Pong

Team Name:

Manager:

Recorder:

Presenter:

Analyst:

This is a Process Oriented Guided Inquiry Learning (POGIL) activity. You and your team will examine a working program. A series of questions will guide you through a cycle of exploration, concept invention, and application. There is strong evidence that this is more effective (and less boring) than a traditional lecture.

By the time you are done with this activity, you and your team should be able to:

- use computational objects to represent physical objects, encapsulating behavior in multiple classes.
- separate the model from the user interface in a graphic game.
- read simple UML class diagrams.
- communicate more effectively.

Your team's recorder is responsible for writing your team's answers to the numbered questions on this form.

Playing the game

Pong was one of the earliest coin-operated arcade games, produced by Atari in 1972. Our version uses a slightly simpler physics model. This is a two-player game, so your manager should divide your team into pairs to play a couple of times. Run Pong.java to play the game.

Note: On Mac OS X, holding down certain letter keys (including ‘a’) pops up a dialog from which accented versions of the letter can be selected. As a result, Pong sometimes hangs if ‘a’ has been held down. If you encounter this problem, quit Pong, enter the line below in a terminal window to disable this feature, and re-run Pong.

```
defaults write -g ApplePressAndHoldEnabled -bool false
```

To reenable accent marks, enter the same line again but with `false` replaced by `true`.

1. Is everyone done playing and ready to pay attention to the team?

You may need to go back and play the game again to answer some of the questions to come, but you should do so *deliberately*, because your team’s manager assigned one or more people to find something out, not merely because you got bored with the conversation or thought you could answer a question better on your own.

2. What does the Z key do?

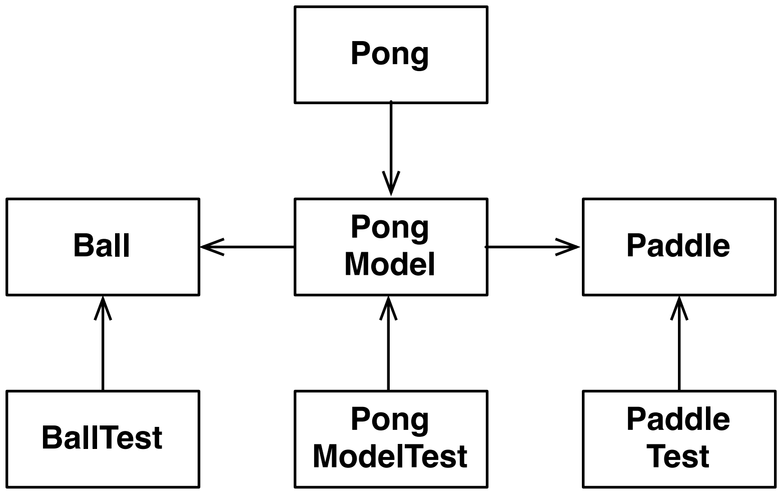
3. How many points are needed to win the game?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team’s presenter should be prepared to present any of your team’s previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

4. Would the game be improved by speeding up or slowing down the ball or the paddles?

Overview

The diagram below, called a *Unified Modeling Language (UML) class diagram*, shows the relationships between the classes involved in this program. Arrows indicate “has-a” relationships. For example, a PongModel has-a Ball.



5. Which classes have-a Paddle?

6. Look at the instance variables within each class. What does a has-a relationship mean in terms of code?

7. Which classes refer to the StdDraw library?

8. If you wanted to modify this game to work with a different graphics library (e.g., for fancier graphics), which classes would you have to modify or replace?

9. Which classes are JUnit test classes?

10. Is your team's recorder writing in complete sentences where appropriate?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

11. Which non-test classes do not have corresponding test classes? Why would it be difficult to write automated JUnit tests for these classes?

Pong

Examine Pong.java.

12. What does the `main` method do?

13. What does the constructor do?

14. Are your team members making eye contact when talking to each other?

15. When does the loop in the `run` method end?

16. How can you speed up or slow down the game by modifying only this class?

17. Is your team's presenter speaking in complete sentences where appropriate?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

18. If you were writing this program, would you write the Pong class before or after the others? Why?

PongModel

Examine PongModel.java.

19. What does the `advance` method tell the paddles to do?

20. What does the `advance` method tell the ball to do?

21. Where is the `advance` method called? (Do you know how to ask your development environment for this information?)

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

22. What is the point of separating the Pong and PongModel classes?

Paddle

Examine Paddle.java.

23. Without testing it, how do you *predict* the game's behavior would change if you removed the lines

    ```
    if (y + HALF_HEIGHT > 1.0) {
         y = 1.0 - HALF_HEIGHT;
    }
    if (y < HALF_HEIGHT) {
         y = HALF_HEIGHT;
    }
    ```

    from the `move` method?

24. Now comment out those lines and run the game. Were you right?

25. Are your team members listening to each other carefully?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

26. In the `move` method, the amount to move the paddle is specified by an argument `dy`. Where is this value specified? Would it be clearer to have separate `moveUp` and `moveDown` methods? Explain.

Ball

Examine Ball.java.

27. Which instance variables are accessible via getters?

28. Which instance variables are accessible via setters?

29. When trying to understand written or spoken explanations, are your team members paraphrasing statements they've just read or heard to make sure they understand?

30. Why does the `move` method need an array of Paddles as an argument?

31. Why does Ball need four instance variables when Paddle only needed one?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

32. In general, is it better for a class to minimize or maximize the amount of information it shares with other classes? Why?

This work was supported by the Google Education and Relations Fund's CS Engagement Small Grant Program grant #TFR15-00411.

The principal investigator, Peter Drake, wishes to thank the following for their useful comments: the members of the CS-POGIL project, specifically Clif Kussmaul and Helen Hu; Maggie Dreyer; and various anonymous reviewers.