Software Engineering Group Project
Design Specification

Author: George Cooper [gwcl], Kieran Foy [kif11], James Owen [jco3], Tate Moore [tam41]
Config Ref: SE_GP17_DesignSpecification
Date: 14th March 2023
Version: 0.9
Status: Review

Commented [D[1]]: Update to correct Reference "SE_GP17_DesignSpecification"

Commented [D[2R1]]: Also Update Copyright to be 2023

Commented [TM[3R1]]: Done both [REVIEW]

Commented [D[4R1]]: Config Ref still appears incorrect

Should be "SE_GP17_DesignSpecification"

Commented [TM[5R1]]: Sorry should have looked at it more. Wanted to copy and paste but it removed the comment when tried. Now adjusted [REVIEW]

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION.

1.1 Purpose of this Document

The purpose of this document is to outline the design specification for the Chess Tutor Application, to be presented along with the final draft of the design presentation in week beginning 20th March 2023.

Commented [D[6]]: Use "Chess Tutor Application" rather than "Group Project"

Commented [TM[7R6]]: Done [REVIEW]

1.2 Scope

~~This document displays the design specification created by Group 17, covering all sections named in SE.QA.05 [1] and following the layout provided in SE.QA.02 [2]. These two documents should be read by all project members working on this document before they begin working on it and should refer to said documents if needed.~~

~~This document should be read by all project members before the day of the design final draft presentation (week beginning 30th March 2023.~~

Commented [TM[8]]: Go to SE.QA.05 / 2 RQAD

Use this for scope?

Also add in UI spec [3] and reqs [4]

Action: redid the scope.

[REVIEW]

The document has been created in line with the General Documentation Standards [1] and covers all sections described in the Design Specification Standards [2].

There are mentions of the use cases given in the User Interface Specification [3] as well as the functional requirements from the Requirements Specification [4]

The documents referenced will need to be looked through and understood before reading on.

1.3 Objectives

The objectives of this document are:

- To justify and describe the breakdown of the system into classes inside a singular program and its purpose, as well as that of each significant class'. 2
- Show that the functional requirements have been met by relating them to the classes. 2
- Using a component diagram, create an understanding of the relationships and dependencies between modules, or in other words how parts of the system fit in together. 3
- Give an outline specification for each class for a developer to understand them. 4
- Show the relationships between classes using UML class diagrams and how they work together using UML sequence diagrams in addition to how they satisfy the use cases [3]. 5
- Explain the planning of difficult or significant algorithms using pseudocode. 5

Commented [D[9]]: Shorten the number of bullet points. Perhaps one for each section?

Commented [TM[10R9]]: [REVIEW] Shortened number slightly, however aren't bullet points supposed to be short, like one point per bulletpoint?

2. DECOMPOSITION DESCRIPTION

2.1 Programs in system

The system is made up of a singular program. The program will allow 2 users to play a game of chess on a single computer.

When the program loads up, they will face a main menu scene with 3 JavaFX Buttons: 'Quit'. 'Load old game' and 'Start new game'.

Left clicking button...

- 'Quit' will exit the program.

Commented [TM[11]]: [REVIEW] I was editing text and I accidentally deleted Dustin's comment about keeping this to sentences instead of my suggestion of bulletpoints.

I have tried to do full sentences but I've ended up with like a mix between sentences and bulletpoints. Suggestions and criticism welcome.

- 'Load old game' will take you to a new load game scene.
    - Use JavaFX FileChooser to pick an old game file to load in and you will be able to either continue an old unfinished game or replay a finished one.
    - Left click three JavaFX buttons to either confirm or unconfirm a chosen file or to go back to the main menu.
- 'Start new game' will take you to a new game scene
    - Use two JavaFX TextFields for inputting player names, one for black and the other white, which is which will be shown with JavaFX labels. This allows players to remember if they are black or white.
    - Left click two JavaFX buttons to either confirm the chosen names or to go back to the main menu.

Confirming and successfully loading an old game or starting a new one will take the players to a new scene where it will display the current chessboard state to the users.

The program will keep track of game statistics including:

- Whose turn it is.
- The colour and name of each player.
- The position of the pieces on the chess board.
- The history of all past moves.
- The taken pieces for each colour.

For an unfinished game, at the start of each new turn the player – whose turn it is – will be prompted to select a piece using right click. Once selected, the program will display all legal moves. The player will then choose a legal move by left clicking on its square or can deselect the piece by left clicking anywhere else.

If a user's king is in…

- Check – the program will alert the payers.
- Checkmate – the program will alert the players and display the winner. The program will give the users the option to save the game.

The program will allow the users to save and quit the game at any time. Saved games will be automatically saved in case of a crash.

For a finished game, the user will be allowed to go forwards and backwards through the game.

2.2 Significant classes

Class Name	Description
Setup	Setup will handle all methods required to setup the game for the user. The class will handle creating a new game or restoring a previous game. The class will build the board using JavaFX, handle usernames and colour choice. The class will also handle restoring a game if the program crashed.
Game	Game will handle all systems required to run the chess match. These include piece movement, detecting checks and piece removal.
Save	Save will handle all functions relating to saving. The class will handle storing all moves in the game to allow for the user to restore and step through the game in its entirety.
Replay	Replay will handle loading and displaying previously played games. It will hand systems required to allow the players to step through previously played games.
Piece	Piece will act as a superclass to all specific piece classes. It will include attribute shared by all chess pieces.

Commented [TM[12]: Change heading to 'Significant classes' as 'each program' suggests more than one program? [REVIEW] - changed.

Commented [TM[13]: Perhaps we should do a controller class

MainMenuController
- Handles UI/Graphics for the main menu scene
- Displays buttons for starting a new game, loading an old game and exiting.
- Handles events fired from clicking these buttons.

NewGameController
- Handles UI/Graphics for the new game scene
- Displays two textfields to input names as well with their corresponding team name using Labels. Lastly a confirm or cancel button.
- Handles all event fired from inputting names and confirming or cancelling.

LoadGameController
- Handles UI/Graphics for the load game scene
- Displays FileChooser to pick a file
- Handles event fired from picking a file, decides if it is a finished game or not.

Commented [D[14R13]: Is it necessary for so many UI controllers or would a single one work? I dont know UI very well so please get back to us on this.

Commented [TM[15R13]: Depends on how implementation and Finn wants to go about UI.

Commented [TM[16]: Save
-saves history (all past moves)
-writes all of it to a .txt

Changed from quit to save

This is actually Replay class

What programming to quit really needs to be done? Surely this is more a graphical thing to be done (in a controller class)

	Its subclasses include Pawn, Knight, Bishop, Rook, Queen and King, each subclass x will act as a blueprint for all x pieces in the game, it will store x specific attributes such as legal moves.
~~King~~	~~King is a subclass of Piece. It will act as a blueprint for all king pieces in the game. The class will store king specific attributes such as legal moves, model and point worth.~~

2.3 Mapping from requirements to classes

Requirement [4]	Classes providing requirement
FR1	Setup, SetupController, NewGame, NewGameController
FR2	Game, Save
FR3	Game, GameController
FR4	Game
FR5	Game, GameController, Pawn, Knight, Bishop, Rook, Queen, King
FR6	Game, GameController, King
FR7	Game, King
FR8	Game, EndScreenController, Quit
FR9	Quit
FR10	Replay
FR11	Replay

3. DEPENDENCY DESCRIPTION

3.1 Component Diagram

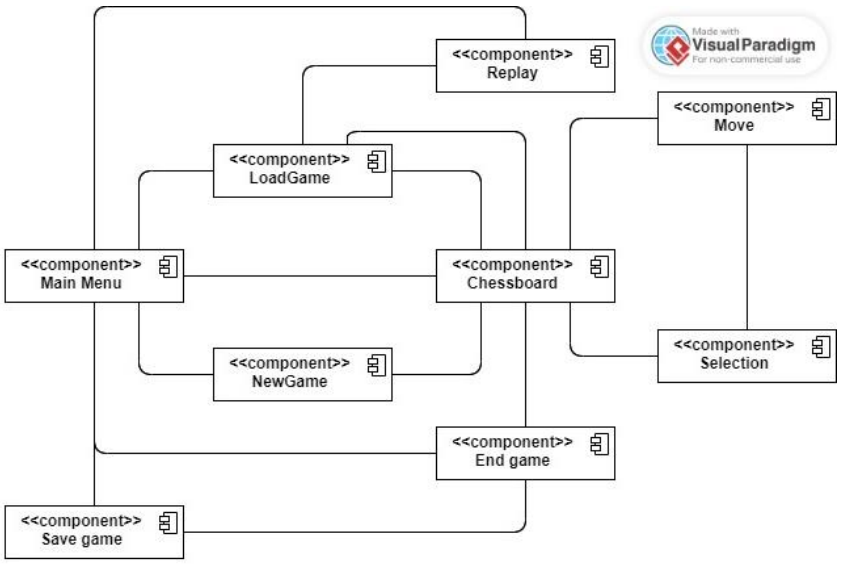


Figure 1: Component Diagram of the Chess Tutor Game System to show the relationships between modules.

Commented [TM[19]: Do we really need the piece subclasses here? High word repetition here.

What if we include that the existence of its subclasses and what they do in parent class Piece's description

Commented [TM[22]: Reference to spec where FRs are described.

Commented [D[23R22]: Reference the System requirements for the FRs under the scope section (1.2)

Commented [TM[24R22]: Action: added reference to references, top left cell 'Requirement [4]' and in scope (1.2)

[REVIEW]

Commented [TM[25]: New Controller classes to display different scenes and handle user input (done through graphical elements)

Commented [D[26]: FR9 talks about saving the game when a user quits. Perhaps saving should also be mentioned here?

Commented [D[27R26]: Maybe the game class also Since it will need to access the state of the board?

Commented [TM[28R26]: Not sure what implementation team is exactly doing for classes.

Commented [D[29]: This functional requirement also mentions storing. perhaps the save class should also be mentioned here?

Commented [D[30R29]: Maybe the game class also Since it will need to access the state of the board?

Commented [TM[31R29]: Not sure what implementation team is exactly doing for classes.

Commented [TM[32]: Diagram associations and components done and interfaces planned and thought out however software used has trouble implementing them.

So in summary, they're pretty much done.

--Update
James has worked on the diagrams

Commented [D[33R32]: The diagrams need updating to include a title, A player Icon and actions performed to link each component. See the example Design Spec for reference.

Commented [D[34R32]: Make sure we reference the software used also

Commented [D[35]: Add more of a description to this caption.

Commented [TM[36R35]: Adjusted [REVIEW]

4. INTERFACE DESCRIPTION

Commented [TM[37]: Psuedocode - gp17 / dev / 20230213 / Chess_Base_+_Pseudocode / Chess Base / src

Big thanks to Kieran btw for creating the pseudocode classes!

4.1 Game

- Type: public
- Extends: **nothing**
- Public methods:
  - checkOrCheckmate() – detects if the King of the opposite colour of the player who just had their turn is in check or checkmate
  - selectPiece(int xPos, int yPos) – highlights a piece on the board and displays its current moves.
  - deselectPiece(int xPos, int yPos) – removes highlighting of a selected piece and stops displaying its legal possible moves.
  - movePiece(int xPos, int yPos) – changes the position of a piece to a given new legal position.
  - removePiece(int xPos, int yPos) – removes a piece from the board when another piece legally captures/takes it.

4.2 Setup

- Type: public
- Extends: nothing
- Public methods:
  - whiteName() – gets user input for the name of the white player.
  - blackName() – gets user input for the name of the black player.
  - boardSetup() – sets up the chessboard array that the game will be played on. Called everytime a new game is started.
  - addPiecesToBoard() – adds all pieces to their starting positions on the chessboard array.

4.3 Piece

- Type**: public**
- Extends**: nothing**
- Public methods:
  - Piece(String colo**ur, int xPos, int yPos, int score) – constructor.**

4.4 Bishop

- Type: public
- Extends: **Piece**
- Public methods:
  - Piece(String **colour, int xPos, int yPos) – constructor.**

4.5 King

- Type: public
- Extends: Piece
- P**ublic methods:**
  - King**(String colour, int xPos, int yPos) – constructor.**

4.6 Knight

- Type: public
- Extends: Piece

- Public methods:
    - Knight**(String colour, int xPos, int yPos) – constructor.**

4.7 Pawn

- Type: public
- Extends: Piece
- Public methods:
    - Pawn**(String colour, int xPos, int yPos) – constructor.**

4.8 Queen

- Type: public
- Extends: Piece
- Public methods:
    - Queen**(String colour, int xPos, int yPos) – constructor.**

4.9 Rook

- Type: public
- Extends: Piece
- Public methods:
    - Rook**(String colour, int xPos, int yPos) – constructor.**

4.10 Replay

- Type: publ**ic**
- Extends: **nothing**
- Public methods:
    - saveGame(**String fileName) – saves the game data to a given save file (fileName) including the**
        - Type, colour and position of the pieces on the board.
        - Type and colour of taken pieces.
        - History of piece moves.
    - loadGame(String fileName)) – loads the game data from a given save file (fileName)
    - addMove(?) – adds a move to the history of past moves.

4.11 Quit

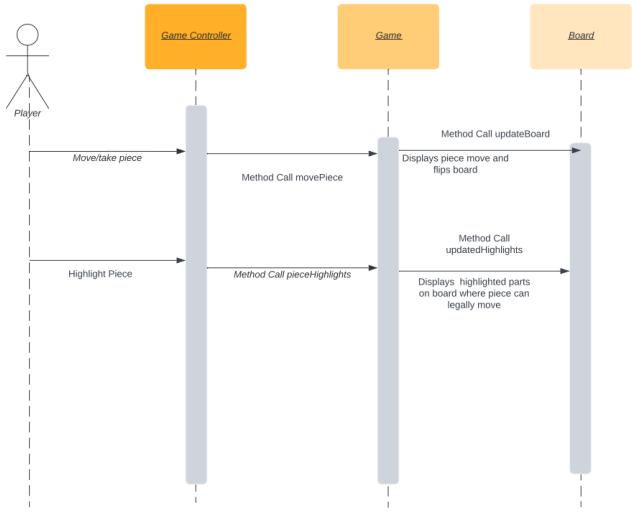
- Type: public
- Extends: nothing
- Public methods:
    - quitGame() – ends the game when players decide to quit.
    - gameFinished() – calls functions for when the game ends in checkmate, a mutual draw occurs or a player resigns.

5. DETAILED DESIGN

5.1 Sequence diagrams

Game Menus Sequence diagram:

Piece Sequence Diagram:



5.2 Significant algorithms

5.2.1 Move Generation

As stated in FR5 pieces must display all legal moves when highlighted. In chess there are some notable exceptions to the normal piece behaviours we will have to run checks on when displaying these moves.

Check/Disccovered check – If the king will get checked it will not display it as a valid move. The program will:

Find the king on the board, Check if any of the opponents pieces can attack the king, do checks to check if any of the opponents pieces can legally check the king using a function for each piece to check if it's a valid move or not.

Function is_in_check:

Find players king on the board

Check if any opponents pieces can attack the king

Funcion is valid move:

Check what piece it is attacking the king

Run through checks seeing if it's a legal/valid move

En Passant – When a pawn is moved two squares a flag will be brought up showing en passant will be legal next turn

5.3 Significant data structures

--The data structure used to store games

--Data structure used to store usernames and passwords

REFERENCES

Commented [D[38]]: Reformat All References. See Test Spec 1.0 for examples

Commented [TM[39R38]]: Redone [REVIEW]

[1] Loftus C.W., "Software Engineering Group Projects –
General Documentation Standards", 2.6, Computer Science Department, 10th January 2023

[2] Loftus C.W., "Software Engineering Group Projects –
Design Specification Standards", 2.3, Computer Science Department, 6th December 2021

[3] Moore T., Enache A., "Software Engineering Group 17 User Interface", 1.0, SE_GP17_UISpecification, 27th February 2023

[4] Loftus C.W., "Software Engineering Group Project - Chess Tutor Requirement Specification", 1.2, Computer Science Department, 28th February 2023

DOCUMENT HISTORY

Version	*Issue No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	7/02/23	N/A - original draft version	GWC1
0.2	11	14/02/23	Filled out section 1: Introduction	KIF11
0.3	11	14/02/23	Filled out section 2: Decomposition Description	GWC1
0.4	11	04/03/23	Adjusted section 2: Decomposition Description Added almost finished diagram (Figure 1)	TAM41
0.5	11	05/03/23	Added to section 4: Interface Description	TAM41
0.6	11	07/03/23	Added in references	TAM41
0.7	11	13/03/23	Added to section 5: Detailed Design	JCO3
0.8	11	13/03/23	Adjusted sections to review meeting feedback	TAM41
0.9	11	14/03/23	Adjusted references and scope.	TAM41