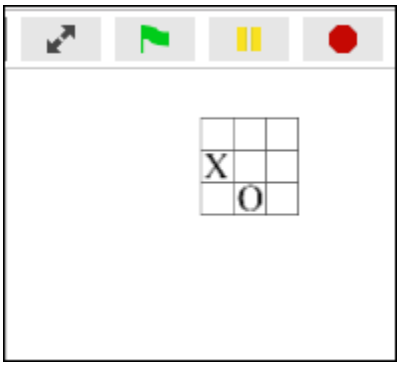
Board Games

Glen Bull and Joe Garofalo

Many board games involve moving tokens across a grid of squares. Chess, checkers, tic-tac-toe, and *Battleship* all follow this format.

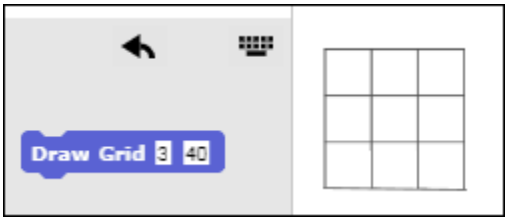


Reconstruction of tic-tac-toe as a digital game involves two basic coding tasks:

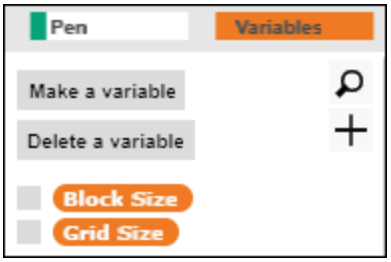
1. Drawing a three-by-three grid, and
2. Procedures to place the tokens.

Drawing a Tic Tac Toe Grid

The *Draw Grid* procedure was constructed using the *Line* and *Over* procedures developed for the prior module, *Graphing Data*. It has two inputs, *Blocks per Side* (3) and *Block Size* (i.e., the size of the blocks in the grid).



In this instance, the grid was constructed with a *Grid Size* of 3 (i.e., 3 rows and 3 columns) and a *Block Size* of 40. These values were assigned to global variables so that they could be used by other procedures.



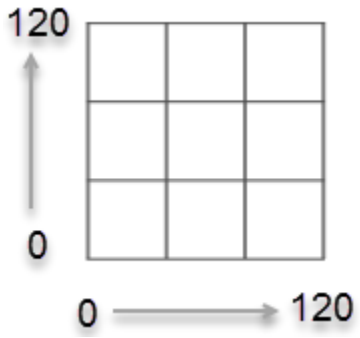
Creating and Identifying the Location of Game Tokens

The game tokens for tic-tac-toe consist of an "X" and an "O". Two sprites were created, one for each shape.

You can access a *Snap!* file with the tokens installed here:

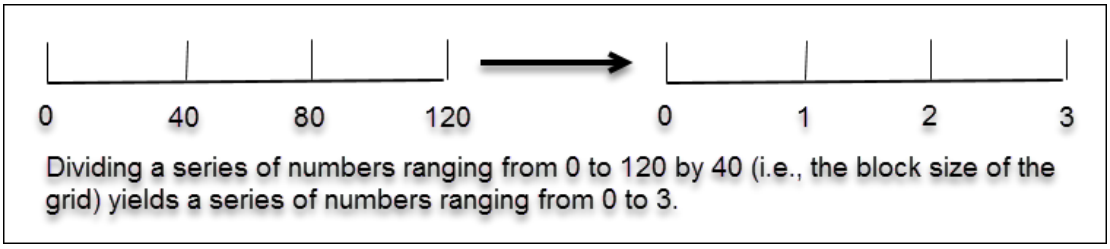
Put Link Here

The built-in *X Position* and *Y Position* commands (found under the *Move* section of the commands palette) can be used to identify the row and column in which a token has been placed.



A grid of three rows and columns with a block size of 40 steps spans the distance from 0 to 120. Dividing the X and Y coordinates by the block size yields the row and column in which a sprite is placed.

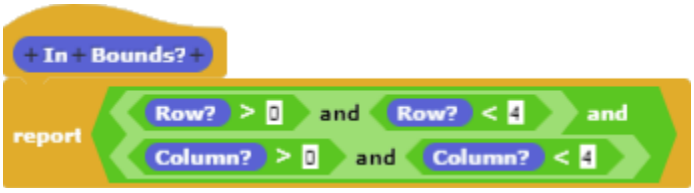
For example, if a token is placed at the X-coordinate of 60, dividing by 40 yields the result of 1.5. Rounding this result yields the integer 2. This result indicates that the token has been placed in the second column. Eventually the token will need to be placed in the center of the square.



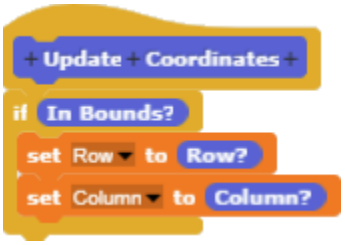
Numbers between 0 and 0.5 round to 0, which would place the token outside the grid. This can be corrected by adding an offset of 0.5 to yield the correct result of "Column 1". The code to accomplish this is shown in the *Column?* procedure.

A similar method is used to identify the row in which the token has been placed.

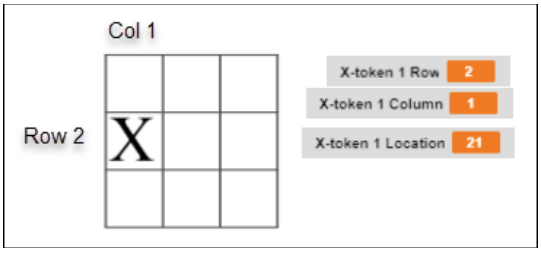
It will be important to know whether the token is on the grid or outside its boundaries. The *In Bounds?* procedure reports whether the result for row and column is between 1 and 3.



If the token is in bounds (i.e., on the game board), the *Update Coordinates* procedure assigns the row and column in which the token has been placed to the global variables *Row* and *Column*.

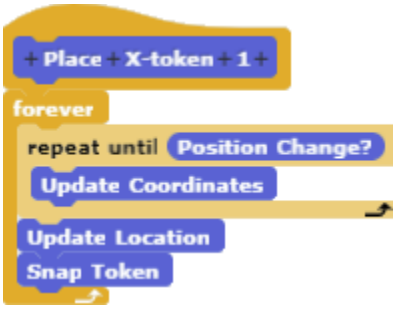


These values are used by the *Update Location* procedure to locate the token on the board. For example, if the token has been placed into Row 2 and Column 1, the *Update Location* procedure assigns a value of 21 to the token's *Location*.



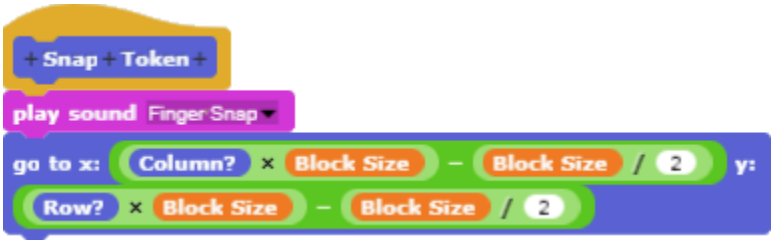
Place Token

The foregoing procedures are used to snap the token into place on the correct square of the game grid. The *Place Token* procedure repeatedly checks to see if the position of the token has been changed.



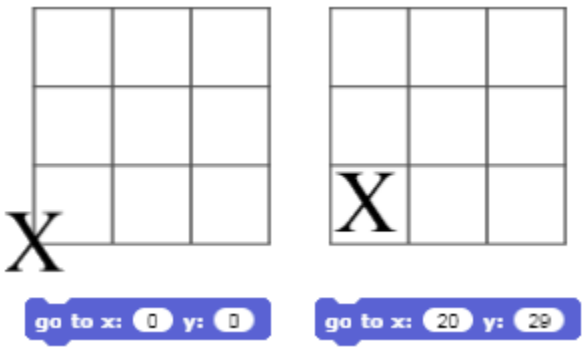
A position change can be detected by checking to see if the most recent coordinates (i.e., row and column) of the token match the previously recorded location of the token.

When a position change is detected, the recorded location is updated using the *Update Location* procedure, and the token is snapped into place on the new game square. The *Snap Token* procedure snaps the token into place.



The procedure provides a sound effect as feedback to indicate that the token is being snapped into place.

- The command *Go to X = 0, Y = 0* sends the token to the corner of the board.
- The command *Go to X = 20, Y = 20* sends the token to the center of the first square.



Therefore, multiplying the row number by the block size and then subtracting half the block size will place the token in the center of the row. For example, if the token is in Row 1:

Row 1 x 40 (i.e, the block size) = 40; 40 – 20 = X coordinate of 20

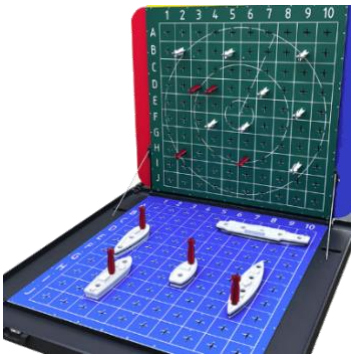
Similar calculations can be used to determine the center of the column.

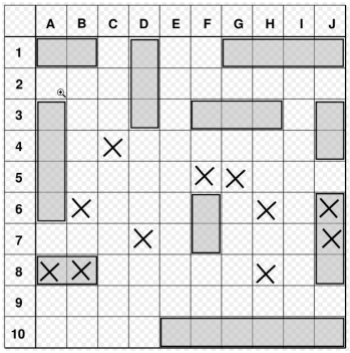
Column 1 x 40 (block size) = 40; 40 – 20 = Y coordinate of 20

Thus, the *Place Token* procedure aligns the token with the center of the square at the intersection of the row and column in which the token has been placed.

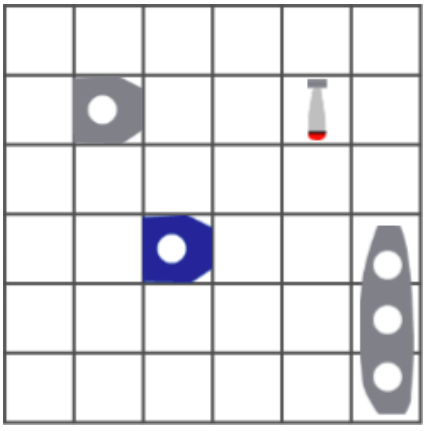
Battleship

Battleship is another board game that is played on a grid. A fleet of ships concealed from an opponent is placed on a grid. The opponent attempts to torpedo ships by calling out coordinates that are marked on the grid. *Battleship* has been played as a pencil-and-paper game on a grid of paper and has also been manufactured in several commercial versions with plastic pieces.





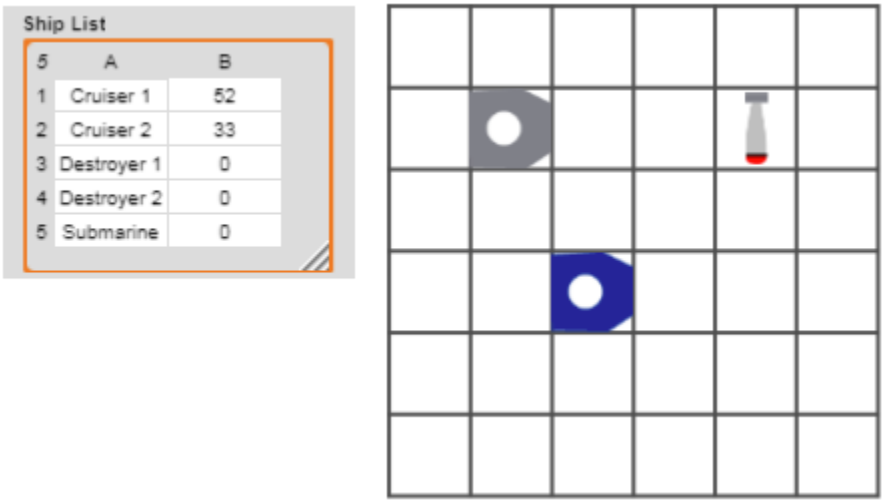
The *Snap!* version of *Battleship* uses ships and torpedoes placed on a grid drawn on the computer screen. The grid is similar to the tic-tac-toe grid except that it is a 6 x 6 grid instead of a 3 x 3 grid. The outline of various ships (cruisers, destroyers, submarines) are used in place of the "X" and "O" tokens used in the tic-tac-toe game.



One player (or, optionally, the computer) places ships on the grid. The ships are then hidden (using the *Hide* command found under the *Looks* section of the command palette). The opposing player then places

a torpedo on a square in attempt to hit a ship. If a ship is hit, an explosion occurs. If a ship is not hit, an "X" is placed in the square to indicate that the square is empty.

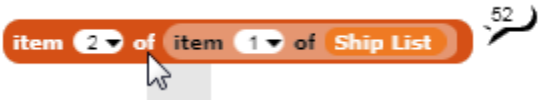
In order to determine if a ship has been hit, a list of squares on which ships have been placed will be needed. (Lists were first introduced in Module 4, *Words and Lists*. Additional attributes of lists were discussed in Module 5, *Graphing Social Data*.) In the illustrated example, the *Ship List* indicates that ships have been placed on *Square 33* and on *Square 52*.



The procedure to place *Cruiser 1* on the grid is identical to the *Place X-token* procedure (from the preceding section on tic-tac-toe), with one enhancement. After the token is snapped into place and the location is updated, the procedure records the location of C*ruiser 1* on the ship list.

Item 1 of Ship List consists of two items: (1) the name of the ship (Cruiser 1) and (2) the location of the ship.

Item 2 of *Item 1* of the *Ship List* points directly to the location of *Cruiser 1*:



Therefore the command *Replace Item 2 of Item 1 of Ship List with Location* will update the value of the location of *Cruiser 1* with the location of the square on which it has been placed.

Collision Detection

The *Place Torpedo* procedure is identical to the *Place Ship* procedure except that at the end, the *Torpedo* procedure checks to see if there has been a collision with one of the ships.

The *Torpedo* procedure checks for collisions by checking to see if the *Torpedo's* current location is the same as the location of any of the ships on the *Ship List*. It accomplishes this by using the *List Contains Thing* command (found under the Variables section of the command palette) to see if the *Torpedo's* location is also listed among the locations recorded on the *Ship List.*

If the *Torpedo's* location and the location of a ship on the *Ship List* are the same, then an explosion can be used to register that a hit has occurred.

Other details can extend the depth of the game. For example, some ships are larger and cross more than one square, so a method for recording all of the squares that a ship occupies is needed. Also, a method for determining when all the ships on the list have been destroyed is needed in order to determine when to end the game. However, these are housekeeping details that can be worked out once the basic game procedures have been implemented and tested.