Team 10 Design Manual

Introduction

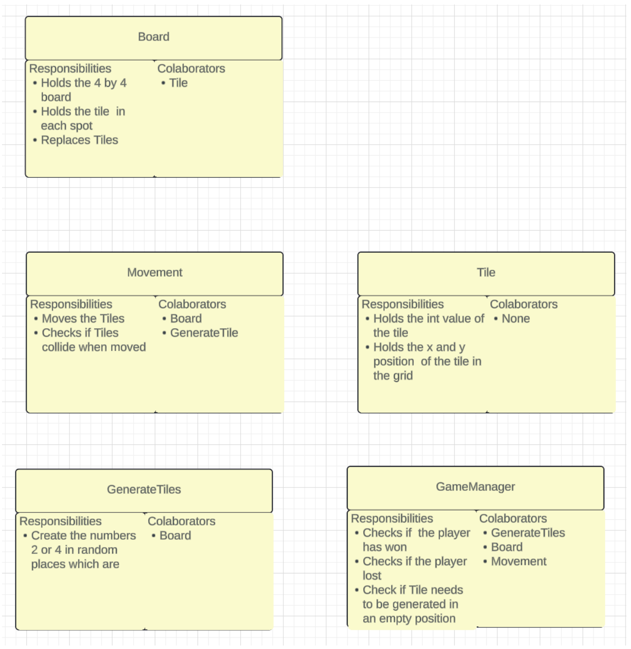
Our team's implementation of the game 2048 consists of several interconnected components that work together to create a fully functional game. The central hub of the game is the GameManager class, it creates instances of other classes and calls their methods to run the game and overall brings all the separate components together. One of these components is the board array which holds all the tiles generated and is bound to the grid in the User Interface. The User Interface was created using Scene Builder and is responsible for providing a visual representation of the game to the user. The UI consists of several directional buttons, a new game button, a grid pane filled with labels that represent tiles and a score label. Within the javaFX controller class there are event handlers that handle input as the user interacts with the UI. Event handlers are used whenever the user pushes a button, when the game ends, or when the WASD keys are pressed. Every move the system creates and places a tile with a value of either two or four in a random empty position on the board to make the game unpredictable. As the player progresses through the game they can see their score increase, the score counting in the system iterates through the board array and produces the sum of all the tiles present, allowing the player to mark their progress, as desired by the user Froukje Regeling who wants to be able to track her score and compete with friends. The GameManger is used to keep track of the current state of the game and whether the player can continue playing, won, or lost. If the player has either won or lost then the GameMaster methods are called within the javaFX controller and an alert is created on top of the board notifying the user to start a new game if they lost or to congratulate them for reaching the 2048 tile.

User Stories

We have several key user stories that we tried to implement into our game. They helped guide what features we wanted to add and what our final product strove to achieve. Although we weren't able to add all of the desired functionality to the game, if we had another sprint we would have been able to include several more user stories. Our first user story was Froukje Regeling who loves to compete with her friends in puzzle games. She wanted a way to track her score or the number of moves it took to reach 2048. We decided to add a way to track the score and not the number of moves because the number of moves doesn't properly reflect how good of a player you are due to the fact that tiles are generated in a random position and it is different for everyone. The score is the sum of all the values on the tiles so a player who is able to create the largest tile will have the best score, which more accurately represents that player's ability. Similar to the last user story, Mason White our second user story, wanted a way to track the number of moves he makes to minimize the number of errors he made and compare it to the minimum number of moves it takes to reach 2048. Again, because the tiles are randomly generated, there is no way to calculate the minimum number of moves it takes to reach 2048. Therefore we aren't able to track the number of errors a player makes, so we decided not to attempt to implement Mason's desired function. Our third user story was Jayden Brar who wanted to play a harder version of 2048. He wanted to be able to choose a power of two that the game would end on, for instance 4096 instead of 2048. We were close to implementing this but we ran out of time adding it to the UI. Although when a player does reach 2048 we allow them to keep playing until they are no longer able to make moves. This isn't exactly what Jayden wanted but it allows him to continue playing until he does reach his desired power of two. Our final user story was William Mortensen. He wanted a bigger board to play on. We weren't able to figure out how to implement this into javaFX because all of our labels in the grid pane were named objects which allowed us to bind it to the board array. If we had unnamed labels it would be much more difficult to bind so we decided not to implement this user story.

Object Oriented Design

Our project was broken up into two areas of focus: the logic behind the gameplay and the user interface created through scene builder. The logic component was composed of five main classes as well as several smaller classes that managed exception handling. The Board is responsible for holding the observableList of tile objects and is responsible for adding tiles and replacing tiles. Movement works with GenerateTiles and Board to make the tiles move, check for collisions, combine, and generate new tiles. The GameManager class is associated by aggregation to the movement and board class which allows GameManager to oversee the conditions of the board and alert the player when the player has either won or lost. The smaller classes not shown in the images to the left are exception handlers that ensure that new tiles aren't placed in positions that are already occupied, that the new tile isn't put in an index outside of the observableList, and to make sure that a new tile isn't added when the board is full. We strove to have



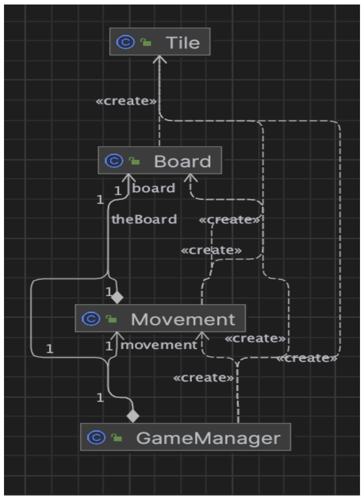
Board	
Responsibilities	Colaborators
• Holds the 4 by 4 board • Holds the tile in each spot • Replaces Tiles	• Tile

Movement	
Responsibilities	Colaborators
• Moves the Tiles • Checks if Tiles collide when moved	• Board • GenerateTile

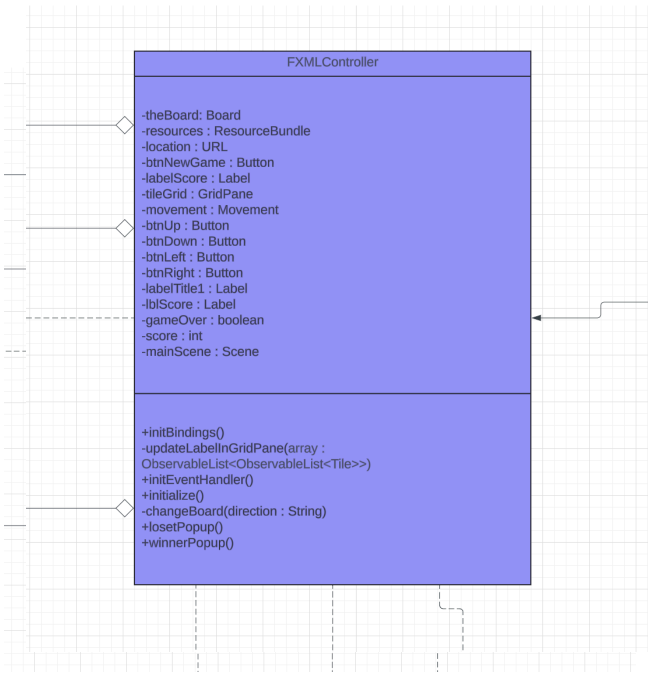
Tile	
Responsibilities	Colaborators
• Holds the int value of the tile • Holds the x and y position of the tile in the grid	• None

GenerateTiles	
Responsibilities	Colaborators
• Create the numbers 2 or 4 in random places which are	• Board

GameManager	
Responsibilities	Colaborators
• Checks if the player has won • Checks if the player lost • Check if Tile needs to be generated in an empty position	• GenerateTiles • Board • Movement



classes with high cohesion and low coupling in order to make our classes easy to manage and change. We believe our classes have achieved high cohesion and relatively low coupling; for the most part the methods in our classes all relate to the purpose of the class and aren't performing actions that should be done in a separate class. The javaFX portion of the program consisted of three classes: 2048.fxml, FXMLMain and FXMLController. 2048.fxml was generated using the application Scene Builder. FXMLMain loads in 2048.fxml from the resources folder and sets the primaryStage scene as the loaded 2048.fxml file and presents it in a window. FXMLController is where all the items in scene builder are bound to objects from the various other classes that deal



with the logic, and where the event handlers produce output that the player can see in the window. In the FXMLController two of the most important methods for the program were initBindings and initEventHandlers. These two methods connected the user interface to the logic of the game. InitBindings binds the observableList of tiles to the labels within the grid pane in Scene Builder. Building off of the binding the event handlers called an instance of the movement class to move tiles throughout the board when the WASD keys or the Up, Down, Left, or Right buttons were pressed in the UI. When the GameManager detects that the player has either won or lost an information alert will pop up with a message and instructions on what to do next. These alerts are created in the winnerPopup or loserPopup methods. Overall the javaFX and logic classes work together to bring the entire game together. The classes mentioned above were developed with proper OOD in mind and the team strove to create the best implementation of 2048 we could by adhering to the guidelines.

Future Improvements

In the future or if we had more time to work on our product we would like to see several other things accomplished. The team would like to refactor several of the classes to reduce redundant code and to improve the coupling of several of the classes. We would also like to see animation be added to the tile movement and the color of the tiles change as their value increases. We would also like to implement Jayden Bar’s user story because we were close to implementing it but ran out of time.