مهسا امینی ۹۸۱۷۸۲۳

در درخت بازی گره ها در سطوحی قرار میگیرند که با نوبت هر بازیکن در بازی مطابقت دارد به طوری که ریشه ی درخت، موقعیت ابتدایی بازی می باشد در زیر ریشه، در سطح دوم، حالت های احتمالی وجود دارد که میتواند از حرکات بازیکن اول ایجاد شود هر گره در سطح دوم، گره های فرزندی دارد که شامل حالت هایی است که با حرکات بازیکن مقابل امکان به وجود آمدن دارد. این کار سطح به سطح، تا رسیدن به حالتی که بازی تمام شده است ادامه می یابد. که در واقع تمام شدن بازی به این معنا است که یا یکی از بازیکنان سه تا از حروف خود را به صورت ردیفی، ستونی و یا قطری در کنار هم قرار میدهد و برنده میشود یا تخته پر است و بازی مساوی میشود. روش مورد نظر برای حل این سوال جمع صفر است به این معنا به بازیکنی که برنده میشود (+1) امتیاز و به بازیکنی که میبازد (-1) امتیاز و زمانی که بازی مساوی میشود به هر دو صفر امتیاز داده میشود.

در روش minimax بازیکن ماکس به دنبال حالتی است که به برد یا تساوی برسد پس برای رسیدن به آن به صورت بازگشتی جستوجو میکند و وضعیت فعلی و حرکات موجود در آن حالت را در نظر میگیرد سپس با هر حرکت معتبری که ممکن است بازی میکند تا زمانی که به یک حالت پایانی برسد اما نکته ی قابل توجه این است که چون ما در این سوال ۲۵ خانه داریم بررسی همه ی حالت ها از توان خارج است و عمق بسیار زیادی داریم پس برای اینکه مسئله قابل حل باشد یک حد برای عمق در نظر میگیریم که برای حالتی که با هرس آلفا-بتا سوال را میکنیم این حد را ۸ و برای حالت عادی این حد را ۶ در نظر میگیریم . چون زمانی که حد را بیشتر از این اعداد در نظر میگیریم زمان زیادی را برای هر حرکت باید منتظر بمانیم.

در کد این تکلیف state را داریم که بیانگر وضعیت فعلی صفحه ی بازی است. و depth را داریم که ایندکس گره ها در درخت بازی است. و player را داریم که میتواند در نقش مین یا مکس باشد.

در ابتدای بازی هر دو بازیکن با بدترین امتیاز خود بازی را شروع میکنند به طوری که اگر بازیکن مکس باشد امتیاز اولیه ی آن منفی بی نهایت است و اگر بازیکن مین باشد امتیاز اولیه ی آن مثبت بی نهایت است.

```
for cell in empty_cells(state):
    x, y = cell[0], cell[1]
    state[x][y] = player
    score = minimax(state, depth - 1, -player)
    state[x][y] = 0
    score[0], score[1] = x, y
```

یکی از قسمت های اصلی الگوریتم تصویر فوق است که x ردیف خانه و y ستون خانه است و depth-1 ایندکس حالت بعدی است و player- به این معنا است که اگر بازیکن min باشد max در نظر گرفته شود و برعکس.

```
if player == COMP:
    if score[2] > best[2]:
        best = score
else:
    if score[2] < best[2]:
        best = score
```

می دانیم بازیکن مکس به دنبال بیشتر شدن است پس امتیاز بیشتر را دریافت میکند و بازیکن min دنبال کمتر شدن است پس امتیاز کمتر را دریافت میکند.

همچنین برای مشخص کردن این که چه حالت هایی بازیکن برنده میشود تابع wins را داریم که تمام حالت های ممکن که میتواند سه تا خانه ی کنار هم به صورت سطری، ستونی و قطری شامل حرف یکسان شود مشخص شده است.

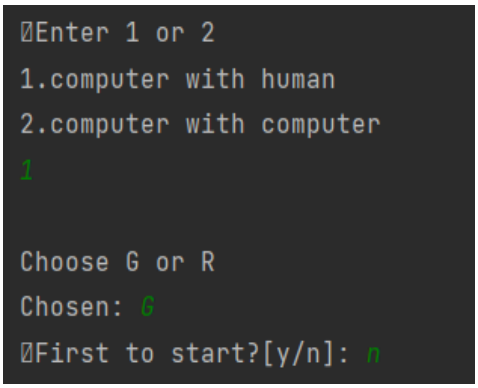
که قسمتی از آن ها را در تصویر پایین مشاهده میکنید:

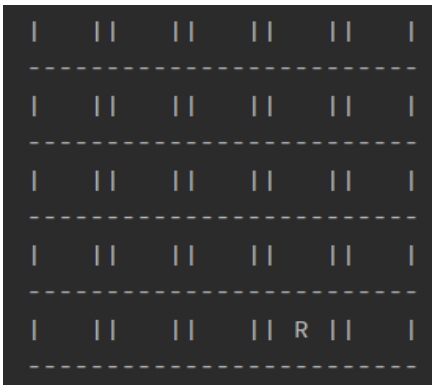
```
def wins(state, player):
    win_state = [
        [state[0][0], state[0][1], state[0][2]],
        [state[0][1], state[0][2], state[0][3]],
        [state[0][2], state[0][3], state[0][4]],
        [state[1][0], state[1][1], state[1][2]],
        [state[1][1], state[1][2], state[1][3]],
        [state[1][2], state[1][3], state[1][4]],
        [state[2][0], state[2][1], state[2][2]],
        [state[2][1], state[2][2], state[2][3]],
        [state[2][2], state[2][3], state[2][4]],
        [state[3][0], state[3][1], state[3][2]],
        [state[3][1], state[3][2], state[3][3]],
        [state[3][2], state[3][3], state[3][4]],
        [state[4][0], state[4][1], state[4][2]],
        [state[4][1], state[4][2], state[4][3]],
        [state[4][2], state[4][3], state[4][4]],
        [state[0][0], state[1][0], state[2][0]],
        [state[1][0], state[2][0], state[3][0]],
        [state[1][0], state[2][0], state[3][0]],
        [state[2][0], state[3][0], state[4][0]],
        [state[0][1], state[1][1], state[2][1]],
        [state[1][1], state[2][1], state[3][1]],
        [state[2][1], state[3][1], state[4][1]],
        [state[0][2], state[1][2], state[2][2]],
        [state[1][2], state[2][2], state[3][2]],
        [state[2][2], state[3][2], state[4][2]],
        [state[0][3], state[1][3], state[2][3]],
        [state[1][3], state[2][3], state[3][3]],
```

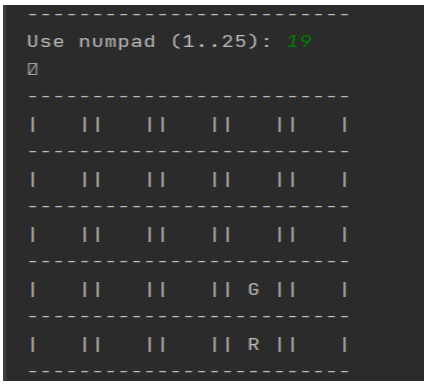
لازم به ذکر است برای حرکت اول کامپیوتر آن را به صورت رندوم در نظر میگیریم:

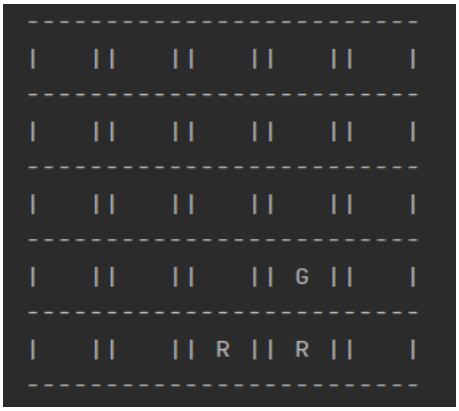
```python
if depth == c and first_flag:
    x = choice([0, 1, 2, 3, 4])
    y = choice([0, 1, 2, 3, 4])
```

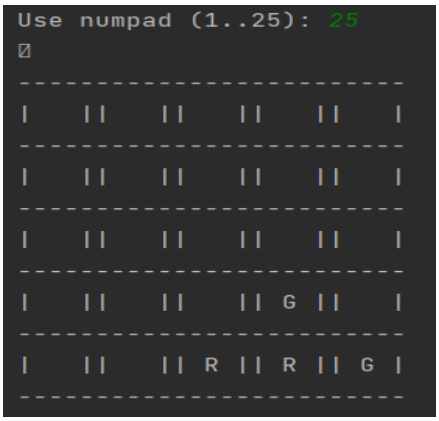
حال الگوریتم را اجرا میکنیم بازی به صورت زیر خواهد بود:

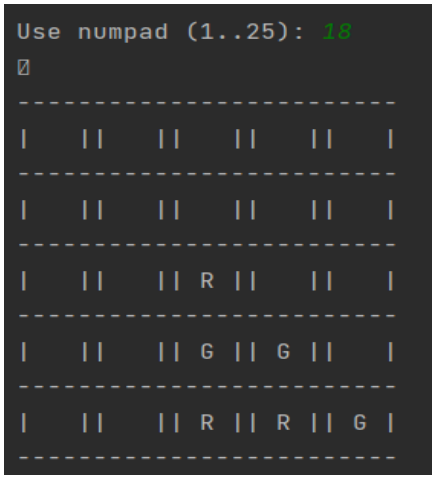


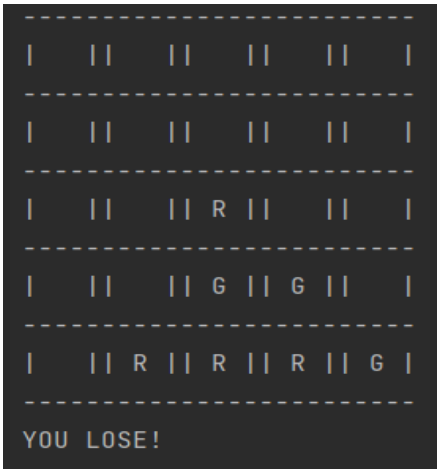




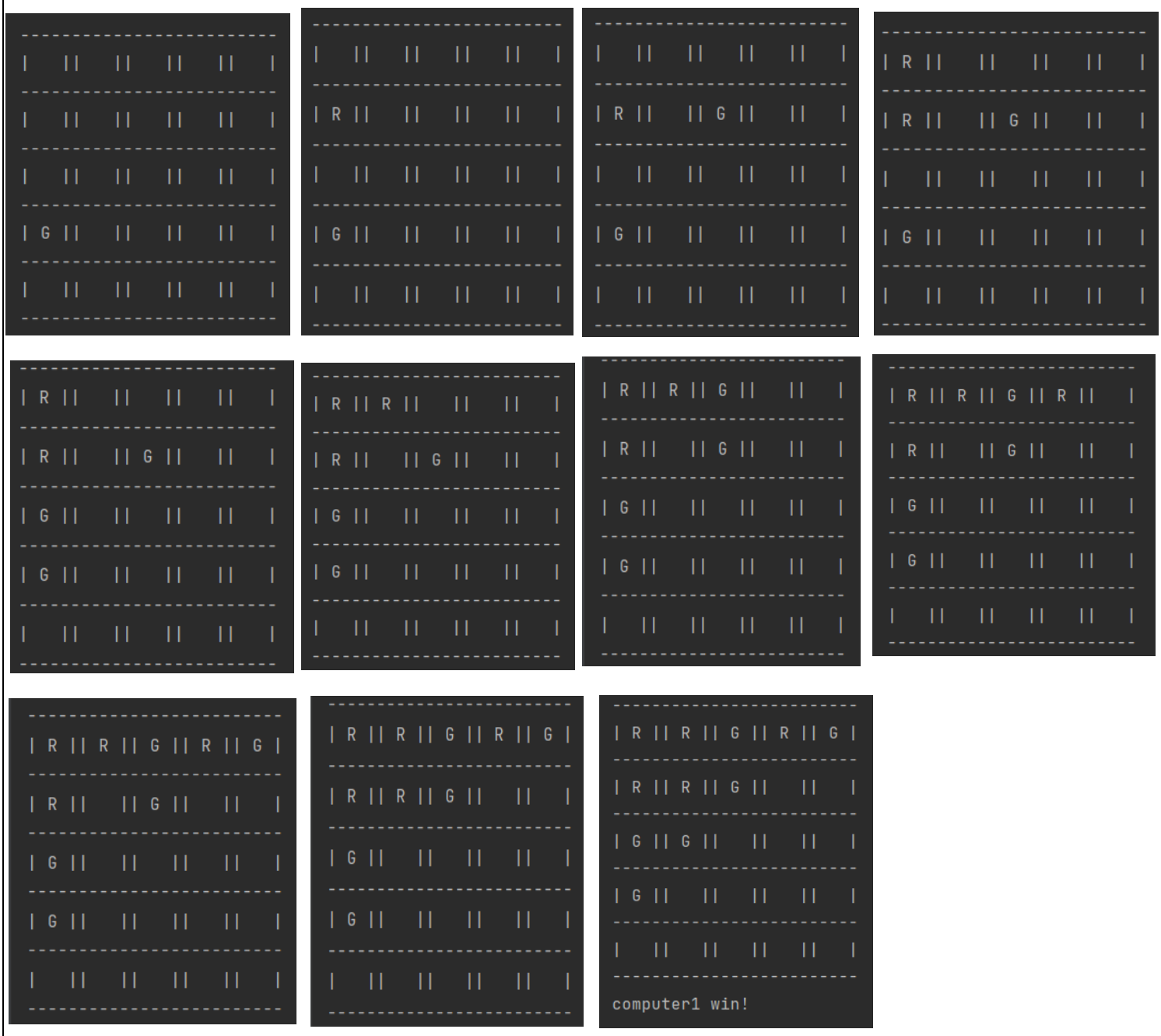








برای حالتی که دو کامپیوتر با هم بازی کنند:



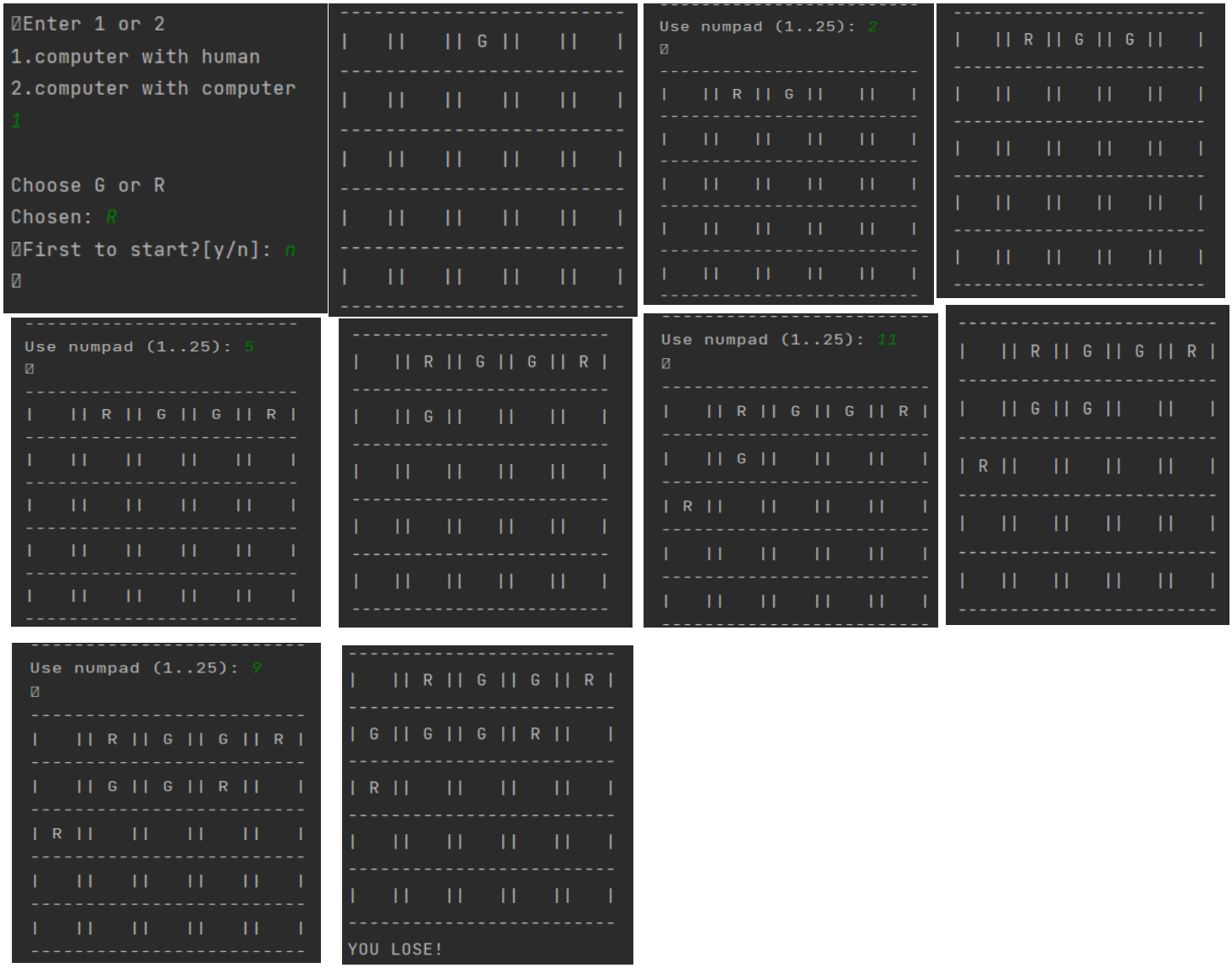
قسمت دو : هرس آلفا-بتا

آلفا بهترین (با بالاترین ارزش) انتخابی است که تا کنون در هر نقطه از مسیر پیدا کرده ایم و مقدار اولیه ی آن منفی بی نهایت است و بتا نیز بهترین (کمترین ارزش) انتخابی است که کرده ایم و مقدار اولیه ی آن مثبت بی نهایت است این الگوریتم گره هایی را که بر تصمیم نهایی تاثیر نمیگذارند حذف میکند به عنوان مثال زمانی که هرس آلفا بتا را پیاده سازی کردیم توانستیم عمق بیشتری را بررسی کنیم چون تعداد حالات را برای ما کاهش میدهد. به تابع minimax علاوه بر مواردی که قبلا داشتیم آلفا و بتا را نیز پاس میدهیم به شکلی که در تصویر زیر آمده است بررسی میکنیم:

```python
if player == COMP:
    if score[2] > alpha:
        alpha = score[2]
        best = score
else:
    if score[2] < beta:
        beta = score[2]
        best = score

if alpha >= beta:
```

الگوریتم را در این حالت نیز اجرا میکنیم خروجی به حالت زیر خواهد بود:



لازم به ذکر است برای اجرای الگوریتم برای حالت دو کامپیوتر باید c و c_count_d را از ۶ به ۵ در ابتدای کد تغییر داد.