Institut für Automatisierungstechnik

C++ Basics and Applications in technical Systems

Final Project
in WiSe 2012/2013

Further information and this exercise as download on:
http://www.elearning.uni-bremen.de

Supervisors:
Adrian Leu - Christos Fragkopoulos - Henning Kampe
{aleu,Fragkopoulos,kampe}@iat.uni-bremen.de

Blocktris Game

Universität Bremen

1 Description

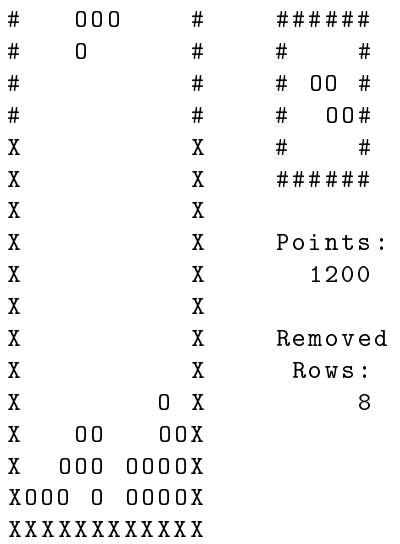

Figure 1: *The Blocktris Game*

The blocktris game seen in Figure 1 is a variant from the tetris game. A user has to place blocks consisting of four connected elements at the bottom of a board so that they build filled lines from left tor right. These connected lined vanish and the rows above move into the freed space. Each removed line is rewarded with points and the goal of the game is to achieve a maximum of points.

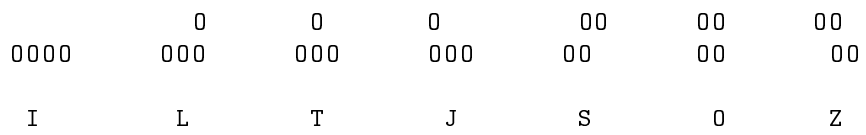

Figure 2: *The blocks in the game*

Blocks in the game consist of four connected elements. They are depicted in Figure 2. A new block is generated each round at the top in the center of the board. There it can be rotated and moved left and right. When the block is in the desired position it can be released and fall downward until it touches another element on the field. Then all elements of the block stay at this position. All complete rows are removed now and the rows above move downwards by the amount of removed rows. If a gap exists in a row and the row above has elements directly over the gap, the elements do not fall down. For each set of removed rows the amount squared times hundred points are awarded. So points are higher when multiple rows vanish at the same time. The block for each round is generated randomly. This is the same for the block for the next round which is displayed to the player at the side of the board. Width of the board is 10 and height is 16. A border is drawn in the example in Figure 1 so the outer dimensions there are 12 times 17. The upper 4 rows are reserved for positioning the block before it is released. Game ends when the current block occuppy any field within these rows after it was released and had fallen down.

The game should be playable by a human user and by an automatic agent. A user interface has to be created to show the board and to get the user input.

2 Implementation

The following has to be implemented:

- A game board with a moveable block,
- a user interface to play the game and
- an algorithm that can play the game.

For the first and third part an interface is specified that must be used and implemented. No operating system specific libraries (e.g. MSVC) may be used.

3 Given Interfaces

Your program has to use and implement the following interfaces.

3.1 BlockType_E

```
// Enum for the different block types.
enum BlockType_E
{
    I_Block_e, L_Block_e, T_Block_e, J_Block_e,
        S_Block_e, O_Block_e, Z_Block_e
};
```

3.2 Movement_E

```
// Enum for the movement of the block.
enum Movement_E
{
    MoveLeft_e, MoveRight_e,
        RotateClockwise_e, RotateCounterClockwise_e,
        Release_e
};
```

3.3 IBoard

```
class IBoard
{
public:
    IBoard() {} ;
    virtual ~IBoard() {} ;

    // Initializes the board with a random current and next block.
    virtual void InitBoard() = 0 ;

    // Returns the field at the given position.
    // A value of 0 means empty, all other values represent
    //  occupied by an element of a block.
    virtual unsigned int GetField( unsigned int PosX ,
        unsigned int PosY ) const = 0 ;
```

```
    // Moves the current block.
    virtual void Move( Movement_E Movement ) = 0 ;

    // Returns true, if the game is finished and false otherwise.
    virtual bool IsFinished() const = 0 ;

    // Returns the next block type.
    virtual BlockType_E GetNextBlockType() const = 0 ;

    // Returns the current amount of points.
    virtual unsigned int GetPoints() const = 0 ;

    // Returns the current amount of removed rows.
    virtual unsigned int GetRemovedRows() const = 0 ;
};
```

3.4 IPlayer

```
class IPlayer
{
public:
    IPlayer() {} ;
    virtual ~IPlayer() {} ;

    // Plays the given board by moving the block.
    virtual void Play( IBoard* pBoard ) = 0 ;
};
```

4 Tasks

- Design an application that combines two game modes:
  - interactive game
  - automatic mode (algorithm plays game)
- Design this application in an object oriented way (using UML) to enable the reuse of their parts.
- Supply a UML class diagram with your work.

5 Additional Information

- Work can be done in groups with up to three people. Sign up your group decision in e-Learning.
- A 0 for a boards field means that it is free. If the value is not 0, one element of a block is at this location.
- At the beginning the board is empty besides the current block at the top.
- The board consists only of the inner play field and not of the outer border.
- Board dimensions are 10 width and 16 height.
- The top left corner of the board has the coordinates x=0, y=0.

- The game ends when a fallen block occuppies any field of the upper four rows.
- Complete rows are removed every time a block has fallen.
- Rows above the removed rows descend by the amount of removed rows.
- Points for removed rows are the amount of removed rows squared times 100.
- Blocks fall directly down when they are released.
- Blocks are chosen randomly.
- The next block is shown to the player.
- For the automatic mode design an algorithm which plays the game automatically.
- The automatic player should gain as much points as possible.
- Your "Blocktris Game" has to implement the given interface, because we will test against it.
- The supplied header for the defined interfaces has to be used. Inconsistency of the implemented classes to the interfaces is a major design fault. **The supplied header files must be used and must not be changed.**
- It is not allowed to cast the objects given via the interfaces to another class! We will use it with our own classes and then the cast will not work.
- A makefile to build the application must be provided. Also sending in the Qt project (.pro) is OK.
- Target platform for your application is a linux system. Keep your application system independent to avoid problems. System independent librariey, like e.g. Qt, may be used. For more exotic libraries you have to ensure, that we have them on the reference system.

HowTo: Send in the Exercises

(Exercises are only accepted, if they apply to the following)

- Copying is not permitted. Solutions will be accepted not later than the announced deadline.
- Special formatting requirements: Use spaces and tabs to align the lines of code within command blocks. For more details see **IAT programming guidelines**.
- zip/rar the project-folder(s) and send everything in one file via Email or hand it in via usb-stick. Before zipping, delete the object-files (*.o) respectively debug/release-folders (Visual Studio). **Additionally** a paper printout has to be delivered to the supervisor.
- Preferably write your programs with Qt Creator. If you not use Creator, you have to provide the source code (again delete object-files) together with a working Makefile. If you use CASE-tools (like Rhapsody or Bouml) send in a full set of model-files **and** the generated C++ source code.