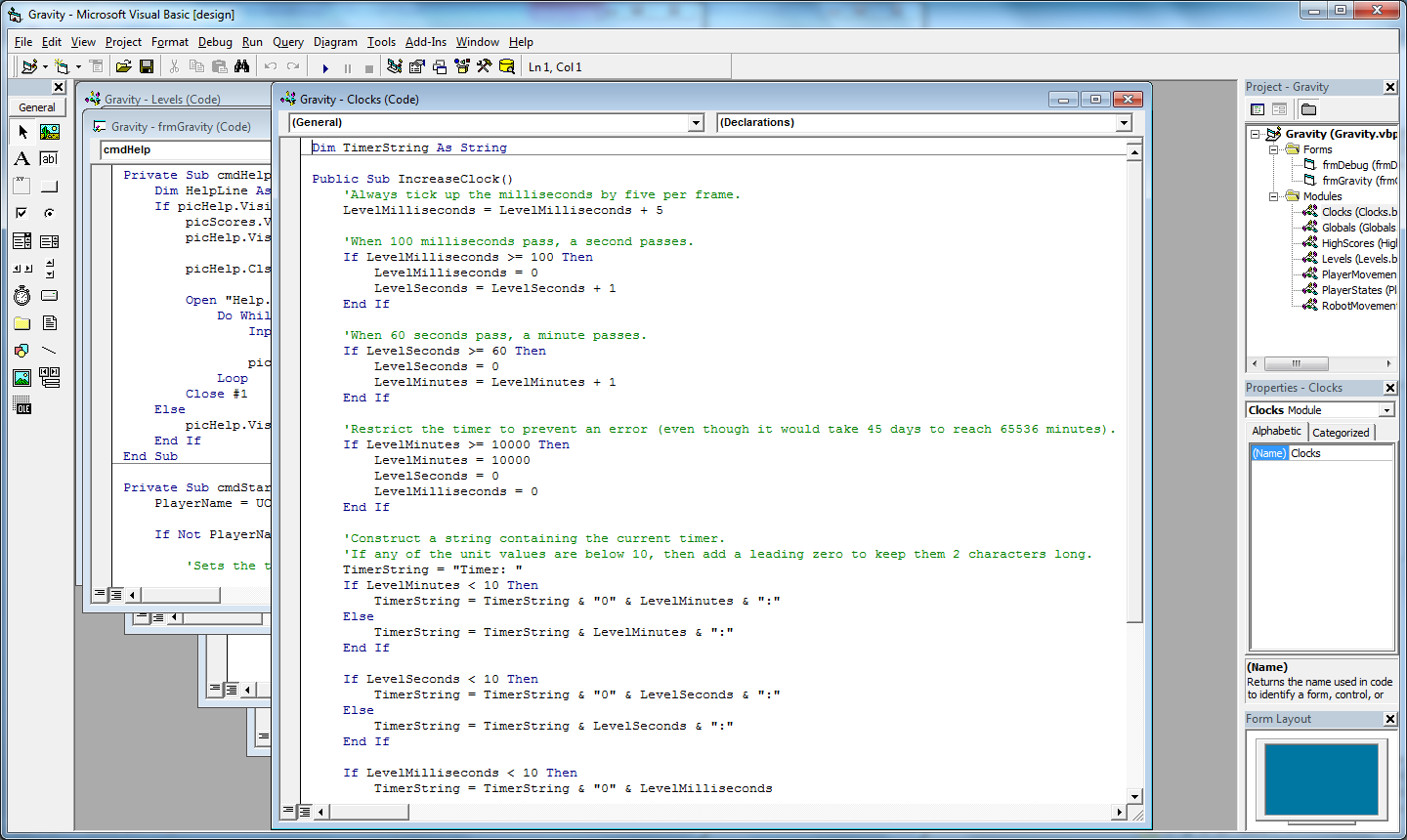
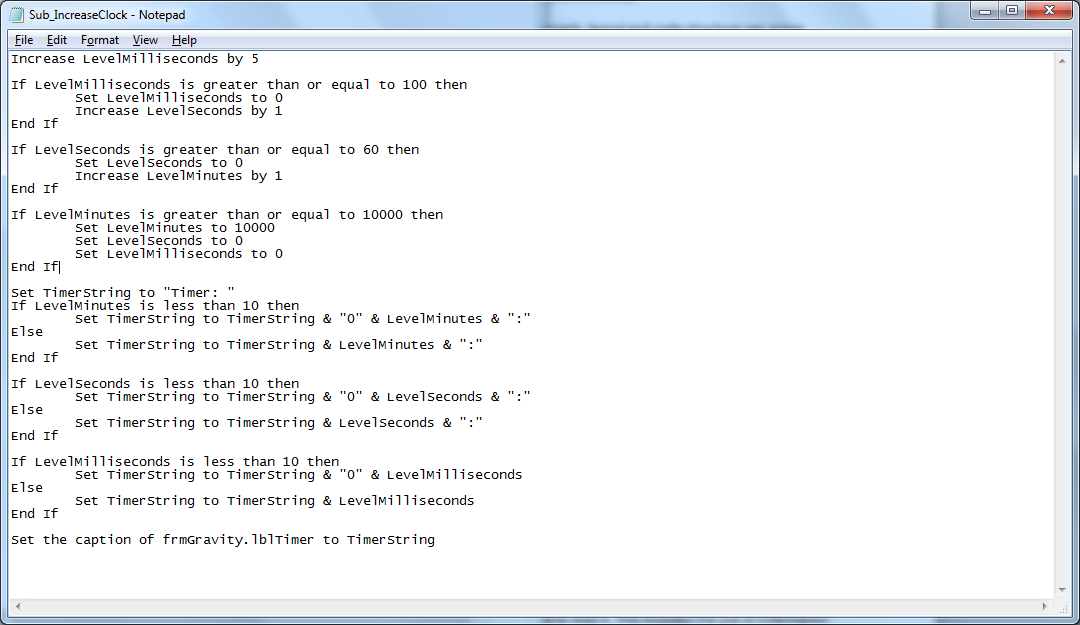
**Benefits of Structure and Design**

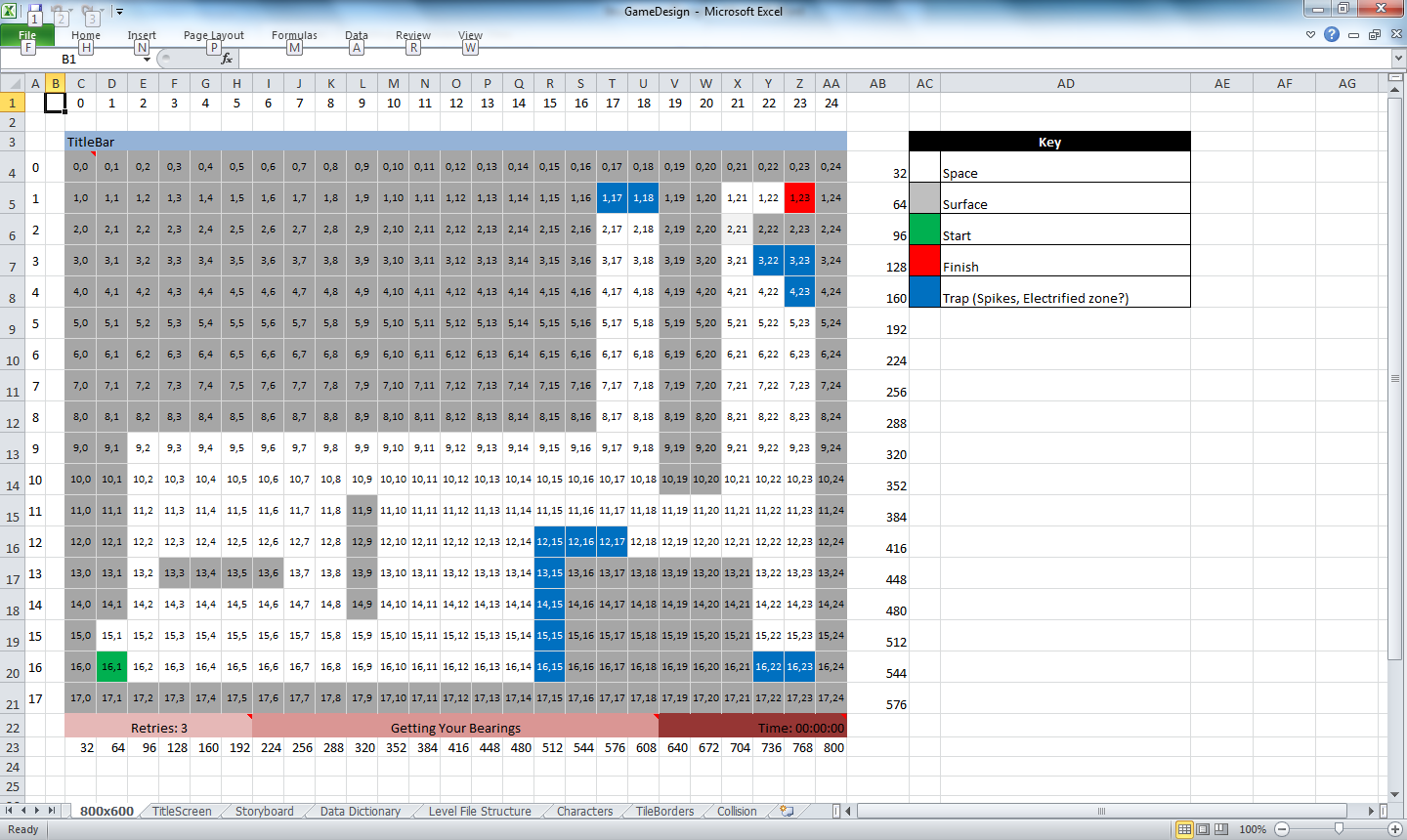
Designing a game with a specific and consistent approach, layout and code structure can prove beneficial to those who would take the reins of any program’s continued development from the prior developer, and ease their understanding of the program’s code without spending a significant amount of time studying it.

Such a design can include the use of pseudocode, as well as comments written within the code itself. Pseudocode is essentially a plain-English representation of the more complex code that forms the program, and can represent the seemingly nonsensical syntax in a manner that can be easily understood. Comments, meanwhile, exist within the code, are ignored by the compiler or interpreter and can provide brief explanations of the functionality of each block of code, sparing both new and existing developers the time that would be spent trying to figure out what a piece of code does.

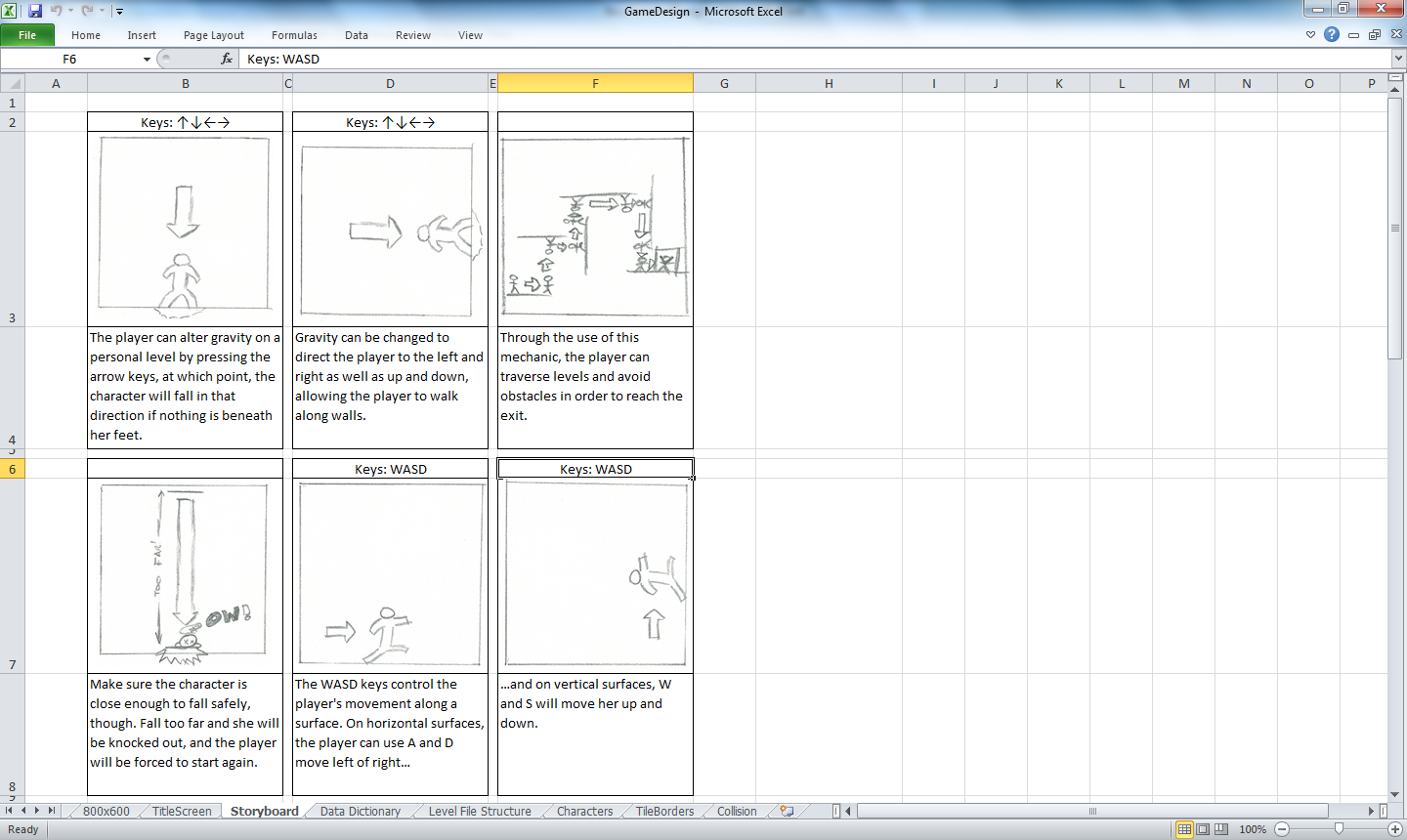


*Examples of pseudocode (left) and the equivalent Visual Basic code, with comments (right). The code in both examples are one and the same, but the pseudocode is presented in plain English rather than mathematical formulae that dominates the actual program code. The comments, meanwhile, offer a brief description of what each block of code is supposed to achieve.*

Meanwhile, visual designs such as form layouts and storyboards can show a new developer the intended functionality of a program without having to first read and understand the code. Storyboards serve to describe the functionality of a program and its reactions to specific events through the use of hand-drawn images and descriptive captions, while drawings and sketches of display or form layouts, along with appropriate labels for their individual elements, can serve to show what how a program should appear to users.

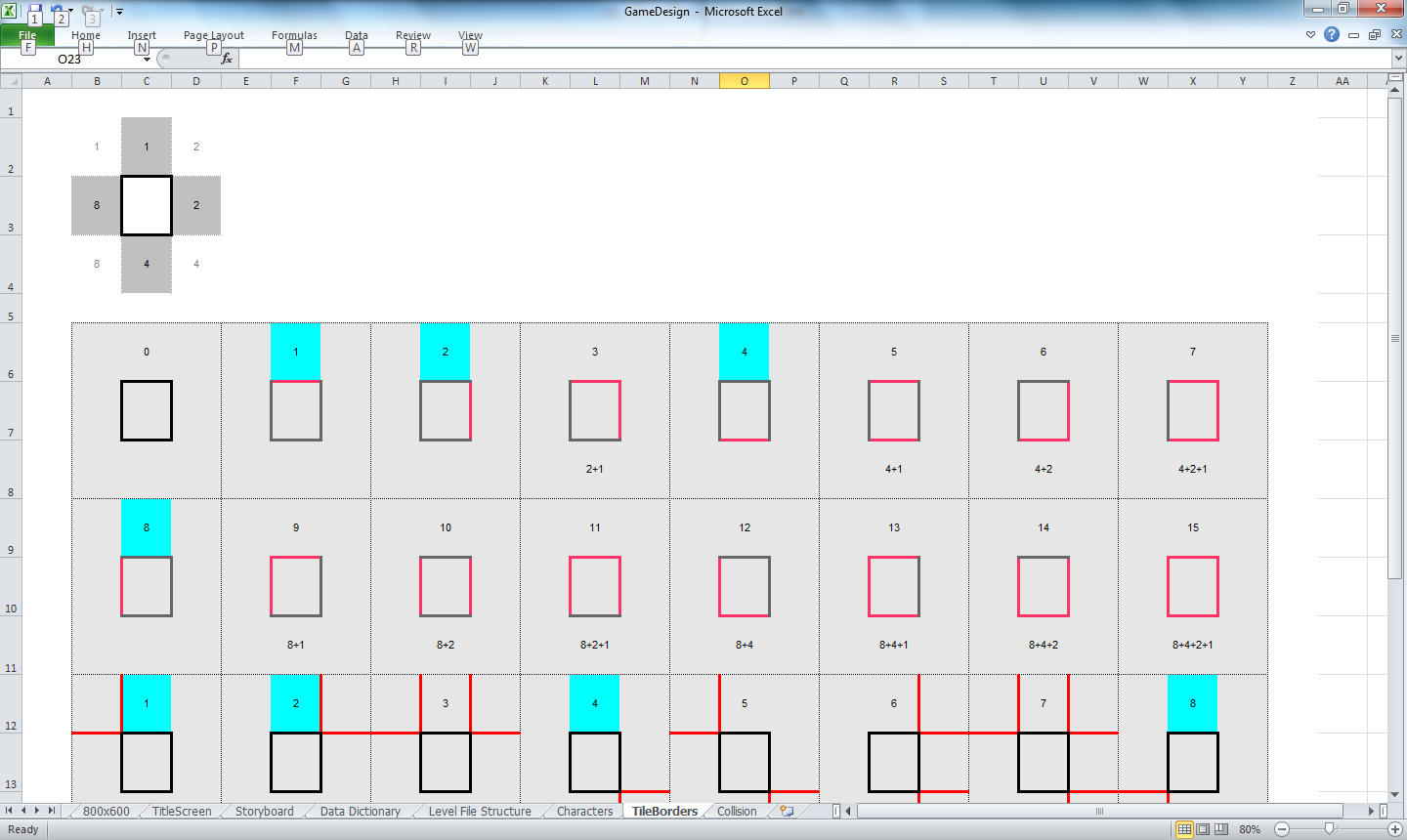


*A design for the layout of a game, which includes colour coding for each tile that makes up a level and their grid references. Because Excel was used for this design document, features such as cell comments were used to include element attributes that would display when hovered over.*



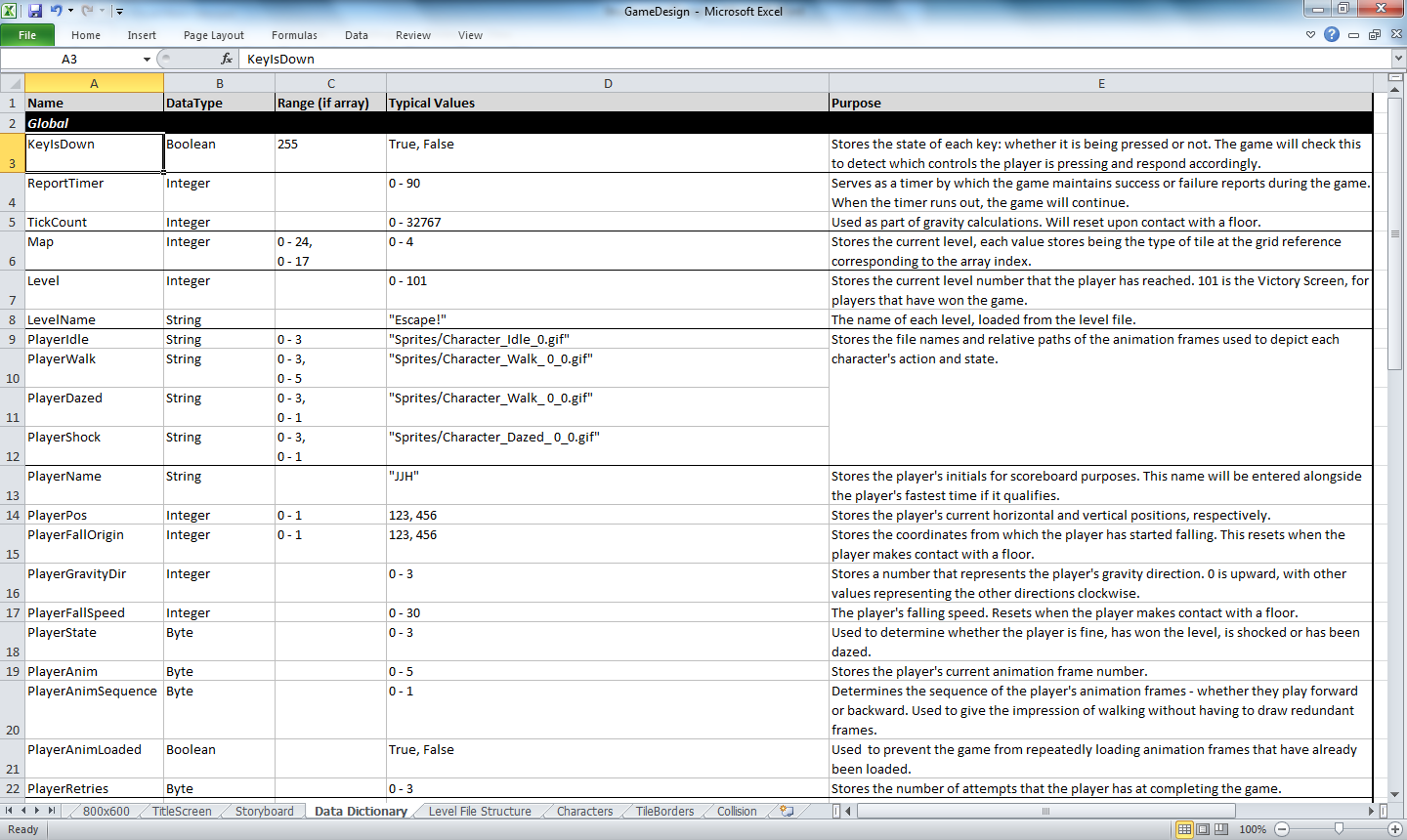
*Part of the storyboard used to describe the game’s reactions to events such as key presses, and how a player would use this functionality to succeed.*

A program’s design document can also include designs for functionality that may be, in a sense, “invisible” to users, such as internal calculations. Such designs can include graphical representations of those calculations in order to describe them in a manner that would have otherwise been difficult to explain verbally.



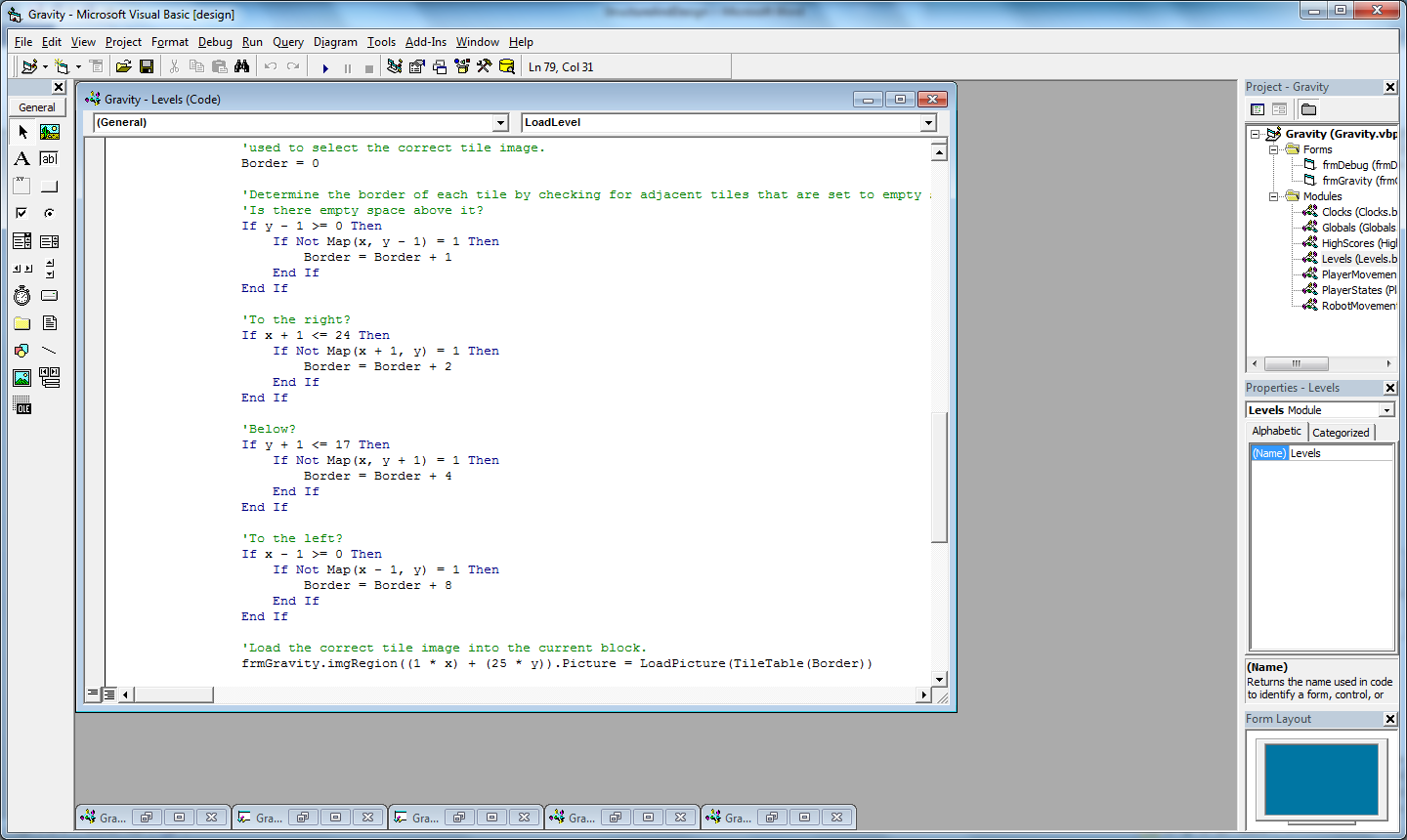
*An example of such a design, used to determine the unseen calculation of the borders that surround solid wall tiles, based on which adjacent tiles are empty space. There are 16 different border combinations, and the values assigned to each side are added together to find the number corresponding to the necessary combination.*

One of the key aspects of a program’s design is the creation of a data dictionary. This is essentially a list of the many variables that a program may use over the course of its runtime, and describes each of their data types, typical values and formats, its scope within the program (whether it’s global or local to a specific module or procedure) and a brief description of the variable’s purpose. This can allow a developer to understand the data that is currently used by the program and avoid the creation of “duplicate” variables in the process of its expansion.



*A portion of a data dictionary. This one includes the data types, ranges (if they are arrays rather than single variables), their typical values and their purposes. This particular data dictionary separates variables by their scope, which forms the white-on-black sub-heading at the top of each group.*

In addition to the design itself, it is of the utmost importance that the code itself maintains a consistent structure that can allow developers to easily read it. This includes the use of indentation to help distinguish individual loops, evaluations and procedures, and the addition of blank lines to visually separate code lines and make them stand out amongst one another. This structure can help to avoid the risk of the code becoming a “wall of text” within which developers can find themselves getting lost in the search for a particular piece of code.

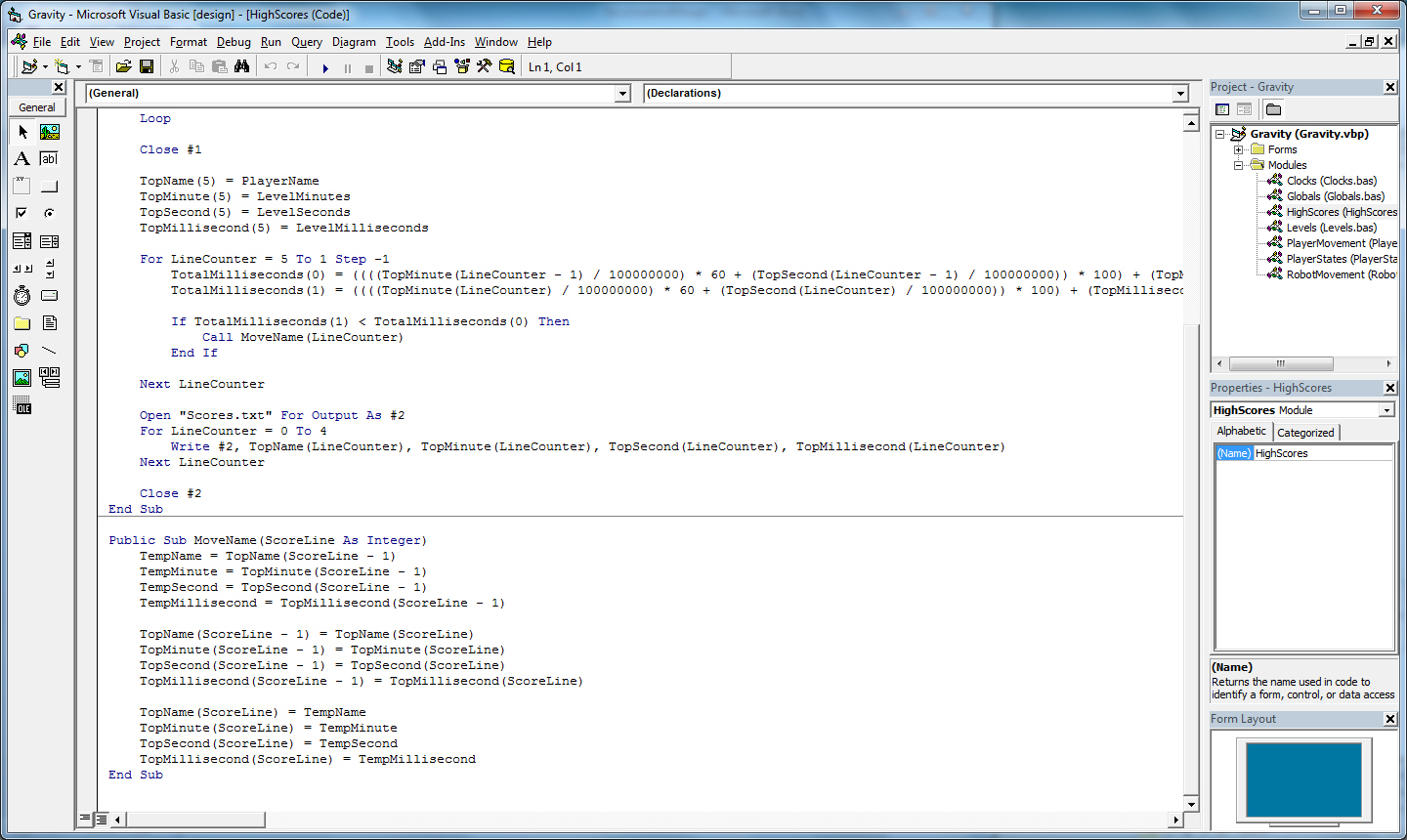


*An example of structured code. The code that is executed when the conditions outlined in each If statement prove true are indented by four spaces. Because this code is itself an additional If structure (a “nested If”) the code that executes when its own evaluation returns true is indented a further four spaces. This helps to distinguish to both new and existing users which code belongs to which structure. While a developer might be able to understand this example easily enough, even if it lacked such indentation, larger structures without indentation would look like an indecipherable wall of text.*

Code can also be separated into individual functions and subroutines, and those procedures can also be placed within separate modules. These procedures, written to perform specific tasks within the program are then called from elsewhere within the program, including the main loop, whenever they are necessary.

There are a number of benefits to this, one being that it can allow the developer to adhere to the principle of “don’t repeat yourself,” or DRY, which aims to reduce the repetition of code where it can be avoided, which in turn can ease the maintenance of a program: several potential repetitions of a block of code would be consolidated to a single procedure, with a single line calling that procedure where needed (and in the case of functions, passing the result of that function to a variable). As a result, rather than making the same change across several iterations of that code (with the risk of missing one or more, or accidentally making inconsistent changes), developers would only need to make the one change.

Functions and subroutines are not restricted to repeated blocks of code. They can also be used to contain code that is only ever used once in the program, with the aim of arranging code into their own self-contained blocks, each with their own purpose within the program. This can aid maintenance by making this code easy to find and identify, and allowing that code to be maintained independently of the rest of the program. In addition, new procedures can be added to the program to extend its functionality, these procedures either written exclusively for the program or being more general-purpose and available for use in other programs where required.



*An example of a subroutine, designed to swap around high scores. Although this code is used five times in a row, storing it in a separate procedure means it only needs to be written once, and thus only one copy of this code exists to be updated when the need arises. Additionally, that the procedure exists for one purpose, coupled with an appropriate name, makes finding this code an easier task than if it had existed within the main game loop.*