

**ELEN2009A: Suicide Checkers in C++**

*School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa*

**Simphiwe Shezi – 2462667**

**Abstract –** This document is about the design and implementation of suicide checkers.

**Keywords:** *suicide checkers,*

1. Introduction

Implementing an algorithm that plays to lose encourages players who are not good at that particular game to continue playing [1]. *Suicide checkers* is one example of a game whereby the aim is to lose. *Suicide checkers*, also known as *anti-checkers, giveaway checkers,* and *losing draugh*ts, is identical to conventional checkers but the only difference is that the aim here is to lose [1]. This document presents the implementation and analyses of Suicide checkers in C++. The project specification, design, and implementation of the game are presented in section 2 below, section 3 shows the results, and then the analysis and discussion are done in section 4.

1. Specification, Design, and implementation

*2.1. Specifications*

This section outlines the project requirements, constraints, and assumptions.

*2.1.1. Requirements*

It is required that two algorithms be implemented, in C++, to compete against each other for the *Suicide checkers* game. The game accepts an input file with board sizes and records every move made by each algorithm in an output file. The output file contains the size of the current game in the first line, the alternating player moves on the following lines, the number of pieces left for each algorithm, then the winner at the end. A suitable time plan for the project must be formulated.

*2.1.2. Constraints*

The game, together with the algorithms, must be implemented in C++. The board size must be an even number ranging from 6 to 12. Global variables are not allowed in the code. The board must use the labeling system as per the brief. Console output is not allowed. Pieces only move forward diagonally, towards the opponent. Playing is only allowed on the black part of the board. The king feature is allowed when one player gets to the end of the opponent’s side and can move both forward and backward diagonally. A piece making a capturing move jumps the opponent’s piece and lands on the following, empty, diagonal position. A piece is only allowed one move up if it is not capturing anything. A piece must be removed from the board after it has been captured. A capturing move is compulsory, an algorithm must capture whenever there is capturing opportunity.

*2.1.3. Assumptions*

Algorithm 1 will use the character ‘o’ to play and algorithm 2 will use the character ‘x’ to play. Algorithm 1 uses ‘O’ to denote king and algorithm 2 uses ‘X’ to denote king. Pieces are not allowed to move beyond board boundaries. Multiple jumps are allowed as long as there is capturing opportunity after a jump. An algorithm can choose one out of multiple capturing opportunities. Assume the ‘.’ characters represent part of the board that playing is not allowed and ‘#’ Characters represent the black part of the board.

*2.2. Design and Implementation*

*2.2.1. Board*

The board is designed with a 2-D vector because a vector allows for flexibility in size. The vector is initiated with ‘.’ characters and ‘#’ characters in alternating turns. Figure C1 shows an example output of this, using a board of size 12. Figure 1 below shows the UML diagram for the Board class.

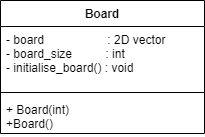


Figure 1: Board class UML diagram.

The board is initialised as per discussed above. A constructor is used to validate and set the board size, making sure it is even and ranges from 6 to 12.

1. Results
2. Discussion
3. Conclusion
4. References

[1] Bosboom, J., Congero, S., Demaine, E.D., Demaine, M.L. and Lynch, J., 2019. Losing at Checkers is hard. *The Mathematics of Various Entertaining Subjects (MOVES 2017)*, *3*, pp.103-118.

**APPENDIX A: Project time management**

**APPENDIX B: Flow charts**

**APPENDIX C: Console references**

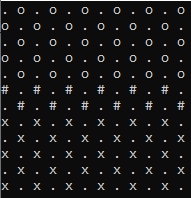


Figure C1 : 12x12 board in console