Xi'an Jiaotong-Liverpool University

西交利物浦大学

CPT204 Advanced Object-Oriented Programming

Final Project

Task Sheet 2 – Supplementary

CPT204-2223 Final Project Task Sheet 2 – Supplementary Info

- This document contains supplementary information on CPT204-2223 Final Project
- You must read **Final Project Task Sheet 1 pdf** first!
  - after that, this document will give you more explanations on the code structure and more details on your tasks
  - also, it is recommended to watch the **video demo** by Haoyue first!

- If you have any questions, please ask in **Final Project Forum**
  - please check **Final Project Forum** and **LM Announcement** frequently for updates!

CPT204-2223 Final Project Skeleton Files and Demo Video

- Extract the **CPT204-2223_Final_Project_Skeleton_Files.zip**
  - the Java skeleton code files are found in **folder ataxx**,
    and so create *New Project from Existing Sources* in IntelliJ on that folder as usual
  - in **folder library**, there is a file for JUnit testing, and so *import* this library as usual for the given test case files
  - in **folder demo**, there is a *demo video mp4* file,
    and a text file that lists commands used in the demo video

Ataxx Code Overview (1)

- As the game involves many files, we will explain the main files and operations needed to understand the code structure and game implementations
  - we apply modular design, classes and methods can work independently
  - some files may contain game-playing machinery that *not* necessarily be understood
  - we start by explaining the routines of **command reading**
- The game starts at **Main.java** where game instance is be created and **game.play()** is called
- In **Game.java**, we observe the **play()** method:

  setManual(RED)

  setAI(BLUE)
  - which means in the beginning, RED plays first as a manual player and BLUE plays next as an AI player

Ataxx Code Overview (2)

- Continuing, in **Game.java**:
  - **setManual** and **setAI** will create an instance of **Manual** and **AIPlayer**, and call **setAtaxxPlayer** to assign players according to their color and being manual/AI players
  - in **play()**, while loop finds the final winner, and execute move or next command (until receiving command to **exit** the game):
    - if no winner yet, it lets the current player to be the next player **runCommand(getAtaxxPlayer(ataxxBoard.nextMove()).getAtaxxMove())** and runs its move;
    - if the game is over (meeting an end condition):
      - if the winner is not yet announced, then announce it
      - after that, get and run the next command

runCommand() and Command Types

- Commands in **runCommand(String command)** are case-insensitive
  - e.g. one can type in **BOARD** or **board** with the same effect

Command Type	Format in Terminal	Explanation
NEW	new	start a new game with an initial board
AI	ai <red/blue>	set Red/Blue player to be an AI player
MANUAL	manual <red/blue>	set Red/Blue player to be a human player
BOARD	board	print the board **with labels**
BLOCK	block <cr>	set blocks according to the rules in Task Sheet 1
SCORE	score	print the current **number** of red and blue pieces on the board
BOARD_ON	board_on	**print** the board after each move
BOARD_OFF	board_off	**not print** the board after each move
PIECEMOVE	c_0r_0-c_1r_1 (i.e., c2-b3)	**move** the piece of current color on the board (must be legal)
QUIT	quit (or q)	quit the game
ERROR	(any other input)	print "Unknown command" in the terminal

States of a Square, PieceState, and Player Types

- The states of a square (c, r) in a board are enumerated **PieceState** types:
  - **EMPTY**: no piece at location (c, r)
  - **BLOCKED**: there's a block at location (c, r)
  - **RED**: there's a Red piece at the location (c, r)
  - **BLUE**: there's a Blue piece at location (c, r)
- Used in **Player[] ataxxPlayers = new Player[PieceState.values().length]** in **Game.java**, but there are only two types: RED and BLUE
  - ataxxPlayers[0] = RED
  - ataxxPlayers[1] = BLUE
- Each type has **opposite()** method to return its opposite color
  - e.g. RED.opposite() is BLUE

Players

- Represented by color Red/Blue in the game in abstract class **Player.java**
- The implementor classes
  - **Manual** extends **Player**: **Manual(game, color)**
  - **AIPlayer** extends **Player**: **AIPlayer(game, color, seed)**
    - students need to implement AI Player by themselves for ***Task A.2***
    - seed is used to implement the psedorandomness of the AI
    - currently, seed is always incremented by 1 and students may change arbitrarily according to their AI implementation

getAtaxxMove

- Called in runCommand(getAtaxxPlayer(ataxxBoard.nextMove()).**getAtaxxMove()**)
- In **Player.java**, it is declared as an abstract method
  - to be implemented independently in **Manual** and **AIPlayer**

getAtaxxPlayer

- Called in runCommand(**getAtaxxPlayer(ataxxBoard.nextMove())**.getAtaxxMove())
- In **Game.java**, Player **getAtaxxPlayer**(PieceState state)
  - returns a player by indexing the attaxPlayers[] array
  - the index is return by enum ordinal() method
    - i.e. RED.ordinal() is 0 and BLUE.ordinal() is 1
- In **Board.java**, ataxxBoard.nextMove() returns the state/color of the player which is to move next

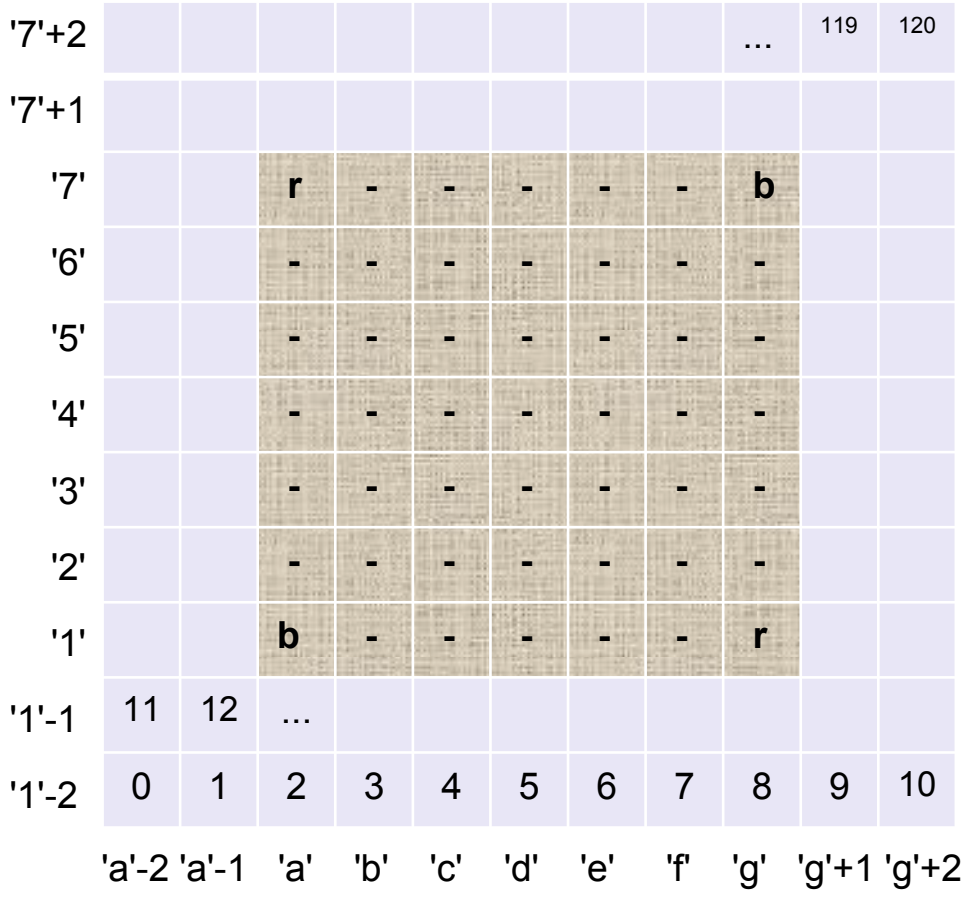
Ataxx Board

- Board in **Board.java** is represented by 11-by-11 array of PieceState
  - with predetermined values as shown in figure on the right
  - why? to avoid special edge cases
- Each piece in board is labeled by:
  - ***char value*** of **column label** c ('a' - 2 < c < 'g' + 2)
  - **row label** r ('1' - 2 < r < '7' + 2)
- Squares *outside* 7-by-7 "real" board (in 'a' - 'g' and '1' - '7') are blocks
  - so that moving to these locations won't cause out-of-bound special edge cases

	'a'-2	'a'-1	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'g'+1	'g'+2
'7'+2	X	X	X	X	X	X	X	X	X	X	X
'7'+1	X	X	X	X	X	X	X	X	X	X	X
'7'	X	X	**r**	-	-	-	-	-	**b**	X	X
'6'	X	X	-	-	-	-	-	-	-	X	X
'5'	X	X	-	-	-	-	-	-	-	X	X
'4'	X	X	-	-	-	-	-	-	-	X	X
'3'	X	X	-	-	-	-	-	-	-	X	X
'2'	X	X	-	-	-	-	-	-	-	X	X
'1'	X	X	**b**	-	-	-	-	-	**r**	X	X
'1'-1	X	X	X	X	X	X	X	X	X	X	X
'1'-2	X	X	X	X	X	X	X	X	X	X	X

Ataxx Board Array

- Instead of a 2-D array, we use a 1-D array
  - the equivalent 1-D indices are given in shown in the figure
  - why? for efficiency in copying the board
  - there are helper methods accessible for you to access a square by column and row labels
    - no need to worry how to translate those into 2-D/1-D indices



Move Objects

- All kinds of Move objects are created by a private constructor in **Move.java**
  - into OVERALL_MOVES array
  - a factory method then returns the requested Move objects
  - so the same Move object used later by your AI Player will only be created ***once*** – for efficiency
- If the requested Move is **not** a legal move
  - the factory method will return **null**

Methods in Board.java

- index() turns column and row labels into index in 1-D array
- getNeighbor() returns index in 1-D array of neighboring square given the distance
- read comments for understanding more helper methods!

- **createMove(String move)**
  - to make a Move object to be used in **Game.java**
    e.g. createMove("c3-d4")
  - it first check whether it is a legal move and whether there's a winner, and check whether it is a pass, to add to list of total moves for displaying
  - find the opposite color

Creating Move

- **createMove(String move)** (continues)
  - if it is a **jump**:
    - set its 'from' index into empty and its 'to' index to next move
    - change the color of surrounding pieces
    - increase the number of consecutive jumping
  - if it is a **clone**:
    - set its 'to' index to next move
    - change the color of surrounding pieces
    - reset the number of consecutive jumping
    - increase the number of corresponding color
  - record the next move and update winner
  - note that it uses **isJump()** and **isClone()** which you will complete
    - explained later in this task sheet

CPT204-2223 Final Project Part A.1

- In the following pages, you will find details about Final Project Part A.1 that you need to complete
  - there are 4 tasks / subparts that you need to complete
  - you will submit your code to Learning Mall autograder during Submission Day (read *Task Sheet 1* and *upcoming Announcements* for more details)
  - **your code will be tested on a new full set of test cases during grading**
  - partial grades will be given if you pass some tasks or pass some test cases

  - **more information and requirements are found in Task Sheet 1 pdf and future LM Announcements**

Part A.1.1 Getting the Number of Colors

- Complete the method **getColorNums(PieceState color)** in **Board.java**
  - it is called in **getScore()** in **Board.java** to print the current score when given the command `score`
  - it takes either RED or BLUE
    - you can ignore other states
  - e.g., getColorNums(RED) → 2

- Find the partial test cases in **ScoreTest.java**

Part A.1.2 Setting a Block

- Complete the method **setBlock(char c, char r)** in **Board.java**
  - it is called when we want to put blocks with the command **`block`**
  - it puts a block at the given position, and its reflected squares symmetrically according to rules in Task Sheet 1
  - we have given the code to throw an error when the location is not legal, such as already occupied by a piece or a block
  - e.g., setBlock('c', '3')

- Hints: Consider using the method setContent and the variable unblockedNum
- Find the partial test cases in **BlockTest.java**

Part A.1.3 Clone or Jump?

- Complete the method **isClone()** and **isJump()** in **Move.java**
  - it is called in the first constructor **Move** in **Move.java**
  - it takes 2 parameters: String location0 and String location1
    - location0 is the origin location such as "c1"
    - location1 is the target location such as "d2"
  - it returns true if and only if the move is a clone/a jump
  - e.g., isClone("c1", "d2") → true
    isJump("c1", "d2") → false

- Find the partial test cases in **MoveTest.java**

Part A.1.4 Getting the Winner

- Complete the method **getWinner()** in **Board.java**
  - it is used to find the winner of the game and is called in **play()** in **Game.java** and **createMove()** in **Board.java**
  - it returns PieceState objects:
    - **null** if the game is not finished
    - **RED** or **BLUE** if the game is finished and there is a winner with that color
    - **EMPTY** if the game is finished but there is no winner since red and blue have the same number of pieces
  - It also stores the result in instance variable **winner**

- Hints: Consider using couldMove, getColorNums, getConsecJumpNums
- Find the partial test cases in **WinnerTest.java**

CPT204-2223 Final Project Part A.2

- You are given in your skeleton code a very simple AI, which moves randomly
  - it generates all possible legal moves, and pick one uniformly at random

- Work with your team members in a team of two/three students, to create your own AI player by modifying the codes in **AIPlayer.java**
  - Specifically, you could create your new methods and call your own methods in **findMove()** method

- **Read Task Sheet 1 pdf and future LM announcements for further information and requirements**

Optional Part: Ataxx GUI

- Complete this part in **GUI.java** for extra points
  - if you have completed the previous parts and have extra time
  - to actually play the game in graphical interface

- Here we provide the way to run the Ataxx game with a GUI interface
  - to enable GUI of Ataxx game you need to first create a jar file
  - and then run that jar file by using command line argument `--display`
  - the **complete steps** can be found in the next slide

Optional Part: Ataxx GUI (continues)

Steps to create a jar file and to run Ataxx with GUI:

1. Click "File->Project Settings -> Artifacts" in the IntelliJ
2. Click "+" button and click to add JAR by "From modules with dependences..."
3. Choose the current module and select the main class;
   and keep other options default and just click "OK"
4. Click "Include in project build" and then click "OK"
5. First click "Build -> Build Project" in the navigation bar, and then click "Build -> Build Artifacts..."
   and choose the action "Build" to create the corresponding .jar file of this program
6. Use "cmd" to open the terminal (in Windows: Click Windows Icon, type cmd, hit enter)
7. Enter the correct path of the document file including your .jar file (here we call it
   "CPT204FinalProjectDemo.jar" for convenience)
8. Under this path, enter "java -jar CPT204FinalProjectDemo.jar --display" to check your GUI
   interface

Good Luck!

- Thank you for your attention and all the best for your final project!