

**Project 1.3.9 Tools for Collaboration**

Introduction

|  |  |
| --- | --- |
| Version control is a powerful tool for developing any project or product. If you are collaborating with one or more other people, it is even more important. Have you ever found it difficult to keep papers well organized? What happens when 20 people all work on the same document? | Github.com/about  The GitHub Team |

Materials

* Computer with Enthought Canopy distribution of Python® programming language, Internet browser, and GitHub client software
* GitHub individual account and membership in organizational account's team

Procedure

1. When creating digital products with other people, keeping track of who revised which files from which other files is essential. **Version control** is a system for keeping track of who revised what. A version control system makes it easier to bring the revisions made by a group of people back together into one cohesive unit.

GitHub is a platform for handling version control. It has become the de facto platform for collaboration for most projects involving a large group of software developers collaborating on a single project.

In this project your entire class will collaborate to create one program, all starting from one copy of a program that already exists. You will work in teams of two people, and each team will work with its own copy of the original. Each team will contribute one algorithm to improve the original program. Then all the teams’ improvements will be merged together into a single version of the program.

The classroom collaboration for this project is a lot like having multiple teams each working on a different feature of a software package. What is it about this project that motivates us to use a version control system like GitHub?

1. The existing code simulates a round-robin tournament among 15 players. The game is called the **Iterative Prisoner’s Dilemma** (IPD), a fundamental problem in **game theory**. Game theory is an approach to studying the social sciences, using computation to understand and predict people’s behavior. Game theory is used, for example, to understand and predict economic phenomena like stock market fluctuations and political phenomena like revolutions. Nations use game theorists to advise national leaders and negotiators in order to help them understand the group’s dynamics and predict results for various actions they are considering taking.

Game theorists use algorithms to describe people’s decisions. Consider your decision to buy lunch each day. If the cafeteria director decides to raise the lunch price above a certain point, you will opt not to buy school lunch, right? At what price? Are there other factors? Express your algorithm for the decision to buy school lunch in terms of price and other factors. You could use any language you care to, but do not use the computer. Your expression must be conveyed in writing or through a graphical representation.

1. The Iterative Prisoner’s Dilemma is a series of rounds of the **Prisoner’s Dilemma.** The Prisoner’s Dilemma was one of the early algorithmic problems in game theory, first posed in 1950. In the Prisoner’s Dilemma, you and a partner have committed a crime together, and you are caught without evidence. The police question you and your partner separately.

* If both of you collude together and refuse to talk the police, you both will go free.
* If you both confess your crime and betray each other, the liability of the crime will be split between you. You both will receive the standard punishment.
* If you stonewall the police, hoping to collude with your partner but your partner betrays you, you will receive an unusually severe punishment while they go free and get to keep the stolen goods.
* If you betray your partner while they attempt to collude with you, you will receive the cash and freedom.

Your result:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Your Action | |
|  |  | Collude | Betray |
| Partner’s Action | Collude | **R**eleased  *0 pts* | Set Free  + **T**reat  *+100 pts* |
| Betray | **S**evere Punishment  *-500 pts* | **P**unishment  *-250 pts* |

What is the best outcome possible for you in one round of this game? When does it occur?

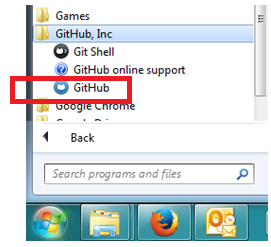
1. In the Iterative Prisoner’s Dilemma, many rounds are played, one after another, with the same partner in crime. Betraying in one round will sometimes cause the partner to lose trust and be less likely to collude with you in future rounds. Once partners start betraying each other, round after round they end up turning each other in and doing poorly. Considered together, your and your partner’s outcomes are better if you collude round after round.

The Prisoner’s Dilemma lets social scientists study the conditions under which people will act either selfishly or in the best collective interests of the group. Think of one situation in which a person in your life could act selfishly or in the interest of a common group, and explain how this relates to the Prisoner’s Dilemma.

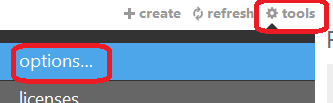
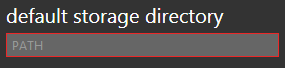
1. Our recent capacity to generate, collect, store, and analyze huge amounts of data quickly has caused dramatic changes in all career fields. Simulation was an unimportant tool in social sciences in 1960. Explain why the rise in computational power has changed the career fields in social sciences such that simulation is now a fundamental and common tool for many social scientists.

**Part I. Setup**

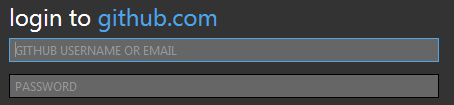
1. Form teams of two as directed by your teacher. Meet or greet each other to practice professional skills. Set team norms.
2. You will now acquire two copies of a program to simulate the IPD. One copy will be for your own experimentation. Another copy will be linked to all the other student pairs’ linked copies. First, set up the connection with GitHub as follows.
3. Launch GitHub for Windows.



1. Select **tools** > **options** and set the default storage directory to a folder for your GitHub work, as directed by your teacher.

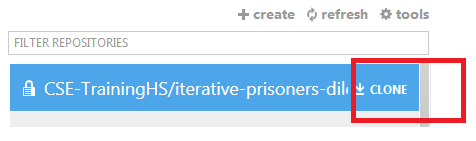
1. Select **login** and enter one teammate’s GitHub credentials. Whenever someone else is entering a password, protocol dictates that you turn your head away. Insist upon privacy.

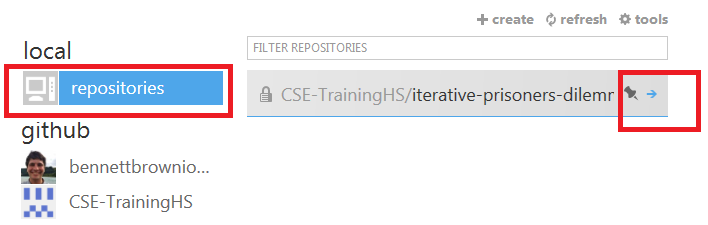
1. You should see a list containing three groups of repositories: the repositories on your local machine, the repositories in your individual account on GitHub, and the repositories in your high school organizational account on GitHub. Select the repositories on GitHub in the school’s organizational account as shown below.



1. Clone the iterative-prisoners-dilemma repository. This repository was in the cloud, and cloning it will create a local repository on your machine.



1. Open the local clone repository by selecting the local repositories and clicking on the blue arrow for this particular repository.



You should see prisoners\_dilemma.py. This file is located in the repository folder in the default storage directory for GitHub for Windows that you established in Step 7. Don’t edit that file until instructed to do so. Changes in that file will later be pushed up to the GitHub cloud repository and merged with other teams’ changes, and you should only modify certain parts of the code.

To give you free rein, copy the *Python* file to the directory you have been using for *Python* files as directed by your teacher.

**Part II. Explore the Simulation**

1. Counting the original, you now have write **permissions** to three copies of the IPD simulation listed below. File permissions give a user the ability to **read, write, or execute** the data in a file.

* Copy 1, in the cloud on GitHub, shared with the class
* Copy 2, on your local machine, within your GitHub for Windows default storage directory
* Copy 3, on your local machine, untracked by GitHub

Write out the directory path for prisoners\_dilemma.py

Copy 1: Don’t worry about this one. It’s not part of this course.

Copy 2:

Copy 3:

1. Open the simulation program that is untracked by GitHub as follows. Launch Canopy. Open a Canopy editor window. Open Copy 3 of prisonersDilemma.py in the *Python* code editor.

Execute the code, which defines several functions.

At the iPython prompt, execute the following command.

In []: play\_tournament(4)

Data from the simulation are printed to the screen and stored in a data file. On screen you should see the following information.

Each column shows the score earned per round against each other player.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | P0 | P1 | P2 | P3 |
| P1 | 0 | 100 | 0 | 0 |
| P2 | -500 | 0 | -376 | -376 |
| P3 | 0 | -74 | 0 | 0 |
| P4 | 0 | -74 | 0 | 0 |
| Total | -500 | -47 | -375 | -375 |

|  |  |  |
| --- | --- | --- |
|  | Mean score per round | Team strategy names |
| Player 0 | -125 pts | loyal |
| Player 1 | -12 pts | backstabber |
| Player 2 | -94 pts | loyal vengeful |
| Player 3 | -94 pts | loyal vengeful |

Looking at the second table, which player finished with the highest average score? Note that all the scores are negative.

The simulation program represents each decision by one of the two characters 'c' or 'b' to collude or betray. Before making the decision each round, each player (or algorithm) can consider what has happened in previous rounds. The algorithms have the previous rounds’ information in the form of a string. For example 'ccb' indicates that the player colluded in the first two rounds and betrayed in the most recent round. Each player can consider two strings: one for their own history and one for their crime partner’s history. Consider the following two histories:

Your history 'ccccc'

Your crime partner’s history 'ccccb'

1. You can tell that five rounds have been played already. What is each player’s score?
2. The four players simulated in Step 14 used three different algorithms, as labeled in the second table in Step 14: loyal, backstabber, loyal vengeful. The code for playing a tournament calls get\_action() with getting\_team\_name=True in line 602. But you don't need to worry about that code; just focus on the strategies. Three parts of *Python* code implementing the three algorithmic strategies are shown below, in the else-blocks starting on lines 115, 131, and 149. Only the third algorithm considers what has occurred in previous rounds. It uses history and opponent\_history, which are strings as described in Step 14.

|  |  |
| --- | --- |
| 111  112  113  114  115  116 | *# This example player always colludes*  **if** player == 0:  **if** getting\_team\_name:  **return** 'loyal'  **else**:  **return** 'c' |

|  |  |
| --- | --- |
| 127  128  129  130  131  132 | *# This example player always betrays*  **if** player == 1:  **if** getting\_team\_name:  **return** 'backstabber'  **else**:  **return** 'b' |

|  |  |
| --- | --- |
| 144  145  146  147  148  149  150  151  152  153  154  155 | *#This example player is silent at first and then*  *#only betrays if they were a sucker last round.*  **if** player == 2:  **if** getting\_team\_name:  **return** 'loyal vengeful'  **else**:  **if** len(opponent\_history)==0: *#First round: collude*  **return** 'c'  **elif** history[-1]=='c' **and** opponent\_history[-1]=='b':  **return** 'b' *# betray if severely punished last time*  **else**:  **return** 'c' *# otherwise collude* |

A ten-round match between player 0 and player 1 results as follows:

Player 0: 'cccccccccc' Score = -5000

Player 1: 'bbbbbbbbbb' Score = +1000

Analyze the code for these algorithms and record the results you expect from a ten-round match of the IPD between player 1 and player 2, following the previous example matching player 0 to player 1.

1. Now open the file tournament.txt. The simulation program will have stored that file in the same directory as the simulation program itself. The file probably opens by default in the application Notepad, but any text editor will do. At the top of the file is the record for team 1 vs. team 0, showing their final scores per round, the names for thier strategies, and a record of their decisions in b and c strings. When the simulation runs, it runs 200 to 300 rounds of the dilemma between each pair of strategies.

team 1 vs. team 0

100 vs. -500

backstabber vs. loyal

bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

Scroll down until you find the data stored for team 2 vs. team 1. Is it different than your prediction? If so, how?

1. At the iPython prompt, execute the following command.

In []: play\_tournament(15)

* 1. How many distinct strategies are being simulated now?
  2. Which strategy earned the highest average score?

1. Notice that a strategy’s success is dependent upon the strategies it is collaborating with and competing against. Discuss this with one or more other teams in the class and describe why this occurs.

**Part III: Develop Your Own Algorithm**

1. With your partner brainstorm algorithms for making the collude vs. betray decision. Recall the ground rules for brainstorming:

* Quantity over quality
* No evaluation of ideas, but piling on is okay
* Record ideas with tag lines of one or a few words

1. Review your team norms for the driver and navigator roles in pair programming.
2. Pick three of your ideas to implement in the *Python* code in Copy 3 of the simulation program. Complete the following for each idea:
   1. Write pseudocode and strategize how you will implement the algorithm in *Python*.
   2. Implement your algorithm in Python in the section of the code reserved for one of the players 5, 6, and 7. You will be replacing the code in the else-block using your algorithm.
   3. In that section of code, also give a name to your algorithm’s strategy.
   4. Test your algorithm by calling the play\_tournament(n) function. If it appears to be working, examine the tournament.txt file to see how your algorithm performed against each other strategy.
   5. Record the performance of your algorithm against each of the other strategies:

|  |  |  |  |
| --- | --- | --- | --- |
| You  They Scored  Scored | Your algorithm #1 | Your algorithm #2 | Your algorithm #3 |
| Loyal |  |  |  |
| Backstabber |  |  |  |
| Loyal Vengeful (LV) |  |  |  |
| Betray every 3rd |  |  |  |
| LV with permanent 2nd impression |  |  |  |
| LV occasionally greedy |  |  |  |

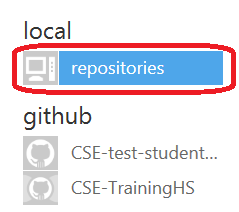
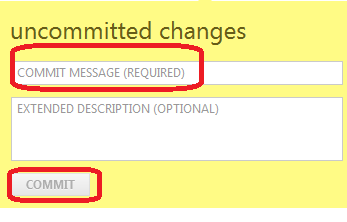
1. Decide on one of your algorithms to contribute to the whole-class tournament. Explain your reasoning for choosing that algorithm.

**Part IV. Contribute Your Strategy to the Class Collaboration**

1. Open Copy 2 of the program. This is the local copy linked to the GitHub repository. You will find it in the default local directory for GitHub, as chosen in Step 7b.

Be careful not to place your cursor anywhere in the file other than the section allocated for your pair’s work. This precaution is necessary because if two teams both change the same line in the file, one of the teams will be unable to push their changes into the shared repository in the cloud without using the Git shell.

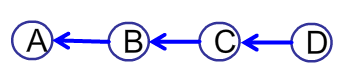
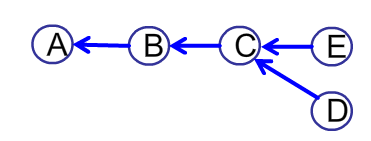
1. In the section allocated to your pair, edit the code so that it uses your algorithm. Save the file.
2. Launch GitHub for Windows and view the local repository, as shown below on the left. Commit the changes and use a commit message like “changed player 5 strategy”.

1. Select **sync** to push your changes to the cloud repository. After you have clicked the “sync” button, the repository may take a couple of minutes to upload.



1. Git, the software on which GitHub is built, creates a tree of revisions building on earlier commits. The commits can be in a sequence, one after another, as shown on the left, or they can be along separate branches of development, as shown on the right.

In the graph of commits on the right, commits D and E are independent revisions of commit C. The arrows go backward in time, showing which commits are being built upon. How many “current” versions do you think there are in the repository graphed on the right?

1. Once everyone has contributed their strategy, synchronize again with the cloud repository. Execute the function and examine the tournament.txt file. Who won? How did your strategy do overall?
2. Against which strategies did your strategy perform well? Which strategies defeated you? Explain why the winning strategy or strategies performed well.

29. Write a pseudocode strategy to beat one team that you fared poorly against in the competition. Explain why you think this new strategy would or would not do well overall.

Conclusion

1. What is version control?
2. Why is version control so helpful when collaborating with people?
3. How has the Internet changed the way in which people write software – even software not meant for use related to the Internet? Advocates of the Scrum process still recommend that a team of developers be located in the same place to collaborate. Why do you think face-to-face collaboration is still considered superior?