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 extends a given abstract class
- 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
- Use the Visual Studio debugger to go through a program line by line

Debugging

Because this assignment doesn't include formal tests to check your program's behavior, it's important that you are able to take an active role in verifying that your game works as expected. To support you in this process, we've included a short activity on the Debugging slide designed to help you get comfortable using VSCode and its debugger.

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
- Chopsticks
- Connect Four

See Wikipedia, Freeze-Dried Games, or Pencil and Paper Games for more inspiration. If you would like to implement a game other than the three listed above (Chomp, Chopsticks, or Connect Four), 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 20** to allow enough time for review and approval before the deadline. (We will monitor the thread and approve on a rolling basis.)

⚠ The **reflection** portion of this assignment asks you to **find someone to play your game, once finished, and observe their experience**. Note that this will take some time and coordination to arrange, so make sure not to leave this part of the reflection until the last minute!

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

The `AbstractStrategyGame` abstract class also contains the following implemented methods. Your class should work with the provided abstract class and should not modify it.

▶ 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 provided 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**.

We have provided you with **two** sample implementations of tic-tac-toe (`TicTacToe1D.java` and `TicTacToe2D.java`). Both of these samples implement the correct functionality for the game, but differ in their design and underlying structure. We encourage you to look over both of these files to see some examples of how one might implement an abstract strategy game, and consider the tradeoffs that result from the two different approaches. **You may not implement tic-tac-toe as your game.**

As you implement your own abstract strategy game, you will need to consider similar tradeoffs of different approaches and make design decisions as you write your code!

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.

⚠ Take extra care to ensure that the correct files are added and work with the **provided** `Client.java` and `AbstractStrategyGames.java` files

For the three suggested games, the basic and extended requirements are as follows:

Chomp

▶ Expand

Chopsticks

▶ Expand

Connect Four

▶ Expand

If you would like to implement a different game, you will need to 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 three suggested games. Post in this Ed thread to propose a different game.

Testing Requirements

There are no formal JUnit testing requirements for this assignment, but **Question 4 of the Reflection** should be answered in-depth. Additionally, it is your responsibility to make sure the game you implement functions appropriately. We still highly encourage you to use JUnit to verify this for yourself programmatically.

NOTE: If you want to use Ed to run your JUnit tests, make sure to name your testing file `Testing.java`, so Ed can find it!

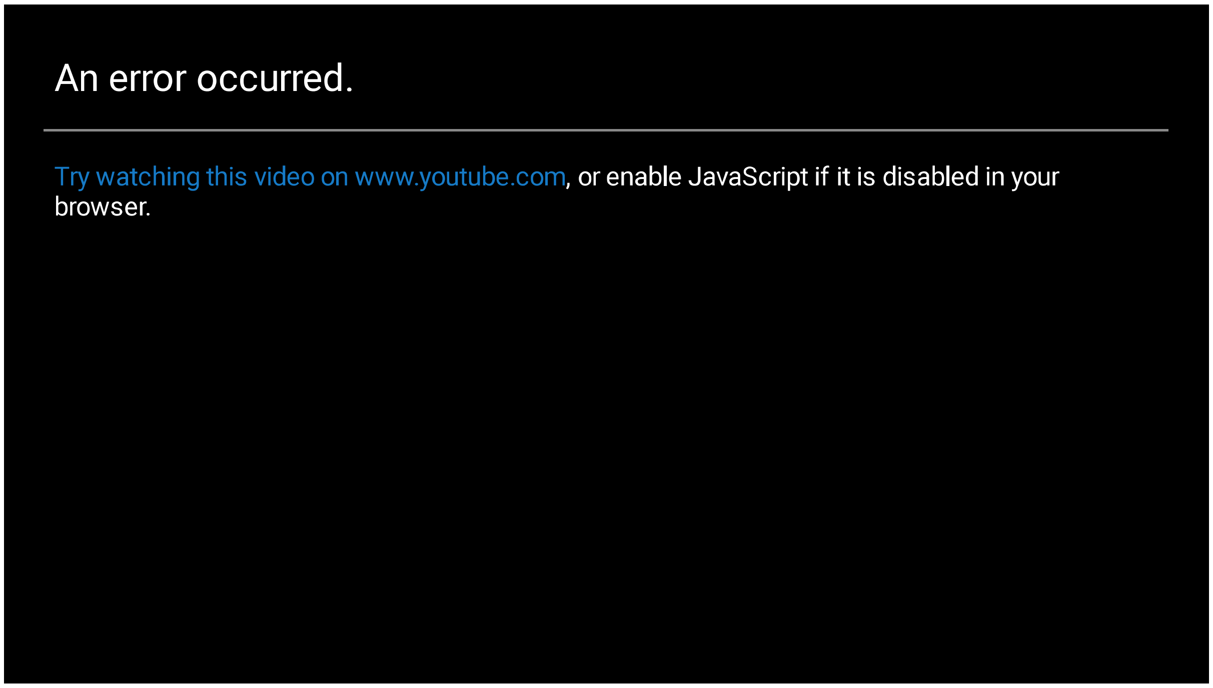
NOTE: The tests provided are only surface-level checks. Once again, it is still **your responsibility to guarantee that you have a working game and updated line 6 of the `Client` with your game constructor to receive credit.** If you are implementing a custom game, you do not need to pass the first and third test (since it only checks that you have a file named `Chomp`, `Chopsticks`, or `ConnectFour`.

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.
- You should avoid hard-coded numbers in your implementation. Instead of hard-coding specific numbers, which we call using magic numbers, it's usually better to use a variable or some property of an object.
- 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***.

Spec Walkthrough



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

NOTE: This walkthrough was created for a previous iteration of this assignment. You might notice that things are slightly different (e.g. the sample tic-tac-toe implementations have changed). A good majority of the assignment is the same which is why we're still using it, but if you have any questions please let us know!

Debugging

Because this assignment doesn't include formal tests to check your program's behavior, it's important that you're able to take an active role in verifying that your game works as expected. To support you in this process, we've included a short activity designed to help you get comfortable using VSCode and its debugger.

Hopefully, you've had a chance to complete the Software Setup for this course. If not, no worries—you'll want to finish that first before moving on with this slide.

Once you have set up the Visual Studio "IDE" (integrated development environment – an application that helps you code), download and open the file below. You will have to determine the correct code needed to defuse the bomb by using the debugger. Time is of the essence, and the fate of the world rests in your hands!

Download starter code:

C1_Debugging.zip

NOTE: You should only need to update `line 30` of the provided file: `defuse("00000");` to the defusal code you determine using the VSCode debugger!

WARNING: We're trying our best to encourage you to use the VSCode debugger here, so you may find that printlns don't actually print anything! This is expected behavior :)

This Debugger Guide (linked on the website) will help guide you through the process!

Abstract Strategy Games

Download starter code:

C1_AbstractStrategyGame.zip

Testing Requirements

There are no formal JUnit testing requirements for this assignment, but **Question 4 of the Reflection** should be answered in-depth. Additionally, it is your responsibility to make sure the game you implement functions appropriately. We still highly encourage you to use JUnit to verify this for yourself programmatically.

NOTE: If you want to use Ed to run your JUnit tests, make sure to name your testing file `Testing.java`, so Ed can find it!

NOTE: The tests provided are only surface-level checks. Once again, it is still **your responsibility to guarantee that you have a working game and updated line 6 of the `Client` with your game constructor to receive credit.** If you are implementing a custom game, you do not need to pass the first and third test (since it only checks that you have a file named `Chomp`, `Chopsticks`, or `ConnectFour`.

Reflection

⚠ The **reflection** portion of this assignment asks you to **find someone to play your game, once finished, and observe their experience**. Note that this will take some time and coordination to arrange, so make sure not to leave this part of the reflection until the last minute!

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
- ○ Chopsticks
- ○ Connect Four
- ○ 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. Include at least one specific detail about this alternate implementation (e.g. what fields would you have? How would you represent a move being made?) .

Then, list at least one advantage and one disadvantage you think this alternative has compared to the implementation you chose.

No response

Question 4

Write a short test plan for your game. Your plan should include a list of important cases to check, the inputs necessary to test those cases, and why those cases are important to test. At minimum, your plan must cover the following cases:

1. Player 1 winning
2. Player 2 winning
3. At least one additional edge case of your choice

In order to earn credit, your format should generally match this example answer for `TicTacToe2D`:

▶ Expand

No response

Question 5

The following 3 questions will be concerned with HCI (Human-computer Interaction) a subfield of CS related to application design and guiding principles for good design.

You're going to conduct a "behavioral mapping" study with your current implementation. (This is just a fancy way of saying "observation of someone playing your game"). You'll have to follow the following steps:

1. First, get a user to play your game - peer, friend, family, roommate, TA at the IPL, etc.
2. Tell them that you'd like them to test out a game that you made, and their goal is to win.
3. Provide no further instruction than that (you want their unbiased reactions and attempt to use your system without explanation).
4. Play the game with them (~5 mins in length).

While playing, take note of the following questions:

- Were there any instances in which the user was confused as to how to play the game or how the game worked?
- Were there any instances in which the user made a mistake / was confused about how to enter input for the game?

Report your findings (indirect feedback) here along with how the user of your study relates to you (peer, friend, family, roommate, TA at the IPL, etc.)

No response

Question 6

Now, ask the user for direct feedback regarding your implementation of the game. Some questions include:

- Did you understand how to play the game from the provided instructions alone?
- What was most confusing about how to record a move?
- In a perfect world, how do you think the interface should change to make this game easier to play?

Report your findings.

No response

Question 7

Based on your responses to the previous two questions, if you were to make one change to your implementation centered around **usability**, what would it be? (This answer doesn't have to be feasible with your current knowledge of Java / CS - can be anything!)

No response

Question 8

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

No response

Question 9

What did you struggle with most on this assignment?

No response

Question 10

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

No response

Question 11

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

No response

Question 12

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 13

[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