Creative Project 1: Abstract Strategy Games

Specification

Background

Strategy games are games in which players make a sequence of moves according to a set of rules hoping to achieve a particular outcome (e.g. a higher score, a specific game state) to win the game. Strategy games usually give players free choice about which moves to make (within the rules) and have little to no randomness or luck (e.g. rolling of dice, drawing of cards) involved. *Abstract strategy games* are a subset of strategy games usually characterized by

1. Perfect information (i.e. all players know the full game state at all times)
2. Little to no theme or narrative around gameplay

Popular examples of abstract strategy games include: Chess, Checkers, Go, Tic-Tac-Toe, and many others.

In this assignment, you will implement a data structure to represent the game state of an abstract strategy game of your choice.

Learning Objectives

By completing this assignment, students will demonstrate their ability to:

- Define a data structure to represent complex data
- Write a Java class that implements a given interface
- Produce clear and effective documentation to improve comprehension and maintainability of a class
- Write a class that is readable and maintainable, and that conforms to provided guidelines for style, and implementation

Choosing a Game

You may implement any abstract strategy game you choose, subject to the following requirements:

- The game must be playable by exactly two players.
    - It is OK if the game you choose can be played by a different number of players as well, but you will implement the game for exactly two players.

- Players must take turns making moves.
- Both players must make moves following the same basic rules (i.e. gameplay must be symmetrical).
- There must be no hidden information and no randomness in gameplay.
- The game must have a clear end condition.
- When the game has ended, there must be a clearly determined winner (or the game ends in a tie).

Here are some suggestions for games to implement:

- Chomp
- Connect Four
- Paper Tennis

See Wikipedia, Freeze-Dried Games, or Pencil and Paper Games for more inspiration. If you would like to implement a game not listed above, please post in this Ed thread to request approval. Note that you **may not implement the game tic-tac-toe** (see below). Requests for a custom game must be made by **11:59pm on Sunday, April 14** to allow enough time for review and approval before the deadline. (We will monitor the thread and approve on a rolling basis.)

Required Abstract Class

You will implement a class to represent your chosen game. Your class should extend that `AbstractStrategyGame` abstract class, which contains the following abstract methods:

▶ Expand

Your class should also include at least one constructor, which may take any parameters you deem necessary. You may implement any additional private helper methods you like as well.

Implementation Requirements

Your game should be able to be run using the client program in `Client.java`. You should modify line 6 of this file to construct an instance of your class, and you may create any additional variables or data to pass to your constructor as parameters, but you should not have to otherwise modify the file. Implement your class so that this client works **as written**.

A sample implementation of tic-tac-toe has been provided that you may use to see how certain implementation choices might be made. Because of this, **you may not implement tic-tac-toe as your game.**

Grading Guidelines

As described in the Creative Project Grading Rubric, your implementation must meet basic requirements to earn an S, and must have an extension implemented to earn an E. For the three suggested games, the basic and extended requirements are as follows:

Chomp

▼ Expand

Basic requirements:

- The game board is a two-dimensional grid with some squares "chomped" and some not.
- Choosing a square on a player's turn causes all squares to the right and below it to become "chomped".
    - i.e. If the grid's coordinates increase to the right and down, with the top-leftmost square acting as the origin, this action consumes all squares with coordinate values greater than or equal to the chosen square's coordinates.
- The player that consumes the top-leftmost square (the origin) **loses** the game. This action triggers the end condition.
- Your program file should be named `Chomp.java`, and therefore your class should be named `Chomp`

Extended requirements:

- The game board has three layers that must be cleared for a game to be completed.
    - The game board includes how many layers are remaining under each square, with a clear indicator of when there are no layers left on a given square.
    - Choosing a square only chomps squares on the same layer. Chomping a square reveals the square directly below it, if there is one.
        - For example, chomping a square on layer 1 should only remove the other squares with strictly greater coordinate values on layer 1, and should reveal corresponding squares on layer 2. Layer 2's squares should all remain untouched by this action.
    - The player that consumes the top-leftmost square from the **final** layer **loses** the game. This action triggers the end condition.

Connect Four

▼ Expand

Basic requirements:

- The game board is a 7-wide by 6-tall grid.
- Players' tokens are placed in the bottom-most available space in the column selected for a move.

    - Your program should only be prompting the user for column input. The program should **not** take any row input.
- Four of the same player's tokens in a row either horizontally or vertically triggers the end condition.
- Your program file should be named `ConnectFour.java`, and therefore your class should be named `ConnectFour`

Extended requirements (choose at least one):

- At the start of each turn, the active player may choose to either remove one of their tokens from the board or place a token (but not both). The player is able to remove one of their own tokens from the bottom row of any column, shifting all other tokens in that column (if any) down by one row.
- Four of the same player's tokens in a row diagonally triggers the end condition (in addition to the horizontal and vertical end conditions).

Paper Tennis

▼ Expand

Basic requirements:

- The game board consists of positions numbered from -2 to +2, with the game piece initially starting at 0.
    - The game piece cannot be at position 0 except at the beginning of the game.
- Each player gives a bid on their turn by choosing a number between 0 and their current score (inclusive).
- After both players have bid, the game piece moves one space towards the player with the lower bid.
- The value of each player's bid is subtracted from their score.
- The end condition is triggered when either both players have a score of 0, or when the game piece moves beyond -2 or +2. The player on whose side of the board the game piece ended (including if it left the board on that side by moving beyond -2 or +2) **loses**.
- Your program file should be named `PaperTennis.java`, and therefore your class should be named `PaperTennis`

Extended requirements:

- The game consists of three rounds played as above, with the winner of each round earning 1 point for winning, and an additional point for winning via moving the game piece beyond -2 or +2.
    - The player with more points after all three rounds is the overall winner.

If you would like to implement a different game, you will need specify what the basic and extended requirements will be as part of your proposal. Your proposed requirements should be similar in scope and complexity to the requirements for the four suggested games. Post in this Ed thread to propose a different game.

Testing Requirements

- There are no formal testing requirements for this assignment. However, it is your responsibility to make sure the game you implement functions appropriately. We'd highly encourage you to use JUnit to verify this for yourself programmatically.

Assignment Requirements

For this assignment, you should follow the Code Quality guide when writing your code to ensure it is readable and maintainable. In particular, you should focus on the following requirements:

- You should make all of your fields private and you should reduce the number of fields only to those that are necessary for solving the problem.
- Each of your fields should be initialized inside of your constructor(s).
- You should comment your code following the Commenting Guide. You should write comments with basic info (a header comment at the top of your file), a class comment for every class, and a comment for every method other than main.
    - Make sure to avoid including *implementation details* in your comments. In particular, for your object class, a *client* should be able to understand how to use your object effectively by only reading your class and method comments, but your comments should maintain *abstraction* by avoiding implementation details.
- Any additional helper methods created, but not specified in the spec, should be declared ***private***.

Reflection

The following questions will ask you practice **metacognition** to reflect on the topics covered on this assignment and your experience completing it. For each question, focus on your plan and/or process for working through the assignment along with the CS concepts. Think about things like how you organized your working time, what sorts of things tended to go wrong, and how you dealt with those errors or mistakes.

Please answer all questions.

Question 1

REQUIRED
You MUST answer this question to receive credit for the assignment

Which game did you implement?

- ○ Chomp
- ○ Connect Four
- ○ Paper Tennis
- ○ Other

Question 2

Describe how you implemented the state of your chosen game and why you chose that implementation.

No response

Question 3

Describe an alternate implementation you could have chosen for your game. What advantages and disadvantages do you think this alternative has compared to the implementation you chose?

No response

Question 4

Describe how you went about testing your implementation. What specific situations and/or test cases did you consider? Why were those cases important?

No response

Question 5

What skills did you learn and/or practice with working on this assignment?

No response

Question 6

What did you struggle with most on this assignment?

No response

Question 7

What questions do you still have about the concepts and skills you used in this assignment?

No response

Question 8

About how long (in hours) did you spend on this assignment? (Feel free to estimate, but try to be close.)

No response

Question 9

Was any part of the specification or requirements unclear? If so, which part(s), how was it unclear, and how could it have been made more clear?

No response

Question 10

[OPTIONAL] Do you have any other feedback, questions, or comments about this assignment?

(Note that we may not be able to respond to questions here, so please post on the message board if you would like a response!)

No response

□ Final Submission □

□ Final Submission□

Fill out the box below and click "Submit" in the upper-right corner of the window to submit your work.

Question

I attest that the work I am about to submit is my own and was completed according to the course Academic Honesty and Collaboration policy. If I collaborated with any other students or utilized any outside resources, they are allowed and have been properly cited. If I have any concerns about this policy, I will reach out to the course staff to discuss *before* submitting.

(Type "yes" as your response.)

No response