Introduction to Programming - Project Part 1

K Ganer, C Sivanandam, H Flindt
Supervisor: Peter Schneider-Kamp
DM536

19. december 2017

Indhold

1 Specification

2 Design

Due to the size and scope of the assigment, each group-member tried to develop different the needed parts. The most interesting solutions are added to this report.

2.1 checkFull() by turnCounter()

Since the full size of the board is known in advanced to the beginning of the game, and each turn uses one square, there is a finite amount of turns available in any size of TTT. Therefore the need to check if all squares in the game are used is not relevant, but only a counter of amount of squares i.e. turns possible is needed. An minor advantage of a turnCounter() based checking system is that even though the size of the TTT-board is of a legible size, a checkFull() that should look at each square in the board. This check sequence would grow with a power of two when the board increases in size, and unless programed with memory, would have to check each square each time. The turnCounter() only needs to increment each time. This is mostly a tour de force in problem finding that for this assignment might not be all that relevant, but non the less a theoretical board for 10.000 players would need checkFull() to go through one hundred million squares and either remember the state of each in a list, or do a full check sequence each time. The turnCounter() would in comparison just need to count to 10.000. A problem would arise with the turnCounter() if players were allowed to change the placement of their piece, but since this fundamentally changes the game no further thought were given to this.

2.2 checkWinning() by math

Since each player is represented by a number, it was thought to be mathematically possible to recognize a winning row, column, or diagonal on the board. To ensure that a row of three followed by two zeros would result in a win for a player, either player three or one, it was decided to develop a checkWinning() that used the product of three relevant squares. Subsequent this number should be transmuted to a winning amongst the players of the game. Since the function ought to return either zero or the number of the winning player, different approaches could be developed. While six if-conditions could check for this, this would become a cumbersome programming task, especially in theory if a higher number of players were to be apple to play the game. There for it would be tested if a program using the cubicroot of the product of all three squares could be the solution. The class should have an intrinsic ability to check any board size from three and up in all directions possible for three in a row, and be to return an integer if needed to checkResult(), based on a float value without loss of data. (i.e only return the integer one if it is 1.0 and not 1.2 etc.) Swing() Due to hardware restraints a GUI version of the TTT was also implemented besides the app. While a functional version was given as part of the assignment, several cosmetic changes were needed to ensure a more user friendly experience.

2.3 GUI

Due to hardware restraints a GUI version of the TTT was also implemented besides the app. While a functional version was given as part of the assignment, several cosmetic changes were needed to ensure a more user friendly experience.

3 Implementation

3.1 checkFull() by turnCounter()

The checkFull has the if-condition which returns true as long as the turnCounter() is below the amount of squares in the game. And each time it is called it calls the turnCounter() which increments by one, from a starting point of zero, each time it is called returning the current turn number.

3.2 checkWinning() by math

The checkWinning() class was implemented as mathematical for-loop which would check each row, column, and diagonally up and down respectively. The main script of each for-loop is quite similar but with variation for the incrementing and direction of the check. Mathematically the checkWinning() consisted of checking if the cubic root of the product of three squares in a row, column, or one of the diagonals were equal to the integer of that root. If so it would return that integer value, which would correspond to the player number which had made the winning combination. If not it would return zero. The overall setup was with a for-loop with a nested for-loop. While the first for-loop selected the starting point the nested for-loop would move the interval being checked by one to a distance of two from the edge of the board. After this the first for loop would increment, or decrement and the nested loop would check the new row, column or diagonal for a winning sequence.

3.3 GUI

Different parts of the GUI color scheme were changed as to make it easier to see what moves were legal in the setting of the game. A few extra action functions in relation to the colors were also implemented. While an earlier version of the GUI was expanded with background music and sounds based upon actions this was removed again due to compiling errors when sharing the game, cause of the specify needed of the file-path, combined with a minor latency issues when loading each sound file.

```
1        her kan i vise kode som i forklarer.
2        indryk virker fint her.
```

4 Testing

4.1 checkFull() and the GUI

A complication arose when implementing the GUI, as the GUI did not recognize when a draw was reached, as should be the case with the checkFull() and turnCounter(). By changing different variables methodically backwards from this first call of checkWinning(), to checkFull() and finally turnCounter() the error was located. It was discovered in the way the GUI generated the board compared to the CLI. The CLI generates the board and calls checkWinning() after player ones turn incrementing turnCounter() till a certain value. If the checkResult() has returned a winner by that point, the else-if-statement comes true thus returning the draw statement. The problem appears to be that the turnCounter() was incrementing one to little when the GUI was launched. Thus an extra call to turnCounter() was launched when the GUI first generates the user interface. This ensured that a draw would be reached for the GUI.

4.2 checkWinning() by math

A special case was discovered that falsely would result in a win for player 2. It arose when three squares were occupied by player one, two and four. These three would result the same number when checkWining were run as if all three squares were occupied by play two. This was resolved by adding the number two to all terms in the cubicroot equation. No result for 2-6 players could then be equal in any combinations. Subsequent removing the added number two would return the checkWinning() result needed.

5 Conclusion

6 Appendix (source code)