عنوان درس:
مبانی کامپیوتر و برنامه‌سازی

مدرس: زینب السادات موسویان

موضوع: پروژه پایانی

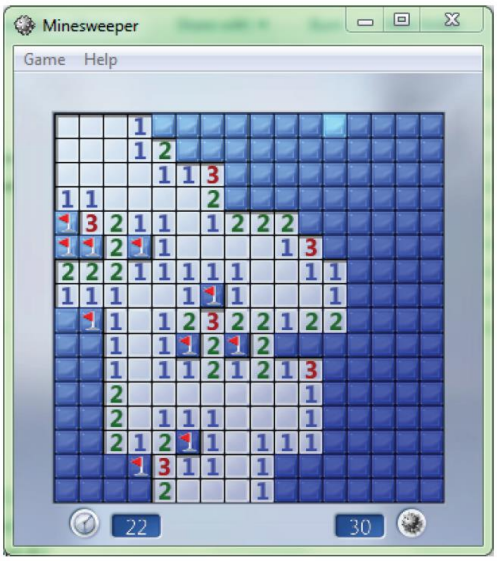
موعد تحویل: ۸ اسفند ماه

دانشکده ریاضی، آمار و علوم کامپیوتر

بازی مین‌روب

در این پروژه قصد داریم در قالب پیاده‌سازی یک بازی به نام مین‌روب، مفاهیمی که در درس یاد گرفته‌اید را مرور کنیم.

بازی مین‌روب (MineSweeper) یک بازی فکری تک‌نفره است. این بازی شامل یک صفحه‌ی مربع (یا مستطیل) شکل است که در تعدادی از خانه‌های آن مین قرار داده شده است. ابتدا همه‌ی خانه‌ها پنهان هستند و بازیکن باید تمام مین‌ها را پیدا کند. در هر حرکت ما می‌توانیم یک خانه را که حدس می‌زنیم شامل مین نیست آشکار کنیم. اگر حدس ما درست باشد، مجموع تعداد مین-های موجود در خانه‌های همسایه‌ی آن خانه نمایش داده می‌شود. اگر به اشتباه خانه‌ای که شامل مین است را آشکار کنیم، بازی را خواهیم باخت. در حین بازی، خانه‌هایی را که حدس می‌زنیم شامل مین هستند با پرچم مشخص می‌کنیم. اگر بتوانیم تمام مین-های موجود در صفحه را به درستی بیابیم، برنده‌ی بازی خواهیم بود. برای آشنایی بیشتر با این بازی، می‌توانید به نسخه‌ی آنلاین آن در MineSweeperOnline.com مراجعه کنید.



شکل ۱: نمای کلی بازی

در ادامه‌ی مطلب، تلاش کردیم تا گام به گام شما را وارد فضای پروژه کنیم. شما هم تلاش کنید با دقت صورت پروژه را مطالعه کرده و با هر گام، تصویر ذهنی خود را از پروژه واضح‌تر کنید.

موفق باشید!

۱. جدول بازی

برای پیاده‌سازی بازی، نیاز داریم که اطلاعات خانه‌های جدول را به گونه‌ای ذخیره کنیم. برای مثال، وجود مین، وجود پرچم، آشکار بودن و یا پنهان بودن، همه ویژگی‌های مختلف در مورد خانه‌ها هستند که باید به گونه‌ای تمام آن‌ها را ذخیره کنید. هرکدام از این ویژگی‌ها را می‌توان در متغیرهایی از نوع لیست دوبعدی (لیستی از لیست‌ها) ذخیره کنید. از آن‌جایی که تمام توابعی که بعداً معرفی می‌شوند، به این اطلاعات نیاز خواهند داشت، باید متغیرهای مربوط به جدول بازی هم به صورت global تعریف شوند. نمونه‌ای از متغیرهای لازم برای بازی در زیر تعریف شده‌اند. البته شما می‌توانید در طول پروژه هر تعداد دیگری ازمتغیرها که نیاز دارید به این بخش اضافه کنید، و یا داده‌ها را به هر شکل منطقی دیگری نگه‌داری کنید.

```
# global variables
global size, board, mines
```

در نمونه کد بالا، متغیر اول اندازه‌ی جدول، یعنی تعداد سطرها و ستون‌ها را نگهداری می‌کند. در این پروژه، ما فرض می‌کنیم که جدول بازی مربع شکل است و بنابراین تعداد سطرها و ستون‌های آن برابر است. متغیر board برای نگهداری وضعیت خانه‌ها از لحاظ آشکارسازی و قرارگرفتن پرچم استفاده می‌شود. به طور مثال، هر خانه از این جدول می‌تواند شامل عدد ۰ به معنی پنهان بودن، ۱ به معنی آشکار گردیدن، و ۲ به معنی قرار گرفتن پرچم باشد. متغیر mines نیز برای نگهداری مکان‌های مین مورد استفاده قرار می‌گیرد.

۲. تولید جدول تصادفی

پیش از شروع بازی، باید جدول بازی را ایجاد کرده و مین‌ها را به صورت تصادفی در خانه‌های آن قرار دهید. برای این کار تابعی به شکل زیر بنویسید که اندازه‌ی جدول و تعداد مین‌ها را به عنوان ورودی گرفته و جدول بازی را بسازد. شما باید همه‌ی متغیرهای مربوط به جدول بازی را در این تابع مقداردهی کنید.

```
def create_board(board_size, num_mines):
    '''Create a random board of given size with given number of mines'''
```

۳. نمایش جدول

برای نمایش جدول بازی در خروجی، تابع زیر را تعریف می کنیم.

```
def print_board(show_mines = False):
    '''Print game board; also show mines if show_mines is True'''
```

این تابع، به ازای هر خانه‌ی بدون پرچم یک کاراکتر '#' و به ازای هر خانه‌ی پرچم‌دار یک کاراکتر 'p' چاپ می‌کند. به طور مثال در جدول ۴×۴ زیر خانه‌های (۰, ۲) و(۲, ۲) حاوی پرچم هستند.

```
# # # #
# # # #
p # p #
# # # #
```

پارامتر ورودی این تابع مشخص می‌کند که آیا مین‌ها نیز باید در جدول نشان داده شوند یا خیر. در صورتی که مقدار این پارامتر False باشد، مین‌ها هم مثل خانه‌های خالی با همان کاراکتر '#' ، و در صورتی که مقدار این پارامتر True باشد با کاراکتر'@' نشان داده می‌شوند. (خانه‌های مین که بر روی آن‌ها پرچم گذاشته شده است با همان 'p' نشان داده شوند.) مثال زیر را ببینید.

```
# @ # #
# # # #
p # p @
# @ # #
```

۴. پرچم‌گذاری

همان‌طور که دیدیم، بازیکن می‌تواند در طول بازی بر روی خانه‌هایی که حدس می‌زند شامل مین هستند، پرچم قرار دهد. در این بخش توابع موردنیاز برای گذاشتن و برداشتن پرچم‌ها را پیاده‌سازی می‌کنیم. تابع زیر در سطر و ستون داده شده یک پرچم قرار می‌دهد.

```
def put_flag(row, col):
    '''Put a flag in the given cell'''
```

تابع زیر پرچم موجود در خانه‌ی داده شده را حذف می‌کند.

```
def remove_flag(row, col):
    '''Remove flag from the given cell if any exists'''
```

۵. آشکارسازی

در این بخش، توابع مربوط به اجرای بازی را پیاده‌سازی می‌کنیم. ابتدا لازم است که بتوانیم تعداد مین‌های اطراف یک خانه از جدول را محاسبه کنیم. تابع count_mines این کار را برای ما انجام دهد. به این صورت که مختصات یک خانه را گرفته و تعداد مین‌های موجود در (حداکثر) هشت خانه‌ی اطراف آن را برمی‌گرداند.

```
def count_mines(row, col):
    '''Count the number of mines in cells adjacent to a given cell'''
```

تابع زیر خانه‌ی داده شده از جدول را آشکار می‌کند (مشابه کلیک کردن بر روی یک خانه در بازی گرافیکی). اگر خانه‌ی انتخاب شده خالی بود، خروجی تابع تعداد مین‌های موجود در خانه‌های اطراف آن خانه خواهد بود. در صورتی‌که خانه‌ی انتخاب شده حاوی یک مین بود، خروجی تابع ۱- است.

```
def reveal_cell(row, col):
    '''Reveal a cell. If it is bomb, return -1, else retuen number of
       mines around the cell'''
```

هنگام فراخوانی تابع print_board به ازای خانه‌هایی که آشکار شده‌اند، به جای '#' باید تعداد مین‌هایی که در اطراف آن خانه وجود دارد گذاشته شود. برای مثال اگر در جدول قبلی، تابع reveal_cell را برای خانه (۱, ۱) و سپس (۳, ۰) فراخوانی کنیم، با صدا زدن تابع print_board(True) خروجی زیر را خواهیم داشت.

```
# @ # 0
# 1 # #
p # p @
# @ # #
```

نکته: همان‌طور که مشاهده می‌کنید، خانه‌ی (۳, ۰) دارای مقدار صفر است. در بازی اصلی مین‌روب، هرگاه شما بر روی خانه‌ای که عدد آن صفر است (یعنی نه خودش و نه هیچ‌کدام از همسایگانش مین ندارند) کلیک کنید، خانه‌های جدول تا رسیدن به مرز جدول یا خانه‌های غیرصفر همگی آشکار می‌شوند. در این قسمت از پروژه، برای سادگی کار شما، این قسمت امتیازی خواهد بود! اگر عدد خانه‌ای که آشکار می‌کنید صفر بود، به صورت بازگشتی تمام خانه‌های همسایه‌ی آن را که صفر هستند آشکار کنید.

۶. وضعیت برد

در این قسمت می‌خواهیم وضعیت بردن بازی را چک کنیم. بازیکن موقعی بازی را می‌برد که همه‌ی خانه‌های خالی جدول آشکار شده باشند و روی همه‌ی خانه‌های حاوی مین در نقشه پرچم گذاشته شده باشد. تابع زیر باید بررسی کند که آیا چنین حالتی پیش آمده است یا خیر. اگر پیش آمده باشد تابع مقدار True و در غیر این صورت مقدار False برمی‌گرداند.

```
def won():
    '''Check if the player has won or not'''
```

۷. بازی نهایی

حالا می‌خواهیم با استفاده از توابعی که نوشتیم، یک بازی کامل و قابل اجرا بسازیم. کد این قسمت در تابع زیر نوشته می‌شود.

```
def main():
    '''The main loop of the game'''
```

ابتدا از ورودی دو عدد بخوانید که به ترتیب مشخص کننده‌ی اندازه‌ی جدول (تعداد سطرها و ستون‌های آن) و تعداد مین‌های موجود در جدول است. با استفاده از تابع نوشته شده در بخش های قبل، یک جدول تصادفی با مشخصات داده شده تولید کنید. سپس تا تمام شدن بازی، دستورهای کاربر را از ورودی بگیرید و آن را اجرا کنید. دستورهایی که کاربر وارد می‌کند در قالب زیر هستند:

```
[r|f|u|x] <row> <col>
```

هر دستور با یکی از کاراکترهای u, f, r و x شروع شده و سپس مختصات یک خانه از نقشه داده می‌شود. شما باید دستوری که کاراکتر داده شده مشخص می کند را بر روی آن خانه اجرا کنید. دستور ها به صورت زیر هستند:

r : دستور آشکارسازی است. یعنی باید روی خانه‌ی داده شده تابع reveal_cell را صدا بزنید.

f : دستور گذاشتن پرچم است. یعنی باید روی خانه‌ی داده شده تابع put_flag را صدا بزنید.

u : دستور برداشتن پرچم است. یعنی باید روی خانه‌ی داده شده تابع remove_flag را صدا بزنید.

x : دستور خروج از برنامه است. در صورت ورود این دستور، نیازی به ذکر مختصات خانه‌ای از جدول نیست.

به عنوان مثال، دستور زیر به این معنی است که خانه‌ی موجود در سطر ۳ و ستون ۲ را آشکار کنید.

```
r 3 2
```

بعد از ورود هر یک از دستور های بالا و اجرای آن توسط برنامه، باید نقشه‌ی بازی را (بدون نشان دادن محل مین‌ها) چاپ کنید. در صورتی که دستور داده شده توسط کاربر معتبر نبود، مثلا کاراکترهای وارد شده هیچ یک از چهار کاراکتر موردنظر نبوده، یا مختصات داده شده صحیح نبود، عبارت زیر را چاپ کرده و به سراغ دستور بعدی بروید.

```
Invalid Command!
```

بعد از اجرای هر دستور باید بررسی کنید که آیا بازی تمام شده است یا خیر. در صورتی‌که بازیکن باخته باشد (یعنی دستور آشکارسازی خانه‌ای که در آن مین است را وارد کرده باشد) عبارت زیر را چاپ کنید:

```
You Lost!
```

در صورتی که بازیکن بازی را برده باشد، عبارت زیر را چاپ کنید:

```
You Won!
```

بعد از چاپ هر کدام از عبارت‌های بالا، اجرای برنامه را تمام کنید.