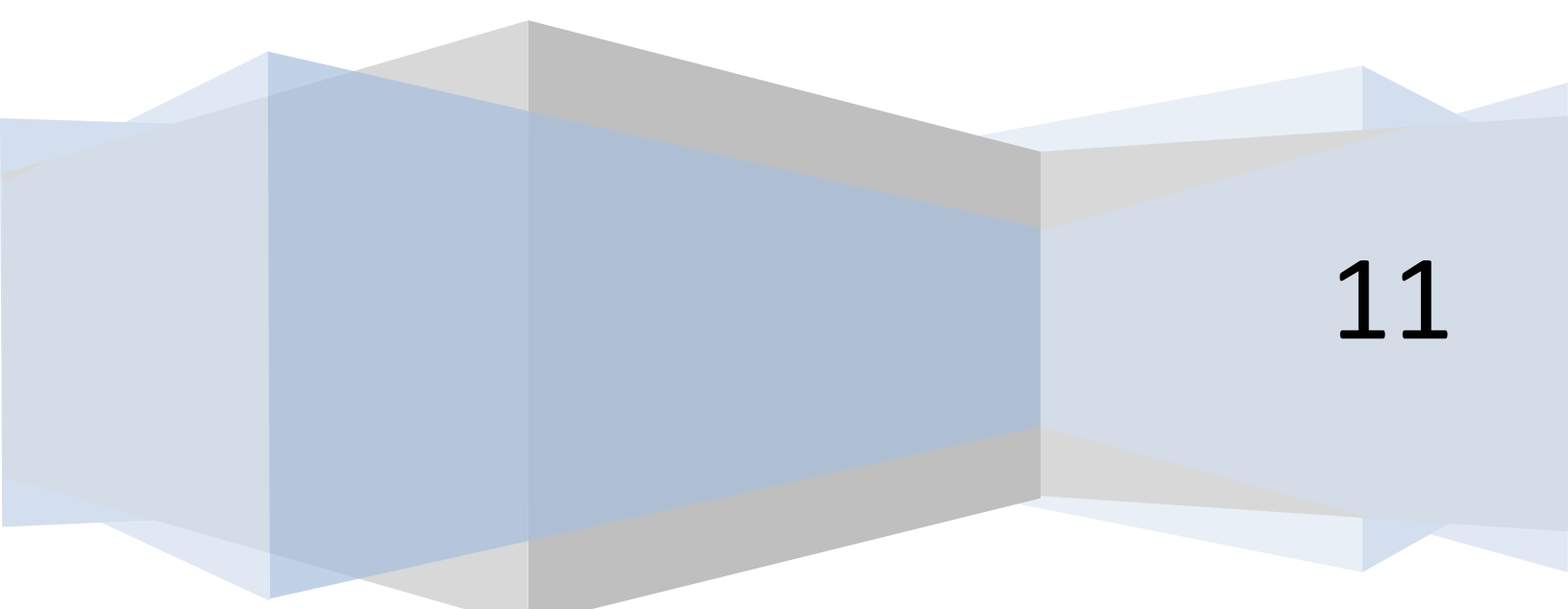


Dr. Allen

# Bombberman 1.0 in Excel

ISYS 540 Final Project

Joshua Duvall



11

# Bombberman 1.0 for Excel

---

## Executive Summary

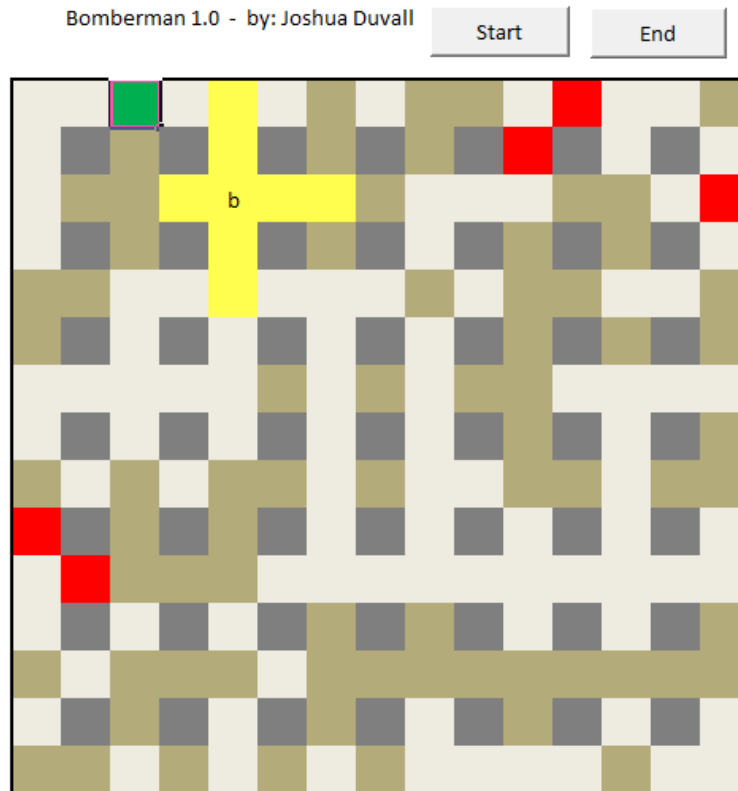
Bombberman is a game where the player tries to navigate a labyrinth comprised of pillars and rocks. The player has the ability to place bombs that blow up the rocks and open up passage ways. Within the labyrinth there are enemies that are trying to get to the players location. If the enemies touch the player, the player dies. However, the player can lay bombs down to blow up the enemies. In order to win, the player must blow up all of the enemies.

I implemented my version of this game using an Excel worksheet. On the work sheet the playing field is laid out with pillars being gray squares, rocks being brown squares, the player as a green square, and enemies being red squares. The player navigates the playing field using the arrow keys and places a bomb by pressing the letter b. The bomb is activated and a timer starts as soon as the player leaves the square where he just placed the bomb. After two seconds the bomb “explodes” cause fire to extend out in all 4 directions, destroying any blocks or enemies in its path. The player, however, needs to be careful not to touch the fire or the enemies or he will die and it will be game over.

The player has the option to customize the playing field prior to starting the game. He can choose the number of rocks to be randomly placed on the playing field (from 25-150) and the number of enemies that will be randomly placed on the playing field (from 1-10). The game uses the Application.onTime event to trigger the movement of the enemies and the countdown on the bombs. This onTime event is one of the essential aspects of the game as it triggers nearly all action. The player also receives points based on the amount of time it took to kill all the enemies and points for the number of enemies killed.

## Implementation

### Game Board



The game board is where all the action takes place in the game. This game board is implemented in the “Game” worksheet and it is initialized when the user presses the Start button. When the board is initialized, everything on the board is drawn. There are five very important arrays: pillars, possibleRocks, rocks, enemies, and fires. The pillars array contains all of the pillar address, stored as strings. These pillars are filled in with a gray color when the game starts. The possibleRocks array contains all of the possible locations that a rock can be placed. Based upon the number of rocks that the user specifies, the rocks array is populated with the specified number of unique rock locations. These rocks are then filled in with brown. The enemies array is populated with the specified number of enemy locations which are randomly placed based on the remaining open locations (not rocks or pillars). These enemies are then filled in with red. The fires array is populated with the locations that fire currently occupies after a bomb explodes and once the fire is removed from the screen the array is emptied.

## Modules and Methods

The project is comprised of three separate modules: `GamePlay`, `BombModule`, and `EnemyModule`. And one worksheet function within the `Game` worksheet.

### GamePlay

This module is responsible for the initialization of the game discussed in the `Game Board` section. It keeps track of the score and most importantly includes the implementation of the timer. Apart from the methods to set up the playing field (rocks, pillars, etc.) this module also contains the following important methods:

- `isPillar`, `isWall`, `isRock`: These methods take in a range object and return a Boolean value if the range that was passed in is a pillar, wall or rock respectively
- `startTimer`, `stopTimer`, `Timer`: These methods are responsible for the timer and calling the method in the enemy module to move the enemies.
- `endGame`: This method stops the timer and ends the game
- `gameOver`: This method is called when the player is hit by an enemy or by the fire from an exploded bomb.
- `gameOverWin`: This method is called when the player destroys all the enemies.
- `moveFrom`: This method is called when the player moves. It updates the `previousLocation` variable and changes the color of the previous cell.
- `moveTo`: This method is called when the player moves. It updates the `currentLocation` variable and changes the new cell to green representing the new location of the player.

### BombModule

The `bombModule` is responsible for handling the bomb, the bomb timers, and the fire that the bomb creates when the bomb is exploded. It also handles the removing of the bomb and the fire from the playing field. It includes the following important methods:

- `setBomb`: This method starts the timer for 2 seconds that triggers the exploding of the bomb.
- `blowUpBomb`: This method blows up the bomb and spreads the fire in all 4 directions to cells out. This method checks to see if there are rocks, pillars or walls in the path of the fire. If there is the fire will blow up the first rock it comes in contact with is in the first cell next to the bomb, it will not continue on to the second cell. Also, pillars and walls stop the fire. The locations that the fire is located are put into the "fires" array. A timer is started and set for 1 second that will remove the fire.
- `removeAllFire`, `removeBomb`: These methods are responsible for removing the fire and bomb from the playing field and removing the locations of the fire from the fires array.
- `isFire`: This method returns true if the passed in range is currently in the fires array. This method is called to see if the player walks on the fire or an enemy is being killed by the fire.

## EnemyModule

This module is responsible for moving the enemies around the game board as they try to kill the player. It is also responsible for setting up the locations and the number of enemies that will be on the playing field. The module contains the following important methods:

- **setUpEnemies, setUpEnemyLocations:** These methods are responsible for setting up the number of enemies and their initial locations on the board to start. They are always placed on a square that is not occupied by a rock or pillar to start.
- **moveEnemies:** This method is one of the most important in the game and the most frequently called. Every second this method is called to move every one of the enemies on the playing field. The method iterates through the enemies array and for each enemy location it will pick a random number from 1-4 and will move the enemy up, down, left, or right accordingly by offsetting the enemy's current location. If the new location is going to be a wall, rock, or pillar, the enemy will move the opposite direction. If that location is occupied it will try moving in one of the other two directions. This way, every enemy, nearly every time will move at least one cell in one of the four directions. This method also calls the method **checkFireForEnemies** to see if fire kills the enemy. Also, the method checks to see if the enemy moved onto the player's current location thus killing the player and ending the game. Also the method checks to see if all the enemies have been killed, thus ending the game in a win for the player.
- **checkFireForEnemies:** This method checks to see if any of the enemies have just walked onto fire. If they have they are "killed" and removed from the enemies array.
- **checkForPlayer:** This method checks to see if an enemy has walked into the player.
- **checkForWin:** This method checks to see if all the enemies have been killed, thus ending the game in a win.
- **isEnemy:** This method checks to see if a cell currently contains an enemy.

## Game Worksheet

The one method in this worksheet is called on the **Worksheet\_SelectionChange** event. This event is triggered when the player moves around the playing field using the arrow keys on the keyboard.

- **Worksheet\_SelectionChange:** This method first checks to see if the "target", or new location that the user is moving to, is fire or is an enemy. If it is, the player is "killed" and the game ends. Then it checks to see if the new cell is a rock, a wall, or a pillar. If it is, the player is returned to the previous location. This makes it seem as if the user cannot move into the walls, rocks, or pillars. If the player moves to a cell not occupied by any of the previously mentioned items, the previously discussed **moveFrom** and **moveTo** methods are called. The method then checks to see if the player left a bomb (press the "b" key) in the previous cell, if he did the "setBomb" method is called. This method also only allows the user to leave a letter "b" in the cell.

## Conceptual Difficulties

While working on this project I ran into a few roadblocks that forced me to think about how I would overcome them.

The first of these roadblocks came when I realized that there was no Timer object available in VBA. I was planning on using a timer object to control a few different things, like bombs and enemies, in the game. After some research I came across the Application.onTime event that will call a method after a specified number of seconds. So what I did to create a “pseudo timer” was have the onTime event call a method called timer that ran certain functionality and then after the code was run in Timer I would call the method that started the onTime event again. Thus creating an infinite loop that could only be broken by stopping the onTime event from continuously calling itself with these two methods. I stop this timer only with the endGame method is called.

Another roadblock that I came across was that I could not figure out how to do multiple enemies and bombs at the same time. As VBA is more procedural based instead of completely object oriented like other modern languages, I was having a hard time visually conceptualizing how I would have multiple enemies moving around. Because of the timers used with the bombs and the difficulty I had with timers getting reset when I would place more than one bomb at once on the field, I decided to only allow one bomb to be on the field at a time. But I was eventually able to allow for any number of enemies to be placed on the field at a time by using an array that contained just the current locations of the enemies. After much thought into trying to get a bomb class to work, I decided that I would just iterate through this enemies array and change the location of each bomb, one at a time every time the “pseudo timer” would “click”.

## Conclusion

This project was an enjoyable one for me as it presented a number of difficulties that I had to think my way through. It was difficult to come up with solutions in a more procedural based language when I am more familiar with more object oriented languages like Java and Flex. The project did a little more than 30 hours to complete. I do not consider this project 100% complete as I would like to continue making the game better and I would like to add features such as allowing the enemies to drop bombs, and maybe move with the intent to kill the player instead of just moving a random direction. Overall, this was a very enjoyable project