Connect 4 Design Manual

Jonas Scott, Owen Reilly, Mikey Myro, Casey King

Introduction

Our system is a game. We created our system in Java and used JavaFX for the GUI and user interaction. We followed a Model View Controller (MVC) design pattern and package structure. The user is greeted with an interface showing players multiple level options for gameplay. These clickable buttons allow the user to launch into a game modeled in various classes. The game is a 4 by 4 board of mouse-selectable tiles, each displaying a unique word. Upon selecting 4 tiles, the “Submit” button will check if the four Tiles belong to the same category. If they do, the Tiles will disappear, and the user is left to select four more Tiles. Otherwise, a counter variable, representing the amount of guesses the user has, is decreased. 4 incorrect submit attempts results in a game over, prompting the user to return to the initial interface.

User Stories

We wanted to create a program that users of all ages would enjoy, whether they are playing for fun or playing for a challenge. We created a function to highlight each word when it is selected so that the player knows the word being picked to continue making connections. We also made the levels at the beginning screen clickable, so the user can pick a difficulty of the game. Finally, we added feedback after guesses. This includes the connections being pulled up together if you get it right, the tiles shaking if you get it wrong, a “one away” popup if your answer is only one away from being right, and a “you lost” popup if you run out of your guesses.

OOD

CRC Cards

These are our early stage CRC cards with our initial class structures and ideas of what each class would look like. We moved some functionality to the view class and out of the board class in our actual implementation. In our design we wanted to make sure that our system had all the functionality that it would need for our user stories. Making sure we kept the GUI simple and easy to understand was important. Additionally we wanted to make sure the feedback was there so that players could easily understand how many guesses they had left, and how close they were to winning, etc.

ConnectionsView	
- Store the GUI root - Store the difficulties Pane - Store the panes for the tiles - Show the connections game, display the stage	ConnectionsModel

ConnectionsMain	
- Main method to run entire program - Init method to initialize all parts of the MVC - Start method to show the JavaFX window	- ConnectionsView - ConnectionsController - ConnectionsModel

Board	
- Store words on the board - Store which tiles they have clicked on - Rearrange the tiles when requested - Store the categories of the current board - Check if the user has the correct 4 tiles selected	- Tile (tiles on the board) - ConnectionGridMaker(Making the connections grid on the board)

Tile	
- Keep track of the tiles position - Keep track of what word is on the tile - Keep track on if the tile is selected - Keep track of what category the tile is apart of	

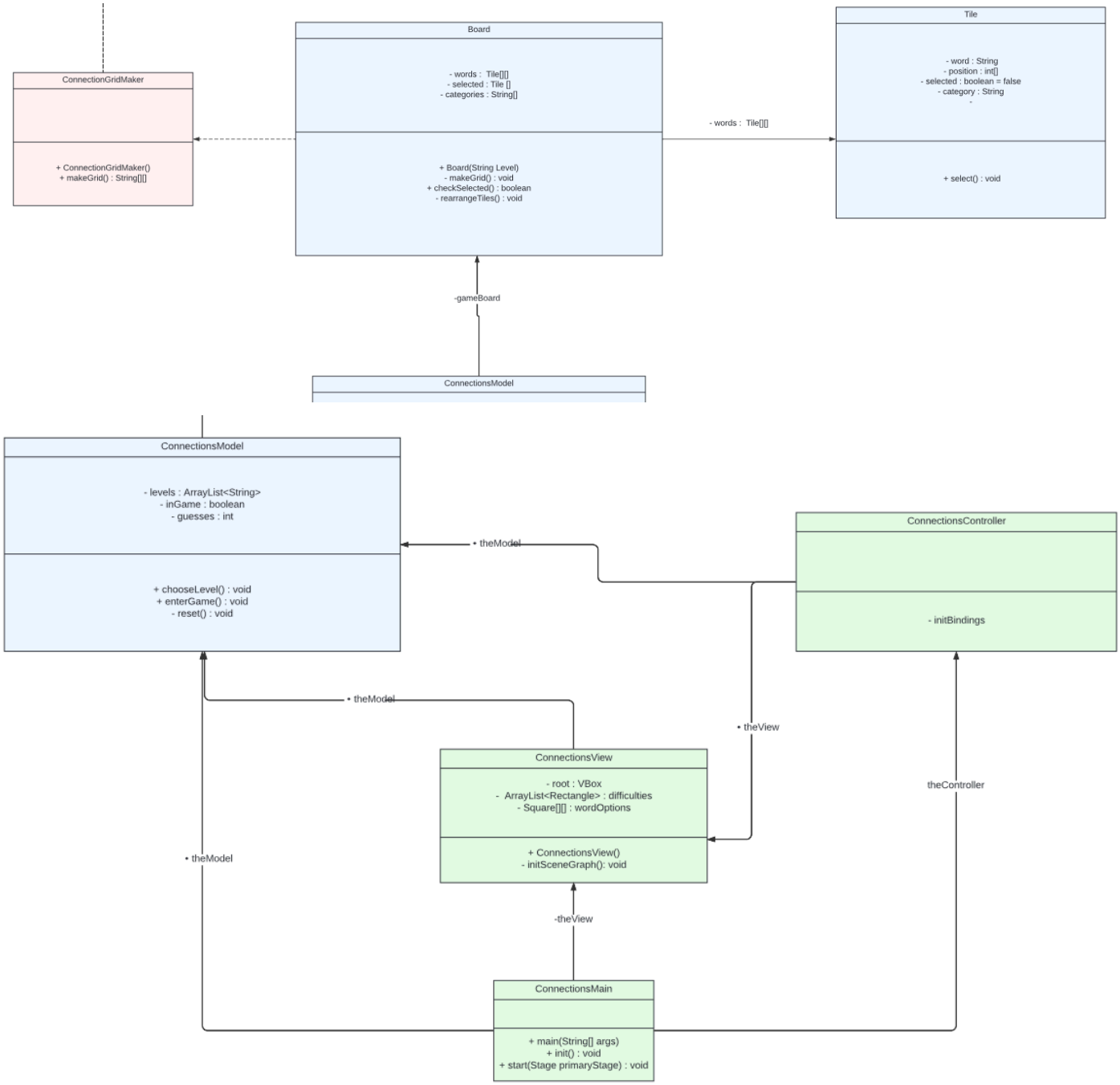
ConnectionGridMaker	
- Make the grid for the tiles to go on as a 2D array of String ArrayLists	

ConnectionsModel	
- Store the levels of connections games - Store if the user is in a game or not - Store the amount of guesses the user has left - This is user interface for choosing which level play and starting their game experience, as well as resetting their game	- Board (the board for the game the user selects to play)

ConnectionsController	
- Controller of the MVC for the game, holds all bindings for the game	- ConnectionsView - ConnectionsModel

Initial UMLs

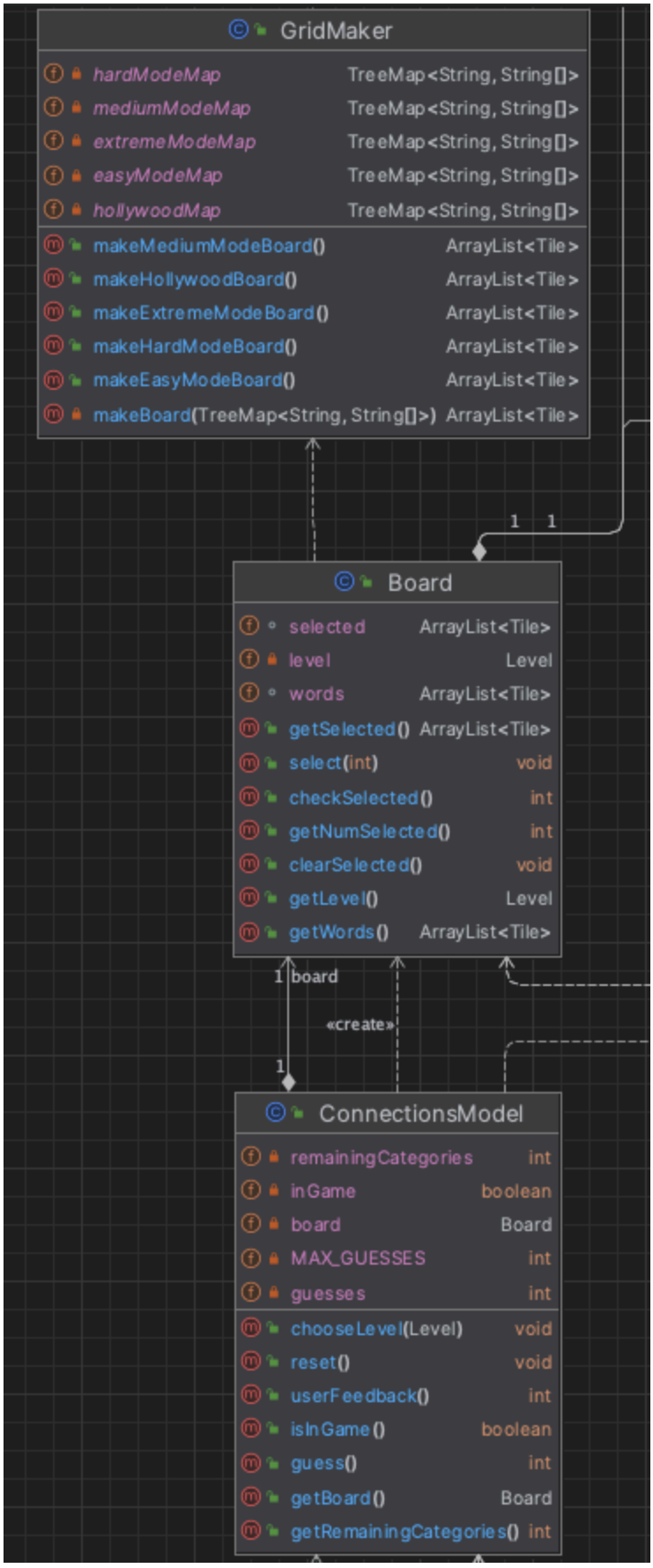
These are the initial UMLs from LucidChart from earliest on in the project showing the most important fields and methods. There is an inheritance between the Board class and the ConnectionsModel class that is cut off between the two images.

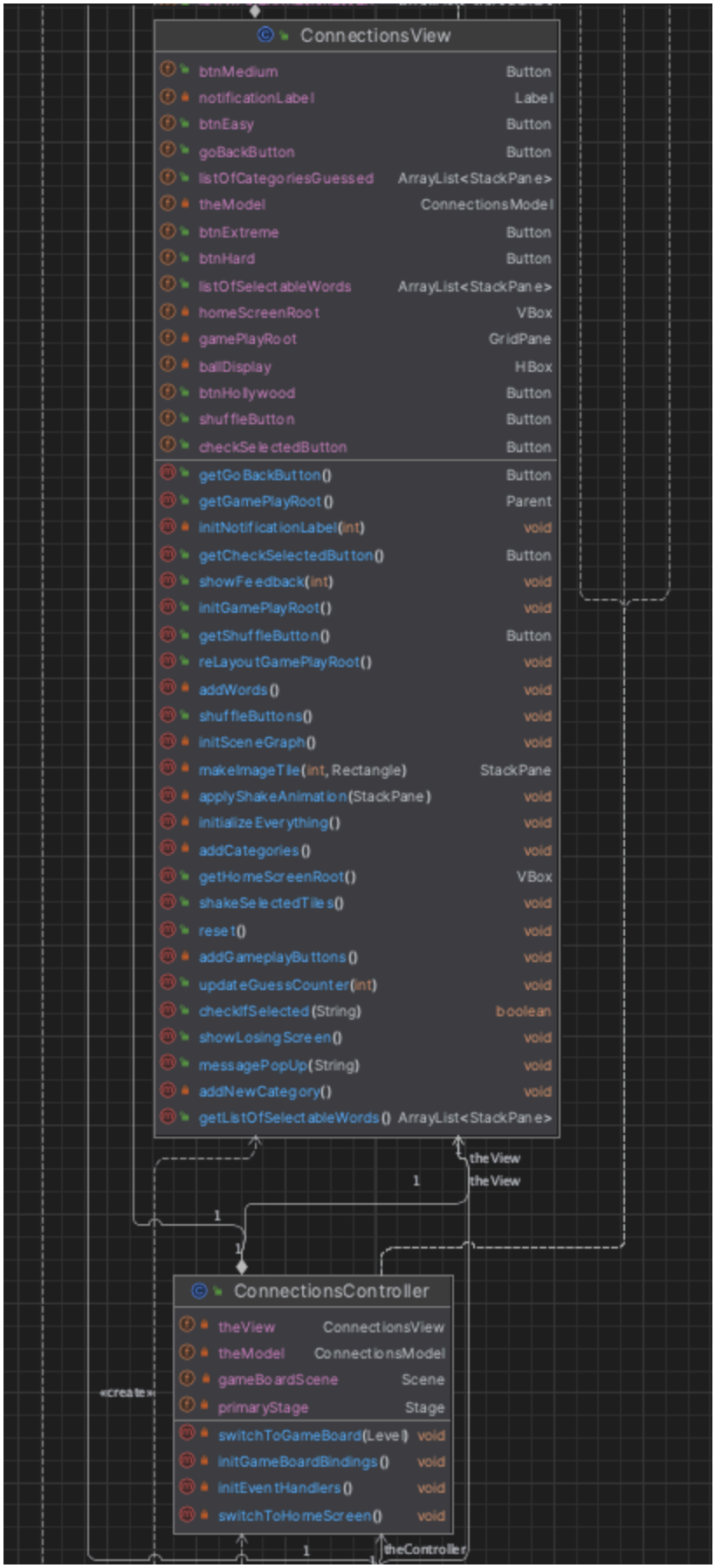


Final UMLs

These are the Final UMLs generated by IntelliJ for our project. We included the most important aspects of the generated UMLs. In our final design the view class took over a lot of functionality, but through refactoring we added a lot of methods that were more helpful than having one very large method.

This is our tile class. A lot of the other classes in our system have dependencies on the tile class as tiles make up the entire gameplay board. Tiles have properties such as changing colors depending on if they are selected and containing the words for each connection. This can be seen in the variables and methods of the tile class. The tile class along with the three classes shown below serve as the setup and the bones of our system. The board and GridMaker classes are used to set everything up and to maintain throughout the game. The ConnectionsModel class is what brings all these classes together in a cohesive manner to control all of the logic of the game.


GridMaker
hardModeMap TreeMap<String, String[]>
mediumModeMap TreeMap<String, String[]>
extremeModeMap TreeMap<String, String[]>
easyModeMap TreeMap<String, String[]>
hollywoodMap TreeMap<String, String[]>
makeMediumModeBoard() ArrayList<Tile>
makeHollywoodBoard() ArrayList<Tile>
makeExtremeModeBoard() ArrayList<Tile>
makeHardModeBoard() ArrayList<Tile>
makeEasyModeBoard() ArrayList<Tile>
makeBoard(TreeMap<String, String[]>) ArrayList<Tile>
1
1
Board
selected ArrayList<Tile>
level Level
words ArrayList<Tile>
getSelected() ArrayList<Tile>
select(int) void
checkSelected() int
getNumSelected() int
clearSelected() void
getLevel() Level
getWords() ArrayList<Tile>
1
board
«create»
1
ConnectionsModel
remainingCategories int
inGame boolean
board Board
MAX_GUESSES int
guesses int
chooseLevel(Level) void
reset() void
userFeedback() int
isInGame() boolean
guess() int
getBoard() Board
getRemainingCategories() int


ConnectionsView
btnMedium Button
notificationLabel Label
btnEasy Button
goBackButton Button
listOfCategoriesGuessed ArrayList<StackPane>
theModel ConnectionsModel
btnExtreme Button
btnHard Button
listOfSelectableWords ArrayList<StackPane>
homeScreenRoot VBox
gamePlayRoot GridPane
ballDisplay HBox
btnHollywood Button
shuffleButton Button
checkSelectedButton Button
getGoBackButton() Button
getGamePlayRoot() Parent
initNotificationLabel(int) void
getCheckSelectedButton() Button
showFeedback(int) void
initGamePlayRoot() void
getShuffleButton() Button
reLayoutGamePlayRoot() void
addWords() void
shuffleButtons() void
initSceneGraph() void
makeImageTile(int, Rectangle) StackPane
applyShakeAnimation(StackPane) void
initializeEverything() void
addCategories() void
getHomeScreenRoot() VBox
shakeSelectedTiles() void
reset() void
addGameplayButtons() void
updateGuessCounter(int) void
checkIfSelected(String) boolean
showLosingScreen() void
messagePopUp(String) void
addNewCategory() void
getListOfSelectableWords() ArrayList<StackPane>
the View
1
the View
1
1
ConnectionsController
theView ConnectionsView
theModel ConnectionsModel
gameBoardScene Scene
primaryStage Stage
«create»
switchToGameBoard(Level) void
initGameBoardBindings() void
initEventHandlers() void
switchToHomeScreen() void
1
theController

Here is shown our view and our controller class. The controller class accounts for all switching between windows in the game as well as all of the bindings and event handlers of the system. The bindings are for the tiles and the event handlers are for all the buttons, invoking the proper methods in the view class for each button press, respectively. The view class contains all functionality related to the display, the functions do not impact functionality but they do impact the visuals. For example the view class contains functionality for showing guesses remaining, creation of all buttons, shaking tiles when an incorrect guess is entered, shuffling tiles when the user wants to, and providing feedback to the user between guesses and upon the conclusion of the game.