PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 - Programación Avanzada (I/2016)
Tarea 2

1. Objetivos

- Aplicar conceptos y nociones de estructuras de datos para el correcto modelamiento de un problema.

2. Introducción

Cuando estabas trabajando para Bucchi Incorporated, te llamó la atención una subasta que tenía muchas apuestas. Te metiste a ver los productos gracias a tu nuevo programa, y notaste que la gran demanda era por un juego que prometía revolucionar la industria, **"SixPYx"**. Deseoso por tenerlo, utilizaste tu programa hackeador[1] para poder asegurarte de ganar la subasta.
El gran día llegó, y recibiste el juego prometido, pero tu decepción fue grande al darte cuenta que solo venía la interfaz y un extraño set de instrucciones para implementarlo[2].
Invertiste todo lo que tenías en este juego, por lo que no te queda más opción que programarlo para poder revenderlo y recuperar tus ahorros.

3. Problema

En esta tarea, deberás modelar las distintas funcionalidades del juego **"SixPYx"**. Lo anterior **debe** ser logrado utilizando conceptos y nociones de **estructuras de datos**.

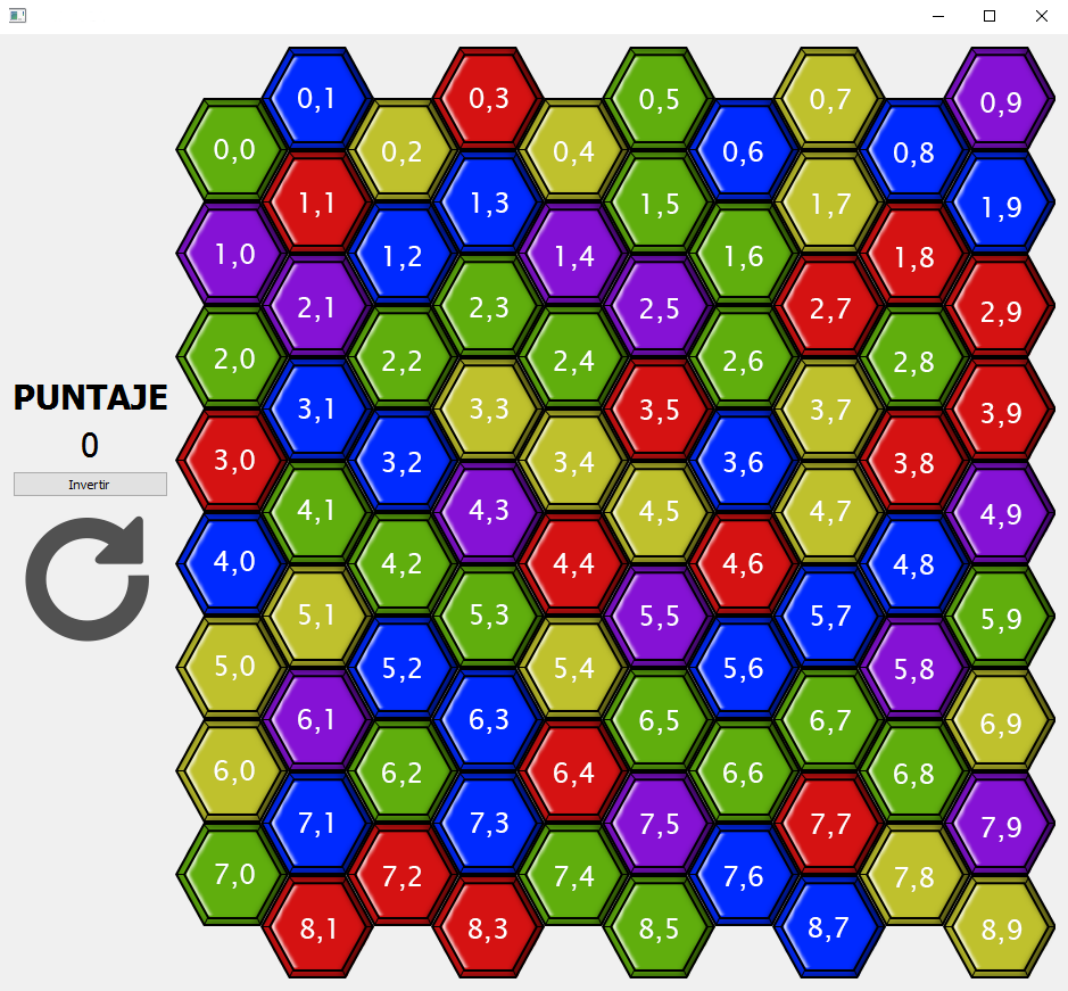
4. SixPYx

SixPYx es un juego estilo puzzle, donde se tiene un tablero compuesto por hexágonos de colores, en el que el jugador debe rotar grupos de estas figuras para formar patrones (clusters o flores), y así eliminarlos del tablero.
El cluster más básico se forma cuando tres hexágonos del mismo color se tocan. Este desaparece del tablero, lo que aumenta el puntaje del jugador. Además, esto hace que las piezas superiores caigan (pudiendo formar más clusters y generando reacciones en cadena), y que aparezcan nuevos hexágonos aleatorios en los espacios vacíos superiores.
Algunos patrones generan piezas nuevas en el juego que tienen funciones especiales, por ejemplo, una *Flor* hace que aparezca una *Estrella*. La Estrella es capaz de rotar todas las piezas que la rodean. El juego termina cuando se forma un cluster o una flor de piezas Black Pearl.

[1] Porque como buen programador, lo lograste hacer
[2] Cortesía del benévolo Bucchi

Además, ya la vida no es fácil. A veces, pueden aparecer bombas, las que de no ser eliminadas a tiempo, harán que pierdas el juego. Todo lo que necesitan saber sobre las figuras y las piezas se encuentra en los apartados 4.2 y 4.3.
Les recomendamos probar este juego[3], para acostumbrarse al desarrollo de la tarea: Hexic

4.1. Interfaz



La interfaz utiliza un sistema de coordenadas `(i,j)` y la clase `Piece` para mostrar los elementos al usuario. Para esto, guarda una referencia que asocia cada posición `(i,j)` a una pieza específica o a un espacio vacío. Más adelante se detalla dicha clase.

4.1.1. Módulo `gui`

Los pocos métodos que vienen en el módulo `gui` son los siguientes:

`init()`: Este método debe ser ejecutado antes que cualquier otro del módulo. Se utiliza para inicializar todos los componentes de la interfaz.

[3] Extrañamente parecido a nuestro querido **SixPYx**...

`set_quality(str quality)`: Con este método puedes cambiar la calidad de las animaciones de la interfaz. Los valores válidos son: "low", "medium", "highz "ultra".

`set_animations(bool animations)`: Con este método puedes activar o desactivar las animaciones de la interfaz. `True`: animaciones activadas y `False`: animaciones desactivadas.

`set_scale(float scale)`: Con este método puedes cambiar el tamaño de la interfaz, en el caso de que esta no quepa en tu pantalla.

`set_points(int points)`: Con este método puedes cambiar el puntaje que se muestra en la interfaz gráfica.

`add_piece(int i, int j, string piece_type=None, function on_move_ended=None, int number=5)`: Con este método puedes agregar piezas a la `gui`. Si no especificas el tipo de pieza, esta será determinada de manera aleatoria. Si las animaciones están activadas, las piezas caen desde arriba hasta la posición indicada por `(i,j)`. Puedes especificar una función para que sea ejecutada cuando termine la animación. El parámetro `number` indica, en caso de que la pieza es una bomba, la cantidad de turnos que quedan antes de su detonación. Si no es una bomba, `number` debe ser `-1`. En caso de que la posición esté ocupada, este método va a arrojar un error.

`get_piece(int i, int j)`: Este método retorna la pieza que se encuentra en la posición `(i,j)`. En caso de que no haya ninguna pieza en aquella posición, retorna `None`.

`pop_piece(int i, int j)`: Este método retorna la pieza que se encuentra en la posición `(i,j)`, y además, quita la referencia a esta en la interfaz. Es decir, después de hacer `pop_piece(i, j)`, el método `get_piece(i, j)` sobre la misma posición retorna `None` [4].

`move_piece(int i, int j, Piece piece, function on_move_ended=None)`: Este método mueve la pieza desde su posición actual hasta la posición `(i,j)`, y agrega la referencia en la interfaz. Puedes especificar una función para que sea ejecutada al terminar la animación[5].

`set_game_interface(GameInterface interface)`: Este método recibe una instancia de la clase `GameInterface` y la utiliza como la interfaz de la `gui`. Esta interfaz es la conexión entre la `gui` y el programador (tú)[6].

`run()`: Este método muestra la interfaz y comienza el main loop del programa. Es el último método que debes llamar.

4.1.2. `GameInterface`

Esta es la clase que utiliza la interfaz para *escuchar* las acciones que realiza el usuario. Tú debes hacer **override** de todos los métodos de esta clase y usar el método `set_game_interface(MyInterface())`. Los métodos son los siguientes:

`piece_clicked(self, int row, int column, int section, Piece piece)`: Este método es llamado cada vez que un usuario hace click en una pieza.

`reverse_clicked(self, bool clockwise)`: Este método es llamado cuando el usuario aprieta el botón Ïnvertir.º presiona espacio. Si clockwise es `True`, las piezas deben girar en sentido horario, de lo contrario deben girar en sentido antihorario.

[4] La imagen de la pieza no desaparecerá hasta que la pieza se elimine de la interfaz mediante el método `pieza.deleteLater()`

[5] Este método **NO** elimina la referencia de la posición antigua, por lo que para mover una pieza correctamente es recomendable usar este método luego de haber popeado la pieza. Ejemplo: `move_piece(i1, j1, pop_piece(i2, j2))`

[6] Tú debes heredar de la clase `GameInterface` y hacer override de sus métodos. En tu clase debes especificar todo el comportamiento que debe tener tu programa cuando el usuario interactúe con la interfaz. Más adelante se explicará con mayor detalle esta clase.

`piece_section_entered(self, int section, Piece piece):` Este método es llamado cada vez que el mouse[7] se coloca sobre una sección de una pieza.

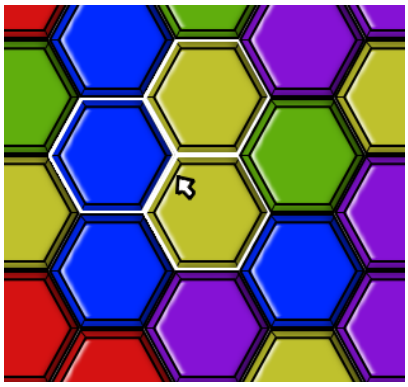
`hint_asked(self):` Este método es llamado cada vez que se apreta la tecla H.

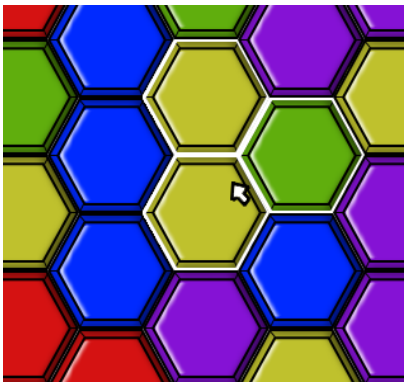
`save_map(self):` Este método es llamado cada vez que se apreta la tecla S.

4.1.3. `Secciones`

Así se encuentran divididas las piezas

Es importante tener esto en cuenta ya que los hexágonos a rotar dependen de la sección en donde fue hecho el click. En las siguientes dos imágenes se puede apreciar un pequeño ejemplo.





Además, se deben mostrar en la interfaz los hexágonos a mover en caso de hacer click en ese instante de la misma forma que se muestra en las imágenes.

4.1.4. `Piece`

Esta clase es utilizada para representar graficamente las piezas del tablero y tiene la siguiente property:

`bool self.selected:` Este atributo indica si la pieza está seleccionada o no. Al seleccionar una pieza, sus bordes se vuelven blancos.

4.1.5. `Bomb`

Esta clase hereda de `Piece`, es utilizada para representar gráficamente las bombas en el tablero y tiene la siguiente property adicional:

`int self.number` Este es el número que se muestra en la bomba.

[7] Cursor, o como quieras llamarle.

4.1.6. Tipos de piezas

 red, blue, yellow, purple, green

 star_{color}

 bomb_{color}

 star

 mirror

 black_pearl

4.2. Figuras

A partir de las piezas básicas dentro del juego, se pueden formar una serie de figuras cada vez más complejas, hasta finalmente conseguir ganar el juego.

`Cluster`

 Es la combinación mas básica del juego. Al tener 3 fichas adyacentes del mismo color, estas se destruyen, lo cual otorga la puntuación básica del juego.

`Cluster con una bomba`

 Es la misma figura que `Cluster` con una pieza reemplazada por una bomba del mismo color. Esta es la forma de deshacerse de las bombas del juego y evitar perder.

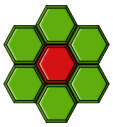
`Cluster con una estrella`

 Si dentro de cualquier combinación existe una estrella, se duplicará el puntaje obtenido por formar la figura.

`Cluster de estrellas`

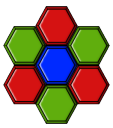
Al tener 3 estrellas adyacentes de cualquier color se destruyen, junto con las estrellas del cluster, todas las piezas adyacentes al mismo.

`Flor`



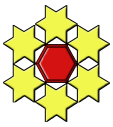
Al rodear con 6 piezas del mismo color a una pieza de cualquier color se forma una "Flor". Esto creará una pieza Estrella en el centro y destruira las 6 piezas a su alrededor.

`Flor Alternada`



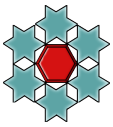
Tambien es posible rodear una pieza de manera alternada por otros dos colores distintos, creando una Flor Alternada, la cual en el centro crea una pieza Espejo.

`Flor de estrellas`



Al rodear una pieza con piezas Estrella se genera la última pieza, un Black Pearl

`Flor de espejos`



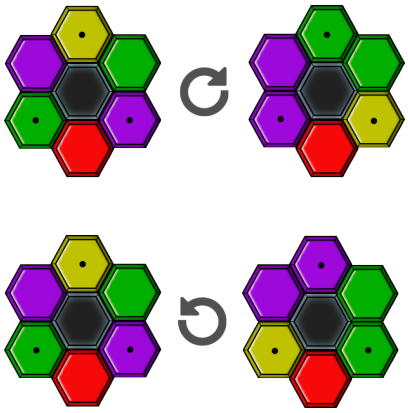
Analógicamente, al rodear una pieza por espejos se genera un Black Pearl.

`Cluster o flor de black pearls`

Finalmente, al formar un cluster de Black Pearl, se gana el juego.

4.3. Rotación

Para jugar, cada turno el usuario debe realizar una rotación de piezas. La forma más básica es seleccionar tres piezas adyacentes y al hacer click estas rotarán hasta encontrar alguna combinación, o bien volver a su lugar inicial.
Este comportamiento puede verse afectado según si existe alguna pieza especial al momento de hacer el click. Si se selecciona una pieza Estrella, esta rotará las 6 piezas adyacentes. La pieza espejo reemplazará cada una de las 6 piezas adyacentes por su lado opuesto respecto de la pieza espejo. La pieza Black Pearl rotará las 3 piezas adyacentes de manera alternada.
A continuación, se muestra la rotación de la pieza Black Pearl:



Además, el usuario puede presionar el botón elegir rotación para cambiar el sentido de la siguiente jugada.

4.4. Puntaje

El sistema de puntaje queda a su criterio y debe ser especificado en el README. Este debe tener un criterio adecuado y relacionado al objetivo del juego. Por ejemplo, que al eliminar más de 3 piezas se otorgue más puntaje (3 piezas $<$ 4 piezas $<$ 5 piezas, etc).
Además, el sistema debe considerar la dificultad de las figuras. Por ejemplo, un cluster debe dar menos puntaje que una flor o una pieza especial, o bien, una flor de espejos vale más que una flor regular.
Finalmente, deben implementar dentro de sus sistema un multiplicador de puntaje. Es decir, la eliminación en cadena de figuras va formando un combo multiplicativo en el puntaje.

5. Mapas

Debes ser capaz de leer un mapa de juego, que tendra ciertas especificaciones para poder iniciar el juego. Esto les podrá ser útil cuando quieran probrar las figuras especiales, tales como estrellas y black pearls.
Para guardar un mapa, debes presionar la tecla S. Cuando la presiones, se llamará al método **save_map(self)** de tu **GameInterface**. La interacción para guardarlos la debes hacer por consola y queda a tu criterio, siempre que mantegas el formato de los mapas y que le des al usuario la opción de nombrar el archivo que se va a guardar.

5.1. Formato Mapa

```
Puntaje
i,j,piece_type,value
```

Los valores de piece_type pueden ser: red, purple, green, yellow, blue, bomb_red, bomb_purple, bomb_green, bomb_yellow, bomb_blue, star, mirror, black_pearl.
Los valores de value serán -1 para las piezas que no son bombas o un número que indica el número que se muestra en la bomba.

6. Inicio del juego

El inicio del juego debe ejecutarse por consola. Se le debe preguntar al usuario si desea iniciar un juego nuevo o cargar un mapa guardado. En caso de querer cargar un mapa se deben mostrar los nombres de los mapas disponibles. El dónde guardarlos y con qué nombres queda a su disposición.

7. Hint

Como en muchos otros juegos, cuando uno no ve la jugada clara te ofrecen una jugada para continuar el juego, pero tú al ser un programador con experiencia, darás como hint la MEJOR jugada[8].
Una vez que encuentres esta jugada, deberás darla a conocer al jugador de una forma gráfica, cambiando los bordes de cada pieza involucrada a blanco. Para esto, tienes que setear el atributo selected de la clase Piece a True (`piece.selected = True`).
Cuando el usuario presione H se llamará al método **hint_asked(self)** de tu **GameInterface**.

8. Developer

Para poder triplicar tus ingresos con la venta del juego y asegurarte una gran reputación en la industria, decides demostrarle al mundo que eres capaz de implementarlo **SIN USAR LAS ESTRUCTURAS DADAS POR PYTHON (listas, diccionarios, sets, etc).** Debes detallar en el README las especificaciones de cómo implementaste cada una de las estructuras utilizadas.

9. Restricciones y alcances

- Tu programa debe ser desarrollado en Python 3.4
- Esta tarea es estrictamente individual y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- Su código debe seguir la guía de estilos `PEP8`
- Si no se encuentra especificado en el enunciado asuma que el uso de cualquier librería Python está prohibido. Pregunte por foro si se pueden usar librerías específicas.
- El ayudante puede castigar el puntaje[9] de tu tarea si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación de algoritmos.
- Debe adjuntar un archivo `README.md` donde se comenten los alcances y el funcionamiento de su sistema (*i.e.* manual de usuario) de forma *concisa* y *clara*.
- Cree un módulo para cada conjunto de clases. Divídalas por las relaciones y los tipos que poseen en común.
- Cualquier aspecto no especificado queda a su criterio, siempre que no pase por encima de ningún otro.

10. Entrega

- **Fecha/hora:** 13 de Abril - 23:59 hrs
- **Lugar:** GIT - Carpeta: Tareas/T02

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).

[8] Se dará a entender como mejor jugada la que otorge mayor puntaje, sin contabilizar los hexágonos por caer.
[9] Hasta −5 décimas.