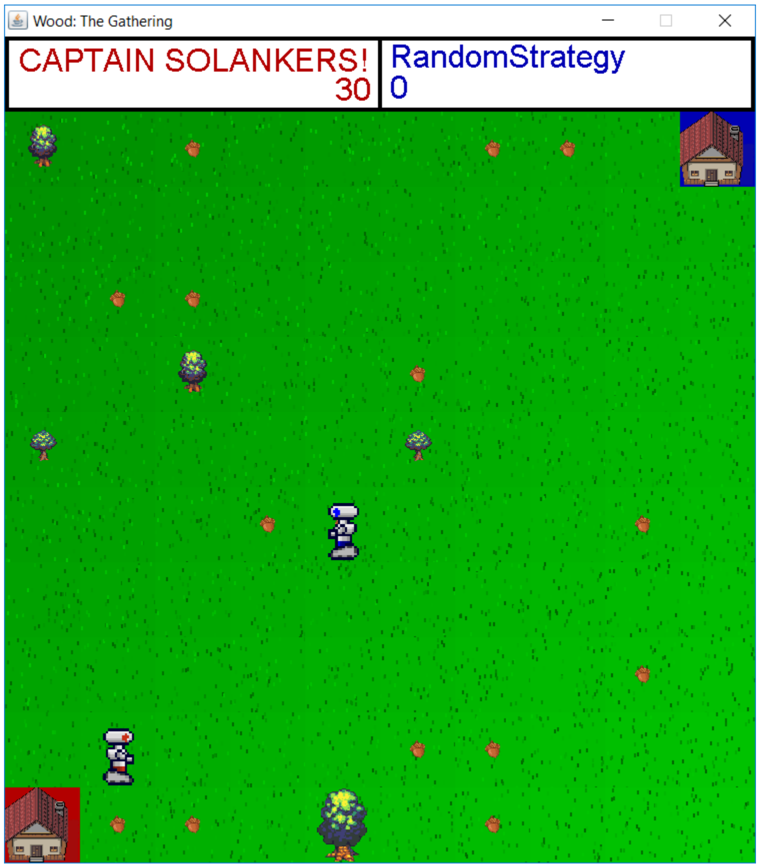
1 Wood: The Gathering

For this assignment, you will be creating at least one intelligent player strategy for a wood cutting game that we have given you. The game is called **Wood: The Gathering**.

We will provide you with everything else needed to run this game. This includes a game engine, graphics engine and associated graphics resources. We will be giving you all of the source code for these pieces so that you may test your strategies. You are not allowed to modify the interface that we provide, **WoodPlayerStrategy**. You are free to modify any other files including the game engine and graphics engine, but do so at your own risk and know that we will be using the original copy that we gave you to run the competition. You should create your Github repository from this link:

`https://classroom.github.com/a/PH9gc7Qy`



2 The Game

The game is a turn based wood cutting competition played on a NxN square set of tiles which is considered the **Game Board**. There are two players per game, a red player and a blue player. The red player will start in the lower left tile of the game and the blue player will start in the upper right tile of the game. Each turn both players will receive information about the board state and will be required to submit a **TurnAction**, which is the action that they want to make on the next turn. Turn actions including moving up, right, down and left, picking up a seed, planting a seed, and cutting down a tree. All actions take one turn to execute.

Movement throughout the game will be in the four directions, **UP, RIGHT, DOWN, LEFT**. Moving one tile takes a single turn. You are free to move wherever you want within the game board. If you try to move outside the bounds of the game board, your player will simply not move.

Scattered throughout the **Game Board** are seeds. In order to pick up a seed, you will need to position your player such that they are standing on the tile with the seed and then execute the **PICK_UP** action. Once you have a seed, you can plant it on any empty tile. In order to do so, you will need to execute the **PLANT_SEED** action when you have a seed in your inventory and you are standing on an empty tile. If you have multiple seeds in your inventory, the seed that you picked up first will be planted first (FIFO). Finally, once you have planted a tree (or your opponent has planted a tree) you can cut it down by executing the **CUT_TREE** action when standing on a tile with a tree on it.

Each player has an inventory which has a max size. Players will receive **Seed** items when they successfully execute the action **PICK_UP** and players will receive **Wood** items when they successfully execute the action **CUT_TREE**. Performing one of these actions with a full inventory will result in the item being permanently lost from the game. Planting seeds will remove seeds from the player's inventory and scoring wood will remove wood from the player's inventory.

Players can score wood by walking back to their home tile. This is the tile that they started on. Once the player is standing on their home tile with **Wood** in their inventory, all the wood items will be removed and points added to their score.

When trees are planted, they will constantly grow and gain value every turn. There is no limit to the size or value of a single tree. The value of the wood item received when a tree is cut down is the value of the tree on that turn.

Finally, since players execute turns "at the same time", potential conflicts such as moving into the same tile will be handled by the concept of red turns and blue turns. On a red turn, the red player will have priority for these types of moves. On a blue turn, the blue player will have priority. Red turns and blue turns will alternate throughout the game. The first turn will be a red turn. So if two players are trying to move into the same tile, and it is currently a red turn the red player has priority and will successfully move into that tile. The blue player will be blocked from moving and therefore do nothing.

3 The Interface

We have given you an interface that your player strategy should implement. You need to implement this interface, otherwise your strategy will not run with our game engine. This interface is called **WoodPlayerStrategy**. It has the following methods:

1. **initialize** - This function will be called at the start of every round, passing in the size of the board, the maximum items that a player can carry in their inventory, the score to reach to win a single round, the starting tile location for the player, whether the player is a red player and a **Random** object for the player to use for generating random numbers if their strategy

needs that.

2. **getTurnAction** - This function should be the meat and potatoes of your player. It will be called every turn to get the action that your player wants to execute. This passes in the current board state as a **PlayerBoardView** object and whether the current turn is a red turn or not. The function then returns a **TurnAction**, which is the action that the player wishes to execute. Returning **null** in this function will cause the player to do nothing.
3. **receiveItem** - This function will be called whenever the player successfully finishes an action that results in them receiving an item. The two actions that can result in this are **PICK_UP** and **CUT_TREE**. This function will not be called if the player's inventory is full, even if they successfully finished an action.
4. **getName** - This function is super simple, just return the name that you want to give your player. Can be as simple as just a string constant. Note that if you manage to throw an exception in this function, your player name will be the exception.
5. **endRound** - This function will be called at the end of every round. This passes in the number of points that your player scored and the number of points that the opposing player scored. You should also use this method to reset any status or state that your player may carry.

4 The Competition

We will be running a competition throughout the week of the assignment (and potentially after). Times when the competition will be run and additional information will be posted on Piazza. Participation and performance in the competition will be a small percentage of your grade.

4.1 How to submit code

Push code to your master branch. We will pull this when we run the competition.

In order for us to run your player strategy in the tournament, you must put a single class that implements the **WoodPlayerStrategy** in the following package. You may also put any other classes you need in that package.

- `package wood.competition`

4.2 How to make sure your code compiles in our competition engine

Do not include any subpackages in the `wood.competition` package. These will not get compiled or run. Do not reference any classes that you personally created which are outside the competition package.

- References to classes we provided like the **WoodPlayerStrategy** class or **TurnAction** enum are perfectly fine.
- If you have other classes you wrote that your strategy needs, they need to be in the competition package.
- This means you either need to move them to the competition package, or make a copy of the class to put in the competition package.

- Yes, repeating code is bad, but points will not be taken off for having duplicate classes in the competition package.
- Your class must also implement a constructor that takes no arguments. You may have logic in this constructor and you may have other constructors, but you need at least one constructor that does not take any arguments. This is the constructor that will be used in the competition engine.

4.3 How to ensure your code is valid

Put only one player strategy in the competition package, otherwise we will not know which Player to run. There should only be one class in the competition package that implements **WoodPlayerStrategy**

- **Do not print things in your player strategy:** This clogs up our logging of the games being run and we will not execute your code if your strategy prints to console
- **Don't cheat:** Do not use things like using reflection to directly modify your player's score. This is just an example, reflection and other similar techniques are disabled.
- **Don't write malicious code:** Things like calling `System.exit()` in your player strategy will not work and will result in a 0 for this assignment.

5 Requirements

To fully complete this assignment, you must write a **WoodPlayerStrategy** that competes in the competition. This strategy must at least meet the following requirements.

- It participate in and not crash in the competition
- It must work on any board sizes between 10 and 30 with different random seeds
- It must consider the case where your trees are cut by the other player
- It must consider the case where you move to the same space as the other player
- It must win by hitting min score vs random 99% of the time.

6 C++ Preparation

In order to prepare for using C++ in the following weeks you will need to show you can build a C++ program. Specifically the HelloWorld program. There is nothing to check in for this simply show that you can compile the program at your code review.