**Proposed project title: Parcheesi**

**Description**: This program will be a Python version of the game Parcheesi for two human players and a computer player. The program will be in the console and include a board that shows player position, safe spots, and the entire board using ASCII.

The rules of the game will be as follows:

* All players start off in jail and have three opportunities per turn to roll doubles to get out of jail. If doubles are rolled, player moves out of jail and may roll again.
* If a player rolls doubles after they are out of jail, they will get another turn
* If a player lands on a space where another player is, the player who was already on that space will be captured return to jail
* Every 5 spaces on the board will be safe spaces, where a player cannot be captured
* Once a player reaches space 63, they are safe for the remainder of the game
* Once a player reaches space 63, they can only roll one die
* To win the game, a player must reach space 69 by an exact roll

**Intended user:** The intended users will be two casual users looking to play a console game against a computer player.

**What problem is the project trying to solve:** The project is intended to be a fun game for users to play in their free time, especially those who enjoy parcheesi but do not have a board, or maybe for two people who do not have a third player and wish to play the game against a computer.

**Which technologies will be needed:** This project will be created using object oriented programming, so files will be used to create a class and subclasses for the human and computer players, which will then be imported into the file containing the main function and turn sequence. The program will be completely in the console, and the board will be printed during and after each turn to show player advancement and whether a player is on a safe spot or not. Objects will have methods for rolling, safe status determination, and the capture of pieces. They will have the attributes “name” and “danger” (on a safe space or not). The computer subclass will have automatic rolls that do not require a player to press enter, but that will simply print results.

# **Use Case Analysis:**

New users will read the rules likely, as parcheesi is not one of the most common board games here in the US. They will then move on to entering their names as players and right into the game, where they press enter to roll. Roll types are determined based on the status and position of a player on the board. Experienced users will simply go straight into the game and follow the same steps as the inexperienced user after skipping the rules. Because of the nature of board games and how rules are determined by position and status, there is not much input required by the user other than the corresponding dice rolls, and the randomness of these rolls will determine what actually happens when a user is playing the game. Users having time and wishing to do so will have an option at the end of the game to play again or exit the program. Aside from the rules, most interactions with the program will be the same, but with random results from the dice.

# **Data design:**

The data that this program is really about is how to roll the dice and how many times, where the player ends up on the board, whether or not they are safe or in danger of capture, whether they capture another piece, or whether they are captured and return to jail. The best way to represent this data will be using objects for both human players and computer players. These players will all need to perform specific dice rolls depending on whether they are in jail, on the board in the normal area, or nearing the end and rolling with a single die. They will also need a method that lets them reroll when they roll doubles, and an attribute of landing on a safe space or not.

|  |
| --- |
| Object class: Player |
| Attributes:  Name  Danger (whether player is on a safe space or not)  Position  Player Indicator (1, 2, or C accordingly)  Wins |
| Methods:  Print board showing player  Check to see if player is on a safe space and update status accordingly |

|  |
| --- |
| Subclass: Human Player |
| Attributes:  Name  Danger (whether player is on a safe space or not)  Position  Player Indicator (1 or 2 accordingly)  Wins |
| Methods:  Roll (determines which roll to perform)  Roll while in jail (3 attempts to roll doubles) with enter to roll  Normal 2 die roll with enter to roll  Reroll 2 dice for doubles with enter to roll  Normal 1 die roll with enter to roll  Print board showing player |

|  |
| --- |
| Subclass: Computer Player |
| Attributes:  Name (randomly selected from a list)  Danger (whether player is on a safe space or not)  Position  Indicator: (C)  Wins: |
| Methods:  autoRoll(Determines which automated roll to perform)  Roll while in jail (3 attempts to roll doubles)  Normal 2 die roll  Reroll 2 dice for doubles  Normal 1 die roll |

This data will need to be output to the user in a readable and non-exhausting way. The ASCII board will be displayed after every roll, showing player position, designator, and the location of safe spots. Player name will be displayed upon their turn, and each human player will appear on the board as 1 or 2, which will appear next to their name in parenthesis for clarity. Whether or not a person is in jail or safe will be explicitly written on the screen during a player’s turn (also when they successfully roll to get out of jail). They will be informed when they roll doubles as well and when they can reroll for doing so. Spacing will be used between each turn to help the user to see that it is their turn and not be distracted by things that have happened earlier with everything running together.

# **UI Design:**

This program will use the console and ASCII art to display a gameboard during each player’s turn. Normal spaces will be shown as “.”, safe spaces as “-”, and player position will be shown using the corresponding indicator (1, 2, or C). Below is an example of the boards printed:

-....-....-....1....-....-....-....-....-....-....-....-....------

-....-....-...2-....-....-....-....-....-....-....-....-....------

-....-....-....-....-....-....-.C..-....-....-....-....-....------

# **Algorithm:**

**Object Class: Player**

|  |
| --- |
| Object class: Player |
| Attributes:  Name  Danger (whether player is on a safe space or not)  Position  Wins |
| Methods:  checkDanger: Check to see if player is on a safe space and update status  printboard: prints the board with the player’s position indicated |

|  |
| --- |
| Subclass: Human Player |
| Attributes:  Name  Danger (whether player is on a safe space or not)  Position  Wins |
| Methods:  getName: input player name and set as attribute  roll: determines which roll to perform  rollJail: Roll while in jail (3 attempts to roll doubles) with enter to roll  diceRoll: Normal 2 die roll with enter to roll  reroll: Reroll 2 dice for doubles with enter to roll  dieRoll: Normal 1 die roll with enter to roll  checkDanger: Check to see if player is on a safe space and update status  printboard: prints the board with the player’s position indicated |

|  |
| --- |
| Subclass: Computer Player |
| Attributes:  Name (randomly selected from a list)  Danger (whether player is on a safe space or not)  Position  Wins |
| Methods:  autoName: Chooses random name from a list  autoRoll: Determines which automated roll to perform  autoRollJail: Roll while in jail (3 attempts to roll doubles)  autoDiceRoll: Normal 2 die roll  autoReroll: Reroll 2 dice for doubles  autoDie Roll: Normal 1 die roll  checkDanger: Check to see if player is on a safe space and update status  printboard: prints the board with the player’s position indicated |

**INITIALIZERS**: The parameter name will be set by the user. The parameters position and danger will start out at default values of 0 and be updated by corresponding methods based on dice rolls.

**METHODS:**

roll Method:

1. If player’s position is 0, call rollJail method
2. If Player’s position is greater than 0 and less than 63, call diceRoll function
3. If player’s position is greter than or equal to 63, call dieRoll function

rollJail Method:

1. Take input of null to roll dice and print “press enter to roll”
2. Create two random numbers between 1 and 6
3. If the numbers are the same, end method and put player at space 1 and call diceRoll method, print “NAME rolled doubles, NAME is out of jail!”
4. If the numbers are different, repeat all steps for a total of three times

diceRoll Method:

1. Take input of null to roll dice and print “press enter to roll”
2. Create two random numbers between 1 and 6
3. Add numbers together and advance object that number of spaces
4. Print “NAME Rolled a (sum of dice)”
5. If random numbers are the same, call reroll method

Reroll:

1. Take input of null to roll dice and print “You rolled doubles, press enter to roll again”
2. Create two random numbers between 1 and 6
3. Add numbers together and advance object that number of spaces
4. Print “You rolled a (sum of dice)”
5. If random numbers are the same, repeat all steps until this is not the case

dieRoll:

1. Take input of null to roll dice and print “press enter to roll”
2. Create two random numbers between 1 and 6
3. If the number rolled is equal to or less than the remaining spaces, advance that many spaces, Print “You rolled a (die roll)”
4. If number rolled is larger, player remains at the same space as before the roll, print “You overrolled, you do not advance”

checkDanger:

1. If player is at a space is less than 63 divisible by 5 change status to safe
2. If player space is less than 63 and not divisible by, change status to danger
3. If player space is greater than or equal to 63, change player status to safe

printBoard:

1. Set a counter “i" equal to 0
2. While i is less than 69,
   1. if i is less than 63, not equal to the player’s position and i is divisible by 5, print “.”
   2. Or else if i is not equal to player position and divisible by 5, print “-”
   3. If i is greater than or equal to 63, print “-“
   4. Or else print player indicator (1, 2, or C)

getName:

1. Ask player to input name, set name as name attribute

**Note**: all methods for subclass computer player beginning with the word “auto” will be identical to those of the human player, except that they will not include a “press enter to roll” input function

Functions:

rules function:

1. Ask player to press r to see rules or to press b to begin playing
2. If player enters r, print rules
3. If player enters b, call main
4. If player enters something different, return to step 1

playAgain:

1. Ask user if they would like to play again y or n
   1. If y, call main function

Call rules function

Main Function:

Call getName for player1

Call getName for player2

Call AutoGetName for Computer

* + 1. Set player position for each player to 0:

While player 1 and player 2 and computer position are all less than 69:

#Player 1’s turn

* 1. Print player “name’s” turn
  2. Call a.roll() method
  3. Call a.checkdanger() method
  4. If player1 position is equal to player2 position and player2 danger status is equal to danger, set player2 position equal to 0
  5. If player1 position is equal to computer position and computer danger status is equal to danger, set computer position equal to 0
  6. Print board for all players
  7. If player1 position is greater than 68, print player 1 wins, print wins for all players and break loop

#Player 2’s turn

* 1. Print player “name’s” turn
  2. Call b.roll() method
  3. Call b.checkdanger() method
  4. If player2 position is equal to player1 position, and player 1 danger status is equal to danger set player1 position equal to 0
  5. If player2 position is equal to computer position and computer danger status is equal to danger, set computer position equal to 0
  6. Print board for all players
  7. If player2 position is greater than 68, print player 2 wins, print wins for all players and break loop

#Computer’s turn

* 1. Print player “name’s” turn
  2. Call c.autoRoll () method
  3. Call c.checkdanger() method
  4. If computer position is equal to player1 position and player 1 danger status is equal to danger, set player1 position equal to 0
  5. If computer position is equal to player2 position and player2 danger status is equal to danger, set player2 position equal to 0
  6. Print board for all players
  7. If computer position is greater than 68, print computer wins, print wins for all players and break loop

Ask player if they want to play again, Y or N, if Y, return to step A.

End of main function: