Marcia Gallant

Milestone 5 Justifications

I added a deepCopy() method to the Tile, RobotTurtle, GameBoard and Portal class, this is because we need to restore the old version of the gameboard if the player does write program but does not win. This makes a completely new game board that will not be changed when the turtle is getting moved in the original gameboard. The old version of the game board is updated to the original game board if a player wins.

There is an addIceWall() and addPortal() methods in the GameBoard class that allows the new objects to be added onto the board game.

There is an IceWallMelter class that was added and has the responsibility of melting the ice walls that are in front of the Robot turtle as long as there is no other object blocking it.

There is a teleport() method in the RobotTurtlePositionUpdater class, this allows the class to calculate the new position when the turtle to teleport if it steps on a portal tile to the correct corresponding portal tile.

There is a setPortalTilesAsNeighbours method in the TileAttacher class. This allows the TileAttacher class to properly keep track of all the tile neighbours because portal tiles only have the corresponding portal tile as their neighbour for every direction.

CardDeckCreator has new cardTypes added to it, so, we implement the new functionality of laser and function frog cards.

GameBoardModel now has addIceWall() and addPortals() that the controller can call to add the ice walls and portals that the user places on the game board. These new methods rely on the methods with the same name in the GameBoard class. The reason why we have them in the GameBoardModel is to not create any extra dependencies with the GameBoardController class and the GameBoard class. Instead, the GameBoardController class only depends on GameBoardModel methods.

The updatePlayersStatusInGame() method call in the moveTurtle() method in the GameBoardModel class has been moved. This is because in the advanced rules the turns do not switch after players pick one card. Instead, players pick multiple cards and have to press write program button to finish their turn.

Methods to add cards the player picked to the list of cards they want as their program or to the list of cards the want as their function frog program have been added to the GameboardModel. This allows us to keep track of all the cards the player picked. There are also clear methods, so, after every turn the lists get reset back to null. This is useful since we do not want cards they picked for their last turn to be playing again before the new cards they pick.

There is a writeProgram() in the GameBoardModel class, It moved the turtle according to the cards the user picked. If the user picked a Function Frog card, it moves the turtle corresponding to all the cards that were stored in the function frog list. This is very important for our program since it is making sure the turtle is moved in the appropriate place and in the correct order corresponding to the cards the user

picked. UpdatePlayersStatusInGame() call has been moved here since after the player writes the program, it is the next players turn.

UpdatePlayersStatusInGame() has been altered so that if the player won, then the oldGameboard now becomes a deep copy of the current game board.

The ObjectMover class has been altered so it knows what actions to perform if players pick the laser card. This is useful since we need to have the program execute the correct functionality if they pick the laser card. It also had a new method called checkIfTeleported() and teleports the turtle accordingly if the turtle stepped on a portal tile.

The WinChecker was altered so that the checkIfWon method now only checks if the player one and does not update the list of players who are playing the game. UpdateWhoWon method was added and it updates the list of players who are playing the game. Both of these methods were altered and turned into a for loop which loops over the list of players, instead of if statements which violated the open closed principle because it would be hard to extend if we wanted to add more players to the game because we would have had to add an if statement for 5 players to both methods. Now, it does not violate the closed principle because we just have to add an extra player to the player list.

In GameboardController the setUpGame method and getBoardSetUpfromUser have been altered to include ice walls and portals. Also, in startGame method, it now also displays instructions before you begin the game, so, you know what you are doing. Also, setUpCardsToBePicked() has been altered to add mouse listeners to 3 new buttons, one that when clicked, the cards that the user selects afterwards are stored as their main program, one that when clicked, the cards that the user selects afterwards are stored as their function frog program. One when clicked, writes program, which means your turn is done and the turtle moves accordingly. If the user did not win, it forces you to use the bug card which resets everything. The addMouseListenersToACard method has been altered to include a way to completely reverse the program if everything needs to be reset.

In GameBoardController, there are new methods added for adding mouse listeners to the new 3 buttons. This is useful because we need to store what actions our program needs to take if they are clicked.

In GameBoardController, validateCardChoice method is gone and has been replaced with 3 different methods, validateCardChoiceForProgram, validateCardChoiceForFunctionFrog and validateProgram. The first 2 methods are called when the user picks a card to be added to their program or when the user picks a card to be added to their function frog program, respectively.

validateProgram makes the model write the program and the view to display the current gamebord. It also checks if the player has won and if not, forces player to use bug card and for them to revert back to their original position. It also makes sure the right cards are enabled or disabled and clears the list of cards for the program and function frog in the model at the end of the method. This method is very important because it is responsible for making sure the model moves the correct turtle according to the cards the user picked.

In GameBoardController, a method called findPlayersTurtle has been added because we used to just use if statements to check through all the player options and this violated the open closed principle because it makes it hard to extend if we must add many if statements to multiple methods to extend our

program to allow more people to play. It looks over the player list to find the current player, so, in order to extend it, we would simply need to add the additional players to the player list. This makes it not violate the open closed principle.

In the GameBoardConverter class, a new method for converter cards to Strings has been added. This is useful because the display class will get the lists of cards for the program and for the function frog and display them on the screen, so, after the turtle moved, the user can look back on what moves they made on the previous turn. It is important to convert them to Strings since the display should not have any dependencies on classes in the model, like the Card class. There is also a helper method added called addSeparator. It is responsible for adding a way to separate each Card string to make identifying the cards the user played easier.

There is a new enumeration class Mode, what Mode the GameBoardController class is in determines whether to cards are stored in the list of cards for the program or the list of cards for the function frog. This is important because we need to be able to determine where to store the cards selected by the user, otherwise, we would not know which cards to play if the user picked the function frog card in their program.

In the GameBoardDisplay class, JPanel p4 now adds the extra 3 JButtons (the ones mentioned previously) and has getter methods for them. It also has a new popup method for the instructions and a new popup method to display the String version of what cards the user picked to be in their program and what cards the user picked to be in their function frog program.