Duell

In Python

Technical Manual

**Duell** is a two-player chess variant played with dice on a board of 9x8 squares. Players take turns moving one of their dice in order to capture their opponent’s pieces, with the ultimate aim of capturing the opponent’s king to win the game.

Full Description: <https://en.wikipedia.org/wiki/Duell_(chess)>

Vivek Pandey

CMPS 366

**Class Descriptions**

**Main Class –** Entry point of the application.

**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 –** Consists of components necessary to draw the updated game board to the console.

**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,
3. Check if any of Computer’s dice can capture that hostile die. If yes, use it.
4. Check if any Computer dice can block that hostile dice. If yes, use it
5. Try to move the king if it is safe around
6. Check if any Computer’s dice can capture any of the other opponent’s dice. If yes, use it.
7. 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.
8. 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**

All the bugs that were discovered during the development and testing phase were fixed right then. At the time of submission, none of the bugs have been left unresolved.

**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!”*

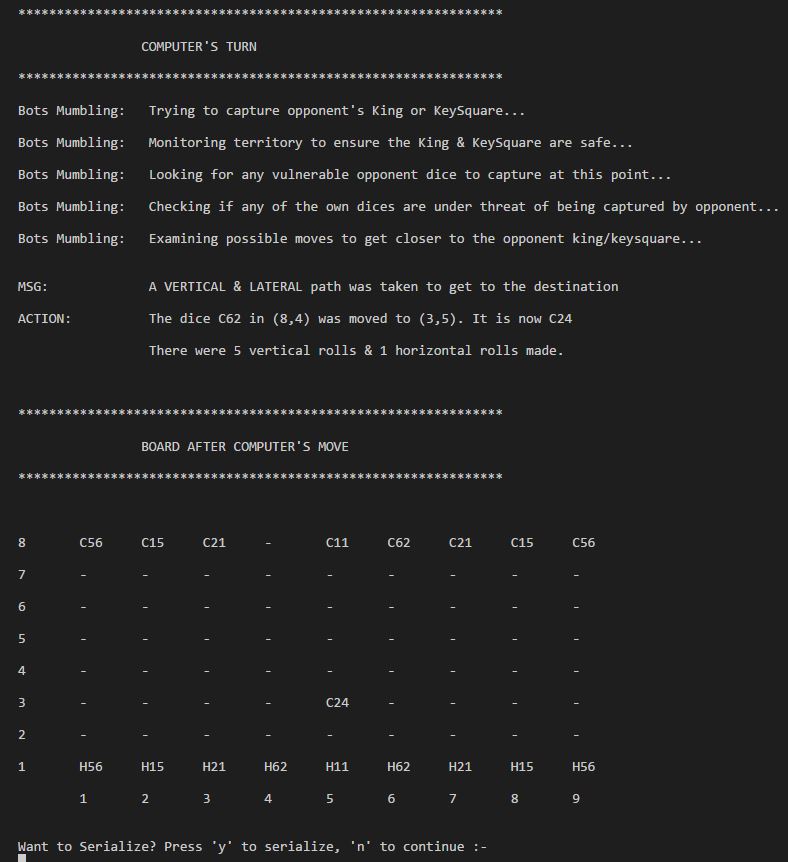
**Missing Features**

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.

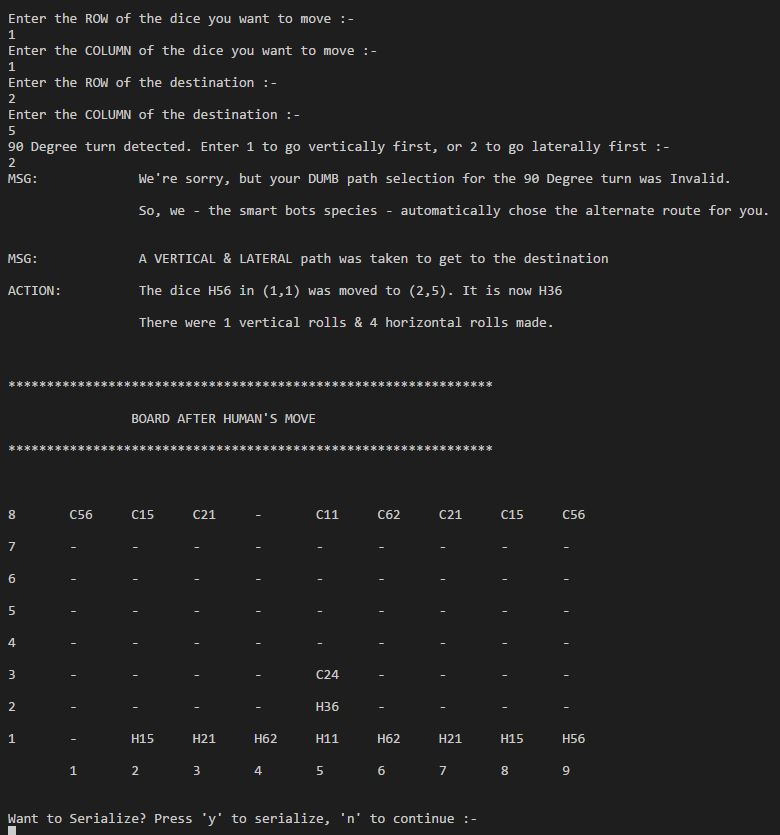
**How to Play**

1. Make sure your system has Python 2.7 (preferably) or higher installed.
2. Go to command line and navigate to the directory with the source code.
3. Launch the main script main.py by typing **“python main.py”**
4. Enjoy the game.

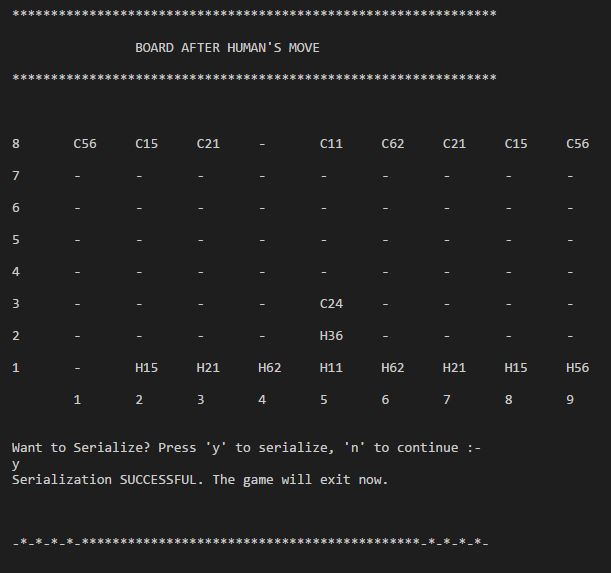
**Screenshots**



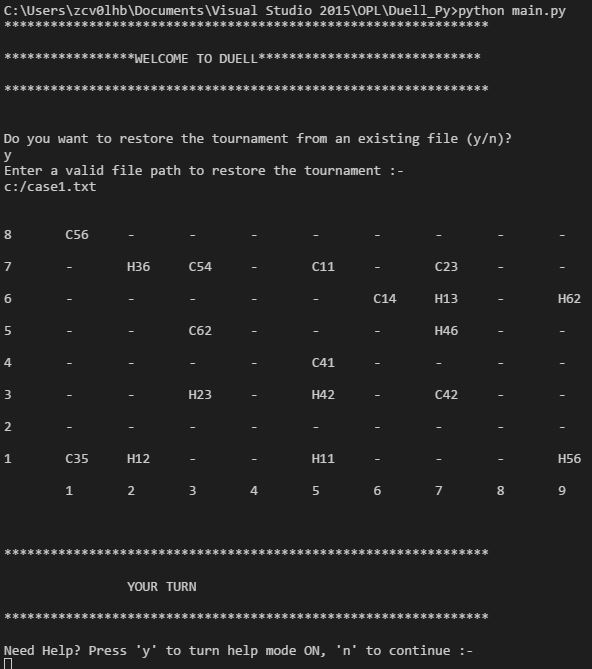
**Fig. Computer’s Turn**



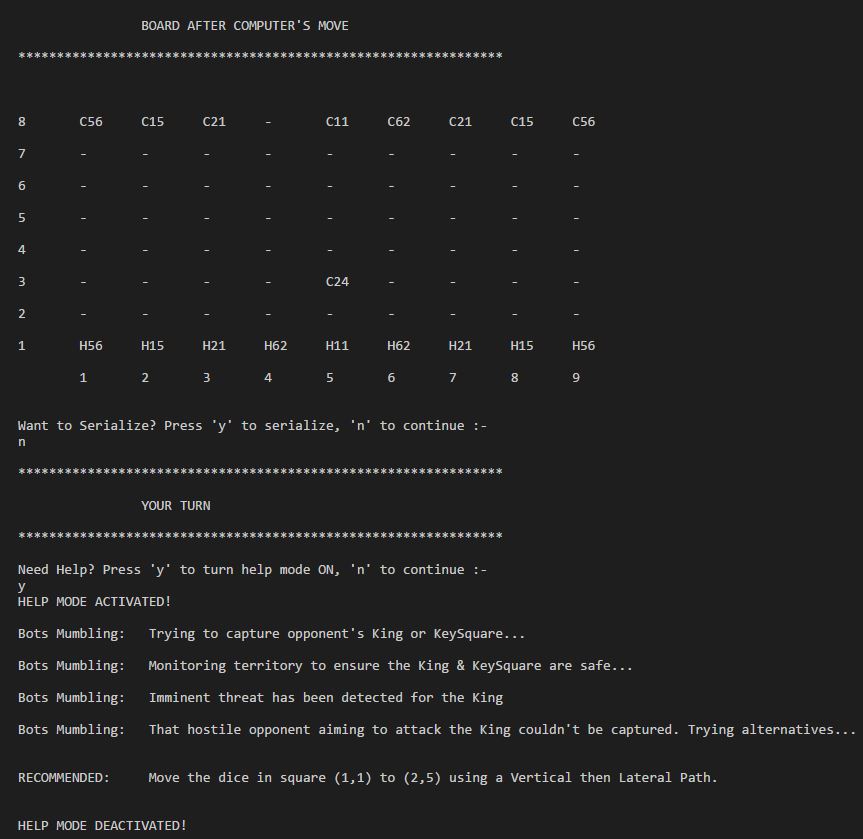
**Fig. Human’s Turn**



**Fig. Serializing to File**



**Fig. Restoring Tournament from File**

****

**Fig. Help Mode**

**Log**

**10/4/2016 (6hrs)**

* Implemented Dice, Square, Board and BoardView Classes.
* Tested their functionality with various test cases and they are fully functional at runtime.

**10/5/2016 (2hrs)**

* Implemented Player class partially for Dice Rolling functions by using my C++ Duell as reference.

**10/6/2016 (12hrs)**

* Fully implemented Player class and tested with few cases.
* Human Class Implemented and tested.
* Computer class implementation had some issues associated with typos which caused semantic issues.
* More testing for player classes can only be done after the game class has been implemented.
* Implemented Game class but it still has random runtime errors that need to be investigated on tomorrow.

**10/7/2016 (1.5hrs)**

* Singleton Notifications class implemented partially.

**10/9/2016 (7hrs)**

* Notifications class implementation completed and its inline functions are injected throughout various classes where necessary.
* Corrected issues associated with input handling in game class.
* Serialization implemented and tested. Works perfecto!
* A single round of game seems to work fine at this point. Need to implement Tournament.

**10/10/2016 (8hrs)**

* Implemented Tournament classes. Had runtime issues due to variable naming conflicts.
* Code Refactoring, Technical manual and documentation complete.
* Ready for submission.