Assignment 11

Blokus: A Tile-Based Strategy Board Game

EC602 Fall 2021

Contents

1 Introduction

1.1 Assignment Goals

The assignment goals are to help you learn about

- two-dimensional data structures
- error handling
- command parsing
- relationships between classes
- class design using static variables and inheritance

1.2 Group Size

The maximum group size is 3 students.

1.3 Due Date

This assignment is due December 10th at 11:59pm (just before midnight).

1.4 Assignment Value

This assignment is worth 8 points.

1.5 Late policy

Late assignments will not be accepted.

1.6 Submission Link

You can submit here: hw11 blokus submit link

2 Blokus

In this assignment, you will implement a customizable version of the board game Blokus.

Blokus tiles are **free polyonimoes**. Here is an excerpt from *wikipedia*:

```
A polyomino is a plane geometric figure formed by joining one or
more equal squares edge to edge. It is a polyform whose cells are
squares.
```

The game we construct is more a model of the physical board game than a true "computer game". We will implement the physical rules of placing tiles on the board, but not the virtual rules of how to play the game.

The format of the tiles is a square text box with `*` indicating the location of a cell or square of the tile, and `.` filling in the picture but meaning the absence of a cell. We will call the enclosing picture the "tilebox."

Here is an example of a tilebox:

```
.....
.....
*....
***..
*....
```

The *weight* of a tile is the number of its cells.

This tilebox contains a single tile with a weight of 5 connected in the shape of a sidewise capital T.

Tiles can be picked up and flipped over, and rotated by multiples of 90 degrees. The cells must be connected side-to-side, not on a corner. Think of them as rigid, flat pieces of colored plastic (since that is actually what they represent in the game).

2.1 The board

The board is also a square box with places for the cells and tiles. The representation of the board is the same as for the tileboxes.

Here is an example of a 7x7 board with two tiles placed:

```
.......
....*..
...***.
....*..
.......
.....*.
....**.
```

2.2 Requirements for Tiles

Each tile is a polyomino with the following requirements:

- all cells must be connected to the rest of the tile with an edge
- tiles must have a positive weight (1 or greater)
- tiles must be unique from all other previously created tiles. A tile which is a rotation, flip, or translation of another tile is considered the same tile.
- polyonimoes with holes are permitted

The empty tile is not considered a tile. Although Blokus tiles only have weights between 1 and 5, we allow for larger polyonimoes (they must be accepted).

If any of the requirements are not met, the tile will not be accepted.

2.3 Requirements for Tileboxes

In addition to the requirements for the tiles themselves, we also specify requirements for the "tilebox" representation used for inputting and showing the tiles.

Each tilebox must

- have one tile
- only have `*` and `.` characters
- be square

3 Program Commands

In the following, lines starting with `>` represent input, and the rest of the text is the program response.

3.1 Game Commands

3.1.1 Quit

This is used to exit the game.

```
>quit
Goodbye
```

3.1.2 Reset

Start the game from beginning using `reset`

```
>reset
game reset
```

All tiles are eliminated, the board is emptied and resized to zero, and the tile numbers will start at 100 again.

3.2 Board Commands

The following commands affect the contents of the board

- play
- resize
- reset

3.2.1 Play a tile

The game is played by placing a tile on the board by specifying the row and column number of the upper-left hand corner.

```
>play 101 4 5
played 101

>play 99 0 0
99 not played
```

If a tile does not exist or cannot be played at the specified location (because something is already there or it is placed off the board), the message "tilenumber not played" is displayed

When a tile is played, it is always placed using its current rotated or flipped state.

The same tile number can be played multiple times on the same board as long as the other rules for tile placement are satisfied.

When playing a tile, the dots of its tilebox are ignored. Here are two examples. Suppose the board is 5x5 like this:

```
.....
..***
..*.*
.....
.....
```

and tile 101 is

```
.*.
***
.*.
```

Then the command

```
play 101 2 2
```

would result in the board

```
.....
..***
..*@*
..@@@
...@.
```

where the new tile is specified with `@` for clarity. Notice that the dots of the tilebox and the dots of the board necessarily overlap. That is OK: both represent air or nothingness.

Here is another example. Suppose tile 102 is

```
***
...
...
```

it can be played with

```
play 102 4 0
```

and the result would be

```
.....
..***
..*@*
..@@@
###@.
```

where the new tile is specifed with `#` for clarity. Notice that the dots of the tilebox extend past the edges of the board, but that is ok.

3.2.2 Resize

The `resize` command can make the board bigger or smaller. The board is restricted to a square shape, and can be any size from 0 and larger.

When the board is made smaller, played tiles fall off the board if any cell of the tile extends off the edge of the new board.

The program should respond with the new board.

```
>resize 5
```

3.3 Tile Commands

The following commands apply to tiles that have not yet been placed on the board:

- create
- show
- rotate
- fliplr
- flipud

Once a tile is created, it must always be located as far up and left as possible after any operation on it.

This means that it must be translated to a position such that at least one cell is in row zero and at least one cell is in column zero.

3.3.1 Create

Tiles can be created at any time. The size of the surrounding tilebox must be specified.

```
>create 4
>..*.
>.**.
```

```
>.*..
>....
created tile 100
```

The `create` command is responsible for implementing the rules about valid tiles.

Here are the messages for invalid tileboxes:

- `invalid tile`: use this message for all other error situations: empty tilebox, tilebox is not square or invalid characters
- `duplicate of 102 discarded`: use this message if the tile is a duplicate of a previous tile
- `disconnected tile discarded`: use this message if two or more tiles are present in the tilebox

Here are examples for these:

```
>create 2
>***
>..
invalid tile
>create 2
>xo
>xo
invalid tile
>create 2
>..
>..
invalid tile
>create 2
>**
>..
created tile 100
>create 2
>*.
>*.
duplicate of 100 discarded
>create 2
>*.
>.*
disconnected tile discarded
```

Notice that the `create` command always accepts N strings even if one of the earlier strings invalidates the tilebox.

3.3.2 Show all tiles

At any time, a tile inventory can be displayed.

```
>show tiles
tile inventory
102
*
101
**
**
100
***
**.
...
```

The ordering of the tile inventory is not important.

3.3.3 Show a tile

Any previously created tile can be displayed by providing the tile ID number:

```
>show 100
***
**.
**.
```

3.3.4 Rotate

The `rotate` operation rotates a tile 90 degrees counterclockwise.

The `TileID` is used to select the tile. Here is a typical sequence ending with `rotate`

```
>create 3
***
..*
...
created tile 100
>show 100
***
..*
...
>rotate 100
rotated 100
**.
*..
*..
```

3.3.5 Flip sideways

When a tile is flipped sideways, left becomes right and right becomes left.

The `fliplr` command performs this operation.

```
>fliplr 100
fliplr 100
**.
.*.
.*.
```

The `TileID` is used to select the tile.

3.3.6 Flip vertically

When a tile is flipped vertically, up becomes down and down becomes up.

The `flipud` command performs this operation.

```
>flipud 100
flipud 100
.*.
.*.
**.
```

3.4 Other details

- the board is initially of size 0 by 0, i.e. it is empty
- the first tile is numbered 100, and each new one is assigned the next available integer
- the upper left corner is row 0 and column 0. row numbers increase going down and column numbers increase going right
- in commands, the first index is the row and the second index is the column

3.5 Error handling

3.5.1 Inside `create_piece`

Most of the error handling involves correctly detecting valid and unique tiles inside the method create_piece.

Here are the requirements. Assume the tilebox is of size N.

- once the size of the tilebox is read in, the program should read N strings, even if one of the initially provided strings is invalid
- any characters other than `.` or `*` will result in the rejection of the tilebox
- if any row of the tilebox is not of size N, the tilebox should be rejected

Keep in mind that the program should also detect multiple tiles in a tilebox and duplicate tiles in a tilebox, and reject those also.

Those are not really "errors" but they are part of the input handling process of the `create_piece` method of the `Blokus` class.

3.5.2 Other errors

You may assume

- all numbers will be numbers
- all provided tile IDs will be valid tile IDs

The play command should handle row and column numbers that are outside the current board by rejecting the piece.

4 Program Requirements

Here is the main program to get you started: blokus_original.cpp

4.1 Class Tile

Here is the required specification of the `Tile` class. It is incomplete: you need to define data attributes and constructors.

```
class Tile {

 public:
  void show() const;  // print out tile in tilebox format
  void rotate();
  void flipud();
  void fliplr();
};
```

4.2 Class `Blokus`

Here is the required specification of the `Blokus` class. It is incomplete: you need to define data attributes.

```
class Blokus {

 public:
  Tile* find_tile(TileID);
  void create_piece();
  void reset();
  void show_tiles() const;
  void show_board() const ;
  void play_tile(TileID, int, int);
  void set_size(int);
};
```

4.3 The user interface

The user interface is defined in the provided `main()` function.

Each user command is mapped to one of the methods defined above.

4.4 No brackets

Brackets (i.e. `[]`) are not permitted. Use the access and modifier methods instead.

5 Downloads

Here are some example input files:

- blokus_input_allcommands.txt
- blokus_input_set1.txt
- blokus_input_set2.txt
- blokus_input_set3.txt
- blokus_input_createshow_1.txt
- blokus_input_createshow_2.txt

and the corresponding output files

- blokus_output_allcommands.txt
- blokus_output_set1.txt
- blokus_output_set2.txt
- blokus_output_set3.txt
- blokus_output_createshow_1.txt
- blokus_output_createshow_2.txt

Here are an additional set of input and output files:

- blokus_input_duplicates.txt
- blokus_input_holes.txt
- blokus_input_disconnected.txt
- blokus_output_holes.txt
- blokus_output_duplicates.txt
- blokus_output_disconnected.txt

They can be downloaded using `cget` or you can get them from the following bundle: blokus_io_files.zip

6 Extra Credit assignment

The following parts of the assignment are extra credit:

- rejecting duplicate tiles
- rejecting disconnected tiles
- resizing the board to be smaller

7 Checker

Here are the checker links:

- hw11_blokus_check.py
- hw11_extra_check.py

The latest version of the checker is 3.0