Marcia Gallant

- Separated the responsibility of updating the crate's position on the board from GameBoard to its own class: CratePositionUpdater.
- Separated the responsibility of setting al the neighbours for all the tiles in the gameboard to its own class: TileAttacher.
- Separated the responsibility of updating the Robot turtle position to its own class: RobotTurtlePositionUpdater.
- Separated the responsibility of calculating what the new position if one exists from the Robot Turtle class which deals with keeping track of the turtles position and made a new class RobotTurtlePosition calculator for the responsibility
- Separated the responsibility of checking if the turtle was able to move successfully (meaning no object or such was in the way) from the GameBoard class and made its own class: GameBoardMoveChecker
- I kept checkIfObjectPresent since that is just a helper method to creating new stone walls and crates which is the responsibility of the GameBoard class inside the GameBoard class.
- I also made a design decision to keep robotJewelClaimed method in the GameBoard class because it is highly unlikely to change and quite overkill to make a class just for a method that makes it so two turtles cannot acquire the same jewel.
- I separated the responsibility of finding out what the invalid positions for crates and stone walls are from the gameboardmodel to its own class InvalidPositionCalculator. I also changed the way we look for invalid positions by storing all of the positions of robot turtles and jewels in a single array and it can now be looped through and it adds the invalid positions. This follows the open closed principle and makes it easier to extend if we ever needed to have more invalid positions at the start.
- I made a design decision to leave the moveTurtle method in the GameBoardModel class because it's an essential method that the controller needs access to and the only class the controller should depend on is the GameBoardModel class.
- RobotJewelCreator now creates the robot jewels and follows the open-closed principle where it can easily be extended and don't have to add if statements like you would have had to before.
- CardDeckCreator is now responsible for creating all the cards and it now follows the open-closed principle where all you need to do is add more card types to the private variables, so, it's easily extendable.
- Playerswitcher responsibility is to find and return the next player. I have also adapted it to the open-closed principle and used a for loop instead of a bunch of if statements.
- WinChecker responsibility is to check if a player has won the game and if all players have won the game.
- ObjectMover responsibility is to make sure the turtle and any other objects are moved as desired or not moved if it's not possible. A design trade off that I made is that we need if statements to implement different functionality for all of the different cards. So, it's not really following the open-closed principle since if any cards are to be added we need new if statements to implement their functionality but this is a crucial part of the game. Each card does something different, so, if statements are necessary for our game to work.
- RobotTurtleCreator responsibility is to create all the robot turtles

- DisplayNotificationOnWhoseTurnItIs in the GameBoardController has been remade based on the open-closed principle so if more players must be added into the game, it's easy to extend it.
- ValidatecardChoice in the GameBoardController has been remade based on the open-closed principle, so, if we need to extend the game to have more players it's easily extendable.
- drawCurrentGame has been changed so it no longer have to check each tileType, now it only must tell if it's a normal tile or not. The reason why it must still check if it's a normal tile or not is because there is a different normal tile image for each square on the board whereas the other tiles mostly only have 1 version and can go on any square or have a couple versions based on which direction they're facing and can still be on any square on the gameboard. In order for my program to work, the normal tiles have identifier numbers which correspond to white spot they are supposed to be in which none of the other tiles have. In order to make the tiles and other objects drawn on the gameboard in only one method would mean I would have to add identifier numbers to each object and have one for each square even though the pictures would be all the same for the objects. This would bloat the imgs folder and seems like unnecessary work. Therefore, I've made a design trade-off to check for the normal tiles or not. It still mostly follows the Liskov substitution principle where all of the subtypes of the Tile class do not need to be known and it is still relatively easy to extend since if I add another tile type, I do not need to check its type.
- I've also updated drawCards method and made it easier to extend by doing a for loop instead of if statements. Now, if we add more cards, we can easily extend it by adding it to String[] cardTypes
- All of the new classes I made were to satisfy the single responsibility principle which is that each class should only have 1 responsibility. This is because if a class only has 1 reason to change it makes the code less fragile since other parts of the code won't be breaking if something else needs to change and then we'd have to fix it all. Now, we only have to focus on fixing that one part if something breaks.
- I've added interfaces such as modelDataInterface which holds all the methods the classes in the controller package needs. ControllerModelDataInterface holds all the methods from the GameBoardModel which the controller needs and ConverterModelDataInterface holds all the methods from GameBoardModel which the converter needs. This useful because it's not violating ISP anymore since we are not forcing the controller package classes to depend on any more than what they need. It also creates a dependency inversion which means it no longer has a compile time dependency and only has a runtime dependency. That way it doesn't have to depend on all the small details.
- I've also added an interface called displayInterface which the controller now depends on instead of GameBoardDisplay. This way the controller no longer has to depend on anymore than necessary and it also creates another dependency inversion.
