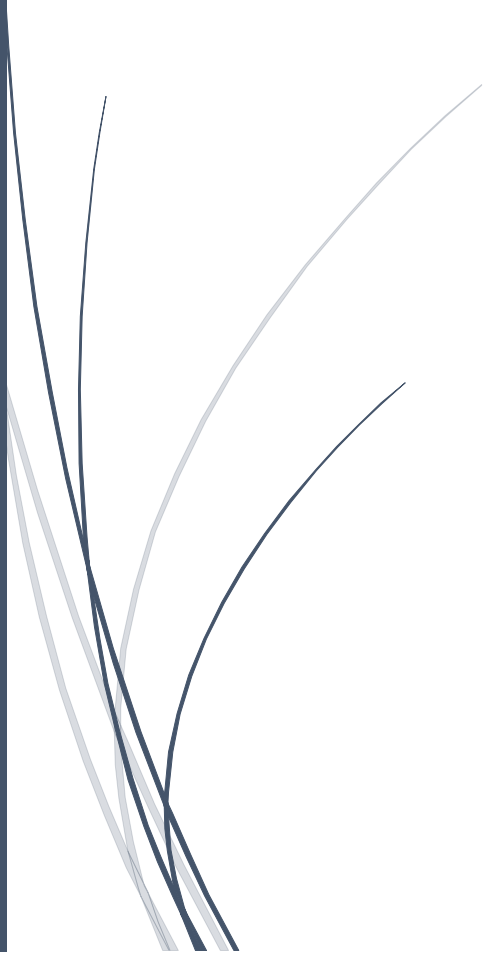
Duell

Technical Manual

Vivek Pandey
CMPS 366



Class Descriptions

Dice Class – Implements the properties of a dice object used in the game. The class consists of variables and functions to store and modify a dice's face values, its coordinates within a game board, its controller (human or computer), its capture status, and whether it is a king.

Square Class – Stores and maintains the location and occupancy status of a square in the game board. A square can be occupied by a dice object only.

Board Class – Initializes and maintains a game board of squares and player dices. It consists of functions to initialize a board, retrieve the contents at different squares within the board, check and set the occupancy of these squares that comprise the game board.

BoardView Class – Draws the updated game board to the console. Also can notify the user about the dices that haven't been captured and are active on the game board.

Game Class -- Implements the necessary details to run a game like conducting initial toss, getting user input for moves, activating help mode for human player, displaying updated game board and turn notifications.

Tournament Class – Runs a series of games, maintains the score for human and computer player, processes game serialization/restoration requests & declares winner.

Player Class – Implements the basic set of strategies for any player like traversing the board and determining path choices, validating and processing moves, changing dice orientation based on the rolls, capturing and eliminating opponent dices.

Human Class – Handles the move requests from a human player by performing input validations and processing them to reflect changes in the game board.

Computer Class – Implements Computer strategies to evaluate, prioritize, select and initiate the best move on behalf of the computer.

Serializer Class – Contains necessary member functions to serialize or restore a tournament to and from a text file.

Notifications Class – Consists of a list of inline functions to display various error messages and notifications to the console.

No Special data structures were used.

Computer's Play Algorithm

The computer uses the following algorithm to make a move.

1) Check if any Computer dice can capture opponent's king or if computer king can get to opponent's key square. If yes, use that dice and end the game.
2) Check if any opponent's dice can get to Computer King or Computer Key Square. If yes,
    a) Check if any of Computer's dice can capture that hostile die. If yes, use it.
    b) Check if any Computer dice can block that hostile dice. If yes, use it
    c) Try to move the king if it is safe around
3) Check if any Computer's dice can capture any of the other opponent's dice. If yes, use it.
4) Check if any of the Computer's dices other than the king are in threat of being captured. If yes, move that dice under threat to a safe location.
5) If none of the above finds a suitable move, calculate which Computer dice can safely get closest to the king in the next move and move it. This is helpful because the dice will keep safely revolving around the opponent king until a proper top value will come up for capture.

Bug Report

Almost all the bugs that were discovered during the development and testing phase were fixed right then. The only one discovered towards the end and can cause the program to crash has to do with the user provided file for game restoration. The parsing algorithm to restore the game from the text file is robust enough to handle minor formatting issues, few additional characters here and there towards the end of the file. But if the file significantly varies from the given specifications, then the program will crash instead of being gracefully aborted.

Apart from that one issue, other user inputs during the game are handled properly to not result in any unexpected crash or anomalous behavior.

Feature Report

Additional Features

The additional feature I have implemented is not necessarily a separate feature but rather a helpful add-on. To help the human player better understand the computer's thought process, I have a "Bots Mumbling" series of messages that get displayed during computer's turn. These short messages display the human player with the steps that the computer algorithm took in order to arrive at that move selection.

Also, to make the game more fun, I have added a sarcastic and scornful tone in the error messages that get displayed in the game. The notification messages, instead of being encouraging, are moderately derogatory intended to show the Computer's contempt to the human species. Some of my favorites are:

"Congratulations! You won. Our programmer must've done a terrible job on algorithms for someone like you to win."

"You just captured an opponent dice. Impressive for a Knucklehead? Eh!"

Missing Features

None. However, the display of the Computer strategy is slightly different than the given specification. Instead of a single "Computer-chose-this-move-because" explanation, the game displays a series of steps that the computer's algorithm took in order to choose its move.

Screenshots

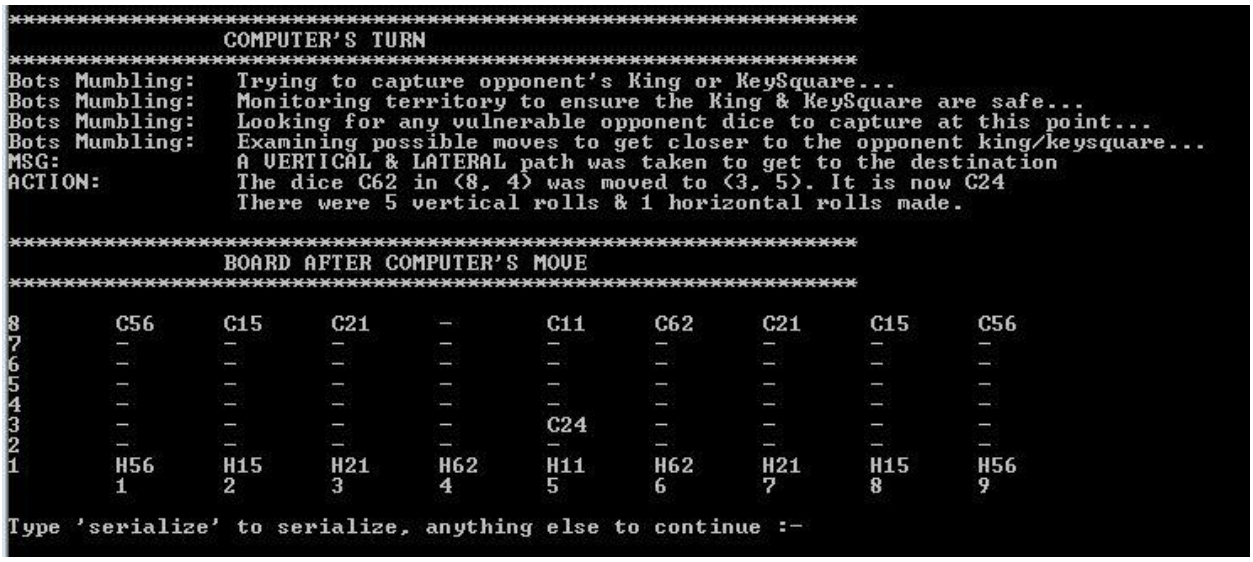


Fig. Computer's Turn

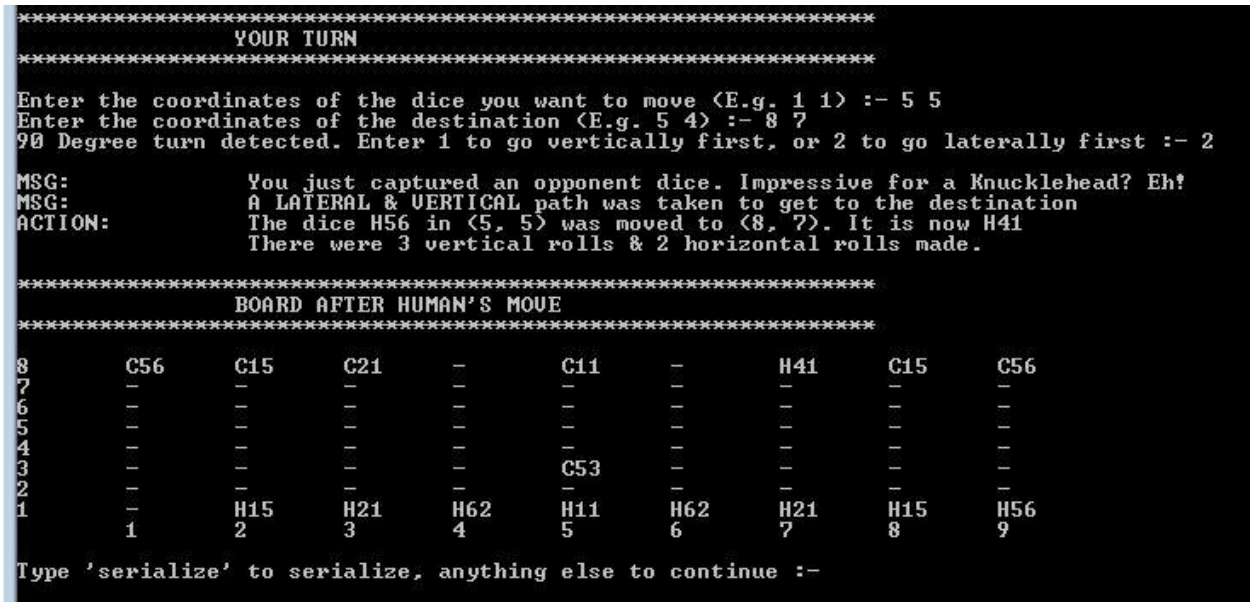


Fig. Human's Turn

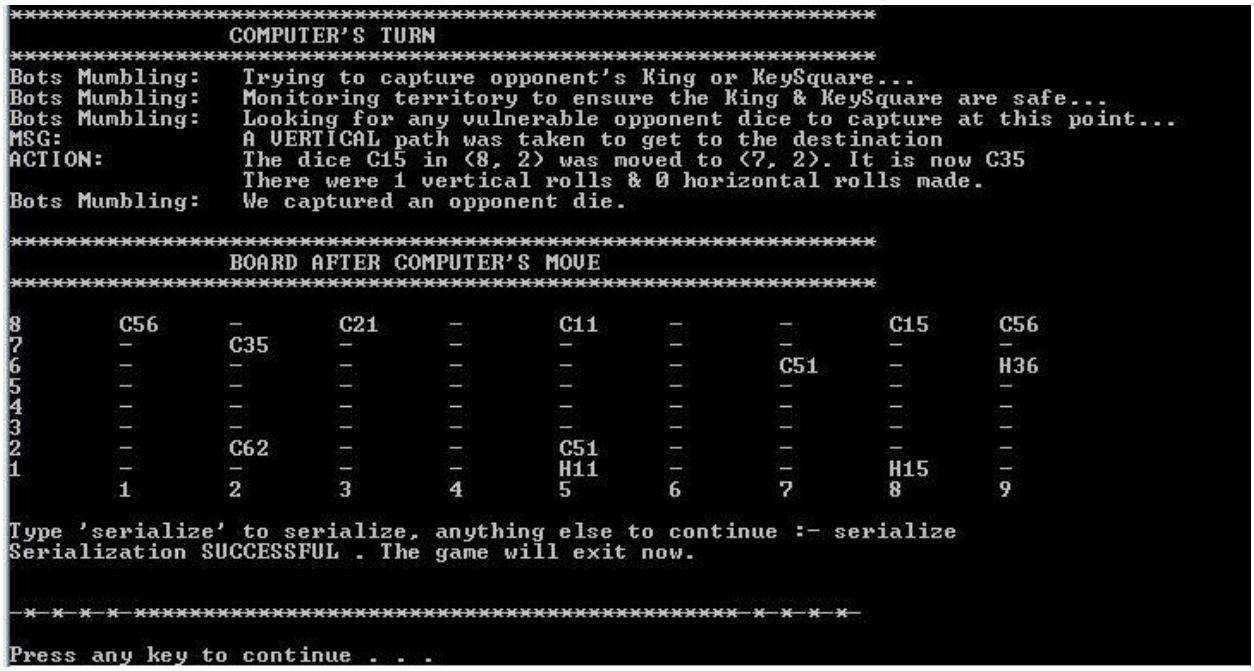


Fig. Serializing to File

Fig. Restoring Tournament from File

Fig. Help Mode

Log

09/09/2016

- Did a preliminary design and brainstorming of ideas on potential strategies for game play
- Partially implemented Dice.h, Square.h, Board.h, Player.h, Moves.h
- Dice.h needs modification in setters based on the notion that setting one face automatically sets the opposite since the sum of opposite sides is 7. Also, a strategy is needed to derive remaining sides based on a known side in the 3D model of the dice.
- Board.h needs to integrate the soldier and king dices.
- The row/column variables in Square class should be linked with the row/column variables in Dice Class to make sure update of one updates the other.
- The moves class might need a second look in the four directional roll functions.

09/11/2016

- Worked primarily on the Board class and created a board with dices located in respective places in respective orientation.
- Found out about the clockwise and anticlockwise nature of dices and used that to find remaining sides of a dice on being provided with 2 sides.
- SetCoordinates, SetRow, SetColumn functions inside Dice class got some further refactoring to check the validity of input parameters.
- DrawBoard function in Board class has been left halfway until a link has been established between the dice and square class.

09/12/2016

- Used pointers to solve the issue of linking the squares and their corresponding dices, if any.
- Some modification within the square and dice classes to integrate pointers to setup a link.

- Completed DrawBoard and UpdateBoard function in the Board class. Also, integrated a multidimensional string array that will be utilized later for serialization.
- The functions within Board class that print results to the console should be separated in a view class.
- Having issues with coming up with a proper way to access the Board class objects from other classes without compromising on Data Encapsulation.

09/15/2016 (2.5hrs)

- Made the GameBoard multidimensional array in Board class private and defined proper selectors/mutator functions to achieve so.
- Created a separate BoardView class and moved the display/serialization functions for the Board
- Merged the Moves class with the Player class with the intention of forming derived classes of Human/Computer later.
- Rearranged the code to meet the Project description of required classes.

09/16/2016 (2hrs)

- Implemented MakeAMove() function along with its supporting KeepMovingVertically() and KeepMovingLaterally() in Player.h
- Implemented a Play function in Human.h
- None of these new additions have been tested with inputs yet.

09/17/2016 (4hr)

- Implemented the Play() function in the Human class.
- Fixed an error of NullPtr exception by modifying the do while conditional in KeepMovingVertically() and KeepMovingLaterally() and adding breaks in switch statements of MakeAMove().
- There are some nullptr errors coming up even while making valid 90 degree moves,

- ISSUE: Moves are being made even when blockade exists on the path; also wipes the blocking dices entirely from the board while making the illegal move.
- Solved the above issues in the evening. Multiple changes in existing functions had to be made throughout the Player class.
- The destination and path validation, dice swaps works properly at this point.
- ISSUE: Any dice can capture any dice right now. Need to make it so only the opponent's dice can be captured.

09/18/2016 (2hr)

- Made the functions in Player class Protected.
- Partially implemented the Game class up to the main do-while loop.
- In the evening, completed the do-while loop. The human part can be played now.
- ISSUE: If user enters char instead of int, it leads to an infinite loop. Need to validate the int input
- NEXT: Fix the existing issues, and implement the Tournament class and ErrorDisplay class. The last thing on the list are Computer's strategy and serialization.

09/19/2016 (4hr)

- Can only capture opponent's dice now. A simple if statement in the IsValidDestination() did the trick.
- Error handling done for user input of co-ordinates.
- Found and fixed the bug of human getting multiple move turns due to improper transfer of controls.
- Added the functionality to allow user to pick a path in case of a 90 degree turn. If the user chosen path is invalid, the code will automatically select the next best path. (Done by adding an optional parameter to MakeAMove()).
- Partially implemented the Notifications class and added notifications to the player class.
- Any big issue aren't pending as of now.
- NEXT: Complete Notifications class and the tournament class.

09/20/2016 (2.5hr)

- Almost fully implemented the Notifications class for the classes written so far.
- Potential issue is that the input validation waits until all the coordinates and path choices have been entered. Nothing fails, it's just weird to let user select path when the destination is out of bounds already.
- Modified code to printing board only after each user makes a valid move
- Implemented the Tournament class and integrated it with the game class properly.
- Handled the user choice to quit or continue after each round.
- Tested thoroughly to make sure the tournament exits/continues as the user wants
- At the point, the game seems to be have all the essential components implemented for it to be a human game.
- NEXT: Serialization and Computer Strategy

09/21/2016 (3hr)

- Implemented first half of serialization in BoardView class where the program asks the user their wish to serialize in every step along the game. Writes to file and exits the game if user wishes to.
- Found bug – If user entered more parameters than necessary, they were being carried over to the next cin request causing discrepancies in values assignment to variables.
    - Fix: Added Cin.ignore() throughout the cins sections of code.
- Next Step: Restoring the serialization file to the gameboard in the beginning of a game

09/22/2016 (2.5hr)

- Moved the serialization components to a separate Serializer class which made things simpler.
- Wrote a function to read the gameboard from the text file and restore to the SerializedGameBoard string array.
- Attempted to use Regular expression to read the scores and next player, but it wouldn't work for some reason.

- The regex_match() is unable to find any matches at all, so it is probably something wrong with the implementation itself rather than the expressions.
- Next step: Get the regex to work so that the values can be restored to the respective variables. Then, the respective flags in the board, squares and dice can be set based on the retrieved values from file.

09/25/2016 (5hr)

- Have a decently working restoration portion of the serialization.
- Issue 1 (Potentially Fixed): King is fixated in [4] position, so the restoration needs to be done carefully to not place any other die in that index
- Issue 2: Need to come up with a formula to calculate the front and rear sides of the die during restoration.
- Combined serialization with the tournament overall and setup, have a full human side of the game.
- Issue: The reading from file itself seems to be fine, however, there seems to be some issue while setting the flags. As a result, while printing the game board from the tournament, some of the dice show incorrect values for top-right.

09/26/2016 (1hr)

- Fixed the issue associated with restoring serialization file. Turns out the botCount was being incremented instead of a humanCount in one part of the code which messed up the indexes being updated.
- At this point, the game has all the features for the human to play. Need to implement the computer strategy next.

09/27/2016

- Modified the function to set the beginning orientation of dice in order to accommodate the differences for human and computer home row dices.

- Changed the BoardView to display Top-left face values for computer dice since the right side is actually left side for user viewing from other side.
- Implemented the Play function in Computer class to capture opponent king if possible, and capture hostile dice if threat detected for computer's King.
- Partially implemented the part to capture other ordinary opponent dice but it still has an issue
- Issue: RunningOverOwnDice error keeps getting displayed while computer is checking possible attack modes to capture opponent dice. It is abnormal because the destination at hand is always an opponent die, so need question of capturing own die at this point.
- Issue: Also, noticed that one of the times the opponent die wasn't captured though it was possible to make that move.
- Next: Fix issues while capturing opponent dice, Implement blocking while under threat, and make it possible to move the king. Also, come up with a recursion to make a proper move if no captures are possible.

09/29/2016 (3.5hr)

- Implemented blocking while under threat and made it possible to move the king.
- Potential Issue: Once there was a weird situation where the blocking move was made even when the hostile die with a top value of 1 was right next to the king. Didn't occur later when I commented out the blocking code itself though. It was weird.
- Apart from that, the game ran quite smoothly.
- NEXT: Implement Breadth First search to find normal moves when no threats/captures are imminent.

09/30/2016 (4.5hr)

- Added the function to calculate the remaining faces during serialization.
- Also fixed the bug associated with accidentally setting king to captured while there are less number of dices in the restore file.

- Implemented the best move search part in Computer's play() function if no defenses, captures are possible.
- Fixed the "blocking in wrong spot" issue noticed yesterday by making the correct function call in the TryBlockingHostileDice function.
- Added print statements to announce the winner at the end of the tournament.
- At this point, the game is basically complete to play Human vs. Computer.
- Next: Implement Help for Human and refractor the code based on the rubric.

10/1/2016 (5hr)

- Worked on notifications display to ensure error printing isn't done when computer is running its' play algorithm.
- Added functions in notification class to display computer's thought process to the user.
- Moved the code properly in between header and implementation files.
- Besides documentation and inlining some functions, the code is presentable.
- Besides the help mode, everything else is implemented in the game as per specifications.

10/2/2016 (7hr)

- Added more description content to the moves as per the game specifications.
- Modified code in Game's GameOverConditionMet() & Computer's Play() function so that KeySquare capture only ends the game when it is done by the key piece of the opponent.
- Added extra inline functions for the "Bots_Think" series in the Notification class.
- Wrote the Technical manual and started final documenting the code.
- Added class and function header throughout all the header and implementation files
- At this point, the game is submit-able
- If possible, comment some more with the pseudo-code and implement Help Mode

10/3/2016 (4.5hr)

- Implemented Help Mode by modifying Computer's Play() function and Player's MakeAMove() function to only calculate and display recommended move and not modify the board in help mode.
- Added an additional strategy in Computer's Play() function that checks for the safety of dices other than kings and moves them to a secure location in case of a threat from opponent.
- At this point, the game implements all of the necessary features, is well documented and is ready for submission.