

CS 1027B, 650 – Assignment 1: Bridge Builder

[Home \(https://content.techsystems.work/\)](https://content.techsystems.work/) / CS 1027B, 650 – Assignment 1: Bridge Builder

Due Monday Jul 3, 23:55pm ET.

- Changelog:
 - No changes yet.

Table of Contents



Introduction Video

Introduction to Challenge

Here are the rules and gameplay details for implementing the game BridgeBuilder:

Scoring Rules:

Behaviour of the Engineer:

Hard Mode Details:

In Easy Mode:

Classes to Implement

GameBoard.java

Engineer.java

Player.java

Download Provided Main.java

View the Game in Action!!

Video

Screenshot

Test Cases: Some tests you should try to ensure your code does run properly:

Getting Started

****Download and Install Eclipse****

****Setting Up the Project****

****Setting Up the Package****

****Setting Up the Main Class****

****Copy the Provided Main Method****

****Setting Up the Other Classes****

****Run the Program****

****Get some help****

Marking Notes

Functional Specifications

Non-Functional Specifications

Files to submit

Grading Criteria

Submission Information

Introduction Video

A1 Intro CS1027, S23



Video: Bridge Builder Game Trailer

Introduction to Challenge

Welcome to BridgeBuilder, a thrilling command-line game where strategy and cunning meet civil engineering. In this game, you'll take on the role of a brilliant civil engineer tasked with constructing a crucial bridge across a vast lake, where a swamp in the bottom right corner impedes construction. However, the challenge doesn't stop there; a rival group has hired their own automated engineer to hinder your construction by erecting obstacles on your path.



Set on a grid representing the lake, your goal is to build a continuous bridge from the left to the right, or from bottom to top. For those seeking an additional challenge and higher scores, diagonal bridges are also an option, but be warned – not all diagonals are treated equally. In this game, the size of the lake – and thus, the length of your bridge – can vary, giving you the freedom to choose the scope of your work and potential for bonus points.

On each turn, a visual representation of the lake will be shown. Your moves are marked with a “+”, while the automated engineer's obstacles are denoted by “0”. Empty spots are shown as “.”, waiting to be claimed for your bridge or blocked by the opposing engineer. For added excitement, the game

offers two difficulty levels. In easy mode, the engineer will randomly choose their points of obstruction. On hard mode, they will cunningly block the point to the right or top of your most recent move.

Points are awarded based on the bridge's path: 5 points for left to right, 7 points for bottom to top, and a staggering 10 points for a certain diagonal path. The catch? Diagonal paths are only awarded points if they go from top left to bottom right. Attempting to go from bottom left to top right is not considered a victory, and the gameplay continues. For games exceeding a 3x3 grid, you'll receive bonus points based on the grid's (lake's) size.

Note the following diagonal bridge would be considered to be built on a swamp and not receive any points or win:

1		A	B	C
2	0	+	0	+
3	1	0	+	0
4	2	+	.	0

But a game board like this would be a win:

1		A	B	C
2	0	+	0	0
3	1	0	+	0
4	2	+	.	+

To create this adventure, you'll be crafting three core classes: **Player**, **GameBoard**, and **Engineer**. The Player class will represent the Player, containing methods for marking moves and calculating scores. The GameBoard class represents the construction zone, encapsulating methods for displaying the board and checking the game's status. Lastly, the Engineer class represents your automated rival, complete with methods for marking their moves and blocking your path, based on the selected difficulty level.

As we start this exciting journey, remember that every decision counts. Each move you make could turn the tide of the game. So put on your hard hat and let's outsmart the rival engineer in this strategic civil engineering challenge. Happy coding and good luck!

Here are the rules and gameplay details for implementing the game BridgeBuilder:



1. The game takes place on a grid that represents a lake, with the goal to construct a bridge either from the left to right or from the bottom to the top of the grid. A diagonal path is also possible from top left to bottom right for a higher score.
2. The size of the lake (grid) is defined by the player at the beginning of the game, representing the scope of work needed to build the bridge. The minimum size is a 3×3 grid.
3. The game is played over several rounds, with each round representing a new game on a fresh grid.
4. The player's first construction point must be on either the left or bottom of the construction zone.
5. On each turn, the player chooses a location on the grid to place a segment of the bridge, marked with a "+" by entering the coordinates separated by a space e.g, "0 A" for the top left

corner.

6. The rival automated engineer will place obstacles on the grid, marked with a “0”, to block the player’s progress.
7. Empty positions on the grid are represented with “.”.
8. The game offers two difficulty levels:
 - In easy mode, the engineer will randomly choose a position on the grid for their obstruction.
 - In hard mode, the engineer will strategically choose a position one point to the right of the player’s most recent position. If that position is full, the engineer will keep moving to the right or top until an empty spot is found.
9. Base points are awarded based on the direction of the completed bridge:
 - 5 points for a bridge from left to right.
 - 7 points for a bridge from bottom to top.
 - 10 points for a diagonal bridge from top left to bottom right.
10. If the game is played on a grid larger than 3x3, the player will receive bonus points equivalent to the grid size minus three upon winning the round (e.g. 1 bonus point for each size bigger than 3, so for a 5x5 grid, they player will get 2 extra points)
11. If the game ends in a tie, where all squares are filled but no bridge is completed, each player is awarded 1 point.
 - To keep the game logic more simple, you do not need to calculate when it becomes impossible for you or the rival engineer to win and end the round early. Just continue to accept inputs until a win condition or the board is full
12. If the rival engineer manages to complete their own bridge, the player loses and is awarded 0 points for that round.
13. The total score from all rounds is summed up at the end to declare the final score.
14. The game is coded using object-oriented principles, and will include the following classes:
 - **Player**: This class represents the player and includes methods for marking a move and calculating the score.
 - **GameBoard**: This class represents the construction zone and includes methods for displaying the board and checking the game status.
 - **Engineer**: This class represents the rival engineer and includes methods for marking its move based on the selected difficulty level and blocking the player’s path

Scoring Rules:



1. If the player successfully constructs a bridge from left to right, they score 5 points.
2. If the player constructs a bridge from bottom to top, they score 7 points.
3. If the player constructs a bridge diagonally from the top left to bottom right, they score 10 points.
4. If the player manages to win on a grid larger than a 3x3, they ALSO receive bonus points equivalent to the grid size minus three.
5. In the event of a tie, where all squares are filled but no bridge is completed, the player is awarded 1 point only.
6. If the rival engineer manages to complete their own bridge, the player loses and is awarded 0 points for that round.

Behaviour of the Engineer:



The engineer's behaviour changes based on the difficulty level set for the game:

1. **Easy Mode:**

- The engineer makes random moves on the grid, placing obstructions in random locations.

2. **Hard Mode:**

- The engineer places obstructions in a strategic manner. They will select the next empty position one to the right of the player's most recent position. If that position is occupied, the engineer will keep moving to the right or next row, until they find an empty spot.

Hard Mode Details:

In Hard Mode, the Engineer applies a more strategic method to place its token on the game board, aiming to obstruct the Player's progress effectively. The behavior is defined by the `makeMove()` method in the Engineer class, with the following rules:

1. Rightward Search:

- The Engineer begins by searching for an empty space on the same row where the Player made their last move, starting from the column to the right of the Player's last position. This search continues towards the right end of the row. If it finds an empty space, it places its token ('0') there and ends its turn.

2. Top-Down Search

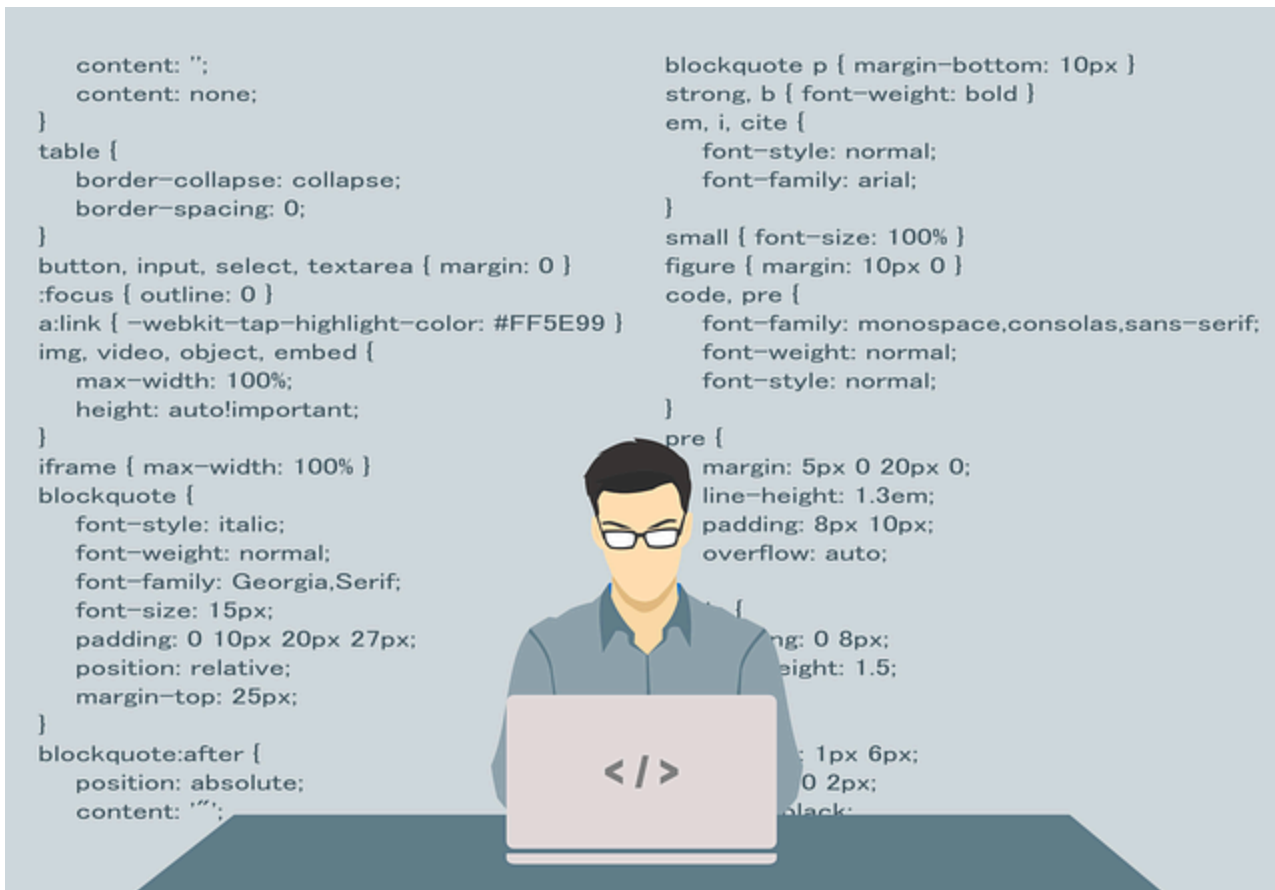
- If the Engineer doesn't find any empty spaces in the row where the Player made their last move (i.e., the row is full), it starts a top-down search. Beginning from the topmost row of the game board, it searches for an empty space in the same column as the Player's last move. The search proceeds downwards until it finds an empty space, where it places its token and ends its turn.

In Easy Mode:

In contrast to Hard Mode, the Engineer in Easy Mode places its token randomly on the game board. It generates random row and column indices until it finds an empty space, where it places its token. This behavior is less strategic, making it easier for the Player to construct their bridge.

Classes to Implement

For this assignment, you must implement three Java classes: GameBoard, Engineer, and Player. Follow the guidelines for each one below.



In all these classes, you can implement more private (helper) methods, if you want to, but you may not implement more public methods. You may not add instance variables other than the ones specified below nor change the variable types or accessibility (i.e., making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

GameBoard.java

This class represents the game board where the game will be played.

The class must have the following private variables:

```
1 | board (char[][])  
2 | size (int)
```

The class must have the following public methods:

```

1 public GameBoard(int size) [constructor]
2 // -- Initialize the board to a 2D char array with '.' representi
3 public void placeToken(int row, int col, char token)
4 // -- Place the given token at the specified row and column on th
5 public boolean isPositionEmpty(int row, int col)
6 // -- Check whether the specified position on the board is empty (c
7 public int getSize()
8 // -- Return the size of the game board.
9 public void displayBoard()
10 // -- Print the game board on the console, including row and col
11 public int checkForWinDirection(Player player)
12 // -- Check whether the player has won the game in any direction
13 public boolean checkForTie()
14 // -- Check whether the game board is full, indicating a tie.

```

Engineer.java

This class represents the computer-controlled player.

The class must have the following private variables:

```

1 token (char)
2 hardMode (boolean)

```

The class must have the following public methods:

```

1 public Engineer(boolean hardMode) [constructor]
2 // -- Initialize the token and set the hardMode.
3 public void makeMove(GameBoard board, int playerLastRow, int playerLc
4 // -- Based on the difficulty level (hardMode), make a move on the
5 public char getToken()
6 // -- Return the token of the engineer.

```

Player.java

This class represents the user-controlled player.

The class must have the following private variables:

```

1 token (char)
2 score (int)

```

The class must have the following public methods:

```

1 public Player() [constructor]
2 //    -- Initialize the token and score.
3 public void makeMove(GameBoard board, int row, int col)
4 //    -- Place the player's token on the game board at the specified
5 public char getToken()
6 //    -- Return the player's token.
7 public int getScore()
8 //    -- Return the player's current score.
9 public void addScore(int increment)
10 //    -- Increase the player's score by the specified increment.

```

Remember to ensure your classes interoperate correctly with the provided Main.java file. Pay close attention to how each class is expected to interact with others. If you need any additional clarification, please feel free to visit TA Consultation Hours.

Download Provided
Main.java
([https://1drv.ms/u/s!As0vrM-
NsbcQkq43ozKME4vx6c7QUQ?e=1INuYs](https://1drv.ms/u/s!As0vrM-NsbcQkq43ozKME4vx6c7QUQ?e=1INuYs))

Here is the Main.java file that you need (<https://1drv.ms/u/s!As0vrM-NsbcQkq43ozKME4vx6c7QUQ?e=1INuYs>). Please do not edit this file, but go ahead and download it.

View the Game in Action!!

If any of the scoring rules or gameplay is unclear, watch the gameplay itself! This is the gold standard to compare your solution with.

This 5 minutes shows the expected behaviour of your program, including the wording for the text outputs and the “engineer’s” behaviour. It does not show all cases, but shows most of the logic you should have.

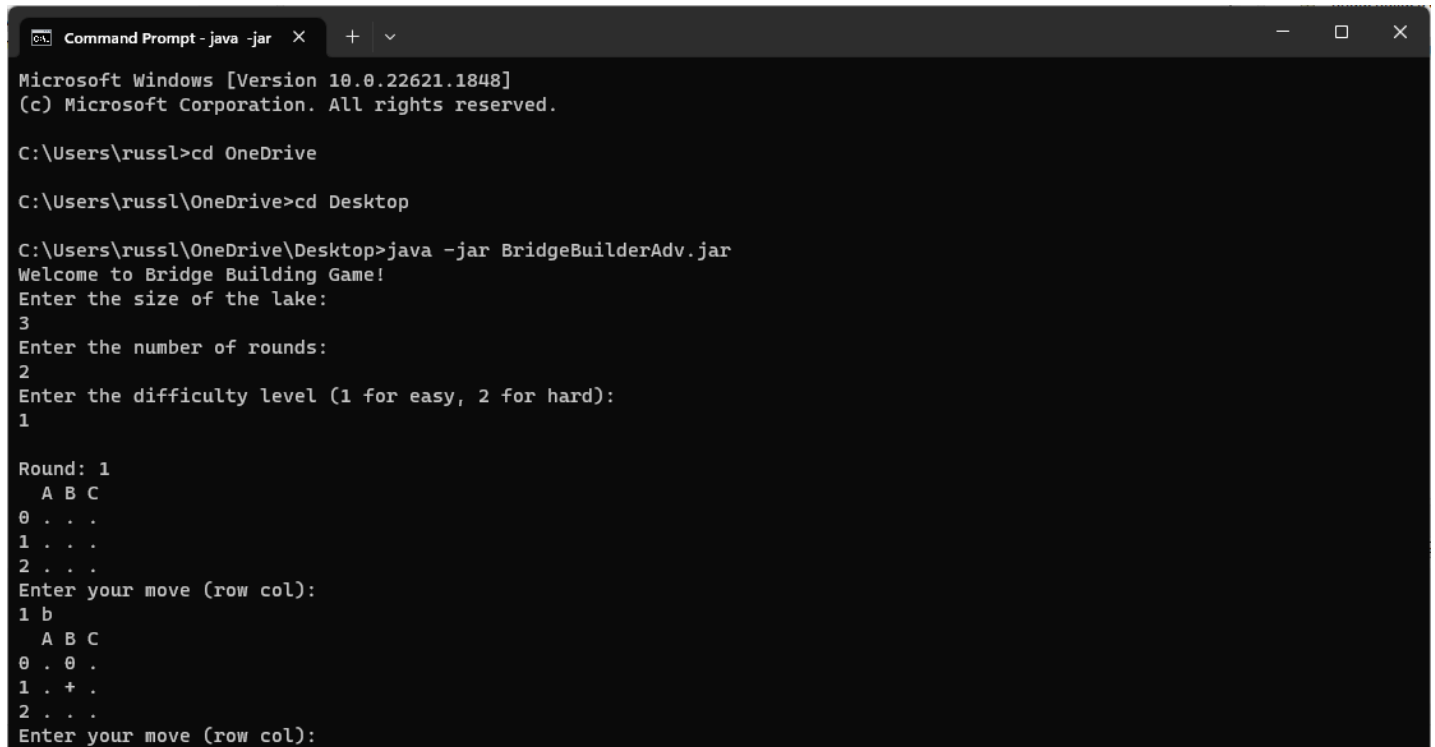
Aim to have your code behave like this. Note that the video is taken in Command Prompt in Windows. It might look different in Eclipse or other IDEs.

Video

There is no sound in this video.

Video of gameplay, provided for example and to clarify the expected wording of the outputs.

Screenshot



```
Command Prompt - java -jar X + v
Microsoft Windows [Version 10.0.22621.1848]
(c) Microsoft Corporation. All rights reserved.

C:\Users\russl>cd OneDrive

C:\Users\russl\OneDrive>cd Desktop

C:\Users\russl\OneDrive\Desktop>java -jar BridgeBuilderAdv.jar
Welcome to Bridge Building Game!
Enter the size of the lake:
3
Enter the number of rounds:
2
Enter the difficulty level (1 for easy, 2 for hard):
1

Round: 1
  A B C
0 . . .
1 . . .
2 . . .
Enter your move (row col):
1 b
  A B C
0 . 0 .
1 . + .
2 . . .
Enter your move (row col):
```

Test Cases: Some tests
you should try to ensure
your code does run
properly:

Note, these are to help you get started testing. Make sure each of these things work, but it's a starting point. Please also do more testing! Make your program more robust than the sample solution!



1. **Test case for GameBoard constructor:** Create a game board and verify that all positions are initialized to '.'.
2. **Test case for GameBoard placeToken:** Place a token at a position and verify it's placed correctly.
3. **Test case for GameBoard isPositionEmpty:** After placing a token, check that the position is not empty.
4. **Test case for Player makeMove:** Test if the player's move places their token at the right position.
5. **Test case for Player addScore:** Add a score to the player and check if the score is correctly added.
6. **Test case for Engineer makeMove in hard mode:** Make sure that the Engineer's move places their token at the right position in hard mode.
7. **Test case for Engineer makeMove in easy mode:** Similarly, make sure that the Engineer's move places their token at the right position in easy mode.
8. **Test case for GameBoard checkForWinDirection:** Verify that the win is correctly recognized in all three directions (horizontal, vertical, and diagonal).
9. **Test case for GameBoard checkForTie:** Fill the board and verify that a tie is correctly recognized.
10. **Test case for main game logic:** Simulate a full game round with predefined inputs and verify the final score and win conditions.

Getting Started

Here are detailed steps to set up Eclipse and start working on the BridgeBuilder assignment. These instructions assume that you've already installed Java JDK on your computer. If you haven't, please install the latest version of Java JDK first (see Lab 0).



****Download and Install Eclipse****

1. Visit the Eclipse download page: <https://www.eclipse.org/downloads/>
(<https://www.eclipse.org/downloads/>)
2. Download the “Eclipse IDE for Java Developers” package which is suitable for your operating system (Windows, Mac, or Linux).
3. Once the download is complete, unzip the downloaded file and run the installer.
4. Follow the on-screen instructions to install Eclipse.

****Setting Up the Project****

1. Open Eclipse. On the welcome screen, close the “Welcome tab” to view the workbench.
2. Click on `File -> New -> Java Project`. Name the project as `BridgeBuilderAdv`.

3. Set the JRE to the latest version of Java you have installed and set the project layout to `Use project folder as root for sources and class files`.

4. Click on `Finish`.

****Setting Up the Package****

1. Right-click on the `src` folder in the Project Explorer in the left sidebar. Click on `New -> Package`.

2. Name the package as `BridgeBuilderAdv` and click on `Finish`.

****Setting Up the Main Class****

1. Right-click on the newly created package `BridgeBuilderAdv`. Click on `New -> Class`.

2. Name the class as `Main`. Make sure the `public static void main(String[] args)` option is checked. This will automatically generate the main method for you.

3. Click on `Finish`.

****Copy the Provided Main Method****

1. Replace the generated `main` method with the provided `main` method from your assignment. Make sure you import the necessary classes such as `java.util.Scanner`. You can download the Main.java file here (<https://1drv.ms/u/s!As0vrM-NsbcQkq43ozKME4vx6c7QUQ?e=1INuYs>).

****Setting Up the Other Classes****

Repeat the process of setting up the Main class for `GameBoard`, `Engineer`, and `Player` classes. This time, don't check the `public static void main(String[] args)` option.

For each of these classes, you can create a constructor with no parameters that does nothing for now. For example:

```
1 public Engineer() {  
2  
3     // Nothing here for now  
4  
5 }
```

****Run the Program****

To run your program, right-click on your `Main` class in the Project Explorer and then select `Run As -> Java Application`.

At this point, your program won't do anything interesting because you haven't implemented the logic yet. But this setup will help you get started with the implementation.

As you implement the logic for each class, you can continue running your program as a Java application to test your changes. Don't forget to save (`Ctrl+S`) before each run.

****Get some help****

Please note that these instructions are for a standard way of organizing Java projects. If you have any difficulty, please talk to a TA in the Lab/Consultation hours every day (except for Canada Day).

Marking Notes



Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes (GameBoard, Engineer, Player, and Main) implemented properly?
- Will the code run properly on Gradescope (even if it runs on Eclipse, it is up to you to ensure it works on Gradescope to get the test marks). Note that Gradescope will not provide you any results of the autograder when you are submitting, except a general confirmation that the file was accepted.
- Does the program produces compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a minimum penalty of 5%
- Including a “package” line at the top of a file will receive a minimum penalty of 5%
- **150 Word Summary**
 - Students are required to provide a 150-word explanation within the JavaDoc of their **GameBoard** class, elaborating their approach towards designing the code, any challenges faced and the steps taken for testing the solution. The explanation should be precise (5), logical (5), and demonstrate clear understanding of the code functionality (10).

Remember you must do all the work on your own. Do not copy or even look at the work of another student, and please develop your code yourself rather than relying on automated tools. All submitted code will be run through similarity-detection software.

Files to submit

- GameBoard.java
- Engineer.java

- Player.java

Grading Criteria

Total Marks: [100%]

- Functional Specifications:
 - [50%] Passing Auto-Graded Tests
- Non-Functional Specifications:
 - [10%] Meaningful variable names, private instance variables
 - [10%] Code readability and indentation
 - [10%] Code comments and JavaDoc
 - [20%] 150 Word Summary of assignment logic in GameBoard.java

Submission Information

Submission will be on Gradescope (a 3rd Party grading tool) on Monday Jul 3 between 9am and 11:55pm ET. No early submissions will be permitted. Instructions for submission will be posted before Jul 3.