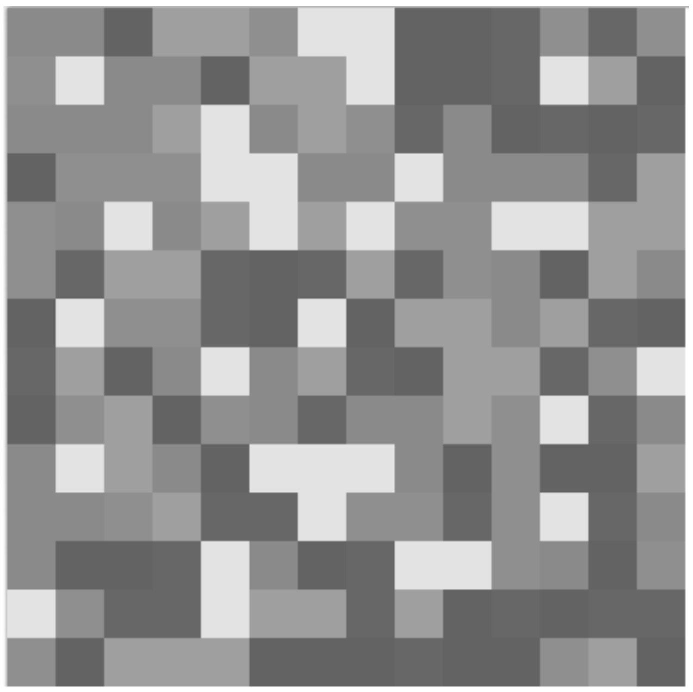
Deconstructing floodit.py

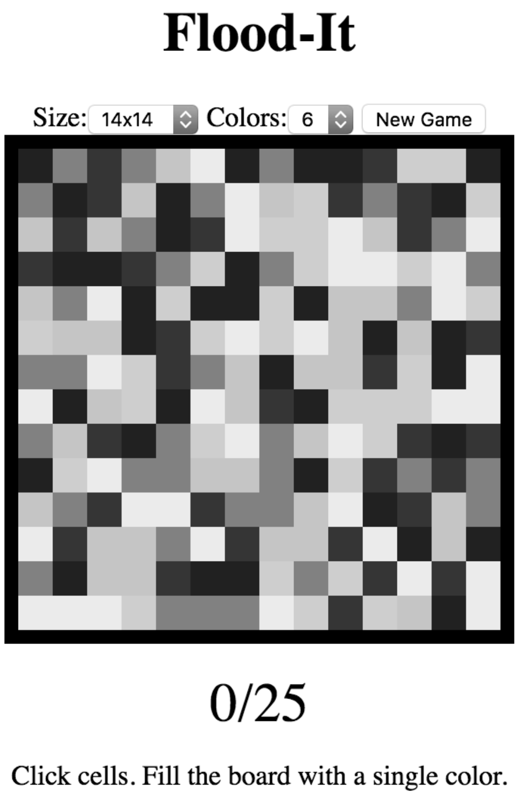


A plaintext explanation and analysis of `floodit.py`, a recreation of the popular puzzle game "Flood it", built with Python and Qt4.

Marshall Briggs
UID: 304417630
University of California, Los Angeles
PIC 16, Spring 2017
Friday, June 9, 2017

1: Introduction and Problem Definition

Flood-it is a popular one-player game, found on smartphones, online, and in the problem sets of many Computer Science courses (*the reason for that will be explained later*).



http://unixpapa.com/floodit/

The objective of the game is simple: the player's "flood" begins in the top left corner. By clicking other cells of a different color, the player can grow his flood to include any adjacent squares of that color. The game ends when the player's flood covers the entire board.

This wouldn't be very difficult on its own (and frankly, not very fun), so instead of allowing an unbounded period to complete their task, the game limits players with a set number of moves. This number can be computed mathematically, as in the example above, or via an algorithm, as in the problem definition (*Speaking of which…*).

`Problem Definition:` Write a version of the game Flood-It! in Python. The player should be able to choose both the board size and the number of colors. Your game should start in a random state. *Instead of the player playing against a fixed number of moves,* you will let the player play against the computer. The number of moves the player needs to beat is the number of moves that the computer would need if it played the game using a greedy algorithm, depending, of course, on the starting configuration of the board.

In the following pages, I will outline my Python implementation of the stated problem definition, all the while avoiding writing *actual code* like the plague. The objective of this report is that you, the reader, regardless of your experience with Python and the Qt4 module, upon completion of this document will be able to produce your own recreation of this game, with relative ease.

2: Dissecting floodit.py

Preface: In an attempt to avoid making false promises, I will hastily amend, somewhat, the statement I made earlier. I cannot teach you everything there is to know about Python and the Qt4 module (*neither of us* has the time for that). But don't fret! A reader with no Python experience should be able to read and understand this with relative ease, and while I cannot explain *everything*, I will do my best to clarify the principles of Python that are (a) integral to your understanding and (b) make programming in Python natural and enjoyable (for this problem definition in particular).

From a bird's eye view, a Python source file is a series of targets waiting to be interpreted. In the source file in question, targets include classes, such as 'FloodIt' and 'GameBlock', and functions, like 'Game', 'CheckWinCondition', and 'main'. When the interpreter processes `floodit.py`, it scans various definitions, searching for the function called 'main.' This function (illustrated below) is the root of our program, and is our entryway into the game.

I. Function: `main()`

With this main function in particular, we accomplish four main tasks:

1. We initialize an object called "`floodit`" of type QApplication, from the QtGui module of PyQt4. The QApplication class manages the control flow and main settings of our GUI application.
2. We set the window icon of our GUI application to an image.[1]
3. We initialize an object "`Window`" of type FloodIt, and display the window in our GUI application.[2]
4. We tell the program to exit as soon as it receives an "exit" status from the GUI Application.

[1] This was a purely aesthetic decision on my part. This function attempts to
[2] We will explore the FloodIt class in depth just a little bit later.

floodit.py; lines 347 - 358[3]

```
-------------------------------------------------

function 'main':
     floodit = an object of type QtApplication
     Set floodit's window icon to "instance.png"
     Window = an object of type FloodIt
     Display Window
     Exit the program when 'floodit' quits

-------------------------------------------------
```

As you can hopefully see, the main function is our entrance into and exit from the GUI application. As soon as the computer runs our program, we launch the GUI; likewise, as soon as our program reports that the GUI has closed, our program exits.

II. Class: `FloodIt`

This is the basic wrapper class for our Flood-It game, containing the variables needed to build and operate key elements of the GUI and keep track of the state of the game, as well the methods needed to interface with the player and trigger actions on-screen. For the time being, I will present only the class interface, including and explaining variable definitions and function prototypes, before later diving into the actual implementation of each function.

[3] I will be making references to specific line numbers throughout this document. Therefore, to the benefit of your understanding, it would be best (but not absolutely vital) for you to have the source file in question, `floodit.py`, at your fingertips. Like I promised earlier, these references won't contain *actual code*; instead, I will replace them with pseudo code that accomplishes the same outcome and is easier to understand.

The first thing to notice about the FloodIt class is that it names itself a child of the “QMainWindow” class, from the QtGui module. Because of this, our FloodIt class can ‘inherit’ (make use of) all the functions belonging to the QMainWindow class, allowing us to add functionality such as adding widgets, applying layouts, and displaying the window in our GUI.

----

floodit.py; lines 125 – 148

```
  class ‘FloodIt’ (parent = QtMainWindow):
  Function ‘initialize’: Does the following
     CenWidget = an object of type QWidget
     Make CenWidget the ‘Central Widget’ of the app
(variable) GraphicsDimensions = 300
(variable) Colors = ["Red","Yellow","Blue”… ]
(variable) BoardColors = 2
(variable) BoardSize = 2
(variable) OpponentRule = "Best Move"
     initialize PlayerGameBoard as an empty array
     initialize PlayerActiveBlocks as an empty array
(counter) PlayerNumberOfMoves = 0
     initialize ComputerGameBoard as an empty array
     initialize ComputerActiveBlocks as an empty array
(counter) ComputerNumberOfMoves = 0
     MovesLabel = an object of type None[4]
     Call the UISetup function
     Call the Menu function
     Display the FloodIt object in our GUI application

Function UISetup:
Function Menu:
Function ClearWindow:
Function Game:
Function SetupPlayerBoard:
Function SetupComputerBoard:
Function SetBoardColors:
Function SetBoardSize:
Function SetOpponentRule:
Function UpdateMovesCounter:
```

----

[4] You may be wondering why I would create an object here, only to leave it empty. I will make use of ‘MovesLabel’ later on, but this particular tactic adheres to a particular characteristic of Python classes: object params can only be initialized *inside the ‘initialize’ function*. If I create ‘MovesLabel’ outside of the class definition, it will not persist outside of its stack frame.

That may be a lot to follow, so I will quickly follow the above pseudo code with plain English definitions.

The first function declared a member of the “FloodIt” class is the initializer (or constructor). This function will be run automatically whenever an object of the FloodIt type is created[5], and is the only place that persistent object parameters can be created (they can be accessed elsewhere, but any variables instantiated in other functions will be deallocated when their particular frame is popped off the stack).

FloodIt’s initialize function assigns the following variables to the FloodIt object (our game):

- **CenWidget:** The central widget of the application. This is a useful variable for setting layouts within the Main Window. When you attempt to give your Window a layout without applying it to the Central Widget, Python complains.
- **GraphicsDimensions:** The dimensions of each of the two boards (player and computer) are GraphicsDimensions x GraphicsDimensions (e.g. 300 pixels by 300 pixels)
- **Colors:** An array of strings containing the 15 usable colors for the Game Blocks: Red, Yellow, Blue, Green, Magenta, Cyan, darkRed, darkYellow, darkBlue, darkGreen, darkMagenta, darkCyan, Gray, Black, and darkGray (Mental note: *Please* remove darkGreen, it's barely distinguishable from ordinary Green).

[5] Since our game will only ever create a single FloodIt object, it is perhaps not immediately intuitive why we would even feel the need to use complicated class interfaces to encapsulate our program. Why not just create objects, instead of assigning them as class parameters? I have several answers to this question.

(1) “That’s just the way Qt works.” (Certainly the answer I came to know and love while working on this assignment.)
(2) Assigning parameters to a persistent class object allows us global access to key variables and objects, without needing to declare them as global variables.
(3) Inheritance from the QtMainWindow, QtGraphicsRectItem, … classes provides invaluable functionality to our GUI; without inheritance, (to bring it back to answer 1), there would be no Qt.

- **BoardColors**: The number of colors that will appear in the current game. This parameter will be set by the user using the GUI, but is default initialized at 2.
- **BoardSize:** The dimensions of the current game (BoardSize x BoardSize, e.g. 24 blocks by 24 blocks). This parameter will be set by the user using the GUI, but is default initialized at 2.
- **OpponentRule:** The greedy algorithm used by the computer in the current game. This parameter will be set by the user using the GUI, but is default initialized to use the "Best Move" algorithm.
- **PlayerGameBoard:** An array of GameBlock[6] objects, representing the player's board.
- **PlayerActiveBlocks:** An array of GameBlock objects, belonging to the player's board, containing the blocks in the player's current "flood." This was meant to grow with every move by the player, but I found it to be simpler (though possibly less functional) to keep this array with size = 1, only containing the game's starting block (Upper left corner, at 0,0).
- **PlayerNumberOfMoves:** The number of moves made by the player so far.
- **ComputerGameBoard:** An array of GameBlock objects, representing the computer's board.
- **ComputerActiveBlocks:** An array of GameBlock objects, belonging to the computer's board, containing the blocks in the computer's current "flood." Like the player's version of this array, it will remain at size = 1 for the duration of the game.
- **ComputerNumberOfMoves:** The number of moves made by the computer so far.
- **MovesLabel:** A QLabel object located (in the application) below the game windows and above the settings' dropdown boxes. Reflects the number of moves made by the player so far.

Finally, the initializer function also applies a few of the class's member functions to the newly created object: `UISetup, Menu, and show.` The `show` function simply tells the FloodIt object to display itself in the active GUI application (`UISetup` and `Menu` will be introduced momentarily).

[6] Don't get ahead of me! We'll cover the GameBlock class next.

Members of the FloodIt class have access to the following functions:

- **Initialize**: Run automatically whenever an object of the FloodIt type is created.
- **UISetup**: Takes a FloodIt object as input, sets up some characteristics of the FloodIt game window (the location, size and title of the window).
- **Menu**: Takes a FloodIt object as input, and sets up and displays the arrangement of the Main Menu screen as follows (arranged vertically, as it would be on screen):
    - A QLabel object, containing the Menu Title
    - A QLabel object, containing a .png image, (Colored boxes)[7]
    - A QLabel object, containing a brief explanation of Flood-It!
    - A QButtonPress object, a button to begin the game
- **ClearWindow**: Takes a FloodIt object and a QLayout object as input. This function iterates through the Widgets inside its QLayout argument, removing each Widget's parents and marking it for deletion. If the object it encounters is a layout instead of a Widget, it recursively calls ClearWindow on that Layout, clearing it of all Widgets. If the object it encounters is of type "None" (one example of a Nonetype object is a spacer), the function does nothing.
- **Game**: Takes a FloodIt object as input, and sets up and displays the arrangement of the Game screen as follows (arranged vertically, as it would be on screen):
    - 3 QLabel objects: two Board Titles (Player and Computer), and which greedy algorithm the computer will be playing
    - 2 QGraphicsView objects, each containing a QGraphicsScene displaying a Flood It! board
    - A QLabel object[8], recording how many moves have been made
    - 3 QLabel objects, 3QComboBox objects, and a QButtonPress object. Each label displays the title of its respective combo box:
        - "Colors:" – Choose how many colors to play with (from 2 to 15 colors)
        - "Size:" – Choose the dimensions of your board (from 2x2 to 30x30)
        - "Greedy Algorithm:" – How do you want your opponent to play? (Best Move or Worst Move)

[7] Another aesthetic decision. On my machine, it finds the image, and displays the image. You won't see the menu's image on your machine.

[8] Hey, it's me, MovesLabel. Remember me?

- **SetupPlayerBoard:** Takes a FloodIt object and a QGraphicsScene object as input, arranges BoardSize x BoardSize colored rectangles (All GameBlock objects, filling them with random colors, depending on the value of the BoardColors variable). Appends each block to the FloodIt object's PlayerGameBoard array, and appends the origin block (at the QGraphicsScene's pixel location [0,0]) to the FloodIt object's PlayerActiveBlocks array. Calls the SetNeighbors function on every GameBlock object (to be introduced later).
- **SetupComputerBoard**: Takes a FloodIt object and a QGraphicsScene object as input, arranges BoardSize x BoardSize colored rectangles (All GameBlock objects, filling them with the same color as their respective block on the Player's game board). Appends each block to the FloodIt object's ComputerGameBoard array, and appends the origin block (at the QGraphicsScene's pixel location [0,0]) to the FloodIt object's ComputerActiveBlocks array. Also calls the SetNeighbors function on every GameBlock object.
- **SetBoardColors**: Takes a FloodIt object and a string, containing text from a Dropdown menu, as input. This function is connected to the "Colors" ComboBox in the Game window, such that every time the user selects an option using the combo box, the BoardColors variable changes to reflect their selection.
- **SetBoardSize**: Takes a FloodIt object and a string, containing text from a Dropdown menu, as input. This function is connected to the "Size" ComboBox in the Game window, such that every time the user selects an option using the combo box, the BoardSize variable changes to reflect their selection.
- **SetOpponentRule**: Takes a FloodIt object and a string, containing text from a Dropdown menu, as input. This function is connected to the "Greedy Algorithm" ComboBox in the Game window, such that every time the user selects an option using the combo box, the OpponentRule variable changes to reflect their selection.
- **UpdateMovesCounter**: Takes a FloodIt object as input. Called every time the player makes a move, this function changes the text of MovesLabel to reflect the current PlayerNumberOfMoves counter (belonging to the FloodIt object).

III. Class: `GameBlock`

Above, I made reference to objects of type 'GameBlock' (particularly, in the `SetupPlayerBoard` and `SetupComputerBoard` functions). In order to register the player's mouse events[9], and to support a grid of rectangles with persistent states (states including color, neighbor, and location information), I had to create a new class called GameBlock.

----

floodit.py; lines 9 - 29

```
   class GameBlock (Parent = QGraphicsRectItem):
   Initialize(parameters: ox, oy, dx, dy, i, j, FLOOD,
                 string):
      # Unlike the FloodIt class, GameBlock's
        initializer requires parameters
      Create a QGraphicsRectItem object at location
            [ox, oy] with dimensions [dx, dy]
(variable) Location = [i,j]
      initialize UpNeighbor = self
      initialize DownNeighbor = self
      initialize LeftNeighbor = self
      initialize RightNeighbor = self
      initialize Neighbors array to contain the four
                 neighbors
      Color = Nothing, right now
      initialize FloodGame as the FLOOD argument
(variable) BoardType = the 'string' argument
      IF BoardType is "Player", THEN:
            initialize Board array as the FloodIt
                 object's PlayerGameBoard
            initialize ActiveBlocks as the FloodIt
                 object's PlayerActiveBlocks
      IF BoardType is "Computer", THEN:
      initialize Board as FloodIt's ComputerGameBoard
      init ActiveBlocks as FloodIt's ComputerActiveBlocks
(variable) BoardSize is FloodIt's BoardSize
```

[9] QGraphicsRectItems do not have access to the mousePressEvent function. Here is another case where inheritance does the programmer a favor; it allows us to create a custom class, in this case the 'GameBlock' class, to expand a Qt class's functionality.

`(continued) floodit.py`; lines 9 - 29

```
Function: mousePressEvent
Function: ComputerMakeMove
Function: CountNeighborColors
Function: SetNeighbors
Function: ColorChange
Function: CheckWinCondition
-----------------------------------------------------------------
```

The GameBlock class is a somewhat ugly interface, but necessary for the proper implementation of Flood-It!'s mechanics. These mechanics do not just include registering mouse clicks; the blocks in the game boards need to hold information about their current state. Such information includes:

- Color: At the very least, each block needs an easily accessible Color variable (easy to check and easy to change)
- Location: Each block needs to know its Location within the PlayerGameBoard / ComputerGameBoard array, so that we can find out who its neighbors are.
- Neighbors: Speaking of neighbors, each GameBlock object needs to know who its neighbors are, so that, when it is involved in a Flood-It! game move, it knows what blocks it is allowed to affect.
- BoardType: Each GameBlock needs to know whether it belongs to the Player's board or the Computer's board, to know how to behave when the mouse is clicked.

The GameBlocks' initializer function accepts 8 extra parameters from the caller:

- ox, oy: Coordinates for the top left corner of the Graphics Rectangle
- dx, dy: Dimensions of the Graphics Rectangle
- i,j: The Game Block's position within the PlayerGameBoard/ComputerGameBoard array
- FLOOD: the FloodIt object the GameBlock belongs to
- string: "Player" or "Computer", the type of the gameboard the GameBlock belongs to

First, the initializer starts by creating an object of type QGraphicsRectItem with the provided coordinates and dimensions.

Then, in an effort to provide the above functionality, the initializer instantiates the following parameters:

- UpNeighbor: The GameBlock's neighbor one block above itself in the FloodIt object's respective (Player or Computer) two-dimensional GameBoard array.
- DownNeighbor: The GameBlock's neighbor one block below itself in the FloodIt object's respective (Player or Computer) two-dimensional GameBoard array.
- LeftNeighbor: The GameBlock's neighbor one block to the left of itself in the FloodIt object's respective (Player or Computer) two-dimensional GameBoard array.
- RightNeighbor: The GameBlock's neighbor one block to the right of itself in the FloodIt object's respective (Player or Computer) two-dimensional GameBoard array.
- Neighbors: An array containing the GameBlock's four neighbors (UpNeighbor, DownNeighbor, LeftNeighbor, RightNeighbor). The four neighbor variables are default initialized to just be the GameBlock itself, and may be changed later within the `SetNeighbors` member function.
- Color: The color of the GameBlock. Default initialized as an empty string, set by the SetupPlayerBoard/SetupComputerBoard function.
- FloodGame: The FloodIt object that the GameBlock belongs to.[10]
- BoardType: Which board the GameBlock belongs to ("Player" or "Computer")
- Board & ActiveBlocks: Depending on the value of the BoardType variable, the initializer assigns values to Board and ActiveBlocks
    - If the GameBlock is of BoardType = "Player":
        - Board = the FloodIt object's PlayerGameBoard
        - ActiveBlocks = the FloodIt object's PlayerActiveBlocks
    - If the GameBlock is of BoardType = "Computer":
        - Board = the FloodIt object's ComputerGameBoard
        - ActiveBlocks = the FloodIt object's ComputerActiveBlocks
- BoardSize: Initiated to the FloodIt object's BoardSize variable

[10] This may be an ugly way of passing information to the GameBlock objects (by including a reference to our FloodIt game in every single GameBlock), but there is a wealth of information contained in the FloodIt class that each GameBlock needs to make use of.

Members of the FloodIt class have access to the following functions:

- **mousePressEvent**: Takes a GameBlock object and a signal of the QEvent class as inputs. This function triggers whenever the mouse is clicked. If the user clicks on a Game Block object belonging to the PlayerGameBoard (and the Color of that GameBlock object is not the same as the current "Flood"), a move will be made. Increments the player's move counter, and calls the CheckWinCondition() function to see if that move caused the player to win. A valid, non-winning Player move triggers the ComputerMakeMove() function.
- **ComputerMakeMove**: Takes a GameBlock object as input. Triggered whenever the player makes a move. This function calls the CountNeighborColors function to count the number of blocks adjacent to the current "Flood" having each color, and makes a move for the computer with regards to those colors counters using the desired greedy algorithm (currently implemented greedy algorithms include: Best Move -- choose the color with the highest color # counter, i.e. the color that belongs to the most neighbors of the current "Flood"; Worst Move -- choose the color with the lowest color counter, # i.e. the color that belongs to the fewest neighbors of the current "Flood." Also, this function increment the computer's move counter and calls the CheckWinCondition() function to see if that move caused the Computer to win.
- **CountNeighborColors**: Takes a GameBlock object, the color belonging to the one GameBlock in the FloodIt object’s ComputerActiveBlocks array, the NeighborColorsCount array to be populated, and a VisitedBlocks array recording which blocks have already been scanned as input.
  The algorithm within this method does the following:

```
Marks the input GameBlock as visited in the
   VisitedBlocks array
Iterates over the Neighbors array of the input
   GameBlock object
If the Neighbor is not the GameBlock itself, and if
  it has not yet been visited:
  IF the Neighbor’s color is not the input
   RootColor THEN:
      Increment the counter of the GameBlock’s
         respective color in NeighborColorsCount
  If the Neighbor’s color IS the input RootColor:
      Recursively call CountNeighborColors on the
         Neighbor (it is part of the current “Flood, and
         we want to count its Neighbor’s colors”)
```

- **SetNeighbors**: Takes a GameBlock object and coordinates [i,j] (the GameBlock's location within the FloodIt object's PlayerGameBoard / ComputerGameBoard array) as input. Sets the Neighbors fields if the GameBlock to be the blocks Above, Below, to the Left, and to the Right of it, *if those blocks are valid blocks on the game board.* If not, the respective neighbor field is just set to be the block itself.
- **ColorChange**: Takes a GameBlock object, a Color to change the object's Color variable, and a RootColor, the color belonging to the one GameBlock in the FloodIt object's PlayerActiveBlocks / ComputerActiveBlocks array, as input.
  The algorithm within this method does the following:

```
Sets the input GameBlock object's Color variable = the
input Color value.
Uses the QtGui QColor function to apply the color change to
the GameBlock's QGraphicsRectItem.
Iterates over the Neighbors array of the input
      GameBlock object
      If Neighbor is not the GameBlock itself:
            If the Neighbor's color is the same as the
            RootColor:
                  Recursively call the ColorChange function
                  on the Neighbor
```

- **CheckWinCondition:** Takes a GameBlock object as input. Considers the color belonging to the block at coordinates [0,0] of the GameBlock's Board variable, and iteratively compares that color to the Color of every GameBlock in the Board. If it finds a Color that is not the same, the function returns False. If it does not find a Color that is different, the function creates a QMessageBox object to display a "Game Over" message. The function then checks to see if the GameBlock is of the BoardType "Player" or "Computer."
  If the object belongs to the Player, then the player managed to Flood his entire board. The function sets the text of the QMessageBox to congratulate the Player and tell him how many moves he made.
  If the object belongs to the Computer, then the computer managed to Flood his board before the Player. The function sets the text of the QMessageBox to inform the Player of his loss, and tell him how many moves the Computer made to beat him.

3: General Program Flow

This will be a brief section, meant to give the reader a general idea of the control flow of the `floodit.py` application.

1. Program is run
2. Target "main" is located and run. FloodIt object is initialized and the GUI Application opens on screen.
3. The FloodIt object's Initializer function sets up the GUI dimensions and arranges the Menu screen.
4. The user presses the Play button, and the Game function is called
    a. The Game function calls the "ClearWindow" function, wiping the Widgets used in the Menu Window
    b. The Game function arranges its widgets on the GUI application, including the Player's Game Board and the Computer's Game Board, triggering the SetupPlayerBoard and SetupComputerBoard functions
5. The two game boards are added to the GUI, instantiating the selected number of colored GameBlocks.
6. The Player begins to interact with the GUI. When the Player clicks on his GameBoard, the mousePressEvent function is triggered repeatedly, triggering the recursive ColorChange function, the ComputerMakeMove function, and all of its recursive targets.
7. The CheckWinCondition function is called repeatedly until a win condition is recognized.
8. The GUI spawns a MessageBox congratulating the winner.
9. The Game function is called again, which in turn calls the ClearWindow function.
10. Repeat steps 4 – 10