوتایی برمیگرداند. خانه اول این مولفه یک مقدار Boolean است که اگر وضعیت پایانی باشیم True است و اگر وضعیت پایانی نباشیم False است. مولفه دوم این تابع شخص برنده است. اگر X باشد یعنی X برنده است و اگر O باشد یعنی O برنده است. اگر نقطه باشد یعنی بازی مساوی است.

همانطور که در کد می‌بینید وضعیت‌های پایانی را بررسی می‌کنیم که شامل برد سطری، برد ستونی، برد قطری و مساوی است.

دلیل این که به جای 0 و -1 و 1 مقدار داخل خانه برنده را ارسال می‌کنیم این است که الگوریتم ما در تابع check بدون توجه به جنس برنده فقط تعیین میکند برنده شده است یا نه و در انتها نام برنده را بر میگرداند.

تابع min_alpha_beta و max_aplha_beta که نقش مهمی در برنامه دارند مقدار دهی -1 و 0 و 1 انجام دادند و با این حرکت از دوباره کاری در این توابع برای وضعیت‌های پایه جلوگیری کردم.

```python
def max_alpha_beta(self, alpha, beta):
    max_value = -10
    place = None
    (status,result) = self.check()

    if status == True:
        if result == 'O':
            return (-1, 0)
        elif result == 'X':
            return (1, 0)
        elif result == '.':
            return (0, 0)

    for i in range(0, 9):
            if self.current_state[i] == '.':
                self.current_state[i] = 'X'
                (m, min_place) = self.min_alpha_beta(alpha, beta)
                if m > max_value:
                    max_value = m
                    place = i

                self.current_state[i] = '.'
                if max_value >= beta:
                    return (max_value, place)

                if max_value > alpha:
                    alpha = max_value

    return (max_value,place)
```

دو تابع min_alpha_beta و max_alpha_beta برای الگوریتم minimax ساخته شده است. در عکس روبه‌رو تابع max_alpha_beta را مشاهده میکنید.

شیوه اجرا این است که ابتدا وضعیت بازی را بررسی میکند اگر وضعیت پایانی بود که به نوعی برگ‌های درخت ما است و مقدار دهی می‌کند.

و در سایر حالت‌ها با کمک حلقه روی خانه‌های صفحه آنجا که خانه مقدار نقطه دارد (یعنی خالی است) مقدار max_value را حساب میکند. برای این کار یک حل بازگشتی ارائه دادیم. به طوری که انگار روش بالا به پایین است. یعنی به ازای ان نقطه خالی میاید وضعیت جدید میسازد و تابع min_alpha_beta را برای آن صدا می‌زند. همین اتفاق برعکسش در تابع min_alpha_beta می افتد و تا زمانی که به برگ برسند این کار را تکرار میکنند.

تابع‌های max_alpha_beta و min_alpha_beta در هر مرحله دو مولفه برمیگرداند. مولفه اول مقدار value است و دومی محل درج مقدار جدید است.

در بخش پایانی نیز بازه (alpha,beta) نیز با توجه به مقدار max_value بروز میشود تا مسئله با سرعت بالاتری حل شود. با انجام چند نمونه محاسبات دیدم یک مسئله با 30 نود در حالت عادی حل مشد و حالا با روش حرس الفا بتا با 20 نود به پایان رسیده است. زمان این برنامه نیز کاهش چشمگیری داشته است و حدودا از 4 ثانیه به 0.35 ثانیه با توجه به سخت افزار کامپیوتر من کاهش یافته است.

برای اجرای کد در حالت عادی کافی است تمام بخش های مربوط به الفا و بتا را پاک کنید.

```python
def play(self):
    while True:
        self.display()
        (self.status,self.result) = self.check()

        if self.status == True:
            if self.result == 'X':
                print('COMPUTER is the Winner!')
            elif self.result == 'O':
                print('YOU are the Winner!')
            elif self.result == '.':
                print("No one is our winner! ")
            return

        else:
            # AI's turn
            if self.player_turn == 'X':
                (m, place) = self.max_alpha_beta(-10, 10)
                self.current_state[place] = 'X'
                self.player_turn = 'O'

            # player's turn
            else:
                while True:
                    place = int(input('Insert the place of O [1...9]: '))
                    place = place - 1

                    if self.is_valid(place):
                        self.current_state[place] = 'O'
                        self.player_turn = 'X'
                        break
                    else:
                        print('The move is not valid! Try again.')
```

تابع play جایی است که بازی در آن انجام میشود. در این قسمت یک حلقه همیشه درست در حال اجرا است تا زمانی که ما به وضعیت‌های پایانی برسیم.

پس لازم است ابتدا وضعیت بازی را بررسی کنم و اینکار را با تابع check انجام دادم.

حال اگر در وضعیت‌های پایانی نبودم بر اساس این که نوبت کدام بازیکن است حرکت میکنند.

اگر نوبت کامپیوتر باشد تابع max_alpha_beta با مقدار الفا و بتا برابر با 10- صدا زده می‌شود (یک عدد بیشتر از کران بازی در ابتدا که مشکلی در مسئله ایجاد نکند).

با صدا زدن این تابع یک مقدار برمیگردد که محل حرکت کامپیوتر در آن تعیین شده است.

اگر نوبت بازیکن باشد نیز از کاربر یک عدد از 1 تا 9 وارد میکند و بعد از مورد تایید بودن آن، آن حرکت را انجام می‌دهد.

```python
def start_game():
    while(True):
        g = TicTocToe()
        g.play()
        n = int(input("Do You want play?(If yes type 1, otherwise print 0.) "))
        if(n==0):
            break

if __name__ == "__main__":
    start_game()
```

در انتها نیز تابع start_game نیز تا زمانی که بازیکن بخواهد بازی کند اجرا میشود. و هرموقع خسته شد با وارد کردن عدد صفر برنامه متوقف می‌شود.