Criteria B Documentation

[Planning 2](#_Toc184834707)

[Requirement 2](#_Toc184834708)

[Design 2](#_Toc184834709)

[Conceptual Model 2](#_Toc184834710)

[Logical Model 2](#_Toc184834711)

[Physical Model 2](#_Toc184834712)

[Test Plan 3](#_Toc184834713)

[Extensibility 3](#_Toc184834714)

[Annexe 3](#_Toc184834715)

# Planning

**Plan of tasks/ GANTT** (depends on WATERFALL/AGILE/SPIRAL choice): planification

**ROT** (depends on WATERFALL/AGILE/SPIRAL choice): real work

# Requirement

**Requirement Use Cases** or **Stories**

**Requirement List**

Any System flow chart or Data flow diagram

# Design

## Conceptual Model

[UML Domain Model](https://lucid.app/lucidchart/752cbd43-3c6e-46cd-a19b-d16646a70124/edit?viewport_loc=28%2C76%2C1149%2C553%2C0_0&invitationId=inv_8b4376c7-4665-4bd6-819d-62d2827cd0bd)

[Sketch, View and Navigation Model](https://www.canva.com/design/DAGXNp2o9Xo/KJwqAfC046WWdrP8NZwvOw/view?utm_content=DAGXNp2o9Xo&utm_campaign=designshare&utm_medium=link&utm_source=editor)

[Data Flow Diagram](https://lucid.app/lucidchart/69b16dec-5292-4b99-a822-00d9f556278e/edit?viewport_loc=-102%2C-167%2C1847%2C889%2C0_0&invitationId=inv_d998b4c8-563a-4ef1-82f8-a465d139f88a)

## Logical Model

[Logical Architecture and Stack](https://lucid.app/lucidchart/9e13bd7f-6d21-4c0b-9ecd-2ccd30207c61/edit?viewport_loc=-63%2C-91%2C1407%2C677%2C0_0&invitationId=inv_fb2eb4ee-79ee-434a-8c63-815b71b9a11e)

[Database Entity Relation Model](https://lucid.app/lucidchart/12de3d77-5cc8-49b0-bb8f-4a696d18da62/edit?invitationId=inv_e7d9de12-fcd9-4ce1-a84a-085ae5470a38)

[UML Class Model](https://lucid.app/lucidchart/dde7b410-8c2e-4106-9aeb-0bcda4acb2ea/edit?viewport_loc=417%2C-1423%2C2194%2C1113%2C0_0&invitationId=inv_cab5ef2b-916b-4225-a4ce-40abdeb89851) -> [UML Class Model v2](UML%20Class%20Model%20v2)

UML Vista Class Model

Algorithm 1 Tic Tac Toe

Algorithm 2 Checkers Game

Game logic class model

Scoring Algorithm

Ranging Retrieval Query

## Physical Model

[MySQL Entity Relation Model](https://app.sqldbm.com/MySQL/DatabaseExplorer/p320771/)

# Test Plan

**Test plan** (depends on WATERFALL/AGILE/SPIRAL choice)

# Extensibility

# Annexe

Algorithm Tic Tac Toe

1. Initialize board size to 3

2. Create a 3x3 mask array filled with '-'

3. Set play to -1 and isPlayer1 to true

4. Print the initial board

5. Initialize count\_play to 0

6. Loop indefinitely:

a. promptNextMove(currentPlayer) and keep it in position

b. Calculate the line and column from the position

c. Update the mask with 'X' for Player 1 and 'O' for Player 2

d. Print the updated board

e. Check if Current Player isWinner(mask, 'X' for Player 1 and 'O' for Player 2, board\_size): is the winner:

i. If true, print "Player 1/2 won" and return

g. Increment count\_play

h. If count\_play equals the maximum number of plays (9):

i. Print "Null Match" and return

i. Switch the player

7. Function promptNextMove(currentPlayer):

a. Loop until a valid move is entered:

i. prompt next position to the currentPlayer

ii. Read the position

iii. If the position is valid and not already played isValidMove(mask, position, board\_size):

- return position

iv. If the position is already played:

- Inform the player

8. Function isWinner(mask, c, board\_size):

a. Check all lines for a win

b. Check all columns for a win

c. Check the first diagonal for a win

d. Check the second diagonal for a win

e. Return true if any win condition is met, otherwise false

9. Function isValidMove(mask, number, board\_size):

a. Calculate the line and column from the number

b. Return false if the position is already played, otherwise true

10. Function printBoard(mask, board\_size):

a. Loop through each cell in the mask

b. Print the cell value or its position if it is '-'

Algorithm Checkers Game

1. Initialize board size to 8

2. Create an 8x8 board array with initial positions for players:

- 'b' for black pieces

- 'w' for white pieces

- '-' for empty spaces

3. Set isPlayer1 to true (Player 1 is black, Player 2 is white)

4. Print the initial board

5. Loop indefinitely:

a. promptNextMove(currentPlayer)

b. Update the board with the move

c. Print the updated board

d. Check if the current player has won:

i. If true, print the winning message and return

e. Switch the player

6. Function promptNextMove(currentPlayer):

a. Loop until a valid move is entered:

i. prompt start and end positions to the currentPlayer

ii. Read the start and end positions

iii. If the move is valid:

- Break the loop

iv. If the move is invalid:

- Inform the player

7. Function isValidMove(board, start, end, isPlayer1):

a. Check if the start and end positions are within bounds

b. Check if the start position has the current player's piece

c. Check if the end position is empty

d. Check if the move is diagonal and within allowed distance

e. Check if the move captures an opponent's piece (if applicable)

f. Return true if all conditions are met, otherwise false

8. Function isWinner(board, isPlayer1):

a. Check if the opponent has any pieces left

b. Return true if the opponent has no pieces, otherwise false

9. Function printBoard(board):

a. Loop through each cell in the board

b. Print the cell value