**Mattel Electronics Aquarius - 10Line Bomb Catcher**

R.T.Templeman 2020

***The Breakdown – How does the game work?***

**1?chr$(11):poke12369,32:h=0:j=12408:b=13268:d$=".":t=0:e=0:bb=b+1:bc=b-1**

**2ifh=0thenh=j+int(rnd(1)\*400):o=12289:p=12307:pokeh-80,25:ift>350thene=1**

**3pokeh,20:pokeh+40,194:pokeh-40,32:h=h+40:ifh=borh=bborh=bcthen10**

**4ifh>13247thenpokeh,32:pokeh-40,32:h=0:forp=0to78:pset(p,65):next**

**5foro=otop:reada$:pokeo,asc(a$):next:k$=inkey$:ifk$<>"z"andk$<>"."then7**

**6d$=k$:pokeh-9,46:ife=1then?"Bombs Caught: "s:?"Play Again?":inputa$:run**

**7ifd$="z"andb>13249thenpokeb,32:pokeb+1,32:b=b-1:pokeb,174:pokeb-1,173**

**8ifd$="."andb<13286thenpokeb,32:pokeb-1,32:b=b+1:pokeb,173:pokeb+1,174**

**9t=t+1:restore:goto2:data" ",B,O,M,B," ",C,A,T,C,H,E,R," ",2,0,2,0," "**

**10pokeh,216:forw=1to200:sound(20,l):next:s=s+1:pokeh-40,32:h=0:goto2**

Complete Source Code



*Breakdown – Each bullet point is a : (colon) in the source code*

**LINE 1**

* Clear the screen
* Clear the top left cursor position with a space. This is to remove the cursor from the play area
* Set variable H to zero. H is used for the starting position of the bomb and will be the current location of the bomb when it starts to move
* Set variable J to 12408. J is used as the current fixed screen location for the initial bomb drop, this will be used as part of a random element, and this fixed location is only used to form part of the randomisation element for all bomb drops
* Set variable B to 13268. B is used as the starting location for your turret. Near the bottom of the screen
* Set string variable D$ to ”.” this is used as the initial key press for when the game begins. This ensures that you start moving straight away and not have to wait for a key press to start drawing your turret graphic
* Set variable T to zero. T is used as a counter. Every time you move your turret, the counter goes up. I am using this as a method of timing each game. After so many T increases, the game ends
* Set variable E to zero. E is used as a stop clause. In other words, when an event occurs, set this variable and then check for its value. I am using it to know when the game has ended
* Set the variable BB, and make it the value of B (bomb drop start screen location) and add 1 to it
* Set the variable BC, and make it the value of B (bomb drop start screen location) and remove 1 from it. These are used to store the position one space to the right and one space to the left of your turret. This will be used to make sure that you can catch the bomb when you are moving

**LINE 2**

* Check to see if H (current bomb location) is currently at zero, and if so then reallocate H to be a random integer based on J (screen location 12328) and a random number between 1 and 400. This will ensure that the bomb starts from a random location every time the main game loops
* Set variable O to 12289. This is the start screen location for the game name banner along the top of the screen (BOMB CATCHER 2020)
* Set variable P to 12307. This is the last screen location for the game name banner along the top of the screen, this is based on the number of characters that you want to display outside the main screen play area
* Place the baddie space invader (ASCII 25) 2 places (80 characters) above the position of the bomb drop (H) so it looks like he is dropping the bomb
* Check to see if T (the number of times that your turret has moved) is greater than 350, if it is, then set the variable NE to 1. This will be used as a marker to signify that the game has ended. If the variable T is not greater than 350, then the NE variable is not set (is false), and the code continues

**LINE 3**

* Push the value of 20, in to memory location H (current bomb position). 20 is the character graphic for the top half of the bomb
* Push the value of 194, in to memory location H plus 40 (bomb location one position further down the screen). 194 is the character graphic for the bottom half of the bomb
* Push the value of 32 (a space) on to the bomb location (H) minus 40 (the position above the bomb). This will give the impression of movement, as the previous part of the bomb will be overwritten by a space
* Add 40 (one screen position down) to H (current bomb position), thus making it look like the bomb has moved down the screen by one position
* Check to see if H (current bomb position) is equal to B (current turret location) or equal to BB (turret location plus 1 position) or equal to BC (turret location minus 1 position). This is to ensure that you can catch the bomb, even if you are moving either right or left at the time the bomb reaches you. If these conditions are true (you caught the bomb!), then jump to line 10. If they are not true (false, you missed the bomb!) then the code will continue to the next line

**LINE 4**

* If H (current bomb location) has crossed the threshold for the bottom of the play area (is greater than 13247) then replace the current bomb graphic with that of a space (32). Once the bomb gets to the bottom of the screen, and you haven’t caught it, then clear its graphic, ready for another bomb drop
* Push the value of 32 (a space) in to H (current bomb location) minus 40 (the space above the H location). This is because the bomb graphic is made up of 2 graphic characters (top and bottom), and we need to clear both graphics
* As the bomb has now reached the bottom of the play area, reset the location (H) back to zero
* Create a small loop with the variable P from zero to 78 (the graphic based length of the screen play area)
* Draw a line based on the length of P near the bottom of the play area (65), but above your turret
* Close the loop for P

**LINE 5**

* Create a loop with the variable O (12289) to P (12307). These are the memory locations for the game text banner at the top of the screen (BOMB CATCHER 2020)
* Read the DATA statement data on line 9, and place the contents within the string variable A$
* Push the ASCII value of A$ in to the memory location of O through to P, thus displaying the banner text above the play area (BOMB CATCHER 2020)
* Wait for a key to be pressed, and place the result in to the string variable K$
* If the “z” key or the “.” Key is pressed, then jump to line 7. All other keys will be ignored and the code will continue to the next line

**LINE 6**

* Create a string variable D$ and place the current contents of K$ (current key press) in to it. This is to allow me to change the contents of D$ without actually changing the key press itself
* Push the value of 46 (a star – actually a .) in to H (current bomb location) minus 9. This is to offset the stars, so that they are not in exactly the same place as the bomb
* If the variable E (end of game marker) is equal to 1, then stop the game play and display your score (S) which will be the number of bombs that you have caught. If E does not equal 1, then the game has not finished yet, so continue to the next line
* Ask the player if they want to play again
* Wait (halt) for return to be pressed (A$)
* If the return key is pressed, then run the game from the start (run)

**LINE 7**

* If the string variable D$ (keyboard input) equals “z” (you want to move left) and your turret is traveling along the screen and is within the play area (greater than 13249), then push a space (32) in to memory location B (your current turret location). Then add 1 to B (your turret location) and do the same thing (push a space in to this memory location). This is to give the illusion of movement. Then re-issue the B (your turret location) variable with minus 1 (your turret has moved 1 position to the left)
* Push the value 174 (the left half of your turret) in to memory location B (your turret location)
* Push the value 173 (the right half of your turret) in to memory location B (your turret location) minus 1. This is to show your complete 2 character graphic for your turret as it moves left

**LINE 8**

* If the string variable D$ (keyboard input) equals “.” (you want to move right) and your turret is traveling along the screen and is within the play area (less than 13286), then push a space (32) in to memory location B (your turret location). Then minus 1 to B (your turret location) and do the same thing (push a space in to this memory location). This is to give the illusion of movement. Then re-issue the B (your turret location) variable with plus 1 (your turret has moved 1 position to the right)
* Push the value 173 (the right half of your turret) in to memory location B (your turret location)
* Push the value 174 (the left half of your turret) in to memory location B (your turret location) plus 1. This is to show your complete 2 character graphic for your turret as it moves right

**LINE 9**

* Add one to the variable T (game timer, turret movement tracker)
* Clear the contents of the READ/DATA statement. Reading the same values from a DATA statement more than once causes the Aquarius to display an out of memory error. To prevent this, I clear the contents every loop using the ‘restore’ command
* Jump to line 2. This closes the main game loop
* Define the game banner text. This is issued as a data statement, so that it can be pushed to the border, outside the play area. This means that the main game loop does not interact with the banner, nor does it overwrite it as its technical outside the main screen. DATA statements can be issued anywhere within the code, even after a GOTO

**LINE 10**

* Push to location H (the current bomb location) the value of 216 (you caught the bomb graphic)
* Create a loop with the variable W between 1 and 200
* Play a sound (you caught the bomb!)
* Increase your score (S) by 1
* Push a space (32) in to location H (current bomb location). This is to clear the caught bomb once the sound has been played
* Reset the bomb location (H) to zero
* Jump to line 2. The next round, with the next bomb… here we go again 😊