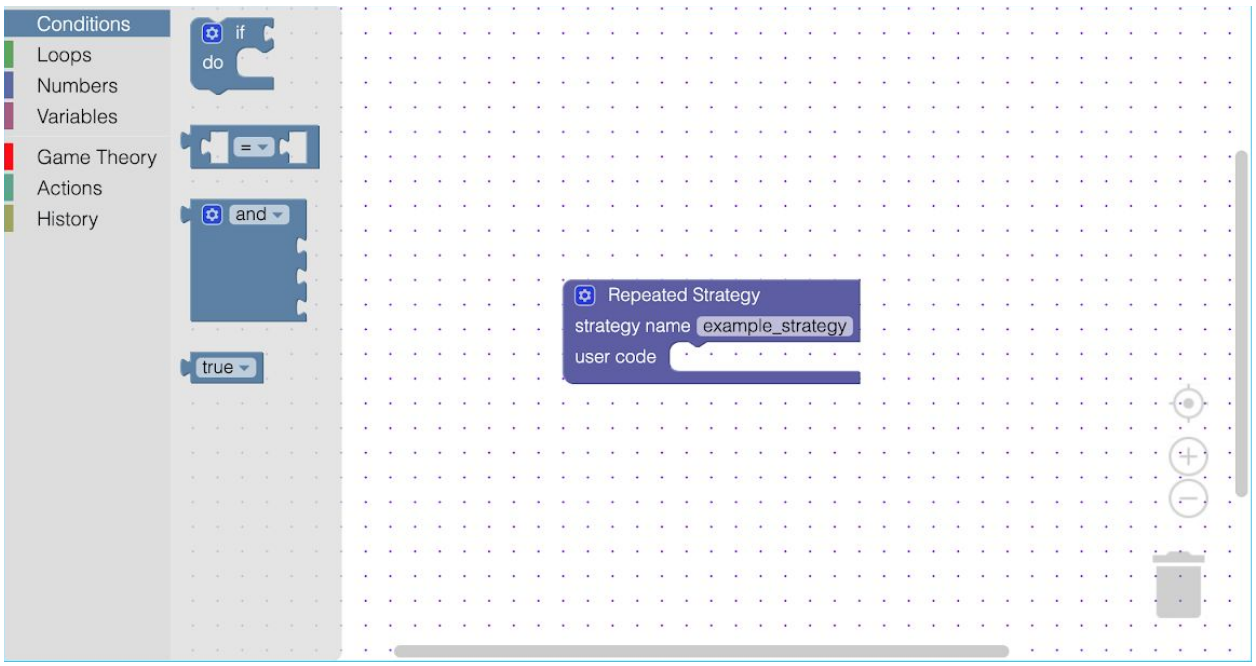
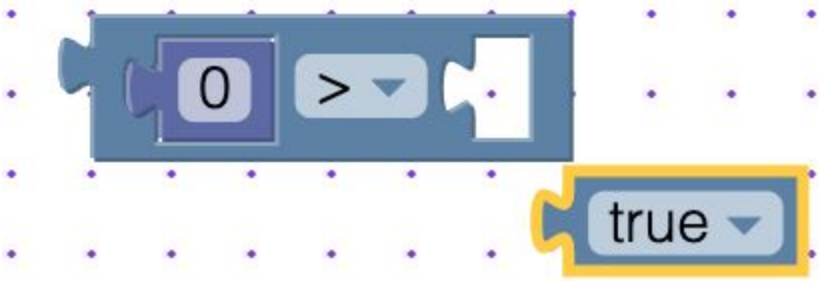
Getting Started

Workshop

The Workshop is made up of blocks (grey area located on the left) and the main workspace (dotted area in the center). Blocks are divided into categories based on their function. There should only be one Repeated Strategy block on the workspace, and any blocks on the workspace must be inside the Repeated Strategy block. Make sure blocks are properly connected and that no loose blocks are lying around.



These blocks abstract code into everyday language. Connected blocks specify a set of instructions that will be translated into code. The blocks have special shapes, and they can only be connected to other blocks of the same shape. Some blocks are even more particular and will only connect to values of a certain type or range. Furthermore, these blocks safeguard against some logical errors by preventing blocks that don't make sense together from connecting.



Example: *Since "0 > true" doesn't make any logical sense, the blocks are prevented from connecting together.*

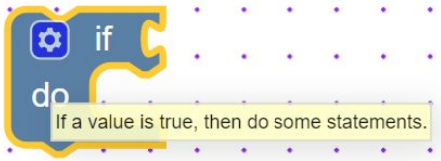
Use the provided blocks to create a set of directions based on different conditions. Remember, the goal is to specify a set of actions for every scenario you could possibly encounter.

Stuck? See here for general troubleshooting tips.

Blockly Tutorial

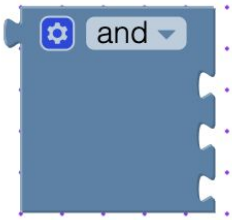
General Blocks

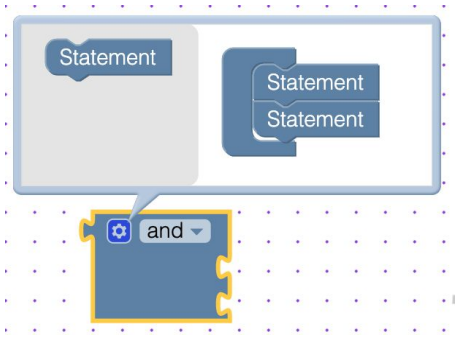
Hover over the block for its description.



Changing the number of inputs and connections

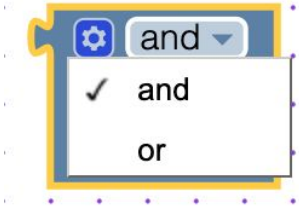
Blocks with the settings symbol on the top-left corner can be configured to increase/decrease the number of inputs and connections.





Changing block values with dropdowns

Simply click on the dropdown to select other available values.



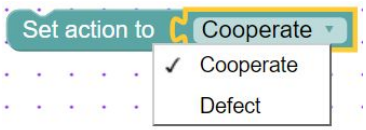
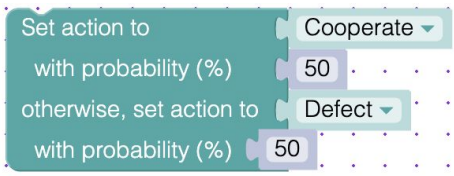
Custom Game Theory Blocks

(See Figure 1)

History Access

Type	Blocks	Input	Output
Action Access	my action 1 period(s) ago Opponent's action 1 period(s) ago	*period(s)* **cannot** exceed the number of periods so far	Returns an action (i.e. Cooperate/Defect).
Payoff Access	My payoff 1 period(s) ago		Returns a value based on the payoff matrix in that period.
Current Position	current period	-	The current position/period in the game (i.e. an integer ≥ 1).

Action Block

Field	Description	Input(s)	Explanation
1. Regular (non-probabilistic)		An action (i.e. Defect/ Cooperate).	Sets the strategy's action for the current period.
2.Probabilistic		1. Action a. 2. Probability 0 ≤ x ≤ 100	Note: the last two fields are not editable. They ensure the probabilities sum to 1 and that every action is accounted for.

Examples

1. With regular (non-probabilistic) action block. (See Figure 2)
2. With probabilistic action block. (See Figure 3)

Repeated Strategy Block

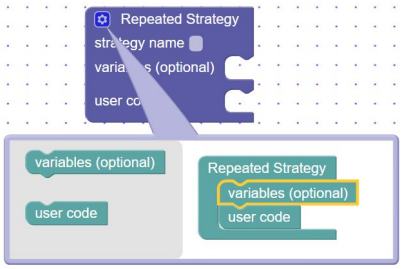
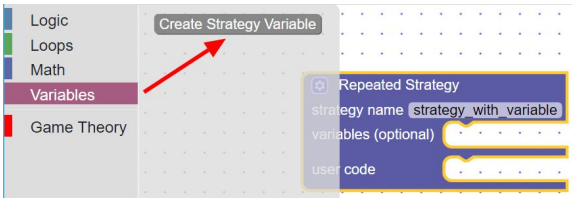
The Repeated Strategy block encapsulates all strategies. Example (See Figure 2).

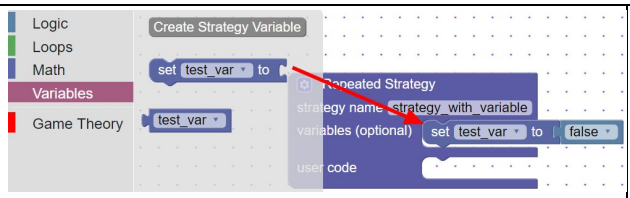
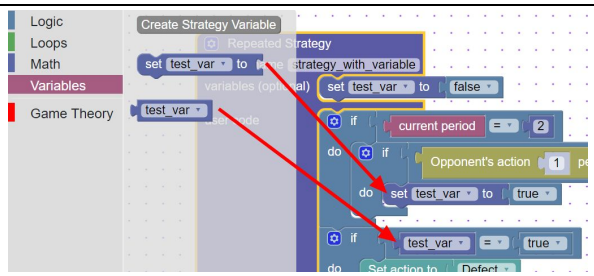
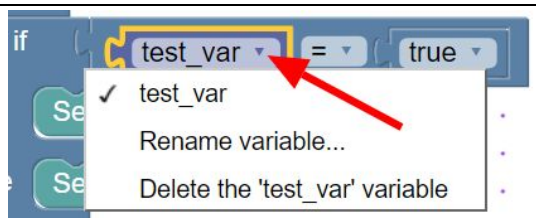
Field	Description	Requirement
strategy name	The name of your strategy.	Names **cannot** have any spaces between them.
user code	All blocks used must go inside this section.	Requires the Set action to block at the **bare minimum**.

Variables

Variables are assigned a value. They can be used as a flag or to keep track of values. The use of variables simplify the steps needed to access data.

Using a Variable

Step	Image	Description
1		Click the settings icon and drag the *variables (optional)* block to modify the *Repeated Strategy* block.
2		Go to *Create Strategy Variable* and give your variable a name.

3		Once the variable is created, it will appear in the *Variable* tab. Use the *Set* block to give it an initial value in the variables(optional) section.
4		Use the *Set* and/or the variable's input connector as desired.
(5)		Variables can be edited/deleted by clicking on the name.

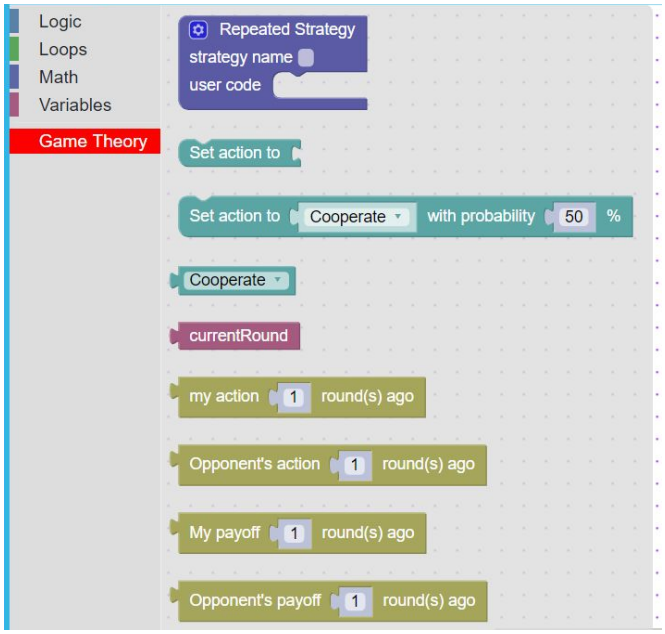
Examples

1. **Tit for Tat Strategy** - Using history blocks and current period . (See Figure 4)
2. **Repeated Pattern** - Using remainder in Math blocks for the repetition. (See Figure 5)
3. **Variables** - Using a variable to keep track of a value or as a flag. (See Figure 6)

**Note: There are multiple ways of implementing these strategies. These examples show just one way the blocks can be used to construct certain strategies.*

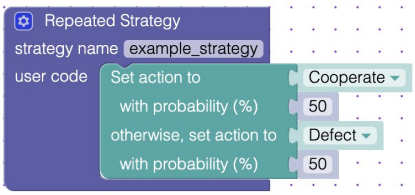
Figures

1. Blocks in the Game Theory tab.

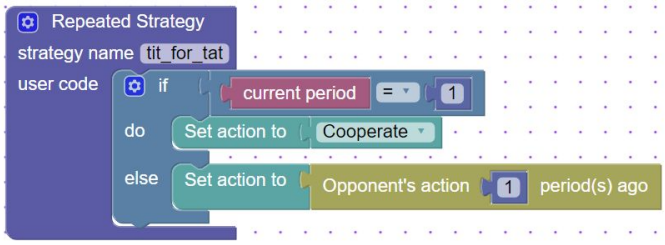


2. Example implementation of **all_C** (always cooperate) strategy.

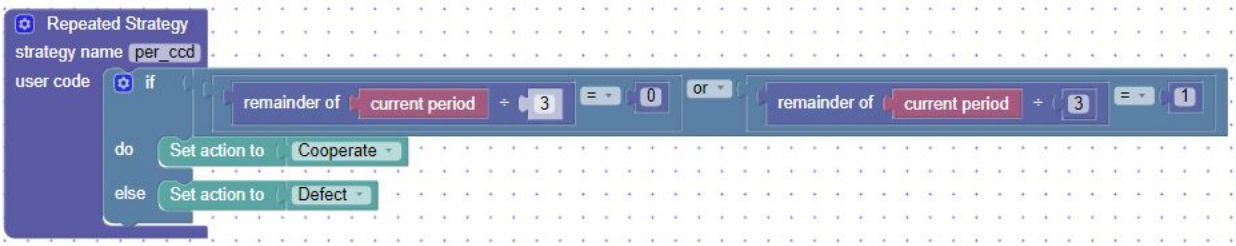
3. Example implementation of **random_choice** strategy (i.e. cooperate/defect with equal probability).



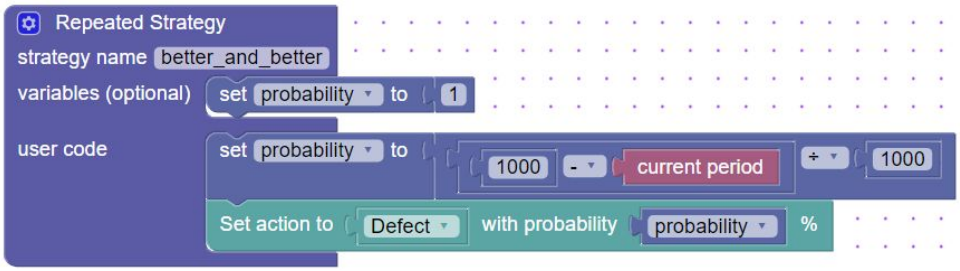
4. Example implementation of **tit_for_tat** strategy.



5. Example implementation of **per_ccd** (cooperate-cooperate-defect) strategy.



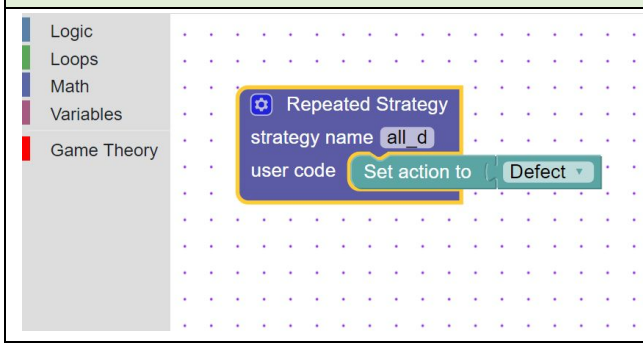
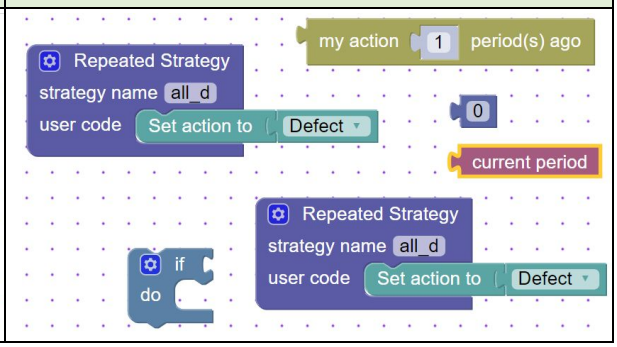
6. Example implementation of **better_and_better** strategy. Variable probability is used to keep track of which probability% to play for clarity.



General Troubleshooting

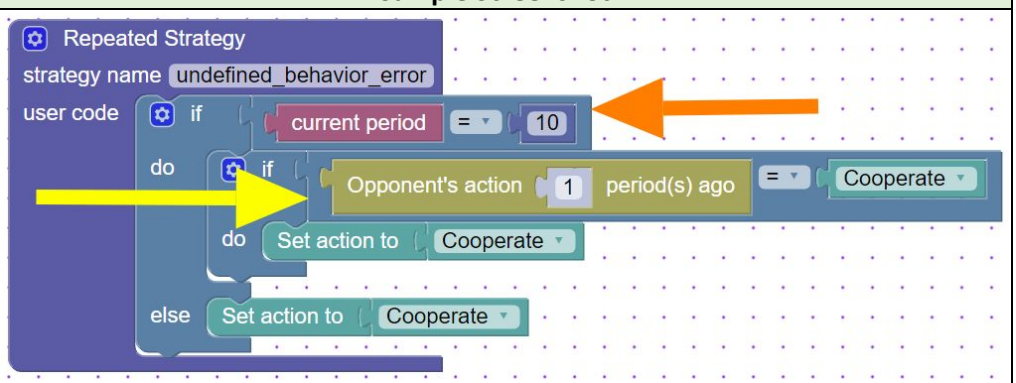
Floating Blocks

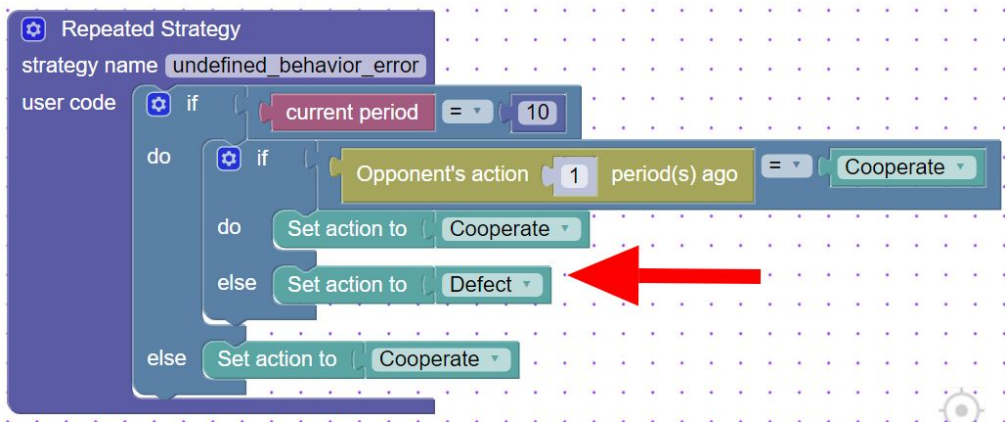
1. Avoid having any blocks outside of the *Repeated Strategy* block.
2. Avoid having multiple *Repeated Strategy* blocks. There should only be **one** in a workspace at a time.

Do	Don't
	

Undefined Behaviors

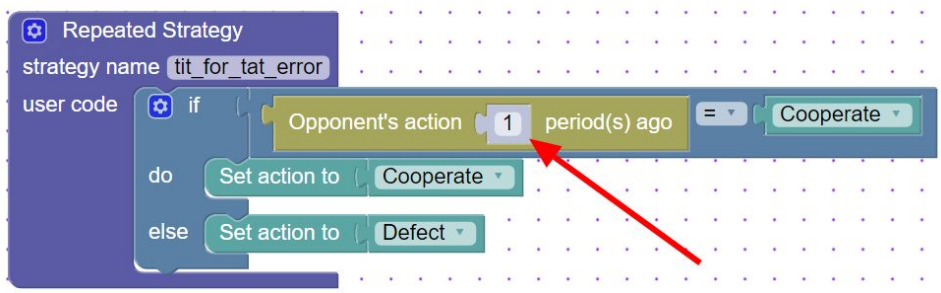
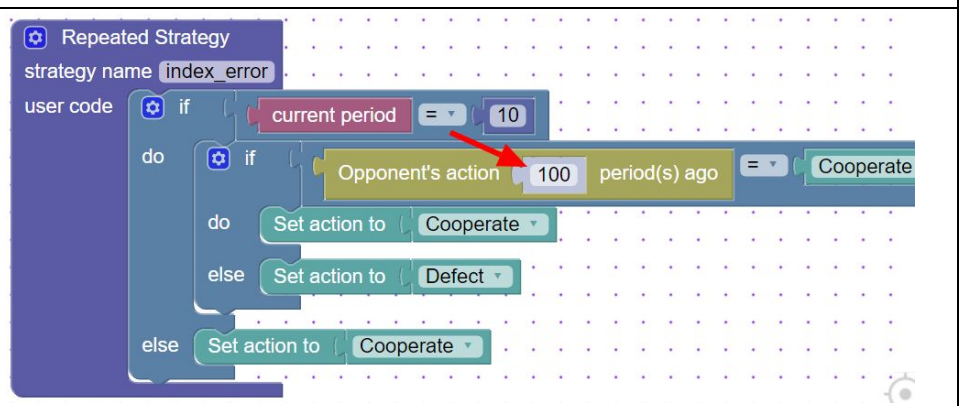
1. Make sure an action is defined for **ALL** possible cases, especially in if-else cases.

Problem	Sample Screenshot
Action not defined for the case where the first condition (orange arrow) is true and the second condition (yellow arrow) is false.	

Fix	

Indexing

1. Make sure the referenced period is greater than the current period.

Problem	Sample Screenshot
Unable to find opponent's action one period ago at the first period.	
Referenced period of 100 periods ago is greater than the current period (10th).	

2. **Keep in mind:** Blocks go by one-based indexing (i.e. periods **start** at 1).

Step-by-Step Example: Strategy Creation

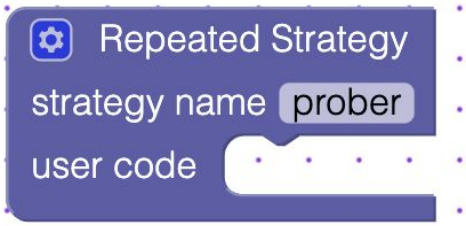
Let's take the strategy Prober as an example.

Prober: *Plays D, C, C initially. Cooperates forever if opponent played D then C in moves 2 and 3. Otherwise plays TFT. (Source)*

Part I: *Plays D, C, C initially.*

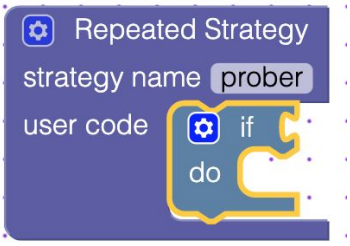
This means defecting (D) on the first round, then cooperating (C) on the second and third round.

I. Start by giving the strategy a name in the **strategy name** box.

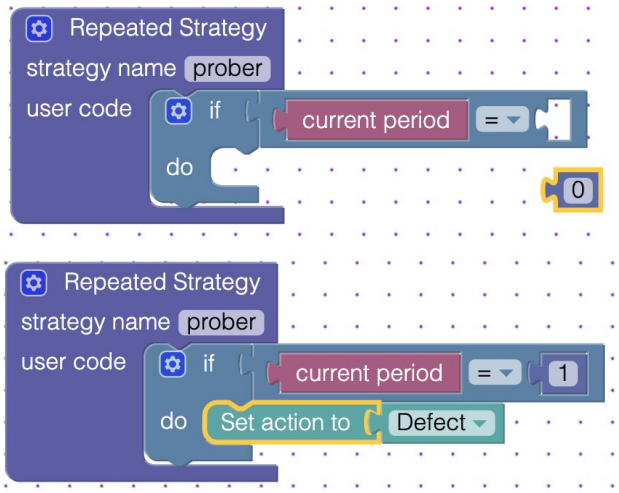


II. Create our first condition: play D in the first round.

A. First, take out the if, do block from the *Conditions* category. Make sure it is connected to the Repeated Strategy block.



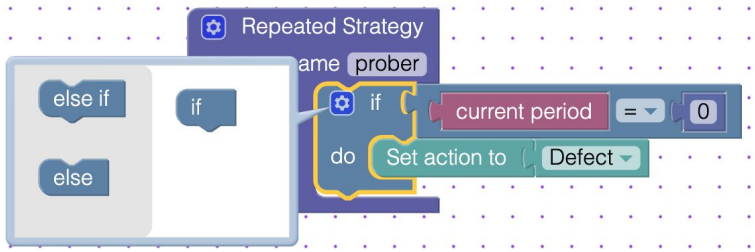
B. Then, add the condition: if current period is equal to 1, play action D (defect).

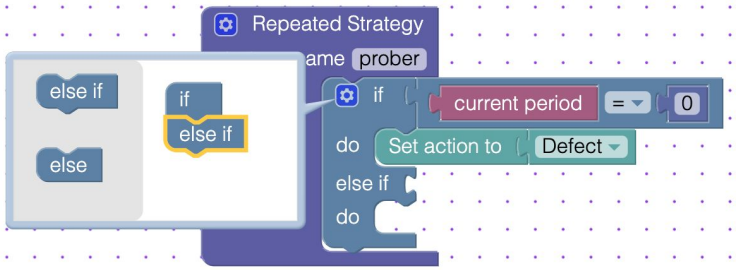


*Make sure to change the default values in the **numbers** and **action** blocks to the desired values (i.e. change 0 in numbers to 1 and Cooperate in actions to Defect).*

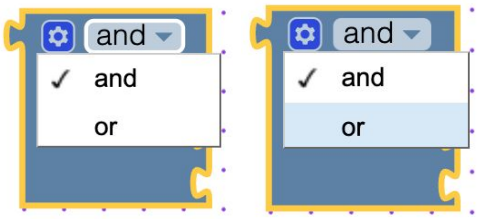
III. Create our second condition: play C in the second and third round.

A. Add another condition to the **if/do** block.

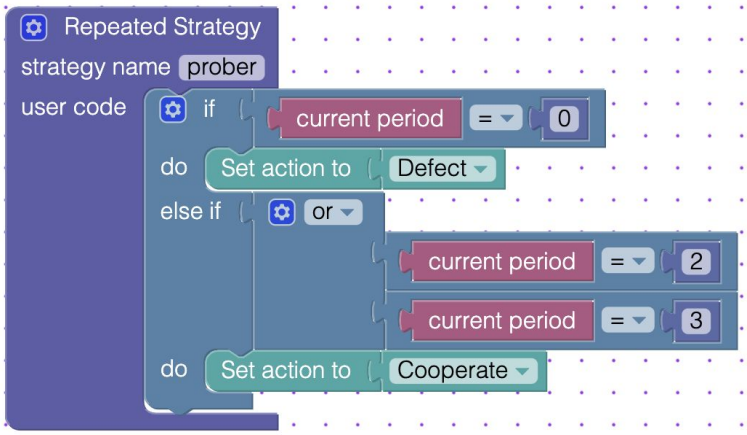




B. We will need an **or** condition to check for periods 2 or 3. Create one using the and/or block from the Conditions category.



C. Fill out the **or** condition and add it to the block.



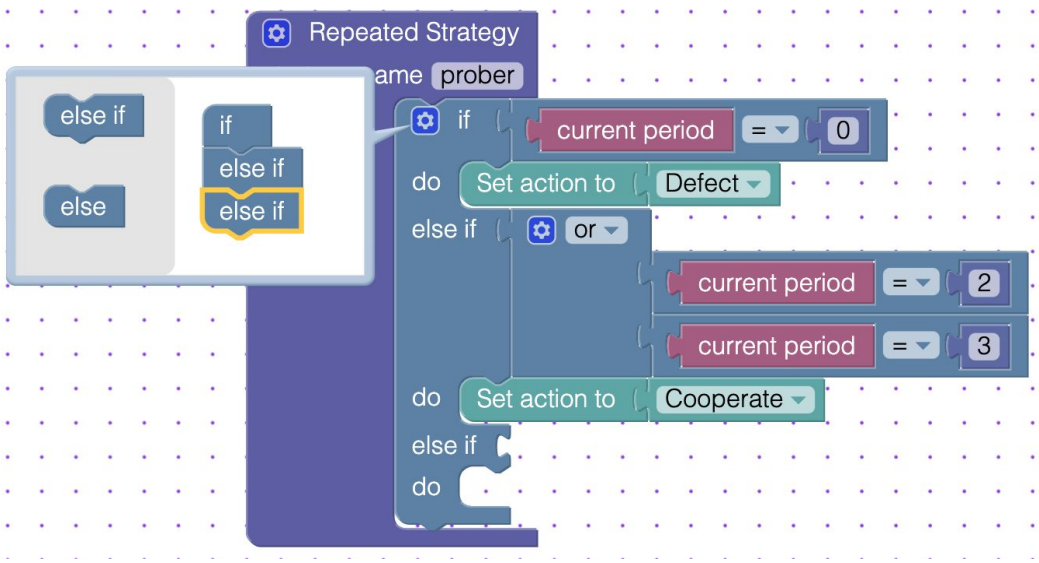
*** Checkpoint: we've now created a strategy that says: Defect in the first round, then cooperate in the second and third round.

Part II: *Cooperates forever if opponent played D then C in moves 2 and 3. Otherwise plays TFT*

Depending on whether the opponent played D and C in the second and third round, we will take two courses of actions.

IV. Add another condition: *if the opponent played D and C in the second and third round*.

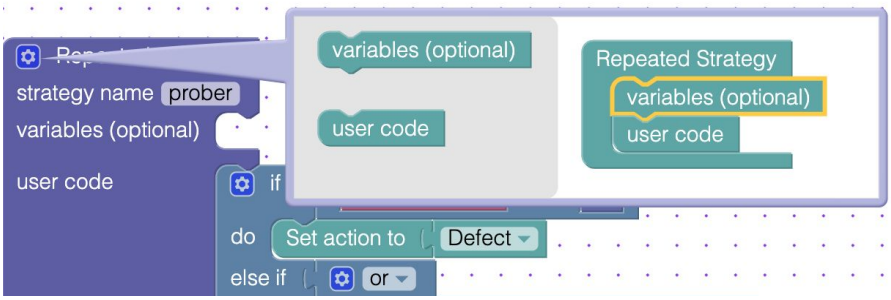
A. Add another **else if** condition:

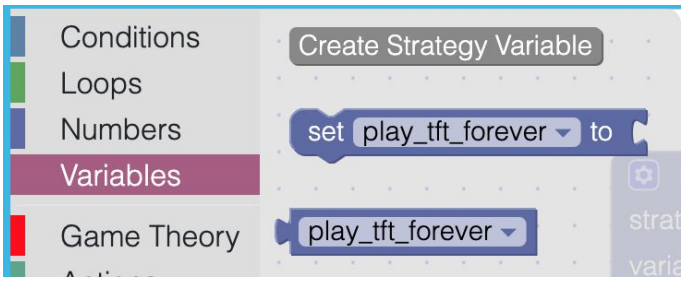


V. Introduce a variable to keep track of whether the opponent played D and C in the second and third round.

Reminder: Variables keep track of values. The variable will keep track of whether our strategy should cooperate forever or play tit-for-tat for the remainder of the game.

A. Add a variable. We will call it ***play_tft_forever*** (where tft is tit-for-tat).





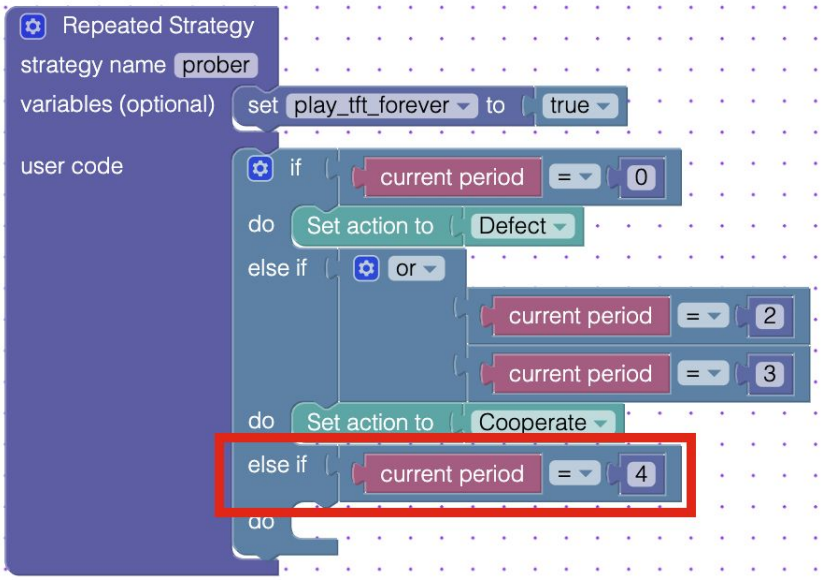
B. Give the variable an initial value.

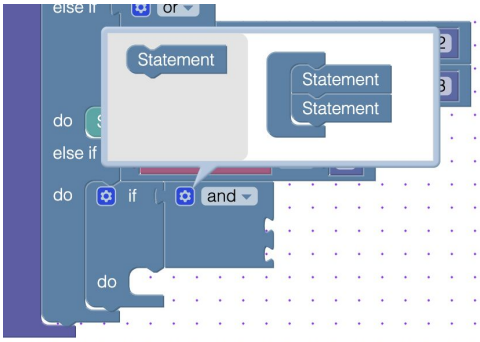
i.e. We will play tit-for-tat forever unless something happens (something being the opponent playing D then C in the second and third round).

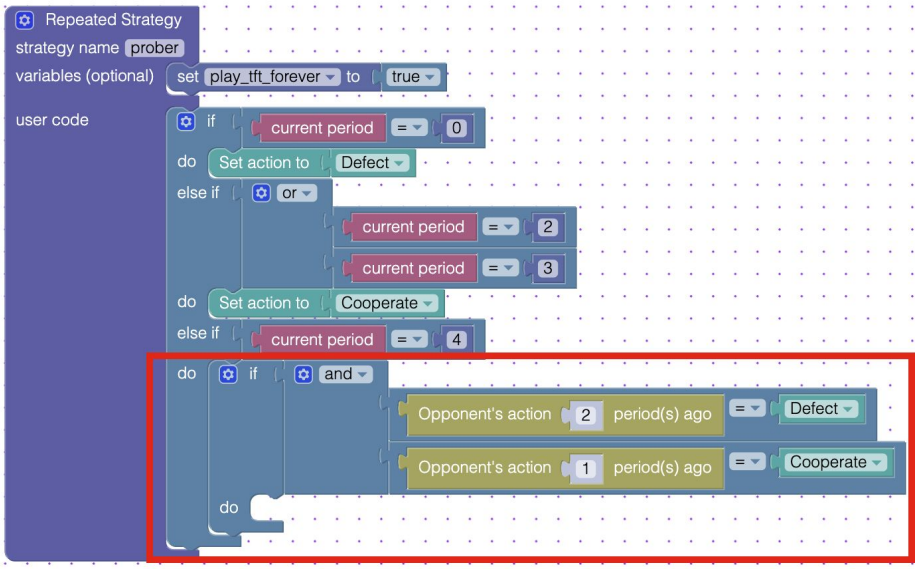
VI. Update the variable when the condition to play tit-for-tat forever changes

i.e. If the opponent played D then C in the second and third round, we will not play tit-for-tat forever anymore.

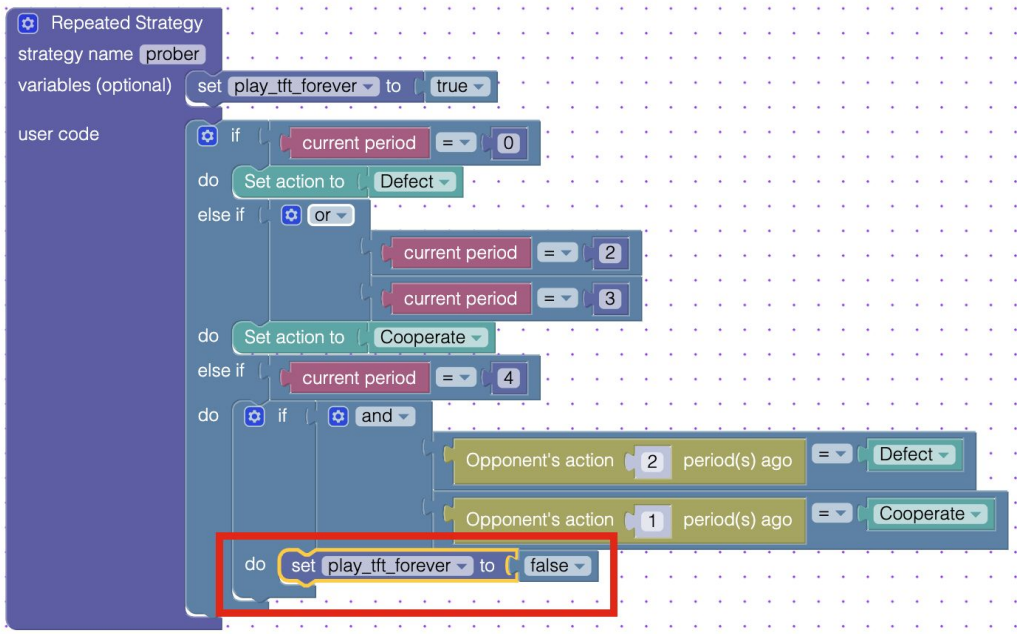
A. In the fourth round, check the opponent's actions in rounds 2 and 3.





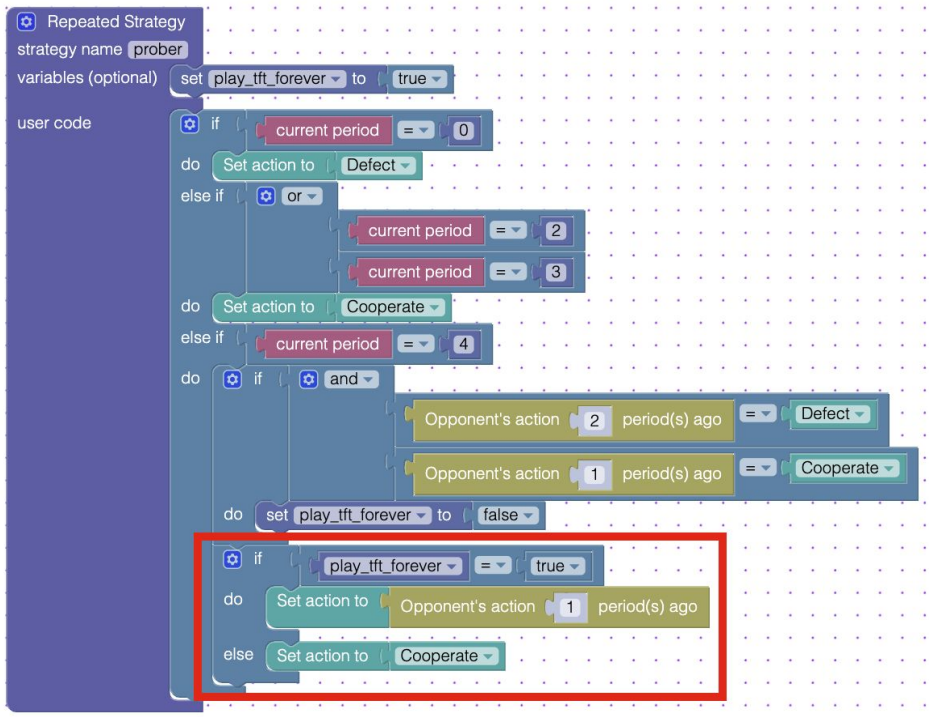


B. Since we will cooperate forever if the opponent's action in rounds 2 and 3 is D then C, we will have to change our variable **play_tft_forever** to false.



C. Now, in the fourth round, we can decide which action to play based on the value of our ***play_tft_forever*** variable.

If the ***play_tft_forever*** variable is true, then we play tit-for-tat (i.e. the opponent's previous action). Otherwise, we will set our action to Cooperate.

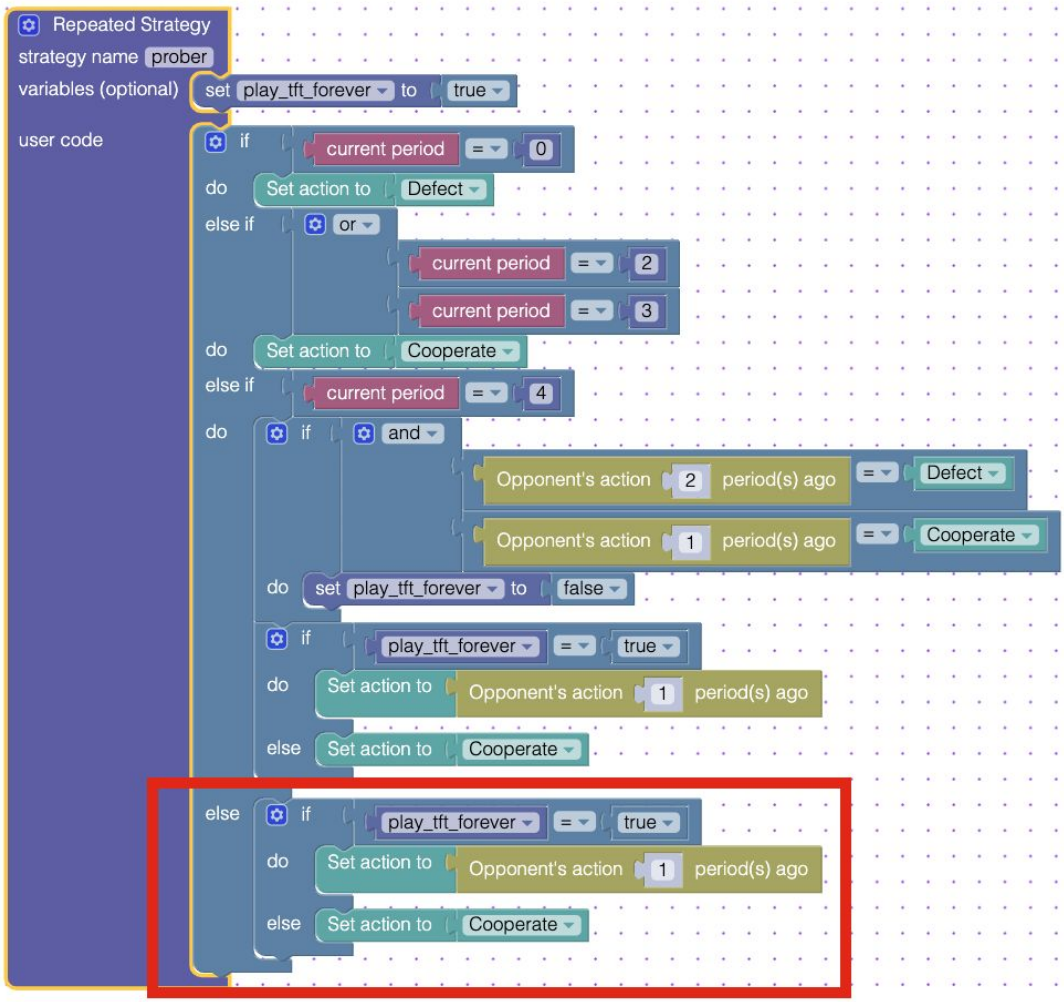


VII. Play an action based on the value of our variable.

A. From the 4th period onwards, the action we play won't change. We can simply refer to the variable to determine which action to play.

An else case is added as a catch-all condition. Any round after the 4th round shall be treated the same.

See image below.



*** Self Check: We've created a strategy that specifies an action for every single round, and handles every scenario the opponent could play.