# DEFA 2020: Shakkipeli (A)

## Harjoitustyösuunnitelma

Teen graaffisen shakkipelin (yksin), jossa kaksi ihmistä pystyy pelamaan toisiaan vastaan. Yritän lisätä peliin kaikki mahdolliset shakkipelin säännöt mukaan lukien Checkmate, Check, Castling ja En Passantin, jos vain aika tauluni siihen riittää.

- Castling -> https://simple.wikipedia.org/wiki/Castling  
 - En Passant -> https://en.wikipedia.org/wiki/En\_passant

### Ratkaisuperiaate

Aloitettaan nappuloiden pääluokasta. Tämä sisältää Pelaajan(nappulan omistajan), nappulan liittouman(BLACK/WHITE), nappula tyypin(bishop, knight .. ect), nappulan koordinaatit laudalla ja staattisen sisäluokan Move.

Nappulan pääluokalla on absractiluokka, jonka aliluokissa kerttoaan kaikki mahdollisen nappulan liikkumiset.

Tehdään pelilauta ja laatta luokat. Laattaluokka sisältää laatan koordinaatit ja nappulan joka on laatan päällä. Pelilauta sisältää moniluotteisen taulukon laatta luokasta ja kaksi pelajaa.

Pelilautaluokassa valmistellaan pelilauta (tehdään 64(8x8) laattaa ja tietyille laatoille asetetaan tietyt nappulat).

Kun tämhän asti ollaan saatu valmiiksi näkyvä pelilauta consolissa Aloitetaan tekemään GUI:ta.

Guihin tulee pelauta, pelilaudan ympärille numerot ja kirjaimet, pelaajan vuoro, menetetyt nappulat paneeli ja valikko mistä voi aloittaa uuden pelin, ladata pelin ja tallentaa pelejä. Jos aikaa jää niin tehdään myös ajastin molemmille pelaajille (Esim speed chess 5min).

Näitteks lisäksi tarvitaan kaksi pomppuikkunaa. Yksi on sotilaan ylennykseen ja toinen

on check/Checkmate ilmoitukseen.

Sitkun kaikki tämä on valmis ja on aikaa niin ruvetaan tekemään shakin erikoisempia

Sääntöjä ja koodin korjailua ja parannusta.

Sääntöjä ja koodin korjailua ja parannusta.

Tärkeimpiin methodeja tulee varmasti lisää esim checkCheck, checkCheckMate.

### Luokat, tärkeimmät metdoit

#### Game:

* Piece.java – Nappuloiden pääluokka
  + Boolean canMove(List<Move> moves, Move destinationMove)
  + Abstract List<Move> getAllAvailableMoves(Board board)
  + void finishMove(Board board, int[] destCoords)
  + List<Move> getVerticalMovesDown(Board board, int maxTiles)
  + List<Move> getVerticalMovesUp(Board board, int maxTiles)
  + List<Move> getHorizontalMovesUp(Board board, int maxTiles)
  + List<Move> getHorizontalMovesDown(Board board, int maxTiles)
  + List<Move> getDiagonalMovesRightDown(Board board, int maxTiles)
  + List<Move> getDiagonalMovesRightUp(Board board, int maxTiles)
  + List<Move> getDiagonalMovesLeftDown(Board board, int maxTiles)
  + List<Move> getDiagonalMovesLeftUp(Board board, int maxTiles)
  + Static class Move
    - Bishop.java
    - King.java
    - Knight.java
    - Pawn.java
    - Queen.java
    - Rook.java
* Tile.java
  + void setPiece(Piece piece)
* Board.java
  + Tile[][] gameboard
  + Tile[][] defaultBoardSetup()
* Player.java
  + Boolean isKingAlive()
  + ArrayList<Piece> getLostPieces()
* FileManager.java
  + Boolean canSave
  + Boolean canLoad
  + Void loadGame()
  + Void saveGame()

#### GUI:

* UserInterface.java extends JFrame
  + UserInterface(Board board, Player p1, Player p2)
* GameBoardPanel.java extends JPanel
  + GameBoardPanel(Board board, DataPanel dataPanel)
  + Void setupBoardCoordinateLabels()
  + JPanel createNewTile(Board board, int[] pos)
  + Void assignTilePieceIcon(JPanel tile, Board board, int[] pos)
  + void refreshTiles(Board board)
  + void selectPiece(JPanel tile, Board board, int[] pos)
  + void delectPiece(Board board)
* DataPanel.java extends JPanel
  + Void setupDataPanel()
  + JPanel playerTurnPanel()
  + JPanel playerPanel(int posY, Player player, String headerText)
  + JPanel lostPiecesPanel(ArrayList<JPanel> lostPieces)
  + ArrayList<JPanel> setupLostPiecesList(Player player)
  + Void refreshDataPanel()