Term Project of AI Tetris for Final Examination of *Advanced Programming Language* 2-2

Junjia Wang

June 27, 2020

Contents

1 Questions and Requirements

This assignment requires the developers to construct a Tetris game which is capable of providing a sequence of operations of each block by the AI algorithm as a part the the program. The input file includes an array of blocks provided

by the professor, which contains the specific type and time for appearance of the blocks. Hence, the program are expected to imitate human players and give the processing methods of these blocks in detail.

2 Construction of the environment

This section shows the fundamental construction of the team developing environment.

2.1 Operating System

Mac OS

2.2 Integrated Environment

Qc Creator (based on Qt)

CLion + Gihub (for team cooperation)

2.3 Build Tools

QMake

Table 1 shows the operating system used, integrated environment, build tools.

Table 1: Construction of Environment

Operating System	Mac OS
Integrated Environment	Qc Creator (based on Qt) CLion + Gihub
Build Tools	QMake

3 Organization of Project

Regarding the work arrangement of the personnel for this experimental course, we strictly followed the work course specifications according to the actual situation of the team members and made the following arrangements. Please refer to the Table 2.

Table 2: Organization of Project

Member	Junjia Wang	Ziyi Hunag	Shuangpeng Ming	Chang Yang
ID	1913138	1913082	1913119	1913187
Position	Group Leader	AI Engineer	Developer Test Engineer	Developer Copywriter
the Division of Responsibility	Construction of game flow and implementation for logic of Tetris, Composition of Report Paper	Design of AI algorithm, Design of Game, Header Files, Optimization	Implementation for game information, display panel, Video record for Test files	Implementation for functions and main features of the Tetris blocks
Responsible Files	**widget.h box.h**	**box.h**	**panel.h**	**block.h**

4 Design of Game

4.1 Interface

We designed the game area coherently. Please check the Figure 1 as reference.

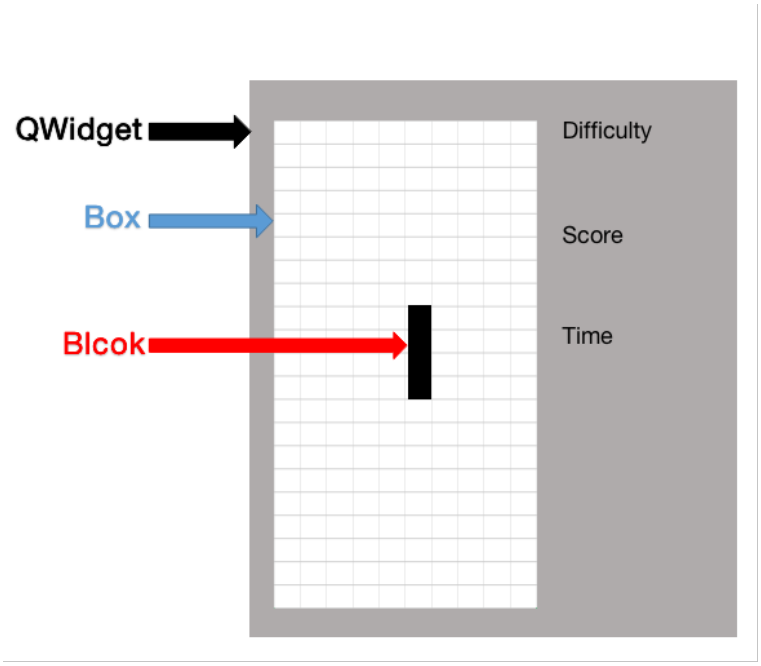

Figure 1: Interface

4.2 Types of Blocks

This section displays the five types of blocks which appears in the game. Please check the Figure 2 to 6 as reference.

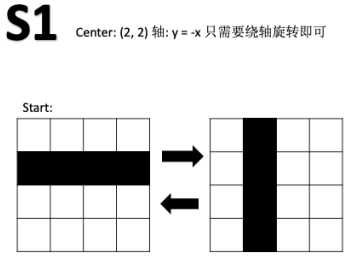

Figure 2: Block S1

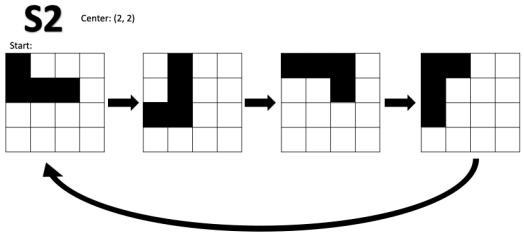

Figure 3: Block S2

P

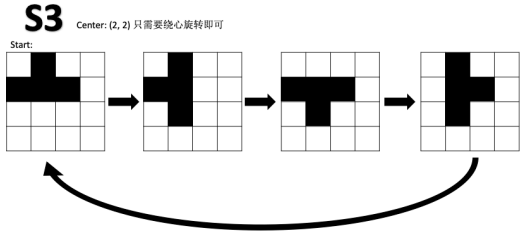

Figure 4: Block S3

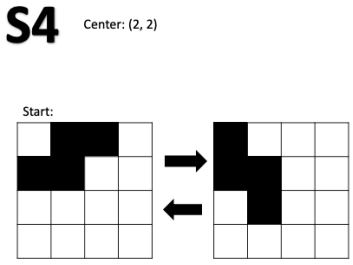

Figure 5: Block S4

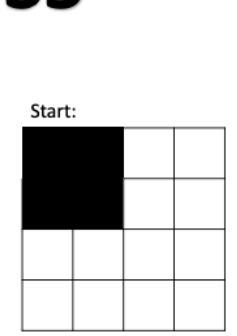

Figure 6: Block S5

5 Features of C++ in Program

The C++ language features, third-party libraries and standard library used to achieve the expected functions will be showed in this section.

5.1 Features of C++ Used

Object-Oriented Features The first feature is **Abstraction**. We make full use of C++'s object-oriented programming ideas by abstractly processing the four **boxes.h block.h panel.h widget.h**.

Second, we make the best use of this feature of **Encapsulation**. For example, in the file **block.h**, we set the type of block to coordinates with private variables. The encapsulated data prevents other classes from calling this type, and prevents program errors and bugs.

Thirdly, is the **Inheritance**. In the call to **QWidget**, **QMainWindow** and

other classes, we use the inheritance mode provided by the **Qt** framework, and use the function polymorphism provided by the **Qt** framework. It realizes the engineering of the project and fully reflects the idea of object-oriented programming.

Macro Macro is used to define constants. Since the expansion of macros is global, this greatly facilitates the use of constants such as block length, falling time interval, window length, and width by various objects for global parameters.

Precompiled Statements Precompiled statements are added in the header file of each class to prevent errors caused by repeated compilation.

Source Files and Header Files We made full use of the advantages of source files and header files in the C++ language, strictly declare data member and function member in the header file, and implement functions in the corresponding source files to achieve strict management of engineering projects, It is conducive to multi-person collaboration, code review, determination of requirements and follow-up operations of the Test Engineers.

5.2 Features of Third-Party Libraries

QWidget The base class of the program GUI, which inherits it, implements the processing of the **Box** and **Block**, and completes the realization of the visualization of the **Box**

QLabel It visualize the four information to be displayed: difficulty, score, time, and prompt.

QTimer It's a timer, using the characteristics of Qt frame *signals* and *slots* for timing and setting the refresh frequency at the same time.

QPaint It performs low-level painting on widgets and other paint devices. And it draws images of **Box** and **Block**.

QPen It defines how a **QPainter** should draw lines and outlines of shapes.

QBrush It sets the color of the game area, the line style of the game area, draws the outline of the square, etc.

QgridLayout It implementates the main window layout setting.

QString It's the Qt's support type for built-in *string*, used to set display text, integral, etc.

QTextStream It provides a convenient interface for reading and writing text, to read information.

QFile It opens file.

QQueue By using the *queue* provided in the **Qt** framework, the custom **Block** performs FIFO operations on the time dimension.

Signals and Slots They are used for communication between objects. We utilized the characteristics of *signals* and *slots* in the **Qt** framework. We set *Box::flash* as the *slot* function, set the *flash signal*, and complete the flashing effect when the line is eliminated. Likewise, We set the function of block falling, block generation, delay, etc., all realized in the form of *slot* function.

5.3 Features of the C++ Standard Library

cstdlib It provides *srand* and *rand* as two useful functions.The *srand* in **cstdlib** uses system time to randomly generate seeds. And the *rand* generates random numbers, which can randomly generate the type of block.

cmath, algorithm Implementation of AI.

6 Introduction to Implementation of Class in Program

This section displays the C++ classes in the system, respective functions and how they cooperate to complete the expected functions.

6.1 Classes of C++

Table 3: Classes of C++

Classes	Functions of the Classes
Block	Records the type of the block, the coordinates corresponding to the various rotation states. It also defines the direction of left, right, down and rotation.
Box	Records the whole chess board and the blocks on the current screen, implements **AI** for block movement, rotation and elimination of block lines.
Panel	Dashboard category, the main feature is to show the current difficulty, score, time and information on the screen, automatically update time. The rest part of the class is updated by the provided parameters from users.
Widget	The Window, it controls a **Box** object and a **Panel** object through the interface to control the game process. Meanwhile, it controls the beginning and initialization of the game.

6.2 Interfaces between Classes

Box is the friend class of **Block**. In the *can_exist* and *ai* method, it can access the coordinate members of the **Block**. It can call the *move_left, move_right, move_down*, and *rotate* methods of the **Block**.

We define a **Box** object in the **Widget**. *init* method in **Widget** calls *init* in **Box**, and *new_block* calls *new_block* in Box. Next, the *timer* drop calls *update_box* to control the box for update, and the timer *creat* controls the generation of the box. Moreover, we define the **KeyPressEvent** class, by using the keyboard to control the movement and rotation of the block, and the start and reset of the game.

Then, we define a **Panel** object, which is able to print the relevant content in **Panel** by passing the parameters, including score, difficulty and status in the **Widget**. At last, **Panel** is capable of the implementation of updating itself through *drop* and *creat*, setting the start of the clock *timer*, and reset itself in *init*.

7 Performance Evaluation of Algorithm

We explicitly mark the number of blocks that can be generated from the top line of game widget, as an indicator of the performance of the program. More precisely, the number equals to the blocks which have been eliminated and the blocks which are still remained in the game area. Please make a review of the data showed in Table 4.

Table 4: Performance Evaluation

Game Difficulty	the Number of Blocks Created
Easy	120
Normal	200
Challenge	600