COMP 424: Code description
Pentago-Twist

Course instructor: Jackie Cheung (jcheung@cs.mcgill.ca)
Project TAs: Erfan Seyedsalehi and Samin Yeasar

Overview

An implementation of **Pentago-twist** (follow the link: **https://github.com/SaminYeasar/pentago_twist**)) is provided. Player games requires: a server (the real board) to be launched, and then two clients (agents) to connect to the server via TCP sockets; this allows for clients to be run on separate computers. We also provide a GUI which displays the game in a nice and intuitive way for learning the strategy behind Pentago-twist. Note that games between agents can be played without the GUI to increase speed. All source code is found in the src directory. You must NOT edit any packages, other than stu- dent player, which can be modified as much as you want so long as it (1) compiles, and, (2) that you do not use any external libraries, but you can use default internal Java libraries (like Math and Util). If you edit other packages you must be aware that your edits will not be used during the tournament; e.g., so if you know what you're doing, you may want to edit Autoplay.java to run many test games of your agent against another agent (or perhaps against itself).

src

- autoplay (Autoplays games; can be ignored)
- boardgame (Package for implementing boardgames, logging, GUI, and server TCP protocol, can be ignored for this project)
- student_player (Package containing your agent)
  - StudentPlayer.java (The class you will implement your AI within)
  - MyTools.java (Placeholder for any extra code you may need)
- pentago_twist (The package implementing all game logic)
  - PentagoBoardPanel.java (Implements the GUI, can be ignored)
  - PentagoBoard.java (Used for server logic, can be ignored)
  - PentagoCoord.java (Simple class representing a board coordinate)
  - PentagoMove.java (A move object for Pentago. Relevant functions)
    - —PentagoCoord getMoveCoord() (Returns the coordinate object for this move.)
    - Quadrant getASwap() (Returns the first quadrant selected for swapping.)
    - Quadrant getBSwap() (Returns the second quadrant selected for swapping.)
  - PentagoBoardState.java (Implements all game logic, most important. Note that PentagoBoardState (PBS) manages logic concerning whose turn it is, rules, and the positions of all pieces)
    - Enum Piece (Three types: BLACK, WHITE, EMPTY)
    - Enum Quadrant (Four types: TL (top left), TR (top right),BL (bottom left), BR (bottom right))
    - Object clone() (Clones the PBS, allowing manipulation of the clone without affecting the original. Must be typecasted to a PBS to work on the clone)
    - Move getRandomMove() (Returns a random, legal move.)
    - List getAllLegalMoves() (Returns all legal moves for the current player from the perspective of the PBS)
    - boolean isLegal(PentagoMove m) (Returns whether or not m is a legal move)
    - void processMove(PentagoMove m) (Processes m on the board and updates the PBS as if the move was played. This allows you to project and determine the impact of moves.)
    - Piece getPieceAt(PentagoCoord c) (Returns the Piece at c on the board.)
    - int getOpponent() (Returns opponent of current player on PBS.)
  - PentagoPlayer.java (Abstract class that all players extend)
  - RandomPentagoPlayer.java (A random player, can be used as a baseline)

Coding tips

The **PentagoBoardState** (PBS) you are given to work with in your **ChooseMove** method is a copy of the real PBS, so you can modify it without worrying about changing the real PBS. However, you should clone the PBS and then process moves on the cloned PBS to assess the impact of doing different moves on the PBS; note that a cloned PBS that processed a move can also be cloned again.

Work flow: Eclipse

We recommend that you develop your agent using Eclipse to avoid complication to compile and run the project. You can also use IntelliJ, which also provides very easy way to run java project. The root directory of the project package is a valid Eclipse project. There are two ways to get it running, based on the following clicks:

- From GitHub: File → Import → Git → Projects from Git → Clone URI !→ put in field URI https://github.com/SaminYeasar/pentag twist → keep pressing Next until the end!
- From source: File → Import → General → Existing projects into workspace and then select the root of the project package.

You may need to set the launch configurations manually as follows: File → Import → Run/Debug → Launch configurations and then navigate to the eclipse directory and select those launches.

Note: GitHub is a great tool for managing your project. However, if you choose to use it, please set the project access to private - this is free for students.

Quick Start Pentago-Twist with Eclipse

If using Eclipse and you've set it up as described above, you can easily start a game. Simply click Run in Eclipse, then click on GUI. This will launch the GUI for you. Then, click the launch tab in the GUI, and select Launch Server. Then, the first client you launch will be the first player; for example, select Launch Human Player next. Then, to set the second player, select the next client class; for example, select Launch Client (**pentago_twist.RandomPentagoPlayer**). You can now play against a random player in the GUI to get a feel for the game. Changing the Clients you select will determine who/what plays who/what - so you can run two humans against each other too, and similarly two agents against each other.

Playing Games

Here we provide documentation for the server and client programs. The commands outlined in this section are the commands that are called by the provided **build.xml** file (if using ant) and launch configurations (if using Eclipse). The details provided in this section are especially useful for advanced use of the provided code, such as playing games where the clients are located on different machines than the server. All of these commands assume that you have compiled the code and stored the class files in a directory named bin located inside the root directory of the project package (Eclipse will automatically do these things). In a nutshell, if you want to run from the command line, you will need three terminal shells open and simultaneously running: one to launch the server, one to launch a Client for the first player, and one to launch a Client for the second player.

Launching the Server

To start the server from the root folder of the project package, run the command:

java -cp bin boardgame.Server [-p port] [-ng] [-q] [-t n] [-ft n] [-k] (1)

where:

- (-p) port sets the TCP port to listen on. (default=8123)
- (-ng) suppresses display of the GUI
- (-q) indicates not to dump log to console.
- (-t n) sets the timeout to n milliseconds. (default=2000)
- (-ft n) sets the first move timeout to n milliseconds. (default=30000)
- (-k) launch a new server every time a game ends (used to run multiple games without the GUI)

For example, assuming the current directory is the root directory of the project package, the command:

java -cp bin boardgame.Server -p 8123 -t 300000 (2)

launches a server on port 8123 (the default TCP port) with the GUI displayed and a timeout of 300 seconds. The server waits for two clients to connect. Closing the GUI window will not terminate the server; the server exits once the game is finished. If the -k arg was passed, then a new server starts up and waits for connections as soon as the previous one exits. Log files for each game are automatically written to the logs subdirectory. The log file for a game contains a list of all moves, names of the two players that participated, and other parameters. The server also maintains a file, outcomes.txt, which stores a summary of all game results. At present this consists of the integer game sequence number, the name of each player, the color and name of the winning player, the number of moves, and the name of the log file.

Launching a Client

As stated previously, the server waits for two client players to connect before starting the game. If using the GUI, one can launch clients (which will run on the same machine as the server) from the Launch menu. This starts a regular client running in a background thread, which plays using the selected player class. In order to play a game of Pentago-twist using the GUI, choose Launch human player from the Launch menu (as described in Section 4). Clients can also be launched from the command line. From the root directory of the project package, run the command:

java -cp bin boardgame.Client [playerClass [serverName [serverPort]]] (3)

where:

- playerClass is the player to be run (default=pentago_twist.RandomPentagoPlayer)
- serverName is the server address (default=localhost)
- serverPort is the port number (default=8123)

For example, the command:

java -cp bin boardgame.Client pentago_twist.RandomPentagoPlayer localhost 8123 (4)

launches a client containing the random Pentago-twist player, connecting to a server on the local machine using the default TCP port. The game starts immediately once two clients are connected. The two approaches of launching players from the GUI and launching players from the command line can also be combined. For instance, one can use the GUI to manually play against the random player (or any other agent) by first launching a human player using the GUI, and subsequently launching the random player, either by selecting it in the GUI or running the appropriate command from the command line. The order can also be switched; the player that connects to the server first will move first.

Autoplay

The provided Autoplay script can be used to play a large batch of games between two agents automatically. It launches a Server with the options -k -ng, and then repeatedly launches pairs of agents to play against one another. To use Autoplay, run the following command from the root directory of the project package:

java -cp bin autoplay.Autoplay n_games (5)

where n games is a positive integer number of games to play. The default behaviour of Autoplay is to play the student agent against the random player, with the random player agent going first every second game. You can modify this behaviour by editing **Autoplay.java**.

Implementing a Player

New agents are created by extending the class **pentago_twist.PentagoPlayer**. The skeleton for a new agent is provided by the class **student_player.StudentPlayer**, and you should proceed by directly modifying that file. In implementing your agent, you have two primary responsibilities:

1. Change the constructor of the **StudentPlayer** class so that it calls super with your student number.
2. Change the code in the method **chooseMove** to implement your agent's strategy for choosing moves (the real work).

To launch your agent from the command line, run the command:

java -cp bin boardgame.Client student_player.StudentPlayer (6)

You can also launch your agent by selecting it from the Launch menu in the GUI. The project specification has further details on implementing your player, and, in particular, what you need to submit.