Ethan La Rue

Robert Gerardi

Niraj Patel

Sahil Patel

<https://github.ncsu.edu/jtuck/309-Final-Project>

**The Game: Sorry!**

Sorry is a game in which 4 players have to get all of their 4 pawns into their own safe zone. Players rotate turns by drawing a card which can be 1-5, 7-8, 10-12, or a sorry card. Once you draw a card you must move your pawn to the appropriate place. If you run out of cards, you reshuffle them and place them in the center of the board. If you land on a slide that is not your color you may slide to the end of it. If you draw a 4 or a 10 you may choose to go backward or forward. The direction to move the pawn from home is in a clockwise direction.

If you land on a space that is occupied by a different color, then that color is sent back to its start, which is its starting position, and the pawn that just took the turn occupies that space. If you are taking turns and land on a space that is already occupied by the same colored pawn, then you forfeit your turn.

If at some point in the game you can’t move any of the pawns, you have to forfeit your turn but you can’t forfeit a turn if you have enough spaces to take a turn. If you land on a slide that isn’t your color, you can move your pawn where the slide ends. When this happens, all the pawns that might be located in the range of that slide are sent back to their starting position, including the pawn of its own color/

There is a safety zone for each color at the end, only the pawn with the same color can enter that safety zone, you can’t enter this safety zone via a backward move but you may choose to move your pawn backward after entering the safety zone if you want. The first one to send all of the pawns home wins the game.

In the game we implemented, we asked the user if they wanted to participate in the game. If the answer is no, we stimulate a game of all computer players, however, if they say yes, we then ask them to pick the pawn color and assign the other remaining 3 colors to the computer.

Failsafes were implemented in case the human player submitted a wrong choice. It will kick back and tell the user that they inputted a wrong choice and allow them to input a new choice.

**The overall design of the product**

Throughout our program, we use many unique data structures, practices, and other means to uphold good and effective execution time and modifiability. The structure of the code goes like this: the user is prompted if they would like to play. If they select no, then the game fully runs to completion using computer players. If you select yes, the game allows you to choose your color and then begins your turn. After the human player has performed their move, picking a pawn to move if they are allowed, then the computer players automatically play, displaying their movements to the screen. The game is then played till one player gets all four pawns in the home.

R1 is achieved through the use of objects for the deck, players (human and computer), and the board, and can be found mostly in GameObjects.h. R2 is achieved by first determining if the user would like to play. If not it leaves all toggles to true signifying that all instances of the player class will be computers. If this is not the case and the player chooses to play it will set the color of their choice to false and divvy up the remaining colors. R3 is satisfied by displaying the turn for each player onto the terminal. Each piece’s location is displayed on the screen, as well as the card pulled and what the player chose as their movement.

We will implement Encapsulation by having different classes with their own functions inside classes. We have abstraction implemented by making each player only access their own pawns and not of different colors. We also implement Inheritance, by making a HumanPlayer and ComputerPlayer class inherit from a player class. C2 is met with the Player and the Board Object will be implemented with the Rule of Zero. The Card Deck Object will be implemented using the Rule of Zero since these three elements are not needed. Other good design principles include proper names for functions, variables, and classes. Lastly, we implemented polymorphism by creating one function that is used for multiple uses. The PlayGame function is virtual in the Player Class and is overridden in the Computer and Human player classes. C3 is taken care of when our code uses virtual functions for the player object. Since each player will have the same components such as color, number of pieces, and the action of drawing a card, we will need to use a virtual function in order to implement each player effectively. HumanPlayer and ComputerPlayer have virtual functions that override the player member functions. These functions are used to play the game for the HumanPlayer and ComputerPlayer, which have different functionality based on the type of player. C4 is achieved by using a vector for the card deck object. This will help us create a real-life scenario for drawing cards. This will work by having a counter that increments each time the player draws a card. This will increment until the counter is at the last spot in the array and then will reshuffle and reset the counter. Finally, C5 is accomplished. GitHub is up and shared with all group members. We have also added Dr. Tuck as the owner of the repo and added the TAs as collaborators.

**Collaboration**

**Ethan La Rue**: Responsible for implementing the deck class which housed and managed all of the cards and also the shuffle feature that used the deck class and returned a shuffled version of it. He was also responsible for the majority of the write-up to be submitted.

**Robert Gerardi:** Implemented the classes, structure, and implementation for the computer player movement, pawn, and player initialization, and created the board and board space, classes. Edited the writeup to reflect new changes and update for consistency. Although most of the commits were by me, a majority of the time, we were working on them together using group code on Client. I created the README.txt.

**Niraj Patel:** Implemented the Player and Pawn classes, structure, and movement for the human player. Also, created all of the HumanPlayer class, including the structure and inheritance of the Player class. Created the prompting for the human player and developed failsafes for user interaction. Edited the write-up to show the necessary requirements that were needed for the project.

**Sahil Patel**: Implemented the turn-based structure for the Sorry Board, which includes the main function. Created how the end of the game will function, along with some of the polymorphism aspects with the implementation of virtual functions. He was also responsible for the introduction and overview of the report.