**Duell C++ Manual**

**Bug Report:**

No bugs to be reported.

**Feature Report:**

**Features Not Implemented:**

All features of the game are implemented.

**Additional features include:**

Keeping track of alive human dice.

Keeping track of alive computer dice.

**Description of Data Structures/Classes**

**Classes:**

Board:

The board class holds the private variables games, diceHuman, and diceComputer which are all maps. These maps are used in order to keep track of where board pieces are located and to whom they belong to. The board class handles moving dice and keeping track of the die and open spaces.

BoardView:

The boardView class handles displaying the game board. It goes through the games map in order to see whether to print a die or a space. If there is a die to be printed, it checks whether it is a 1 for a computer or 2 for a human and prints out the top and right of the die accordingly. The only purpose of the boardView class is for printing.

Computer:

The computer class inherits from the player class and implements the computer player strategy and validates that a move is legal. The computer strategy is to attempt to win the game first by capturing the 1x1 die or the special tile, then the computer will attempt to defend its own 1x1 die or special tile if either can be captured, the computer will then attempt to defend any other die that may be captured, afterwards the computer will attempt to capture any enemy die, and if none of the previous moves are possible the computer will just attempt to move to a possible open space. The move is decided in the function decideMove() which is a virtual function inherited from the player class. The computer class validates that a move is legal before making the move on the board.

Game:

The game class handles each round in a game and its primary responsibility is handling the serialization of a current game. The game class handles asking if a player would like to save, saving a game, and reading in a from save file. The game class also handles deciding which player will play first in a round.

Human:  
The human class inherits from the player class and implements the human player help strategy when the human player asks for help making a move and validates that a move is legal. The human strategy is to attempt to win the game first by capturing the 1x1 die or the special tile, then the human should attempt to defend its own 1x1 die or special tile if either can be captured, it will then attempt to defend any other die that may be captured, afterwards the human will attempt to capture any enemy die, and if none of the previous moves are possible the human will just attempt to move to a possible open space. Which move to be suggested is decided in the function decideMove() which is a virtual function inherited from the player class. The human class validates that a move is legal before suggesting the human should make a move.

Player:

The player class handles the player names, dead dice, and alive dice. It is used as a base class for the human and computer class, and utilizes a virtual function called decideMove() in order for the human and computer to be able to pick which move should be made.

Tournament:

The tournament class keeps track of the rounds won by each player and also determines who won the match. It is used to update the number of wins a computer or human player has, either by winning a round or reading in the wins from a serialized file.

**Structs:**

Dice struct:

The dice structure is a part of the board class and it used to keep track of the die. The die have the top, right, row, and column as part of the structure. Each time a die is moved the row, column, top, and right of the die have to be updated accordingly in order to have the die in the correct position.

**Maps:**

diceComputer:

The diceComputer map is used to store the computer dice, which are mapped to the row and column coordinates.

diceHuman:

The diceHuman map is used to store the human dice, and are mapped to row and column coordinates.

Games:

The games map stores integers which represent game pieces, it stores a 0 to represent a space, 1 to represent a computer player die, and 2 to represent a human player die. The integers are mapped using row and column as coordinates.

**Log:**

**September 9, 2016:**

- Added main.cpp file.

- Created a roster[2] array in main to hold players and initialized a human player for testing purposes.

-Added new private member to board class, map<int, map<int, int>> games to hold board coordinates. Stored integers represent game pieces, 0 is a space, 1 it's a computer dice, 2 is a human dice.

-In the default board() constructor the board is initialized using the games map. Using nested for loops the map is initialized beginning at the top left corner (8, 1) and ends at the bottom right corner (1, 9). If a row is 8, then the map has a 1 integer to represent a computer piece. If a row is a 1, then it holds a 2 to represent a human piece. All other rows have 0s to represent blank spaces.

-Added a displayBoard() function, which uses for loops to display games map to the screen and also is used to display column and row numbers to the screen.

(1 hour)

-Tested displayBoard() function to see if the games map will display correctly to the screen, which was successful.

-Created a struct called dice with the properties of, top, right, row, column, and owner.

-Added two new maps to board class, map<int, map<int, int>> diceHuman and map<int, map<int, int>> diceComputer.

- Created a function called initialHumanDice() which initializes the original row of dice for the human player and stores them in the map diceHuman using the row and column coordinates as keys.

-Created a function called initialComputerDice() which initializes the original row of dice for the computer player and stores them in the map diceComputer using the row and column coordinates as keys.

-Updated displayBoard() to display either the values at either the games map, diceHuman map, or diceComputer map. Display values are based on games, which keeps track of which spaces are blank or occupied.

-Tested new displayBoard() function and was successful.

(1 hour 30 mins)

Total: 2 hrs 30 mins

**September 10, 2016:**

-Created isHumanDice() and isComputerDice() function to tell if a space is currently occupied by a human or computer.

-Created isDice() and isSpace() to tell if a space is empty.

-Created getNumAtSpace() to tell what number is at the space in games.

(1 hr)

Total: 1hr

**September 11, 2016:**

-Added movementDirectionHuman() which calculates whether the current dice piece is being moved forward, backwards, left, right, or some combination of the two.

- Added moveFrontal(), moveBackwards(), moveLeft(), moveRight() in order to move dice appropriately.

- Edited moveDiceHuman() function in order to correlate to different directions and call appropriate movement functions.

(1 hour)

Total: 1hr

**September 12, 2016:**

-Implemented a checkCollisionHuman() function in board class. Tests to see if a path to a set of coordinates is blocked. Currently does not check if the last piece is an enemy piece capable of being captured.

-Implemented isValidMovementHuman() takes the top number of the dice and checks to make sure the total spaces being moved are the same as the top of the dice.

(1 hour 30 mins)

Total: 1hr 30 mins

**September 13, 2016:**

- Added a statement asking whether to move front or lateral first when deciding a move.

- Added an if statement in checkCollisonHuman() to determine which directions to check first, front lateral or lateral front.

-Added getHuamnTop(), getHumanRight(), getComputerTop(), getComputerRight() to get the numbers on the dice.

-Moved displayBoard() to the boardView class in order to separate the classes into more organized classes and keep the logic and view separate. Deleted displayBoard() from the board class and added displayGameBoard() to boardView class.

(1 hour)

-Added setName(), getName(), checkAliveDice(), checkDeadDice(), updateDeadDice() to the player class.

- Added checkKeyPiece() and checkKeySpace() to the player class.

-Added killComputerDice() to the board class.

-Added checkLastPathSpaceHuman() to the board class in order to check if the last space in a movement is an enemy dice, your own dice, the special space, or the special piece. If it's the special piece you win the game.

(1 hour)

Total: 2hrs

**September 14, 2016:**

- Added decideMove() to the player class and computer class. Decided to try to make computer kill the special die or move into special space first if at all possible when choosing a move.

- Added isValidMovementComputer() and checkCollisionComputer() to the board class.

- Added getSpecialHumanDiceRow() and getSpecialHumanDiceColumn() to the board class to ascertain where the dice is located.

(1 hour)

-Added moveDiceComputer() to the board class.

-Added movementDirectionComputer() to the board class.

-Added owner to the moveFrontal, moveBackwards, moveLeft, and moveRight functions.

-Tested moveDiceComputer() to see if the dice would move correctly. Currently the dice checks for forward collisions and if there are none moves the dice available forward.

(1 hour 30mins)

Total: 2hrs 30mins

**September 15, 2016:**

-Commented out the block where the human player is asked for a name and a dice is rolled to make testing faster and more consistent. Human player goes first.

-Added killHumanDice() to the board class.

-Added resetGame() to the board class.

-Added initializeGames() to the board class to initialize the games map back to the original.

-Tested if the game could tell if a human player won, testing was successful.

(1 hour 30 mins)

-Added updateHumanWins(), updateComputerWins(), and updateRoundsPlayed() to the tournament class.

-Added getRoundsPlayed(), getHumanWins(), and getComputerWins() to the tournament.

-Added resetTournament() to the tournament class.

- Did not implement or test any of the functions in the main program.

(30 mins)

Total: 2hrs

**September 16, 2016:**

-Moved movementDirection(), checkCollision(), isValidMovement(), and checkLastPathSpace() to the human class because they involve input validation.

- Deleted the above functions from the board class.

-Moved movementDirection(), checkCollision(), isValidMovement(), and checkLastPathSpace() to the computer class because they involve input validation.

- Deleted the above functions from the board class.

(1 hour)

Total: 1hr

**September 17, 2016**

-Created decideFirstPlayer() and added it to the game class. Decides whether the computer or the player gets to play first.

-Created moveToSpecialEnemyDice() in computer to compute if it's possible for a computer dice to kill the 1x1 dice to win the game.

(1 hour)

Total: 1hr

**September 18, 2016**

-Added moveToSpecialEnemyTile() to computer class to move to the enemy's special tile if it's open in order to win the game.

**-**Added defendSpecialDice() to the computer class. The goal is to see if any opposing dice can take over the special dice and if they can then move a dice to save it. If there's more than one there's nothing you can do to save it anyway.

(1 hour)

Total: 1hr

**September 19, 2016**

-Added defendSpecialTile() to the computer class so the tile is prevented from being taken over.

-Added killAHumanDice() to the computer class so the computer will kill a dice if it is possible to do so.

(1 hour)

-Added defendComputerDice() to the computer class to defend a dice from being captured by an oppent.

-Added moveToOpenSpace() default if no other movement strategies work, just move to an open tile if possible.

(1 hour)

Total: 2hrs

**September 20, 2016**

**-**Added setGamesPiece(), setComputerDice(), and setHumanDice() to the board class.

-Added gameSerializationIn() to the game class in order to read in from a file a current round of a game.

(1 hour)

Total: 1hr

**September 21, 2016**

-Added updateSerialWinsComp() and updateSerialWinsHuman() to the tournament class.

- Added print statements to the computer class to say where the dice was moved from and where the dice is going.

(1 hour)

Total: 1hr

**September 22, 2016**

-Tested serialization with a static text file. Need to add asking for the file name from the human.

-Added determineWinner() to the tournament class in order to see if the human or computer won the tournament.

(30 minutes)

Total: 30mins

**September 23, 2016**

-Added help for the human player, added moveToOpenSpace() and moveToSpecialEnemyDie().

-Added moveToSpecialEnemyTile();

(1 hour)

Total: 1hr

**September 24, 2016**

-Added defendSpecialDice(), defendSpecialTile(), defendHumanDice(), killAComputerDice() to the human class.

(1 hour)

Total: 1hr

**September 25, 2016**

-Added serializationOut() to the game class.

-Added askForSerialization() to the game class.

(1 hour)

Total: 1hr

**September 26, 2016**

-Made changes to the defendHumanDice() function in the human class to better check the pathing.

-Made changes to defendComputerDice() function in the computer class to better check dice pathing.

(1 hour)

Total: 1hr

**September 27, 2016**

-Made changes to checkCollison() in the human class to more accurately check for collisions in the pathway.

-Made changes to checkCollison() in the computer class to more accurately check for collisions in the pathway.

(1 hour)

Total: 1hr

**September 28, 2016**

-Did testing on the serializationOut() and serializationIn() functions.

-Made a change to the serializationIn() function so it correctly assigns the currentPlayer.

(1 hour)

Total: 1hr

**September 29, 2016**

- Added input validation as function getHumanInput() in main while asking human which die to move and to where.

-Added askToPlayAgain() function to main to help lessen repeated code.

-Added file name validation to serializationIn() and serializationOut() functions in game class.

(1 hour)

Total: 1 hr

**September 30, 2016**

-Fixed an error with getDeadDice() returning the incorrect number.

-Added setDie() as a function to the player class in order to help with serialization.

(30 mins)

Total: 30 mins

**October 5, 2016**

**-**Added decideMove() as a virtual function to the player class to handle what move the computer and human should make.

-Added decideMove() to the computer class, returns a string determining which move will be made.

-Added decideMove() to the human class, returns a string determining which move will be made.

-Fixed an error where determineWinner() would incorrectly declare the wrong winner.

(1 hour)

Total: 1hr

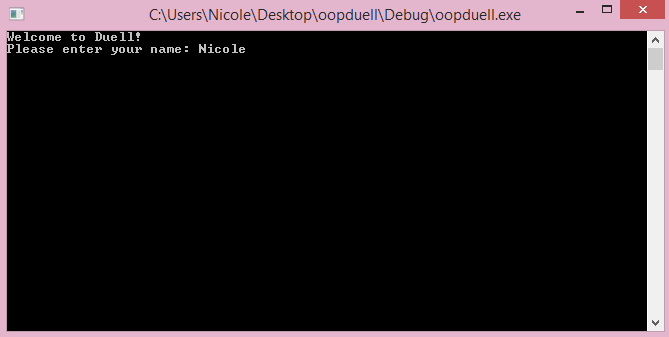
**Total Amount of Time: 28 hours**

**How to Run the Program:**

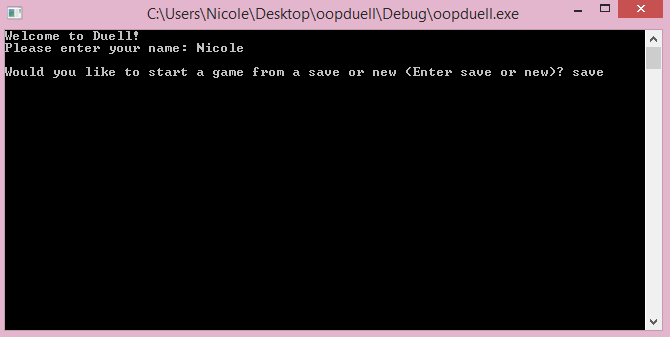
The game will run by running the exe file in the debug folder. The program is run using the main.cpp file in order to handle setting and playing out the game. Text files can be used in order to load in a saved gave or to save a game.

**Screen Shots:**

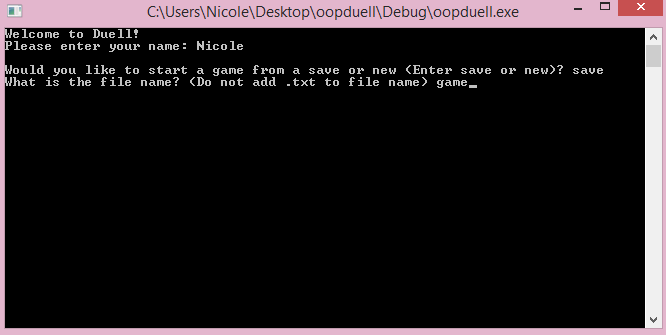
1. Entering human name



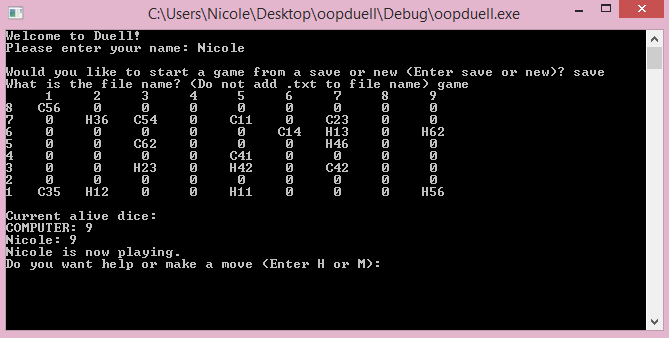
1. Asking to start from a save or new game, answering save



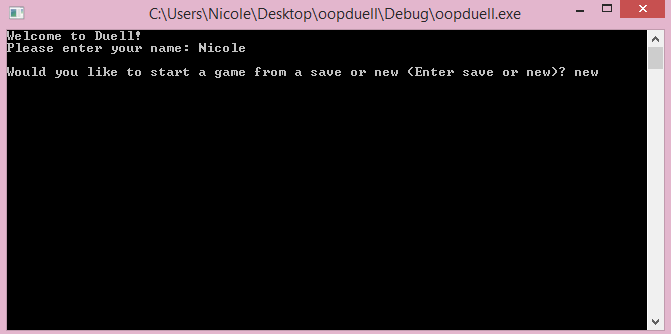
1. Asking the human player for the file name



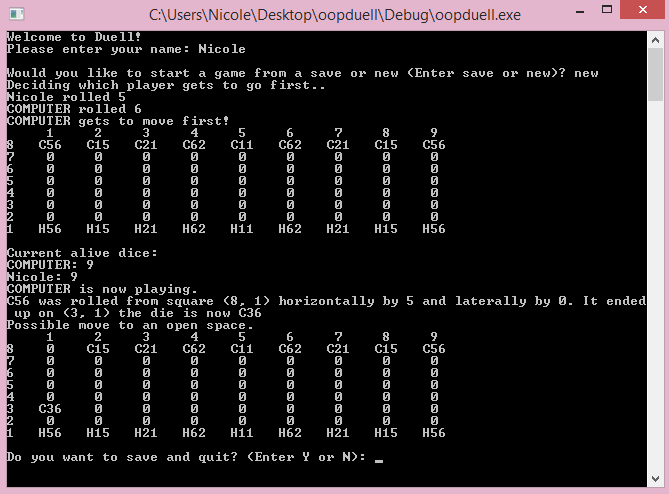
1. After loading the saved game from the file



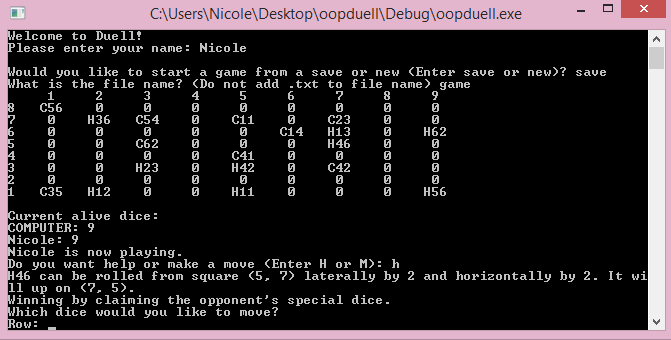
1. Asking to start from a saved or new game, answering new



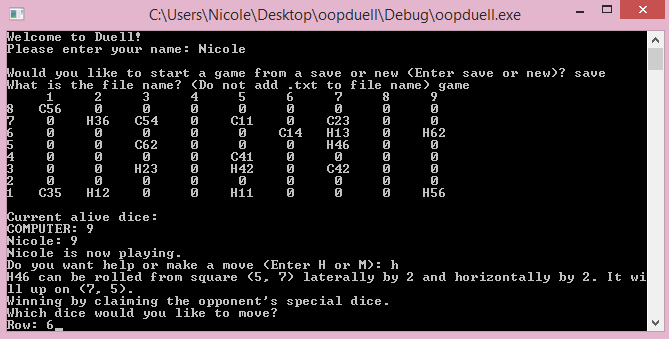
1. After answering new game, loading up the board and deciding which player goes first



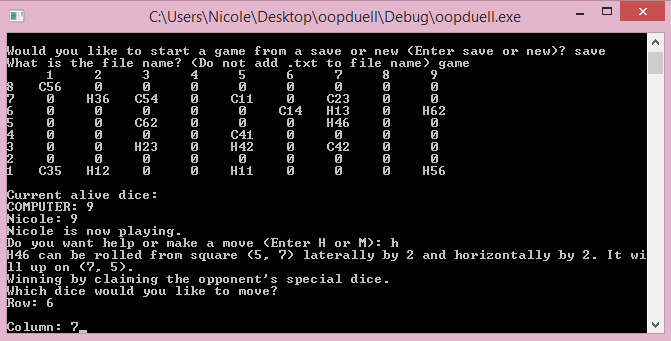
1. The human player asking for help and the computer giving help



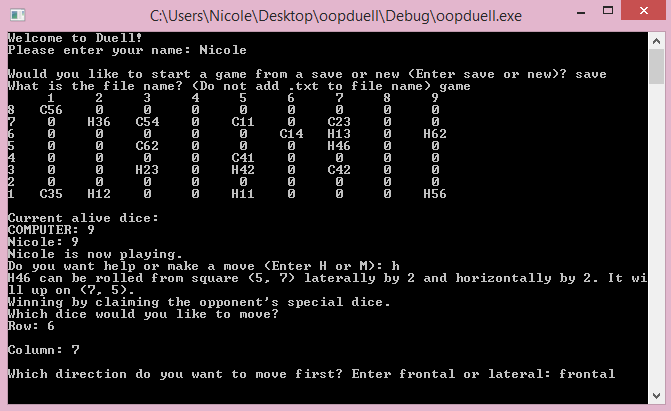
1. Entering the row of the die the human wants to move



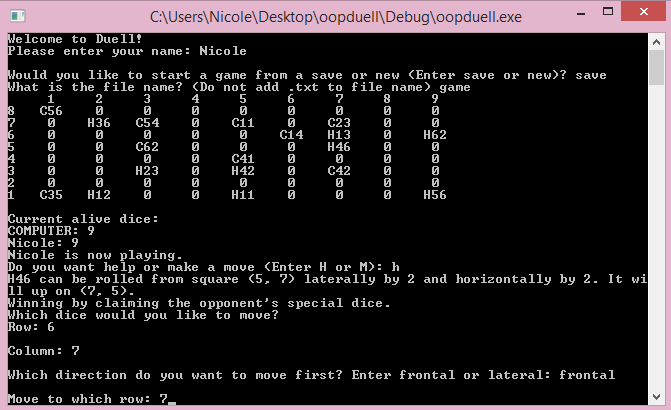
1. Entering the column of the die the human wants to move



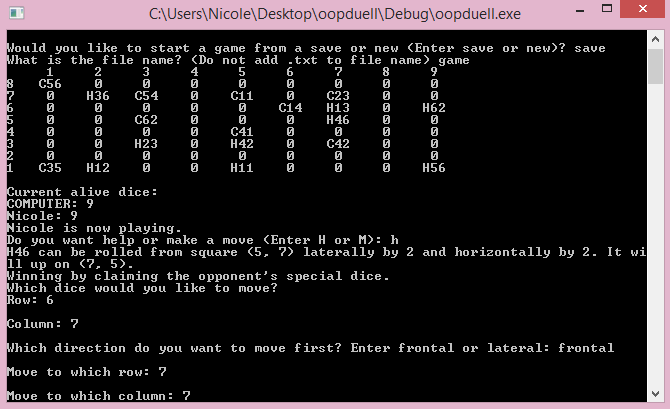
1. Entering moving frontally or laterally first



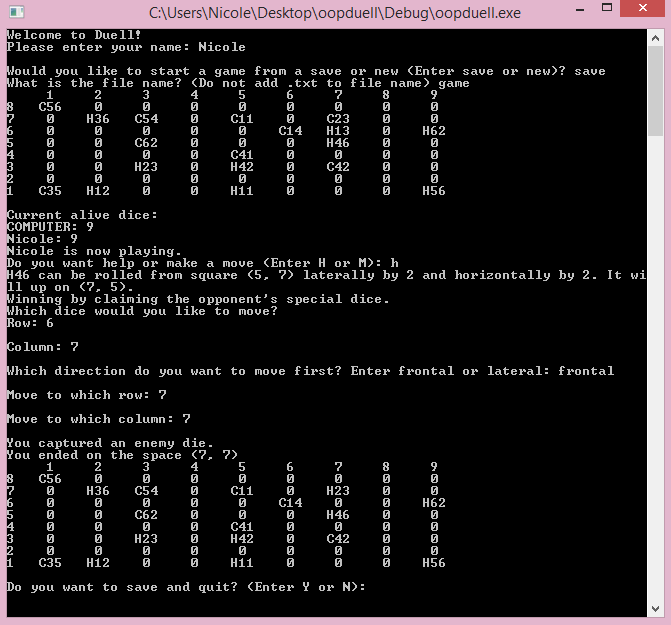
1. Entering the row of the tile to move to



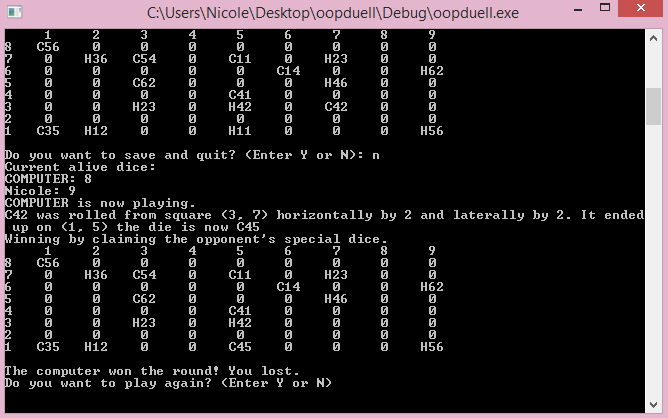
1. Entering the column of the tile to move to



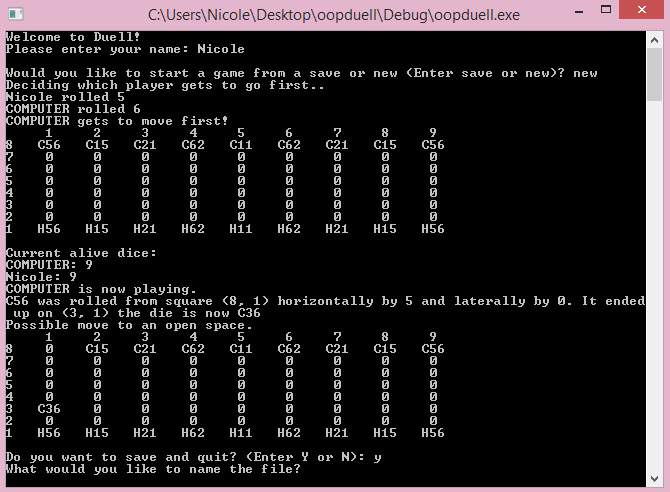
1. A completed human move being displayed



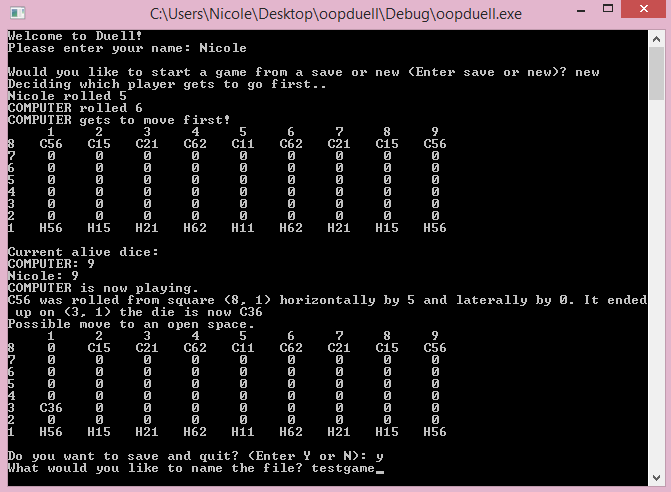
1. The computer making a move



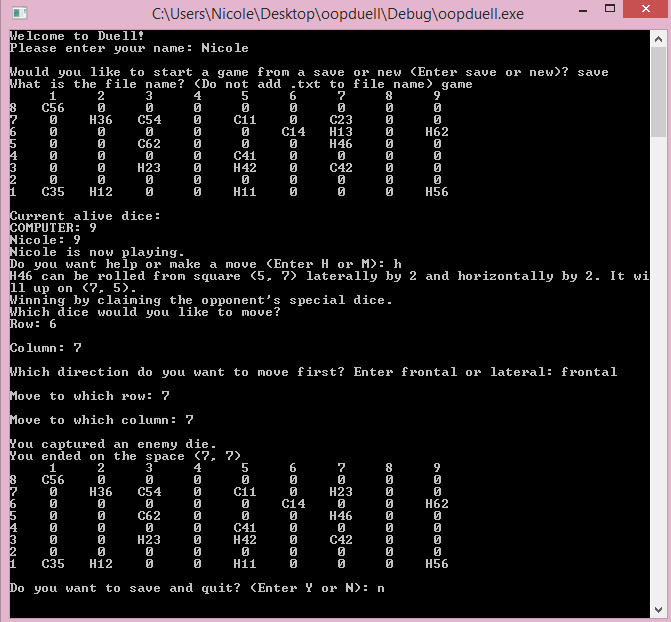
1. The human being asked if they would like to save, answering Y



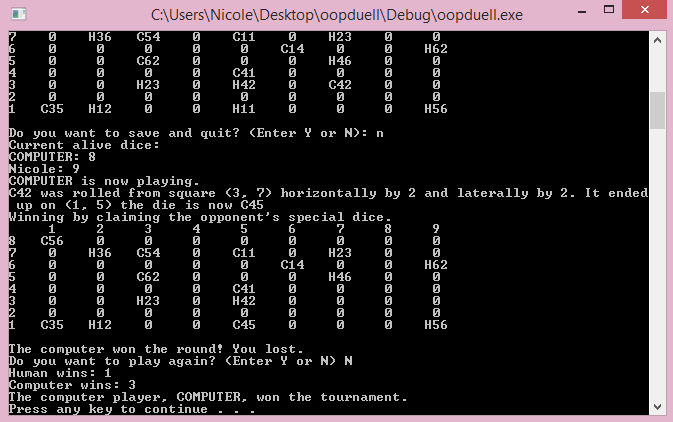
1. The human player entering what to save the file as, the program closes after hitting enter.



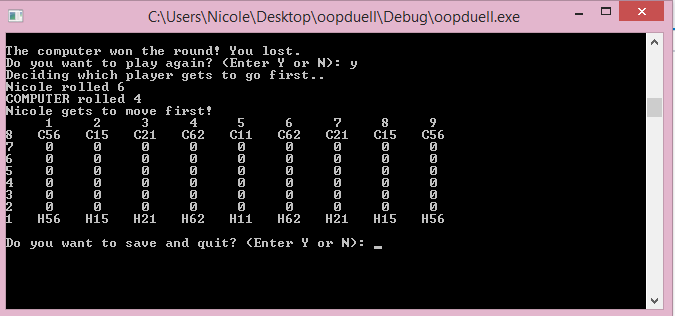
1. The human being asked if they would like to save, answering N



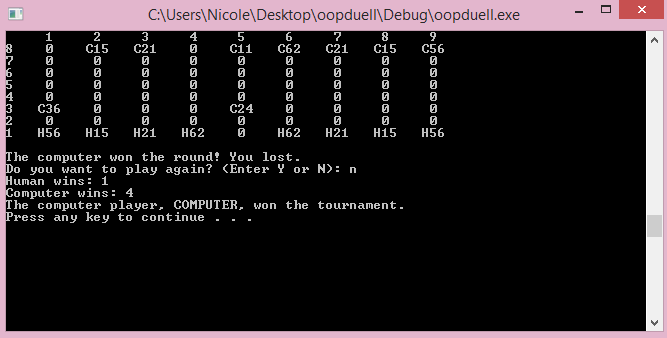
1. The human being asked if they would like to play another round, answering N



1. The human being asked if they would like to play another round, answering Y



1. The end of the tournament declaring the winner



1. Picture of the full board with all elements.

