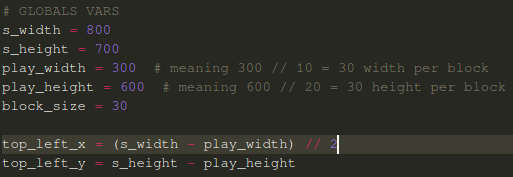
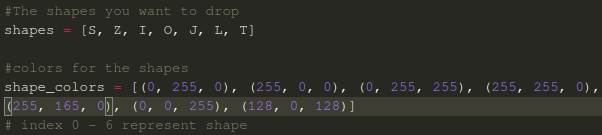
**Tetris Support**

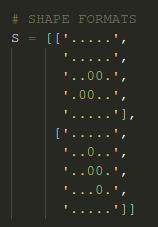
1. Support code

* Trên command prompt trên Windows hoặc terminal trên Ubuntu, nhập câu lệnh **pip3 install pygame** để cài đặt thư viện PyGame, phục vụ xây dựng ứng dụng Tetris mà mình sắp trình bày bên dưới.
* Xác định kích thước cửa sổ hiển thị là 800\*700, và 1 khung nhỏ bên trong kích thước 300\*600 để chơi xếp hình - là tập hợp các ô, mỗi ô có kích thước = 30, gồm 20 hàng và 10 cột, vị trí góc trên bên trái của khung nhỏ nằm ở tọa độ x = 250 và y = 100 trong khung cửa sổ hiển thị

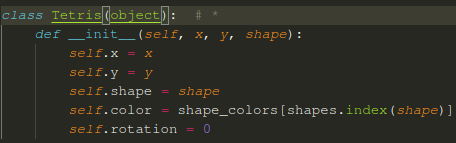


* **shapes** chứa các hình dạng khối hình thả xuống, **shape\_colors** chứa các màu, ví dụ, **shape\_colors[0]** = (0,255,0) là màu xanh lá cây còn **shapes[0]** là hình chữ S đã được định nghĩa hình dạng như hình dưới, các hình dạng khác được trình bày đầy đủ trong code

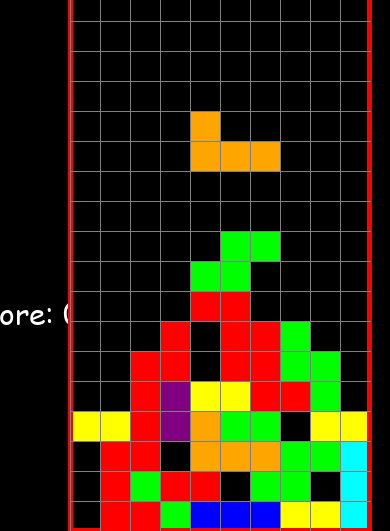


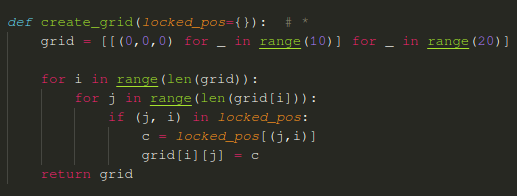


* Tạo class **Tetris**, tạo các biến x, y để lưu vị trí, shape để lưu hình dạng của khối hình thả xuống, color để lưu màu sắc của hình đó và biến rotation để lưu các kiểu quay có thể có của mỗi khối hình:

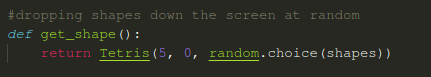


* + Dòng**shapes.index(*shape*)** sẽ trả về vị trí lần xuất hiện đầu tiên của giá trị được chỉ định, nếu **shape** là hình ‘S’ thì **shapes.index(*shape*) = 0**
  + Khi đó **shape\_colors[shapes.index(*shape*)] = shape\_colors[0] =** (0,255,0)
  + Ta sẽ gán từng màu cho từng khối hình, ví dụ phần tử đầu tiên trong **shape\_colors** là (0, 255, 0) - màu xanh lá cây thì gán cho phần tử đầu tiên của **shapes** là chữ S, vậy các khối hình dạng chữ S đều có màu xanh lá cây, làm tương tự với tất cả các khối hình còn lại.
* Hàm **create\_grid():** Sau khi 1 khối hình rơi xuống bảng trò chơi, vị trí các ô nơi khối hình đó rơi xuống trong bảng trò chơi, sẽ hiển thị cùng màu với chính khối hình đang rơi xuống đó, ví dụ:

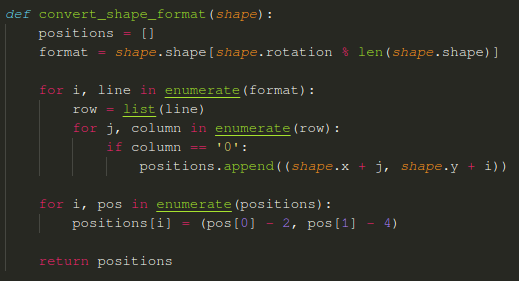


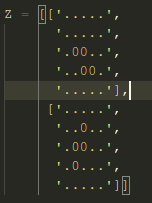
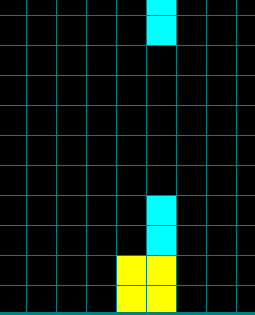


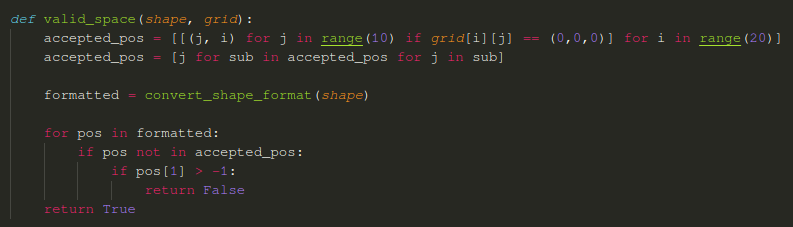
* + Khởi tạo 1 biến **grid** là 1 list nhiều chiều, mỗi list chứa 20 ***list con*** - đại diện cho 20 hàng, mỗi ***list con*** chứa 10 phần tử - mỗi hàng có 10 ô, ví dụ **grid[0][1]** thể hiện **tọa độ góc trên bên trái của ô** thứ 2 hàng đầu tiên
  + Mỗi phần tử trong ***list con*** đại diện cho màu của ô đó và có dạng (0,0,0) - màu đen
  + Vòng for đầu tiên - duyệt qua 20 hàng, vòng for thứ 2 - duyệt từng ô trong mỗi hàng, **locked\_pos ={}** là kiểu dữ liệu dictionary trong python, mỗi phần tử là 1 cặp key-value, trong đó key sẽ là vị trí của ô trong bảng, và value là màu của ô đó
  + Tham số vị trí bị khóa **c =** **locked\_pos ={[j,i]}** chứa vị trí của khối hình rơi xuống trong bảng trò chơi và màu của nó, gán **grid[i][j] = c** =>tìmvị trí tương ứng trong bảng trò chơi với vị trí bị khóa và đổi màu.
* Hàm **get\_shape()**: Thả các hình khối dạng ngẫu nhiên xuống



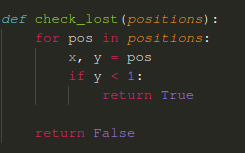
* + **random.choice(shapes)**: Lựa chọn ngẫu nhiên 1 phần tử trong list **shapes** đã định nghĩa bên trên, đó là hình dạng các khối sẽ thả xuống
  + Trả về hình dạng khối bất kỳ sẽ thả xuống và vị trí thả x = 5, y = 0 trong bảng trò chơi.
* Hàm **convert\_shape\_format()**: Thay đổi hình dạng có thể có của mỗi khối hình đang thả xuống



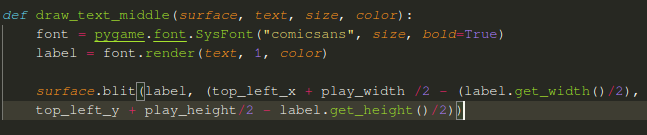
* + **format** đại diện cho các hình dạng khác nhau của 1 khối, mỗi khối có tối đa 4 hình dạng khác nhau, tạo **positions** là 1 list rỗng để lưu vị trí khối hình
  + Ví dụ, khối hình chữ "T" ta định nghĩa bên trên có 4 hình dạng khác nhau => **len(shape.shape) = 4**,  **shape.rotation** bắt đầu từ 0, giả sử mỗi khi ấn mũi tên lên trong bàn phím, **rotation** tăng 1, chia lấy dư cho **len(shape.shape)** sẽ luôn được 1 khoảng giá trị từ 0 đến 3, ví dụ kết quả ra 0 => **format = shape.shape[0]** => ta lấy hình dạng đầu tiên trong định nghĩa của hình chữ “T”
  + Duyệt qua từng phần tử trong **format**, nhưng kèm thêm index của nó, ta sử dụng **enumerate** trong python, ví dụ hiện tại **format** là chữ “Z”, có 2 phần tử trong đó , gán **row = list(line)** ta sẽ được **row** trông như thế này 
  + Vòng for thứ 2 tiếp tục duyệt qua từng phần tử trong **row** vừa tạo, nếu **column =’0’** ta thêm vào **positions** tọa độ hiện tại (x =5, y =0) nơi bắt đầu thả - xem hàm get\_shape() + với giá trị i, j tại nơi bắt được **‘0’** khi đó chương trình mới ‘hiểu’ được khối hình dạng đang thả xuống trông như thế nào
  + Và tại i, j nơi bắt được **‘0’** sẽ làm khối hình thả xuống của chúng ta bị lệch đi từ 1 đến 2 ô so với vị trí x = 5 (do ta bỏ qua 1 hoặc 2 dấu chấm đằng trước các kí tự ‘0’) , nên ta cần trừ đi 2 để các hình thả xuống luôn có tọa độ từ x = 5
* Hàm **valid\_space()**: Không gian hợp lệ, các khối hình sẽ không thể di chuyển ra khỏi bảng trò chơi
  + Đặc biệt chú ý đến dòng **if *grid*[i][j] == (0,0,0)** - nó thể hiện những vị trí hợp lệ mà khối hình có thể thả xuống trong bảng (tức là các ô còn màu đen), ví dụ 1 số ô trong bảng đã bị chiếm giữ bởi các khối hình khác (có màu) nếu không có điều kiện này, nó sẽ đè lên các khối hình đã thả trước đó như thế này 
  + Dòng đầu để duyệt qua hết các ô như các phần trên, dòng 2 convert lại format cho phù hợp, nó sẽ kiểu như thế này [[(1,2)], [(2,3)]] -> [(1,2),(2,3)]



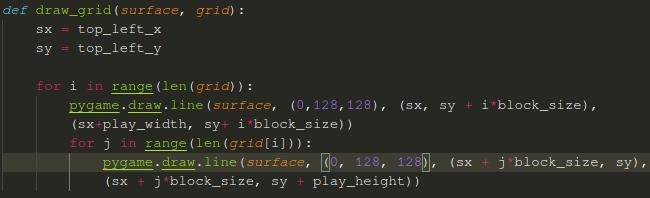
* Hàm **check\_lost()**: Kiểm tra xem ta có thua trò chơi không
  + Nếu tọa độ của khối hình, cụ thể là y mà nhỏ hơn 1 tức là 0, chạm vào đỉnh trên của bảng => thua



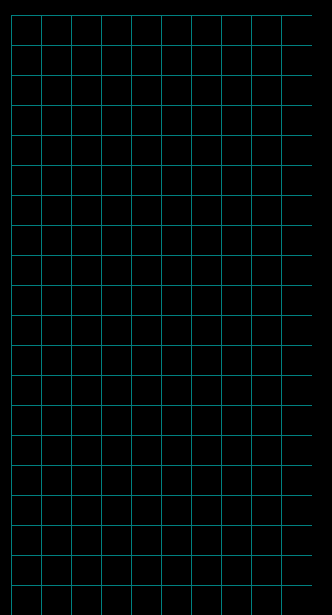
* Hàm **draw\_text\_middle()**: Hiển thị chữ ra màn hình cửa sổ trước khi bắt đầu vào trò chơi



* + Khởi tạo các thông số như font chữ, vị trí hiển thị trên cửa sổ, những yếu tố như nội dung hiển thị, màu sắc hay cỡ chữ sẽ truyền vào trong hàm main\_menu()
* Hàm **draw\_grid()**: Tham số truyền vào là **grid** trả về từ hàm **create\_grid()**, vẽ các đường thẳng xen kẽ để tạo ra các ô trong khung hình trò chơi

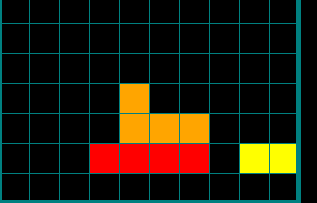


* + Duyệt qua 20 hàng, vẽ 20 đường thẳng có màu là (0,128,128) đó chính là 20 đường hàng ngang, mỗi đường cách nhau 1 khoảng **block\_size** = 30, bắt đầu từ vị trí **top\_left\_x** và kết thúc **top\_left\_x + play\_width**
  + Tương tự, duyệt qua từng ô trong 1 hàng, vẽ 10 đường thẳng có màu (0,128,128) đó chính là 10 hàng dọc, mỗi đường cách nhau 1 khoảng **block\_size = 30**
  + Kết thúc hàm này, ta vẽ được các ô có viền màu (0,128,128) - tạo thành 1 bảng trò chơi xếp hình như thế này



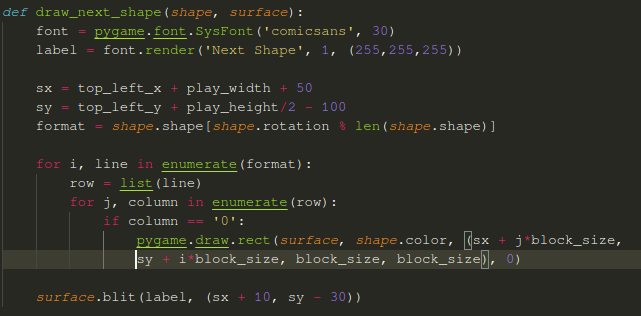
* Hàm **clear\_rows()**: Khi 1 hàng bị các khối màu lấp đầy, theo nguyên tắc hàng đó sẽ bị xóa



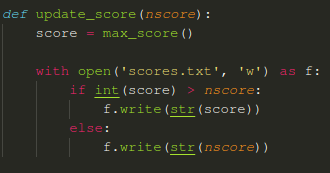
* + Duyệt hàng theo chiều ngược lại, nếu không có ô màu đen nào trong hàng (tức hàng đó toàn ô có màu), ta tiến hành xóa hết các ô trong hàng đó đi sử dụng câu lệnh **del** trong python
  + Ý tưởng là ta khi xóa các hàng bị lấp đầy ở dưới, ta sẽ thêm vào các hàng vào phần trên cùng, để đảm bảo số hàng luôn = 20
  + Sau khi xóa các hàng bị lấp đầy bởi các ô màu, hàng phía trên nó sẽ tụt xuống vị trí của hàng đã bị xóa, khi 1 hàng bị lấp đầy bởi các ô màu, biến **inc** sẽ tăng lên 1 và nó nhảy đến điều kiện **if inc > 0**
  + Biến **ind** được gán bằng vị trí hàng đã bị lấp đầy, nếu hàng nào nằm liền kề phía trên hàng bị lấp đầy (tức là **y < ind**), thì sau khi xóa, hàng đó sẽ thay thế vị trí của hàng bị lấp đầy (tụt xuống vị trí đó) => **newKey = (x, y+inc)**
  + **locked[newKey] = locked.pop(key)**: xóa hàng đã bị lấp đầy và thay thế vị trí của nó bằng hàng ở kề phía trên, nếu không có dòng này, phần bị xóa đi sẽ không bị thay thế bởi hàng trên, dẫn đến

hàng cuối rõ ràng đã bị xóa đi khi lấp đầy, nhưng phần hàng trên có vẻ vẫn chưa tụt xuống

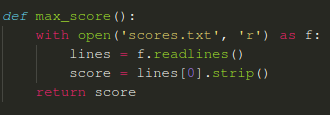
* Hàm **draw\_next\_shape()**: Hàm này xác định các hình chuẩn bị rơi xuống bảng trò chơi, và hiển thị bên phải bảng trò chơi



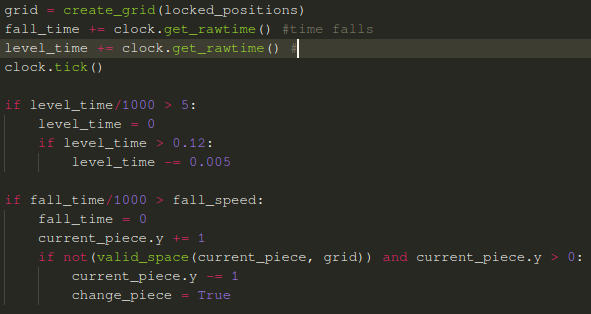
* Hàm **update\_score()**: ghi lại điểm ra file **scores.txt**, nếu lần chơi tiếp theo có điểm cao hơn lần trước, sẽ update vào file này



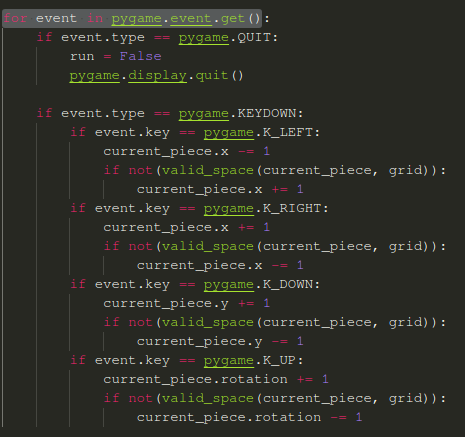
* Hàm **max\_score()**: Lấy giá trị điểm lưu trong file **scores.txt**, cột **High Scores** bên trái bảng trò chơi sẽ hiển thị số điểm đạt được cao nhất trong quá khứ



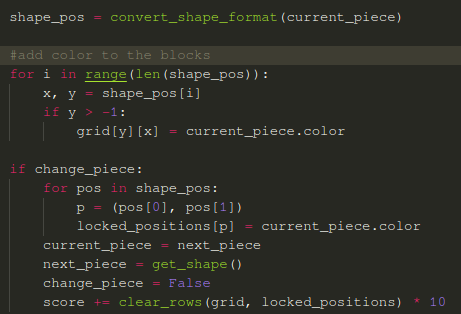
* Hàm **main():**
  + Điều này sẽ khiến cho khối hình của chúng ta di chuyển xuống (rơi xuống), khi chạm vào cuối màn hình hoặc mảnh khác, ta sẽ dừng di chuyển khối đó



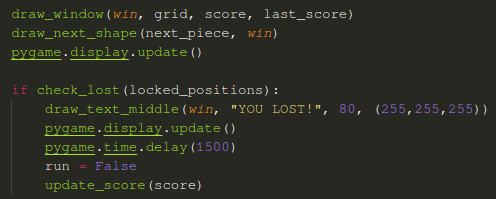
* + KEYDOWN là event mà mỗi khi bạn ấn một phím trên bàn phím xuống, check từng phím lên xuống trái phải trên bàn phím, ví dụ trái phải sẽ di chuyển khối hình, lên sẽ thay đổi hình dạng các khối, và xuống để khối hình rơi nhanh hơn, nhưng tất cả khối hình phải nằm trong bảng trò chơi, không vượt ra ngoài



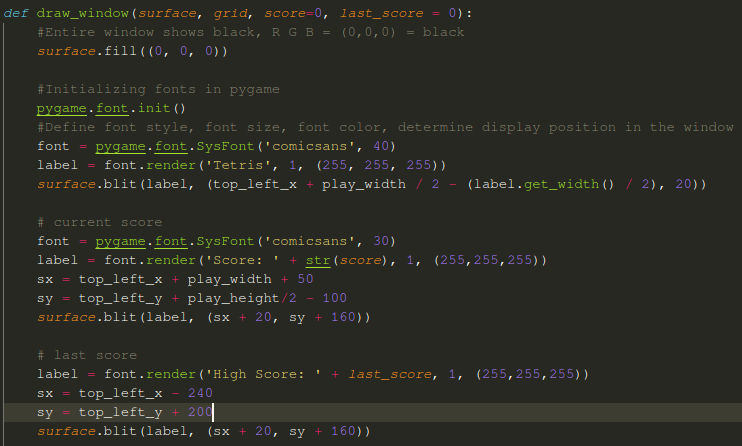
* + Tô màu vào những ô mà khối hình di chuyển qua, hiệu ứng giống như 1 khối hình có màu đang di chuyển, và khi khối hình dưới xuống cuối bảng trò chơi, vị trí đó trong bảng trò chơi sẽ có màu của khối hình, tăng 10đ sau khi lấp đầy 1 hàng



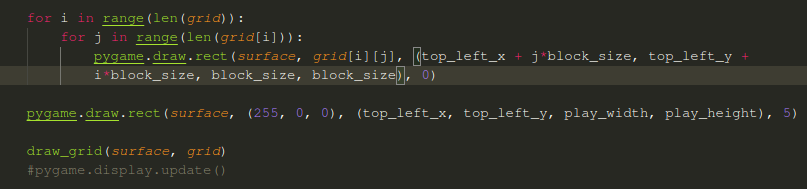
* + **next\_piece = get\_shape()** thả 1 khối hình dạng bất kỳ và truyền vào hàm **draw\_next\_shape()** để hiển thị khối hình sắp rơi xuống vào bên phải màn hình



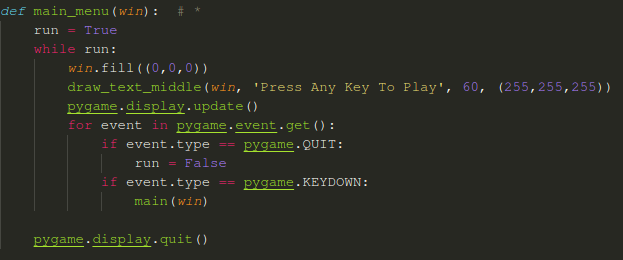
* Hàm **draw\_window()**



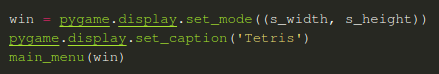
* + **surface.fill((0, 0, 0))**: Tô toàn bộ cửa sổ hiển thị lên bằng màu đen
  + **surface.blit(label, (top\_left\_x + play\_width / 2 - (label.get\_width() / 2), 20))**: Vẽ chữ "Tetris” lên cửa sổ - với vị trí bắt đầu là x = điểm nằm giữa bảng trò chơi, và y = 20
  + Tương tự, ta hiển thị chữ “Score” kèm điểm sau mỗi lần các khối xếp kín 1 hàng, vị trí hiển thị ta setup sao cho vừa mắt, bạn có thể thử thay đổi và quan sát
  + Tương tự với chữ “High Score” cũng như vậy
  + 2 vòng for giúp vẽ các khối hay nói cách khác là hiển thị các khối hình dạng đang rơi xuống bảng trò chơi, nếu không có phần này, mặc dù chương trình “đang thả” các khối hình xuống nhưng ta sẽ không nhìn thấy, hãy thử comment lại để quan sát kỹ hơn
  + **pygame.draw.rect(*surface*, (0,128,128), (top\_left\_x, top\_left\_y, play\_width, play\_height), 5)** :Vẽ đường bao xung quanh các ô với màu tùy chọn



* Hiển thị cửa sổ màu đen cùng dòng chữ **Press Any Key To Play** khi vừa chạy chương trình, gọi hàm **draw\_text\_middel** để hiển thị chữ theo ý muốn lên cửa sổ, nhận được tín hiệu từ việc ta ấn xuống 1 phím từ bàn phím để bắt đầu trò chơi



* Tạo cửa sổ với caption **Tetris**



2. Support test

* **Unit testing**: Unit testing là ở mức đơn vị, ta sẽ kiểm thử 1 số hàm trong chương trình để xem nó có hoạt động đúng với mục đích đề ra hay không. Ta sẽ phải giả lập tất cả những yếu tố xung quanh để phục vụ việc kiểm thử, ta sẽ xây dựng từng file code để phục vụ việc test từng phần (các file test sẽ được đặt tên lần lượt là Tes1, Test2, Test3,... - và test cái gì sẽ được mô tả dưới đây).Vì đây là game nên ta hướng tới việc test thủ công để quan sát thay về test tự động, xem mô tả các test case dưới đây:
  + Test1: Hiển thị các đường thẳng ngang dọc đan xen có màu xác định tạo thành 1 bảng có các ô vuông để thực hiện trò chơi. (**draw\_grid**)
  + Test2: Hiển thị 1 dòng chữ bất kỳ báo hiệu chuẩn bị bắt đầu trò chơi lên cửa sổ với kích thước đặt trước. Vị trí dòng chữ hiển thị (**draw\_text\_middle**)
  + Test3: Vẽ 1 hình chữ nhật cạnh màu với kích thước xác định để bao quanh bảng trò chơi, hiển thị các nhãn 'Tetris' ‘Score’ và 'High Score' vào các vị trí xung quanh bảng (**draw\_window**)
  + Test4: Hiển thị các khối hình màu chuẩn bị rơi xuống vào phần bên phải trong cửa sổ, bạn có thể thay đổi vị trí hiển thị của nó trên khung cửa sổ (**draw\_next\_shape**)
  + Test5: Mở file txt đang lưu điểm sau mỗi lần chơi, nó sẽ trả về số điểm cao nhất trong quá khứ (**max\_score**)
  + Test6: Lưu điểm cao nhất sau mỗi lần chơi vào file txt, nếu lần chơi mới điểm không cao bằng lần cũ, thì vẫn lưu phần cũ (**update\_score)**

**=> Error:**

* Sau Test1, không hiển thị được thành các ô, do các đường không cách nhau => bổ sung 1 tham số block\_size = khoảng cách giữa các đường mà bạn muốn
* Sau Test5, ta thấy score trả về là 1 list, ta cần nó ở dạng string để hiển thị ra bên ngoài, nếu không phần High Score sau này sẽ hiển thị lỗi như thế này ['20']
* **Integration testing**: Đây là phần kiểm thử hướng tới sự kết nối giữa các module, các test case được mô tả như dưới đây
  + Test1: Sau màn hình báo hiệu chuẩn bị vào trò chơi, ấn 1 nút bất kỳ từ bàn phím, giao diện chuyển sang bảng trò chơi với các ô, trò chơi có thể tắt đi khi ta muốn đừng (không giống như phần unittest, ta không thể trực tiếp tắt trò chơi)
  + Test2: Một khối hình bất kỳ có màu “rơi” xuống bảng trò chơi với tốc độ được setup, ta có thể di chuyển khối hình cũng như thay đổi hình dạng có thể có của khối hình dựa vào các phím
  + Test3: Khối hình khác rơi xuống tiếp sau mỗi lần 1 khối hình cũ chạm “đáy” hoặc chạm vào khối hình khác. Ngoài ra, khối hình tiếp theo rơi xuống được hiển thị bên phải bảng trò chơi
  + Test4: Khi các khối hình được sắp xếp sao cho có 1 hàng được lấp đầy, ta sẽ xóa hàng đó đi, ngoài ra, khi các khối hình chồng chéo cao hơn bảng trò chơi => Thua và bắt đầu lại ván mới

**Error**:

* + Sau Test2, ta có thể thấy khối hình có thể di chuyển ra ngoài bảng trò chơi, khiến chương trình bị crash => Phải fix trước khi vào Test3, xây dựng thêm hàm **valid\_space()** nhằm để các khối hình chỉ có thể di chuyển trong bảng trò chơi
  + Sau Test4, ta thấy khi 1 hàng được lấp đầy và bị xóa, các khối hình trong hàng liền kề bên trên không 'tụt' xuống => xây dựng thêm chương trình trong hàm **clear\_row()** để thực hiện điều đó
* **System testing**:
  + Test1: Thực hiện chạy toàn bộ chương trình và trải nghiệm với tất cả các tính năng của trò chơi, đảm bảo tất cả các tính năng được triển khai thành công:
    - Có 7 loại khối hình, mỗi loại có 1 màu xác định, mỗi khối có tối thiểu 1 hình dạng và tối đa 4 hình dạng khác nhau
    - Các khối hình được thả ngẫu nhiên xuống bảng trò chơi, người chơi có thể di chuyển, thay đổi hình dạng theo ý muốn, các khối hình chỉ di chuyển được trong bảng trò chơi, không vượt ra bên ngoài
    - Mỗi khi lấp đầy 1 hàng, hàng đó bị xóa, các khối hình ở hàng liền kề phía trên sẽ tụt xuống để thay thế, mỗi 1 hàng bị xóa sẽ được tăng 10 điểm
    - Số điểm cao nhất trong quá khứ sẽ được hiển thị bên trái màn hình, nếu các hình khối chồng chéo lên nhau và cao chạm đến 'nóc’ của bảng trò chơi => Bạn thua

**Error**: Điểm chưa được cập nhật sau mỗi hàng được xóa đi do lấp đầy

3. Test plan