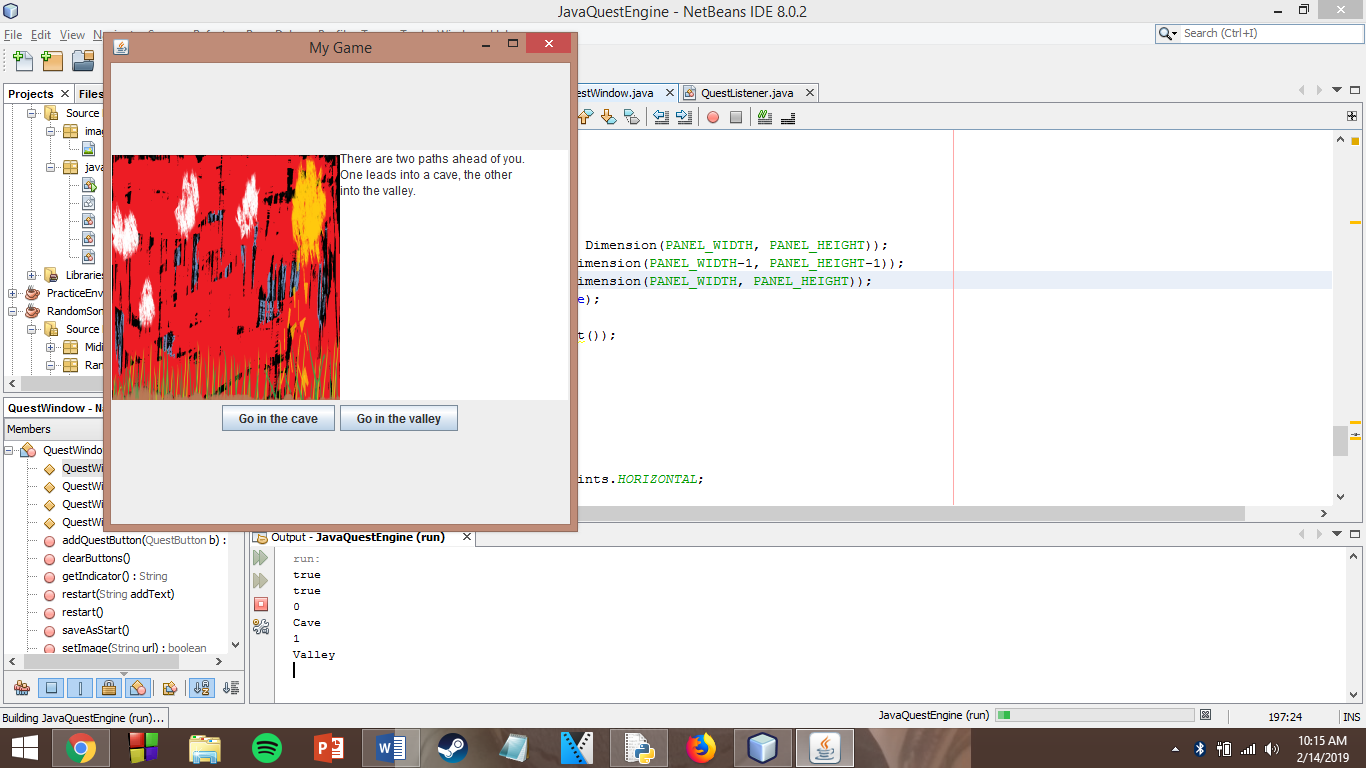
# Documentation for the JavaQuest Engine

## Screenshot



## QuestWindow

### Private Instance Variables

* top – the window itself
* image – the panel on the left-hand side which contains images for your game
* text – the panel on the right-hand side which contains text for your game
* width – the integer width of the whole window
* height – the integer height of the whole window
* indicator – the String holding the identity of the most recent button pressed

### Constructors

QuestWindow()

* Creates and displays a new QuestWindow with the title “Untitled Game” and a size of 475 by 500
* This new window will use the default picture and text
* Example
  + QuestWindow newWindow = new QuestWindow();

QuestWindow(int, int)

* Creates and displays a new QuestWindow with the title “Untitled Game” and a size determined by the two integers that you provide (width, height)
* This new window will use the default picture and text
* Example
  + QuestWindow newWindow = new QuestWindow(400, 500);

QuestWindow(String)

* Creates and displays a new QuestWindow titled the same as the String that you provide and a size of 400 by 500
* This new window will use the default picture and text
* Example
  + QuestWindow newWindow = new QuestWindow(“My Game”);

QuestWindow(String, int, int)

* Creates and displays a new QuestWindow titled the same as the String that you provide and a size determined by the two integers that you provide (width, height)
* This new window will use the default picture and text
* Example
  + QuestWindow newWindow = new QuestWindow();

QuestWindow(String, int, int, int, int)

* Creates and displays a new QuestWindow titled the same as the String that you provide and a size determined by the first two integers that you provide (width, height), and will set the location of the window to x/y coordinates specified by the second pair of integers
* This new window will use the default picture and text
* Example
  + QuestWindow newWindow = new QuestWindow();

### The Menus

#### Methods for Allowing Menu Access

enableMenu()

* Will display the menu bar on the top of the window.
* Example:
  + (Assuming that *gameWindow* is a QuestWindow)
  + gameWindow.enableMenu();

disableMenu()

* Will turn off (make invisible) the menu bar on the top of the window.
* Example:
  + (Assuming that *gameWindow* is a QuestWindow)
  + gameWindow.disableMenu();

#### Methods for Saving Progress

saveGame()

* Will create a file with the name *file* and the extension *.jqsav*.
* This file will contain a list of all the buttons pressed by the user during their play. All of these button identifications will be encrypted using the *saveFileEncryptor()* method.
* This method is meant to be called by the menu item when clicked by the user, but it can also be called like this:
  + (Assuming that *gameWindow* is a QuestWindow)
  + gameWindow.saveGame();

saveGame(String)

* Given a name for the file, will create a file with that name and the extension *.jqsav*.
* This file will contain a list of all the buttons pressed by the user during their play. All of these button identifications will be encrypted using the *saveFileEncryptor()* method.
* This method is meant to be called by the menu item when clicked by the user, but it can also be called like this:
  + (Assuming that *gameWindow* is a QuestWindow)
  + gameWindow.saveGame(“myfilename”);

loadGame()

* Will read a file with the name *file* and the extension *.jqsav*, which must contain the encrypted button identities that were clicked.
* This method will decrypt each line of the save file and run the button identifications from the file, in order of how they were clicked.
* This method is meant to be called by the menu item when clicked by the user, but it can also be called like this:
  + (Assuming that *gameWindow* is a QuestWindow)
  + gameWindow.loadGame();

loadGame(String)

* Given a name for the file, will read a file with that name and the extension *.jqsav*, which must contain the encrypted button identities that were clicked.
* This method will decrypt each line of the save file and run the button identifications from the file, in order of how they were clicked.
* This method is meant to be called by the menu item when clicked by the user, but it can also be called like this:
  + (Assuming that *gameWindow* is a QuestWindow)
  + gameWindow.loadGame(“myfilename”);

saveFileEncryptor(String)

* Returns an encrypted version of the String argument passed to it. This encrypted is used for save files.

saveFileDecryptor(String)

* Returns a decrypted version of the encrypted String argument passed to it. This decryption algorithm is used for loading save files.

### Other Methods

clearButtons()

* Removes all buttons from the window
* Does not return anything

swapTextAndImage()

* If the image is on the left and text is on the right, will put image on the right and text on the left, and vice-versa

saveAsStart()

* Will save the current image, text, and buttons. This allows you to call the restart() method and return the game to the state it was in when the game first started

restart()

* Will reset the game window to what it appeared when you last called the *saveAsStart()* method
* Will **not** change the indicator variable (for last button pressed)

restart(String)

* Will reset the game window to what it appeared when you last called the *saveAsStart()* method, however the text box will contain the String argument passed to it before the original start text
* Will **not** change the indicator variable (for last button pressed)

getIndicator()

* This is your way to find out what button the user has just pressed. If your indicator matches the identity of the button that you made, then you can rest assured that that button was just now pressed.
* It returns the String identity of the button pressed exactly as you wrote it for that particular QuestButton

setImage(String)

* Updates the image JPanel to have the image located at the URL String that it is passed.
* Will return *true* if it found the picture, *false* if it could not find the picture. Use this fact to test your program if an image isn’t showing up.
* Example:
  + top.setImage(“C:\\Users\\Kevin Strileckis\\Documents\\NetBeansProjects\\JavaQuestEngine\\src\\images\\default.png”)

setImage()

* Updates the image JPanel to the default JavaQuest image
* Example:
  + top.setImage();

Similar methods:

* setGrayImage()
* setGrayImage(String)
* setByteImage()
* setByteImage(String)

setText(String)

* Updates the text JPanel to the String that is passed to it
* Example:
  + top.setText(“Lorem ipsum”);

setText()

* Updates the text JPanel to the default String message
* Example:
  + top.setText();

setFontName(String)

* Will change the current font of the text on the screen to the font indicated by the String argument
* Suggestions:
  + Elephant
  + Narkisim
  + Broadway

setFontSize(int)

* Will change the current size of the text on the screen to the font size indicated by the integer argument

addQuestButton(QuestButton)

* Will add the QuestButton Object passed to it to the bottom third of the window. Works best with three or fewer buttons
* Returns true
* Example:
  + QuestButton button1 = new QuestButton(top, 3, "Go in the cave");
  + top.addQuestButton(button1);

setBackground(String)

* Changes the background of the QuestWindow to the color matching the String:
  + Red
  + Yellow
  + Black
  + White
  + Green
  + Blue
  + Cyan
  + Gray
  + Lightgray
  + Darkgray
  + Pink
  + Magenta
  + Orange
* Example:
  + top.setBackground(“Cyan”);

setBackground(double, double, double)

* Changes the background to the color determined by the Red-Green-Blue format based on the three numbers passed as arguments
* Example:
  + top.setBackground(255, 0, 120);

darkenBackground()

* Darkens the background a little
* Example:
  + top.darkenBackground();

lightenBackground()

* Brightens the background a little
* Example:
  + top.lightenBackground();

## QuestButton

### Private Instance Variables

* button – a reference to the JButton component of the QuestButton
* type – an integer representing what kind of effect this button has
  + Type = 1: Change Text
  + Type = 2: Change Image
  + Type = 3: Change Both
* attachedWindow – the window upon which this button is placed
* textChange – the String holding the text with which to update the text panel when this button is clicked
* imageURLChange – the String holding the URL in the system where the image to update the image panel is stored
* identity – the String containing the identity for your button. Defaults value is the number of buttons you’ve created minus one

### Constructors

QuestButton(QuestWindow, int, String)

* Sets attachedWindow equal to the QuestWindow argument
* Sets type equal to the integer argument
* Creates a new JButton object with the String argument as the button text
* Sets textChange and imageURLChange to “UNSET”
* Example:
  + QuestButton button = new QuestButton(top, 3, “Go in the cave”);

QuestButton(QuestWindow, int, String, String)

* Sets attachedWindow equal to the QuestWindow argument
* Sets type equal to the integer argument
* Creates a new JButton object with the String argument as the button text
* Sets textChange or imageURLChange to “UNSET”, depending on what type is equal to
  + If type = 1, textChange is set equal to second String argument; imageURLChange is set to “UNSET”
  + Otherwise, textChange is set to “UNSET” and imageURLChange is set equal to the second String argument
* Example:
  + QuestButton button = new QuestButton(top, 3, “Go in the cave”);

QuestButton(QuestWindow, int, String, String, String)

* Sets attachedWindow equal to the QuestWindow argument
* Sets type equal to the integer argument
* Creates a new JButton object with the String argument as the button text
* Sets textChange and imageURLChange to the second and third String arguments, respectively
* Example:
  + QuestButton button = new QuestButton(top, 3, “Go in the cave”);

### Other Methods

setEffect(String)

* If type is 1, will set textChange equal to the String argument passed
* If type is 2, will set imageURLChange equal to the String argument passed
* If type is 3, will check if textChange equals “UNSET”. If so, then textChange will be set equal to the String argument. Otherwise, imageURLChange will be set equal to the String argument
* Example:
  + button.setEffect(“You enter the cave…”);

setEffect(String, String)

* If type is 1, will set textChange equal to the first String argument passed
* If type is 2, will set imageURLChange equal to the second String argument passed
* If type is 3, will set textChange equal to the first String argument and imageURLChange equal to the second
* Example:
  + button.setEffect(“You enter the cave…”, “images/image.png”);

### Accessors and Mutators

* getType()
* setType(int)
* getTextChange()
* setTextChange(String)
* getImageURLChange()
* setImageURLChange(String)
* getIdentity()
* setIdentity(String)

## QuestBehavior

This is an abstract class that you are meant to extend. If you do so properly and fill its *performBehavior* method with plenty to do, your game can actually run!

### Constructor

QuestBehavior(QuestWindow)

* This constructor will establish the connection between the QuestWindow and the set of behaviors that you define in the performBehavior() method
* When you extend this abstract class, your child class will need to have its own constructor which calls this method

### Other Method

abstract boolean performBehavior(String)

* You will need to implement this method in your child class. It will contain all of the if statements for the various button presses that are possible in your program.
* Receives one String for the identity of the button which was pressed. Use this parameter to control the flow of your game.