Assignment 01

Quad Tree Simulator

Assignment 01

Table of Contents

Assignment 01

Important Notes

1. I will **not** update this document after it is posted as you might print it. Keep an eye out on your email and announcements for any update or changes.
2. Must use JDK 11.
3. JavaFX 11 is downloaded through maven.
4. You can adjust the height and width of your GUI to match your screen.
5. You can use any IDE you like for this lab. However, all instructions are written for Eclipse on Windows. If you use decide on other IDE's or OS it is up to you to figure out how to convert the project.
6. If you are getting **NullPointerException**, it means you forgot to initialize something. You Should be able to figure it out using debug and just reading your error.
7. Whenever you use a getter method you should store it to be used later or directly pass it to another method. There is never a point in calling a getter and not using it.
   int width = canvas.getWidth(); // good
   canvas.getHeight(); // ?? what is the point, this is wrong

Skeleton

1. Do not modify the code I have given you.
   a. If you modify anything and it doesn't work, it is up to you to fix and provide reasoning for the changes.
2. This project is based on JavaFX 11 and JDK 11.
   a. If you end up using a higher JDK version, you must let me know in the comment of your submission.

Output

Your application must run and should look like something from the demo on the assignment page. Don't forget to watch the import summery linked on the Brightspace assignment page also in week 2/video/summary.

Bright Space

Keep an eye out for updates that will be posted on Brightspace assignment page. This document will not be updated after it is posted.

Due Date

Technically due on October 25th midnight however, no late penalty till November 1st midnight.

Late Penalty

10% per day, after day 4 it will be zero.

Assignment 01

Attachments

1) Skeleton of project to complete.
2) This writeup which you are reading.
3) UML diagrams

Submission (No DEMO)

Must submit **two** files. If you have done any bonuses you need to mention it in the comments of Brightspace assignment submission page. If you have errors and your code does not run mention that as well.

1. A zip/archive of your whole project.
   [firstName]-[lastName]-[labSection#].zip
   ex: shawn-emami-11.zip
2. A runnable jar file of your project. You should be able to run it by clicking the jar or using command java -jar [jar path] in command line. Have in mind asset folder needs to be in same directory as jar file.
   [firstName]-[lastName]-[labSection#].jar
   ex: shawn-emami-11.jar

Project import, JAR, and ZIP

If you don't know how to do these steps look at the video in:
"Brightspace/Content/Lecture Material/Week 02/Lecture Videos/Summery - Project import, maven, jar and zip".

Project Import

Unzip your project manually.

1. Unzip the skeleton first and place it where you want.
2. In Eclipse go to file/open project from file system.
3. Click on directory and navigate to unzipped path. Go in directory till you see the src folder. Click select folder.
4. There should now be a new row with the name of project and says import as eclipse project. This is the only row that should be selected.
5. Click finish.

Assignment 01

Requirements

Basic setters and getter might not be given specific instructions, however you must complete them as they appear in the Class Diagrams. Steps bellow will have associated instructions and/or UML diagrams to help you complete them. Aside form the methods you need to complete the class variables as well.

1) Complete the QuadTreeSimulator class.
   a) Complete the method init().
   b) Complete the createColorBar method.
2) Complete AbstractAnimator.
   a) Complete all methods based on UML and instruction provided.
3) Complete QuadTreeAnimator.
   a) Complete init() method.
   b) Complete handle method.
   c) Complete clear method.
4) Complete AbstractScene.
   a) Most methods are basic getters.
5) Complete ColorDetectionScene.
   a) Complete method createScene.
6) Complete QuadTree.
7) Create a Sequence diagram for method QuadTreeAnimator::handle(gc:GraphicsContext, now:long)
   a) You don't need to show initialization of the array. If you want, you can comment it on the diagram like I did.

Bonus

If you do any bonuses make sure to mention them in the comments along with any other bonuses when you are submitting.

1) If your lines are connecting to each other when drawing and you don't want it. You can fix it in QuadTreeAnimator::init. How to fix it is now a small bonus.
2) Change the color of quad tree grid based on the color selected. Maybe a bit darker version. Take advantage of the Sprite class and the built-in methods in Color class.
3) Selected multiple colors.
4) Instead of having 3 colors, use the built in JavaFX color picker and allow the user to choose any color they like.
5) Save the final image drawn to a file.
6) Save the final image with the quadtree on top of it in a file.
7) This bonus is very big, almost and extra assignment. Add a new Animator that instead of detecting lines, it can detect little moving dots in regions f quadtree. Quadtree should update when the dots move.

Assignment 01

Suggested work schedule

Weekly work suggestion.

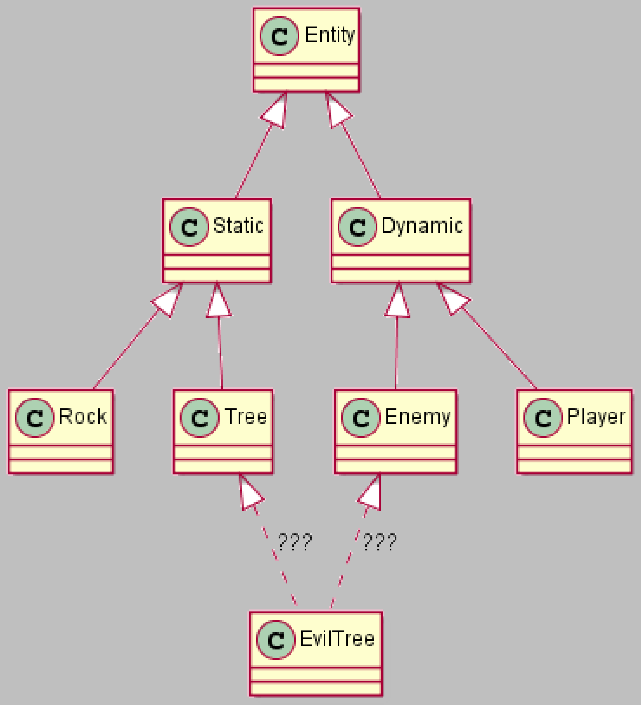
1) Week 3
   a) Read and understand the project instructions.
   b) Complete requirement 1.
2) Week 4
   a) Complete requirement 2 and 3.
3) Week 5
   a) This week is a bit lighter, so you share it with week 4.
   b) Complete requirement 4 and 5.
4) Week 6
   a) Complete requirement 6.
5) Week 7
   a) Complete requirement 7.
   b) Finalize your project and finish any leftover content form previous weeks.
   c) Ready your content for submission.
6) Week 8 – Break
   a) If you decide to leave some work for the break, I suggest requirement 1 and 7.
   b) The support I can give you during the break is very limited and requirement 1 and 7 are easiest ones.

Assignment 01

Designs

Here are some brief descriptions of some of the designs and DPs (Design Patterns) used in this assignment.

In this assignment we are going to focus more on composition over inheritance. Look at the diagram below taken from gamedev.net [1].



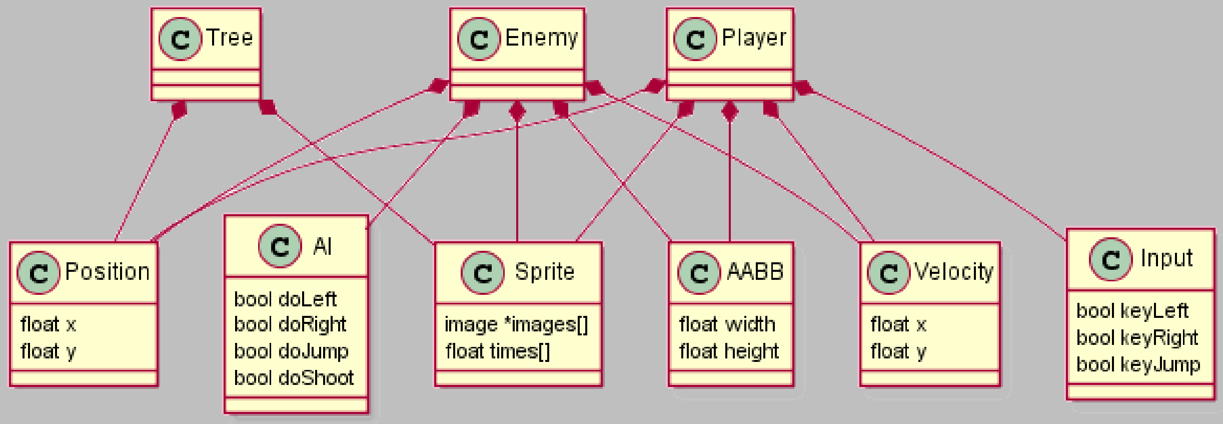
what is wrong with this system? It looks easy enough however the more complex your system becomes the harder it will be to expand your code. In this example what is EvilTree? To solve this, you can inherit form one of the Enemy or Tree but then you must duplicate much of your code to get the behavior of other component. This problem will continue to grow.

Now instead of inheritance if we go with composition, we can solve this issue and create a more expandable system. This Design Pattern is called Entity component system which is an architectural design pattern.

Entity Component System

In this design pattern instead of completely relying on inheritance we rely more on composition. As you can see in diagram bellow entities are composed of traits. This is just an example. Optionally you can read more about this topic at:

https://www.gamedev.net/articles/programming/general-and-gameplay-programming/understanding-component-entity-systems-r3013/



Assignment 01

MVVM Design

Model-View-ViewModel like MVC (Model View Controller) and MVP (Model View Presenter) is designed to separate UI from the logic. In this application Model is the animator, View is the application class, and ViewModel is the AbstractScene. This abstraction allows for separate modification of each layer without impacting other layers. Unlike Controller and Presenter, ViewModel is used as data binding. It does not control any behavior.

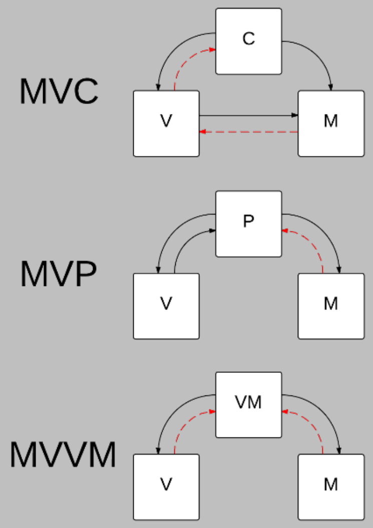


Figure 1: https://www.bigdev.de/2014/02/gui-design-patterns-mvc-mvp-vs-mvvm.html

Observer Pattern

This pattern is used as middle one between and Observable object and its observers. This pattern is a notification system. Observers are notified every time the Observable is changed.

JavaFX uses Observer pattern in Property Objects. Each Node has multiple properties that are tied to different values. For example, a text field has a text property that can be monitored for every time the value of it is changed. By either binding or adding an event handler to a property interested Observer can be notified every time property changes. In this case notification means a lambda that was attached to the property will be executed. There can be multiple lambdas (observers) attached to each property.

Refactoring

This application is built on the idea of lab 1. In lab 1 all the code was written in one class and there was no separation between the components like animator and the main application class. In this refactoring by using MVVM we have refactored the code to more represent a well-built design.

UI components are stored in the application class while the logic is in the animator. These two components are now connected through the AbstractScene.

Link below is a very short Stack Overflow question regarding Design Patterns. I Highly recommend reading it. There 2 long articles referenced in it which up to you to read (Optional).

https://stackoverflow.com/questions/4842667/why-do-some-people-presents-mvvp-as-an-evolution-improvement-over-mvc-whereas-it

Assignment 01

Explanation of packages

In this section I will explain the functionality of all the packages and classes in them.

Utility

1. QuadTree, this class will process an image at pixel level to find a specific color. Node is an inner class of QuadTree. QuadTree divides the screen in 4 equal sectors and each sector will be divided again and again in 4 equal sectors. The rule for this division relies on any colored pixel being present in the sector. The number of times this division will occur depends on the maximum depth defined at initialization of the tree.

Scene

1. AbstractScene, this class holds some common code that might be shared among scenes. In the scope of this assignment we'll only use one scene. However, this application can have multiple scenes allowing for multiple custom behaviors. Now arguably we can move more components from UI to this class to even have more custom look per scene. For example, we can move some GUI Nodes like menu item to scene, so we can have more customized options depending on the scene.
   Two of the methods that will be used often from this class are:
    a. addOption(uniqueName:String, option:Object):void
    b. getOption(uniqueName:String): Object

   Add method adds an object such as IntegerProperty to an internal map called options. The aim is to have one central location for each scene that properties/options can be retrieved based on their name. Here is an example:
   ObjectProperty< Color> colorOption = new SimpleObjectProperty<>(Color.BLACK);
   scene.addOption("color", colorOption);//add the colorOption to scene
   //here the option retrieved from the scene and casted to the desired type.
   ObjectProperty< Color> color = (ObjectProperty< Color>) scene.getOption("color");
   These methods will be used quiet often. You can find the name of options under the heading Options.
2. ColorDetectionScene, this class inherits form AbstractScene. Custom component related to the specific scene are created here instead of in AbstractScene.

Property

1. Drawable, this is a generic interface which holds all the methods that relate to drawing entities. All sprites in this application are subclass of Drawable.
2. Entity, this is an interface which represents anything that can be placed in the Application. For example, QuadTree and FpsCounter are subclass of Entity. This interface holds the common methods between all entities.
3. Sprite, this is an abstract class which implements Drawable. It adds body to all common methods, however leaves draw method as abstract as it must be overridden by its subclasses.

Assignment 01

Entity

1. FpsCounter, this class simply counts the number of frames per second. This class is an entity which is drawable. It extends GenericEntity.
2. GenericEntity, this class implements Entity. It is meant to implement a generic entity which can be customized but ready to use as is. The method update in this class will be called by animator to update the entity. The content of update in almost all cases will be nothing, except for cases where some updated is needed for every frame. In this assignment we don't use this class. It is just here as a support class for the overall structure.

Animator

1. AbstractAnimator, this class extends AnimationTimer. AnimationTimer is a JavaFX class which acts as a timer. It has methods start, stop, and handle. Start and Stop manipulate the active state of timer. Handle method is called by JavaFX automatically every 1/120 of second. Each time this method is called a new frame is drawn on the screen. AbstractAnimator is the abstraction to JavaFX AnimationTimer to hold some common code like clear.
2. QuadTreeAnimator, this class extends AbstractAnimator. This is the heart of the application. it updates and draws everything in each frame. In this animator we have two canvas objects. One canvas is called drawingCanvas which is created internally and is used to store the movements of player mouse for drawing the lines, but it is not actually visible to the user. Other canvas is used for actual display to user. In every frame we take snapshot of the drawing canvas and save it in and image object. The pixels of image object are passed to QuadTree to be processed. Then the image and QuadTree are both drawn on the main canvas.

Assignment 01

Instructions

Here are some details regarding each component in the assignment that needs to be completed.

Options

These are the name of options that are used in the code, you add other options you like to scene.

- quadTreeDepth
- displayFPS
- displayQuadTree
- color

QuadTreeSimulator

init

1. You can find the details of it in the UML attached.
2. At the end, we want to bind the size of the canvas to the size of the root. To do this we going to use the widthProperty() and heightProperty() of our desired nodes. These properties also have methods called subtract() which we will use.
    a. Get the root height property and subtract statusBar height property form it and then subtract optionsbar height property form it. Finally, get height property of canvas and bind it to the result of first bit.
    b. Get the root width property and subtract colorBar width property form it. Finally, get width property of canvas and bind it to the result of first bit.
3. Complete the createColorBar method.

createColorBar

1. Declare a new ObjectProperty with generic type of Color and name it colorOption.
2. Initialize colorOption with SimpleObjectProperty and pass to it Color.Black;
3. Call addOption from scene and pass to it "color" and colorOption.
4. In this method you will create 3 buttons with no names. Here is reminder of some method we are interested in.
    a. Button() //constructor a button with no name
    b. setMinSize(w:double, h:double):void//set a minimum size for the button
    c. setStyle(css:String):void//using custom javafx CSS update look and feel
    d. setOnAction(value:EventHandler<ActionEvent>):void//on button click, execute lambda

    For each button set the minimum size to 40 and 16.
    Each button will have the same style as follow, except change the color of each button.
    "-fx-border-color: #000000; -fx-border-width: 2px; -fx-background-color: RED"
    Each button also will have a setOnAction which will set the value of colorOption to the color you choose for that button.
5. Create a vertical toolbar with minimum width of 50 and add the buttons to it. Here are the methods we are interested in for ToolBar.
    a. ToolBar(Node... items)//constructor that can take open ended number of arguments.
    b. setOrientation(value:Orientation):void
    c. setMinWidth(w:double):void

    return the toolbar.

Assignment 01

AbstractAnimator

Most methods are setters and getters which you should be able to complete.

1. In constructor initialize fps at 10 and 25. Set the fill to black, stroke to white and width to 1. To access fill, stroke, and width, use the method getDrawable on fps first.
2. Override both stop and start methods. In both methods call the super of the method first then if in stop set isRunning to false and if in start set isRunning to true.
3. There are 3 clear methods.
    a. No-arg clear calls clearAndFill with two arguments and pass to it scene.gc() and Color.TRANSPARENT.
    b. clearAndFill with two arguments calls the other clear and feel and pass to it gc, background, 0, 0, scene.w(), and scene.h().
    c. In final clearAndFill call methods setFill, clearRect and fillRect on gc.
4. Complete handle based on UML diagram.

QuadTreeAnimator

Complete the 3 methods.

Init

Described in UML diagram.

Clear

Over the no-arg clear method from abstract animator.

1. Call clear on qt.
2. Call clearAndFill and pass drawingCanvas.getGraphicsContext2D() and Color.TRANSPARENT.

Handle

In this method we describe the process of one frame to be drawn. Clear the canvas and take a snapshot of drawing canvas. This is so we can pass it to QuadTree as pixels to be processed. Draw the image and QuadTree.

1. Call init.
2. Call clearAndFill and pass gc and Color.TRANSPARENT.
3. Create a variable called sp of type SnapshotParameters with default constructor.
4. Call setFill on sp and set the color to Color.TRANSPARENT.
5. Call snapshot on drawingCanvas with sp and null and store the return in a WritableImage called image.
6. If Image is not null call drawImage on gc and pass image and 0,0.
7. Get the “displayQuadTree” option from scene and cast it to BooleanProperty.
8. If “displayQuadTree” option is true
    a. Initialize the buffer with new int[(int) (scene.w() * scene.h()) + 1];
    b. image.getPixelReader().getPixels(0, 0, (int) scene.w(), (int) scene.h(), PixelFormat.getIntArgbInstance(), buffer, 0, (int) scene.w());
    c. get the “color” option from scene and cast it to ObjectProperty< Color>. Name it color.
    d. Call push on qt and pass buffer, scene.w(), and color.get()
    e. Call getDrawable on qt and then draw.

Assignment 01

AbstractScene

Complete the setters and getters based on the class diagram.

1. In constructor initialize options map with HashMap. Then call addOption and pass to it "displayFPS" and new SimpleBooleanProperty(false).
2. The return values for methods w(), h() and gc() are retrieved from canvasNode. Use methods getWidth, getHeight, and getGraphicsContext2D.
3. In addOption,
    a. first check if options map contains key uniqueName.
        i. If it does, throw a new IllegalStateException and as message say the uniqueName already exists.
    b. Use the put method of map to add the uniqueName and option to the options map.
4. In getOption call the get method on options and return the result.

ColorDetectionScene

1. Complete the getter.
2. Initialize the qt in createScene and complete the rest of the method using the UML.

QuadTree

Complete the setters and getters.

1. Do not modify the no-arg constructor. Unless you are messing with the bonuses.
2. In the other constructor chain to the no-arg constructor. Store depth then initialize head Node with 0, 0, 0, width, and height.
3. In clear method just call the clear method of head.
4. In push method we need to convert the Color object to its integer value first. To do so, copy the code below:
   int color = (0xff << 24) | (((int) (c.getRed() * 255) & 0xff) << 16) | (((int) (c.getGreen() * 255) & 0xff) << 8) | ((int) (c.getBlue() * 255) & 0xff);
   then call push on head and pass to it buffer, colSize, and color.

Node

Follow the class Diagram for details.

1. In constructor save all the arguments and set empty to true.
2. Don’t initialize children.

Divide

In this method we divide the QT to 4 equal rectangles. We Only do this if children array is still null. So, if it has already been divided, we don’t divide again. We will have 4 rectangles, top left, top right, bottom left, and bottom right, all the same size. Here is an example of what the bottom right will look.

children[2] = new Node(depth + 1, x + width / 2, y + height / 2, width / 2, height / 2);//bottom right

1. If already divided leave the method
2. Initialize the array children.
3. Initialize each index of the array. You don’t need to use a loop you can hard code the 4 initializations.

Clear

This is a recursive method, meaning it calls itself.

1. Set empty to true
2. Check If it has been divided, which we check by null checking children.
    a. If it has, loop through all children and call clear on each.

Push

This is a recursive method. In this method we have a one-dimensional array of pixels which was original 2 dimensional. Meaning after every rowSize we move to the next row. Traverse the buffer and if an index is found for which the color we are looking, set empty to false and break the loop. Finally, if max depth hasn't been reached and empty is false, divide the node and try to push to all children nodes.

To traverse a one-dimensional array in 2d you can use the loop below:

```
for ( int i = (int) y * rowSize; i < y * rowSize + height * rowSize; i += rowSize) {

        for ( int j = (int) x; j < x + width; j += 1) {
```

and to access an index simply use buffer[i+j].

1. Loop through the buffer
    a. If i+j is bigger and equal to buffer.length, break.
    b. If current index of buffer is equal to color set empty to false and break.
2. If empty is false and current depth is smaller than maxDepth
    a. Divide the node.
    b. Loop through all children can call push on each one of them.