Button Press Timing Single Player

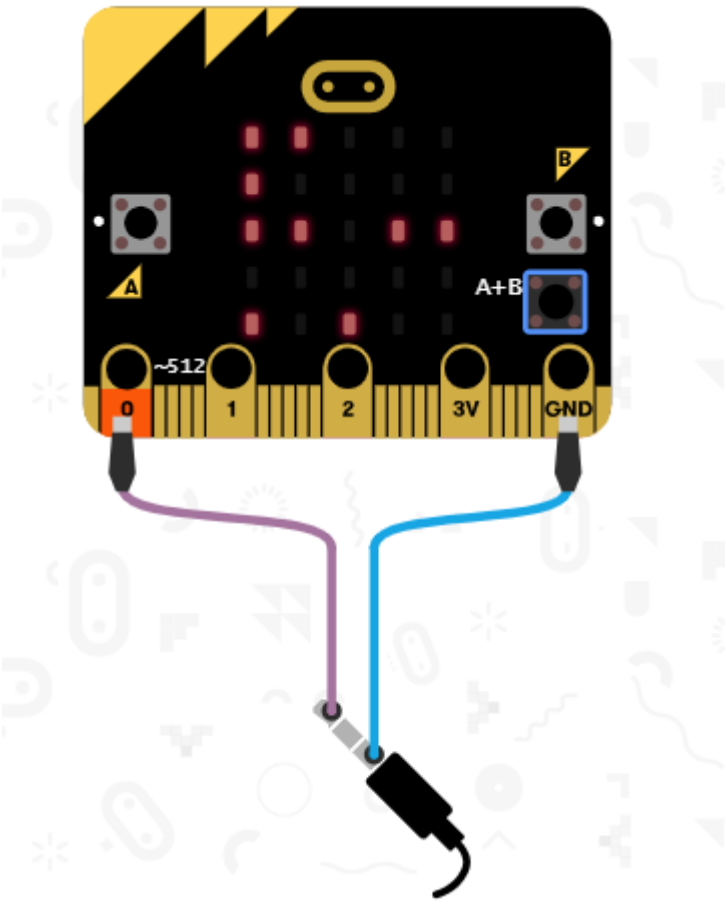
The following instructions will take you through the steps of creating a game where try to press the A and B buttons in time to blocks that fall from the sky. The longer you are able to stay alive the faster theblocks drop. This game makes use of the micro:bit function blocks and also contains extension points where you can develop the game further.

Game Instructions

The screen shot below shows the game being played in the micro:bit simulator. The 2 left columns and the 2 right columns of the LED screen are used to indicate which buttons to press. The middle column of the LED screen is used to help with the timing and rhythm of the game. The A, B or A+B buttons need to be pressed based on the LEDs that are lit up in the bottom row as follows:

- If no left or right LEDs are lit up, no button should be pressed.
- If a single left LED is lit up, the A button needs pressing once.
- If two left LEDs are lit up, the A button needs pressing twice.
- If a single right LED is lit up, the B button needs pressing once.
- If two right LEDs are lit up, the B button needs pressing twice.
- Left and right LEDs can be combined. Therefore, if two left LEDS and one right LED are lit up then the A button needs pressing twice and the B button needs pressing once.

Power is gained for getting the button presses correct and power is lost when the button presses are wrong. The game steadily gets faster as the player plays.



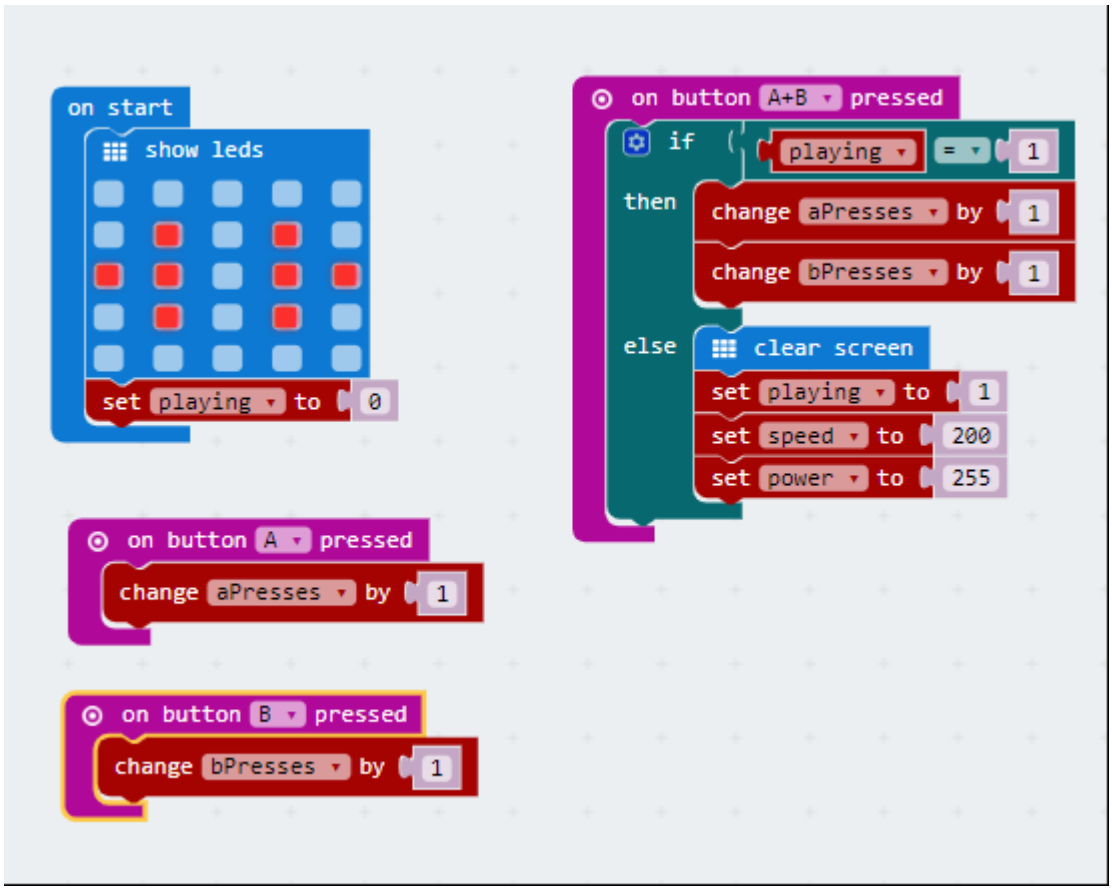
Step 1 – Create the start screen and the button press code

Starting the game will be achieved by pressing the A and B buttons together. Because the A+B buttons being pressed together are also used within the game, an if-the-else block is used to determine whether the game is starting or if an A and B button needs to be registered.

A variable called playing is used to control if the game is currently being played or not. 0 means not playing and 1 means playing. When the game first starts, we do not want it to start playing.

A variable called aPresses is used to record the number of A button presses. Similarly, a variable called bPresses is used to record the number of B button presses.

Two other variables are required to control the speed of the game and the power of the player.



Step 2 – The basic game loop

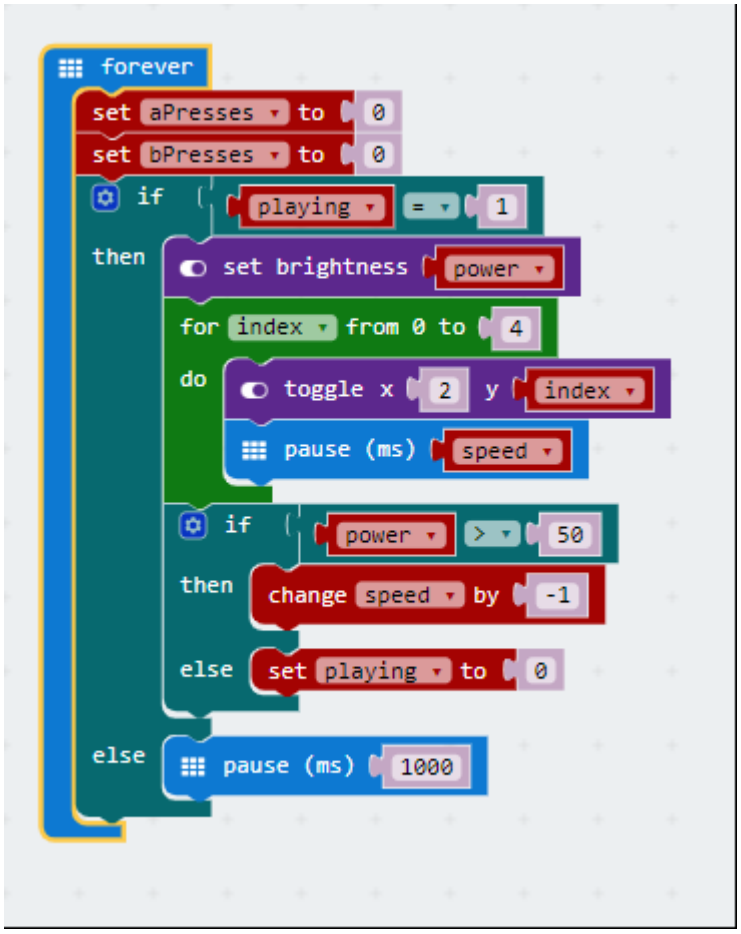
The basic game loop is contained within a forever loop. The start of the game loop resets the aPresses and bPresses variables to zero. The main code for the game loop should only execute it the game is currently playing (i.e. playing is set to 1).

In the game loop, the brightness of the LED display is set so that as the player loses power the LEDs get dimmer. There is also a for loop which controls the middle columns of LEDs. These LEDs are pulsed to help the player keep in time with the game.

Finally, there is an if-then-else block which adjust the speed of the game (making it faster) if the player has enough power left; otherwise the game ends by setting playing back to zero.

One this code is added, if A+B is pressed, then a pulsing middle columns should be seen.

NOTE: The LEDs are numbered in a grid. The columns are numbered zero to 4 from left to right (i.e. the left most column is zero and the right most column is 4). The rows are numbered zero to 4 also but from top to bottom (i.e. the top most row is zero and the bottom most row is 4).

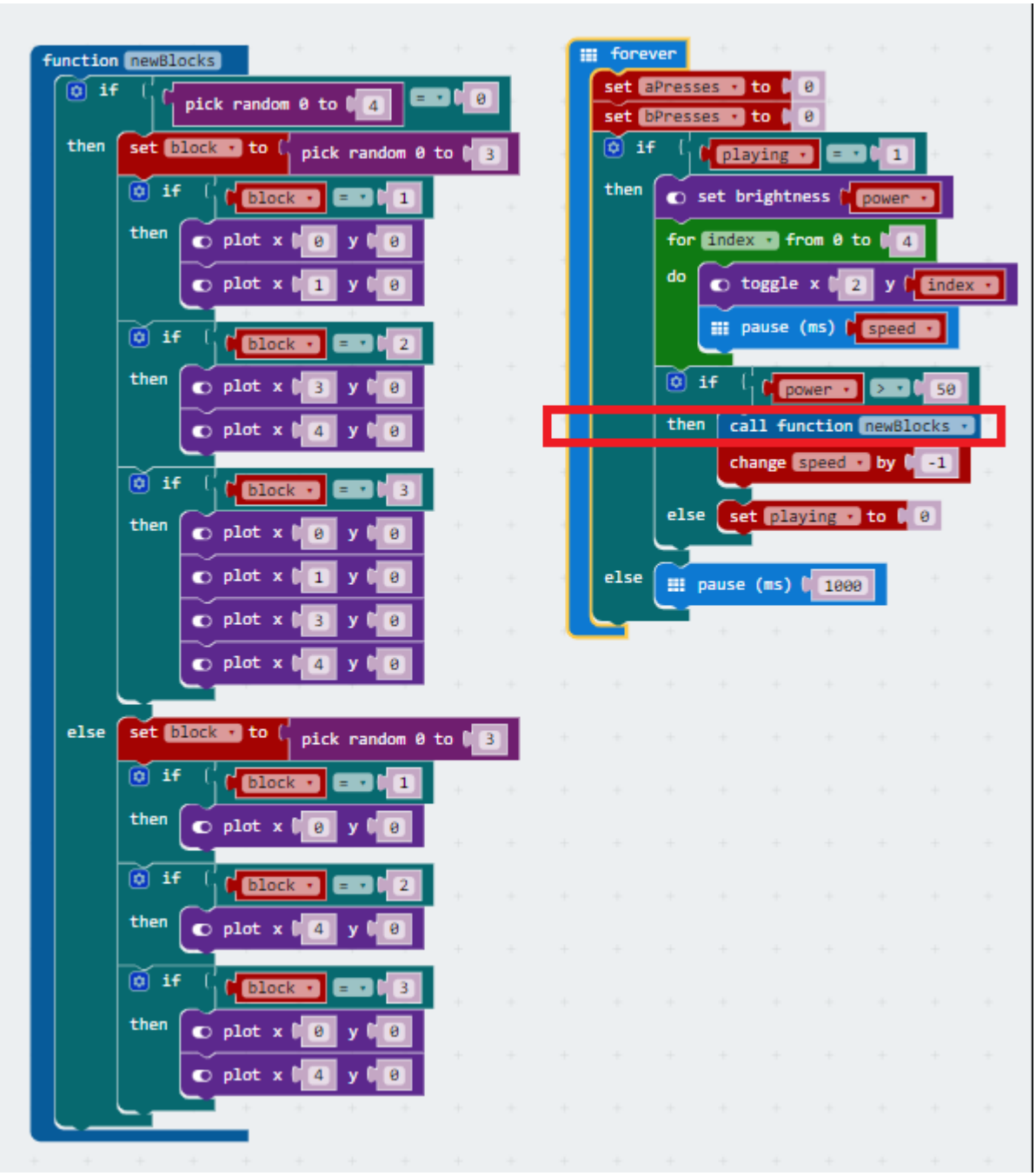


Step 3 – Creating new blocks to drop

From the Advanced blocks section, select Functions and then select “Make a function”. Call this new function newBlocks. This should create a new block within which you can add code and then call it from other places. Functions are a useful way to group together related code.

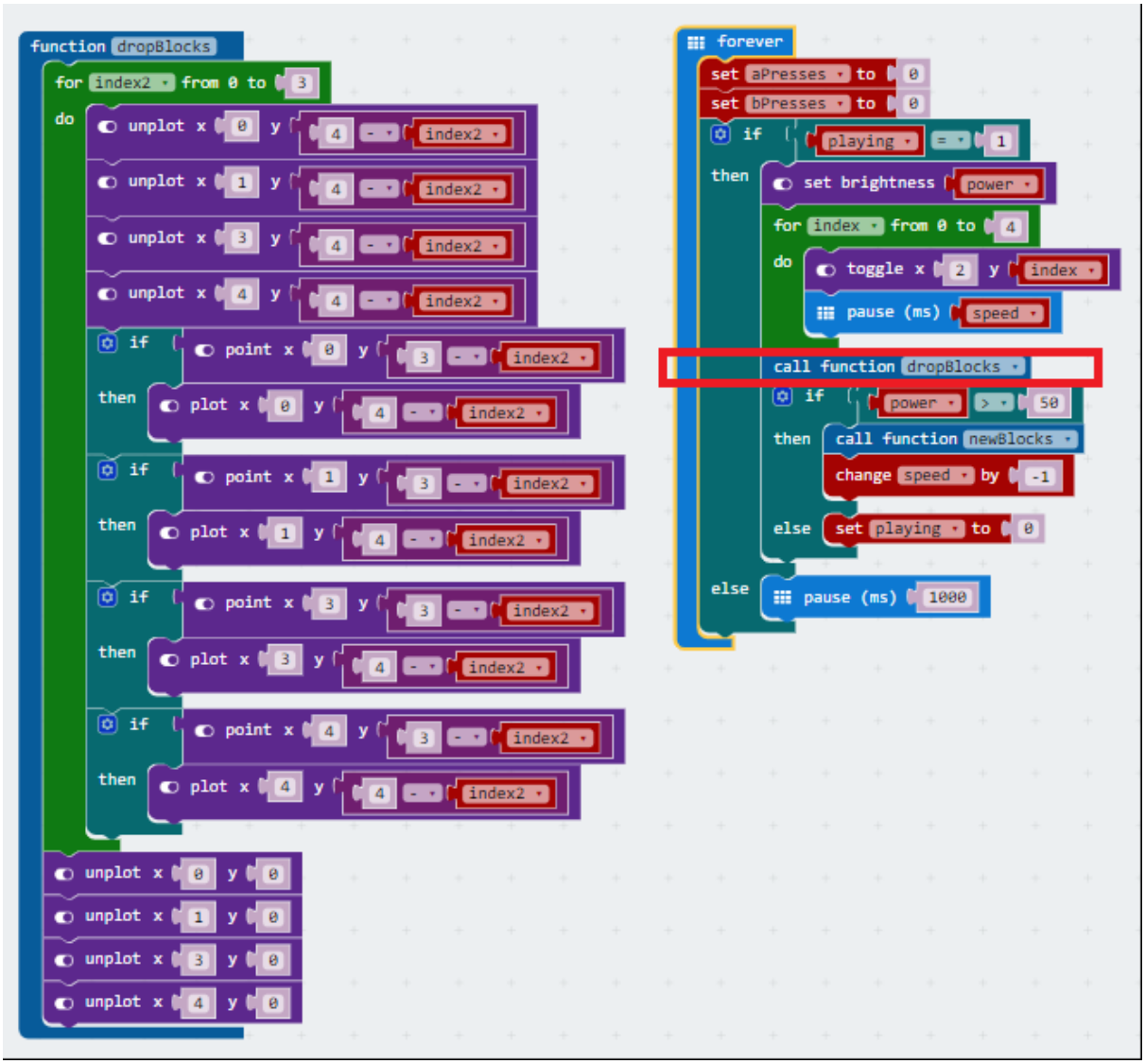
This new function is going to randomly populate some blocks along the top row of LEDs. The first random decision to make is whether to include a double button press or not. Because these are harder, we want them to happen less often and this is done by selecting to have a double only if the random number is zero. If it is 1, 2, 3 or 4, we stick to just an A, B, A+B or no blocks.

A second random selection is then made to determine which blocks to put down. Zero is for no blocks, 1 is for an A or AA, 2 is for a B or BB and 3 is for an A+B or an AA+BB combination. The newBlocks function needs to be called from the game loop (added code highlighted in red).



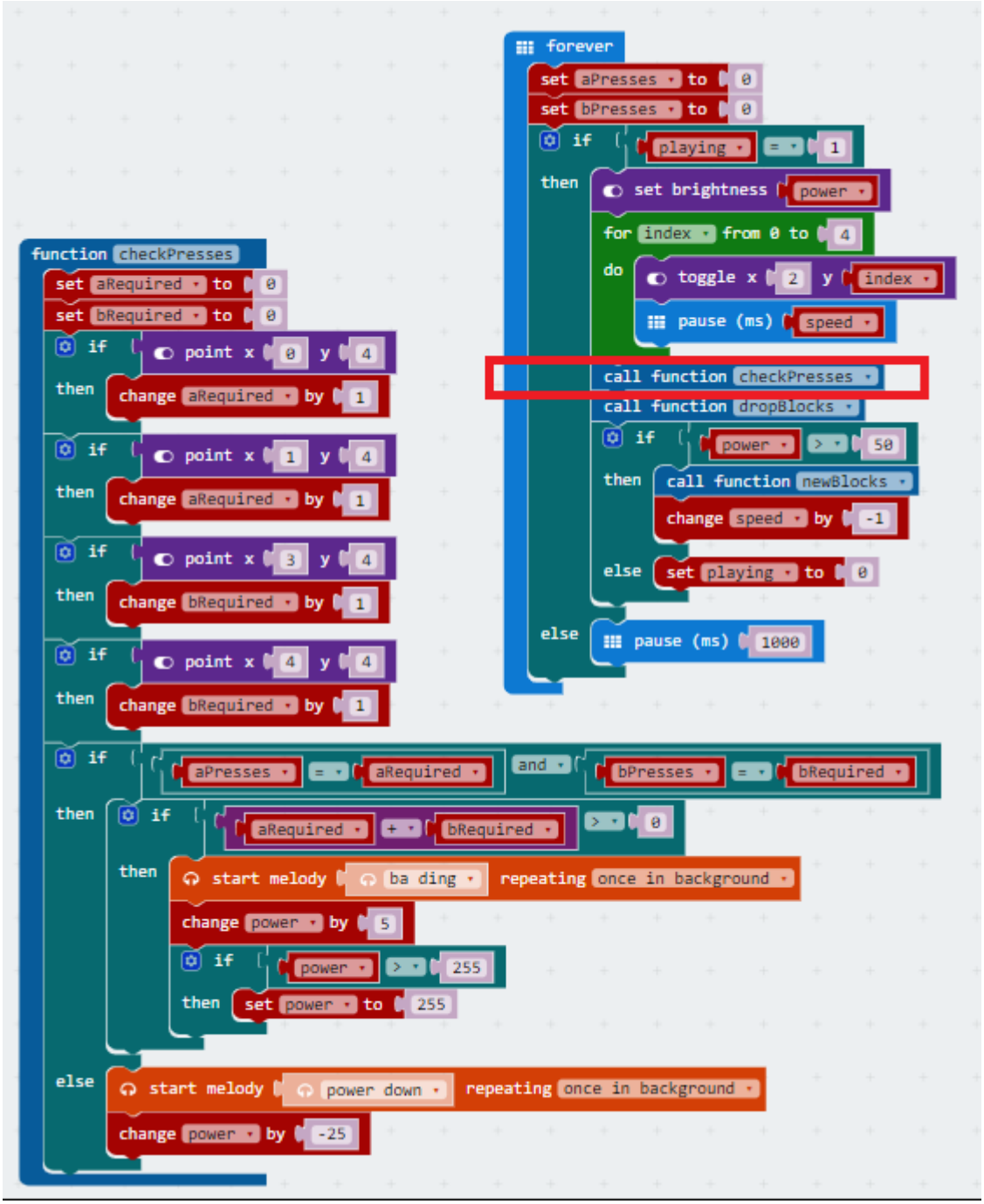
Step 4 – Making the blocks drop

If you run our game, you will see that the blocks do not drop. To drop the blocks, we will create a new functions called dropBlocks. This uses a loop to drop the rows down one by one starting at the last but one row and then working upwards. The top row is then cleared ready to add a new set of blocks to drop. The dropBlocks function also needs to be called from the game loop (added code highlighted in red). When the game is now run, the blocks drop down.



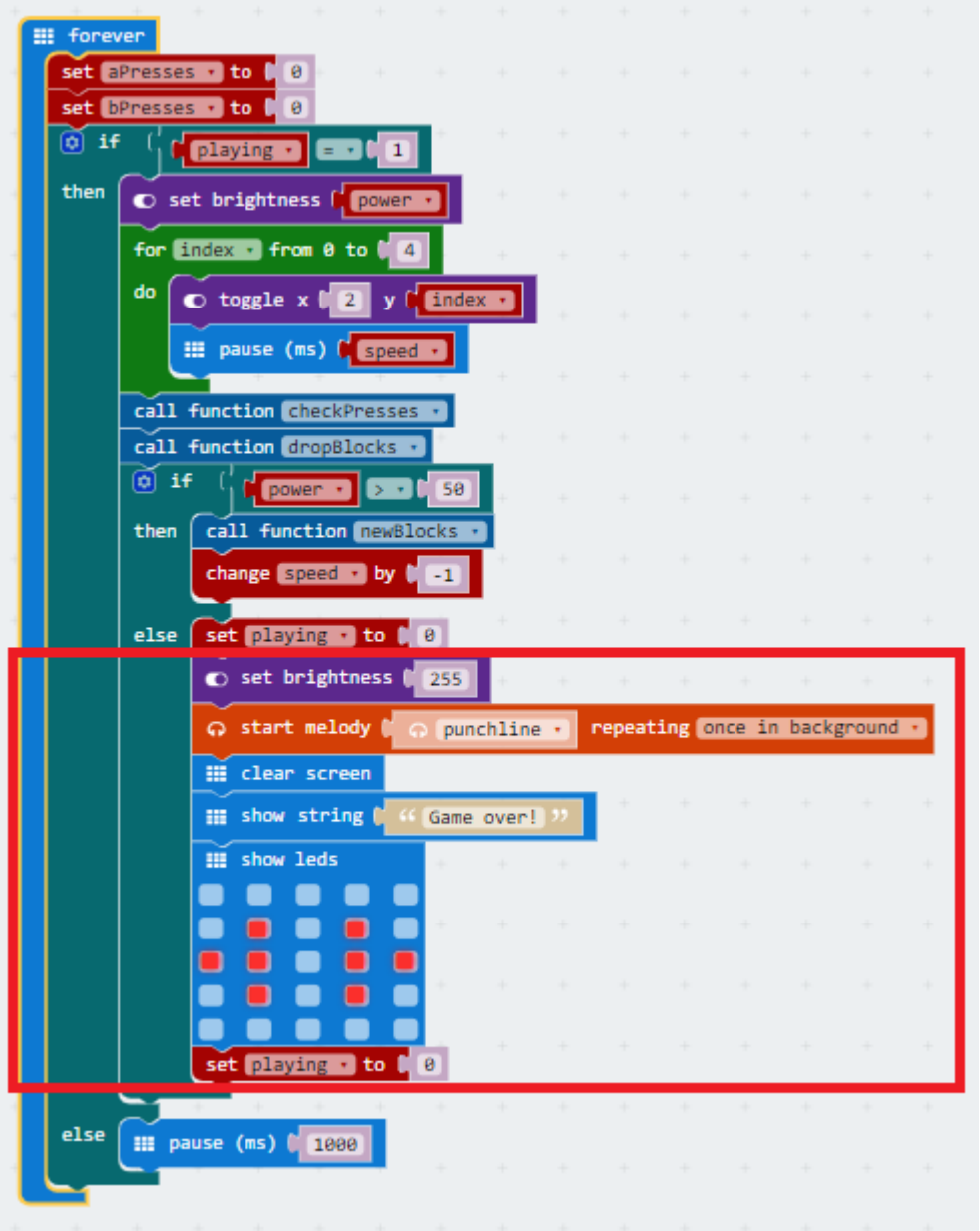
Step 5 – Checking the key presses

It is now time to check that the buttons the player presses are correct for the blocks that have dropped. To do this, another function will be created called checkPresses. Additionally, two new variables will be required: aRequired and bRequired. These two new variables are used to determine how may A or B button presses are expected for the blocks. These are calculated by inspecting the LEDs that are currently lit up on the bottom row. Once we know how many buttons presses of each type are required, we can compare them against the buttons that the player has pressed. If the correct number have been pressed, then we play a melody and increase the players power (up to a maximum power of 255). If the player has gotten the presses wrong, then a sad melody is played, and the players power is decreased. The checkPresses function also needs to be called from the game loop (added code highlighted in red).



Step 6 – Ending the game

All that remains now is to improve the game loop so that when the game ends, the player is informed and the game is restored back to the default starting place. The code that needs to be added to the game loop to achieve this is highlighted in red below.



Extending the game

There are many ways that this game can be extended. Just a few ideas are.

- Extend the block selection so that it also provides AA+B or A+BB combinations.
- Ad a score to the game to show to the player at the end. Score each successful button combination, perhaps less points for simple presses and more points for complex presses.
- Add a high score to the game and let the player know when they beat it.
- Make the game start with easier combinations (just A, B or A+B) and only introduce the more complicated combinations with AA and BB after a certain point.
- Can you make this a 2-player face off game? Reducing you opponents power with correct combinations?