Project FlyingMoose

Laura Anzaldi

Ryan Cunningham

Eric Griswold

Corey Rausch

Laura Suckling

Derek Sweet

5/13/2012

Table of Contents

Chapter 1: Project Statement………………………………………………………3

Chapter 2: Quad Chart…………………………………………………………………4

Chapter 3: Skill Matrix & Risks……………………………………………………..6

Chapter 4: Schedule…………………………………………………………………….8

Chapter 5: Requirements…………………………………………………………….9

Chapter 6: Implementation plan………………………………………………..10

Chapter 7: Testing……………………………………………………………………..11

Appendix……………………………………………………………………………………12

Chapter 1

Project Statement

The goal of this project is for each team to develop a working digital version of the game Checkers for the customer, Dr. Sidhu.

The end-product should be an electronic version of the game Checkers in which the user can play a game of checkers against a secondary player, or against the computer. A single game should follow all of the standard rules of checkers: The board should be an eight by eight grid of sixty four squares of alternating color, each player should have twelve pieces, and all legal moves should be possible.

The game should include features that are considered standard for a digital game. For instance, the player should have the ability to start new games and exit games, surrender/forfeit a match, and should include a game timer. The AI player will provide a casual level of difficulty against the player. Also, a game statistics system will be created and implemented to record the player’s win and loss records, and number of total games played.

The back-end of the game will be coded using Java, and a GUI will be created to provide a user-friendly interface for playing the game. This GUI will indicate whose turn it currently is, and also present the board placement as the game progresses. Interaction with the GUI should rely mostly on the integration of mouse clicks. An options menu should be accessible during play if the user wishes to exit the game or forfeit a match.

The steps that we will take to complete this project include:

1. Develop the Overall Project Plan and Operational Requirement

2. Formulate Testable Requirements and Cases

3. Design the Software

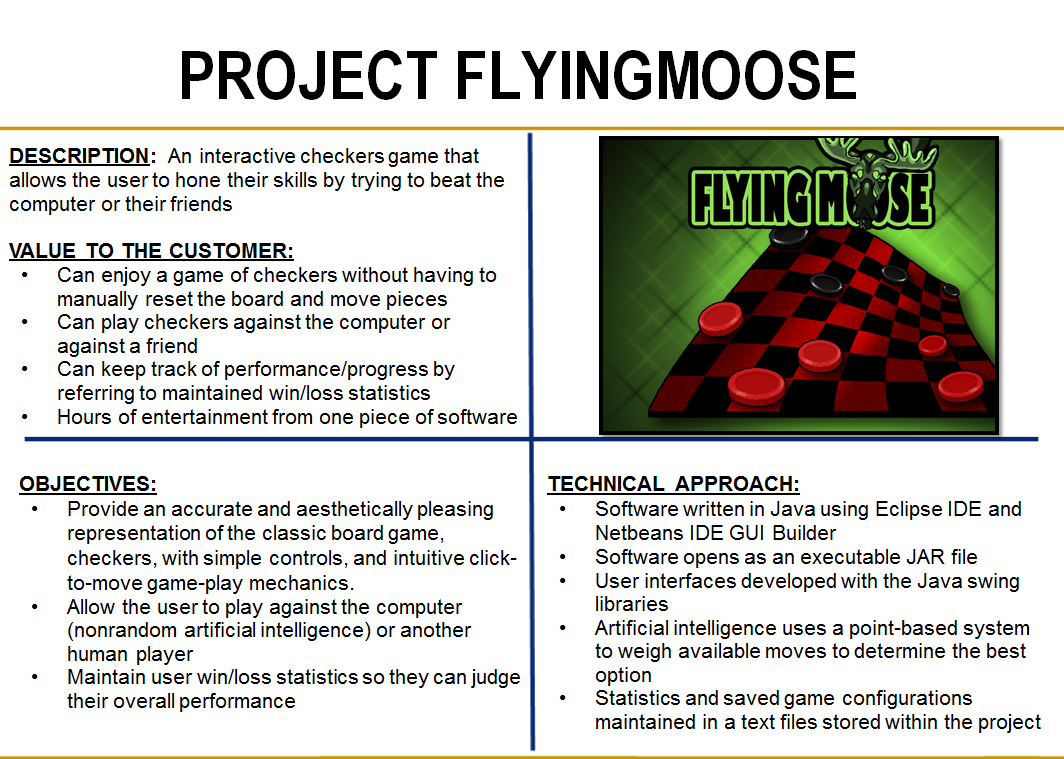
4. Implement the Software

5. Test the Software

6. Demonstrate the Software

Chapter 2

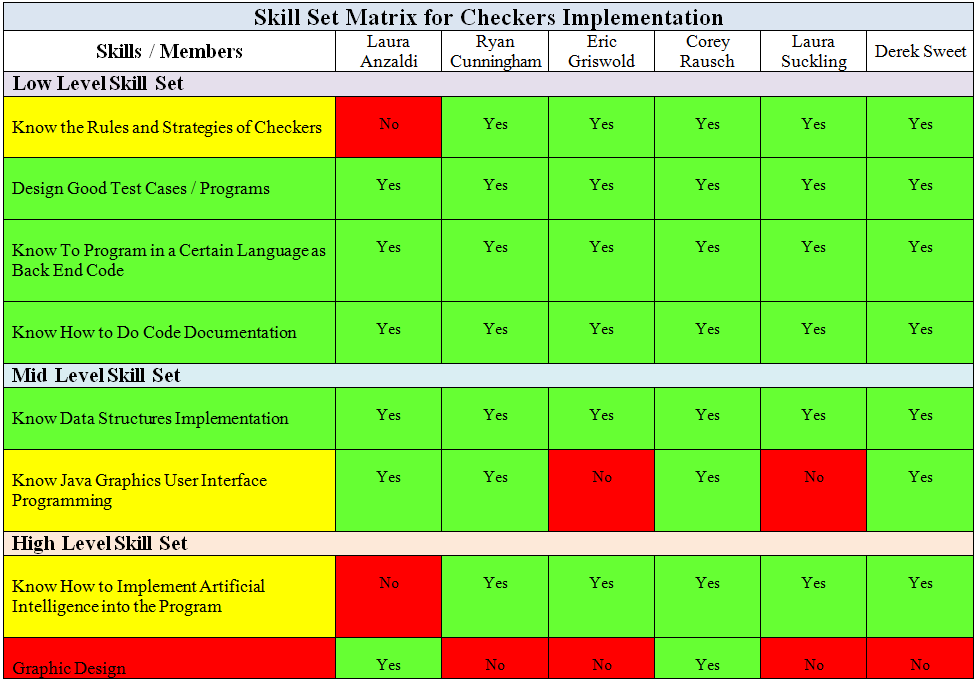
Quad Chart

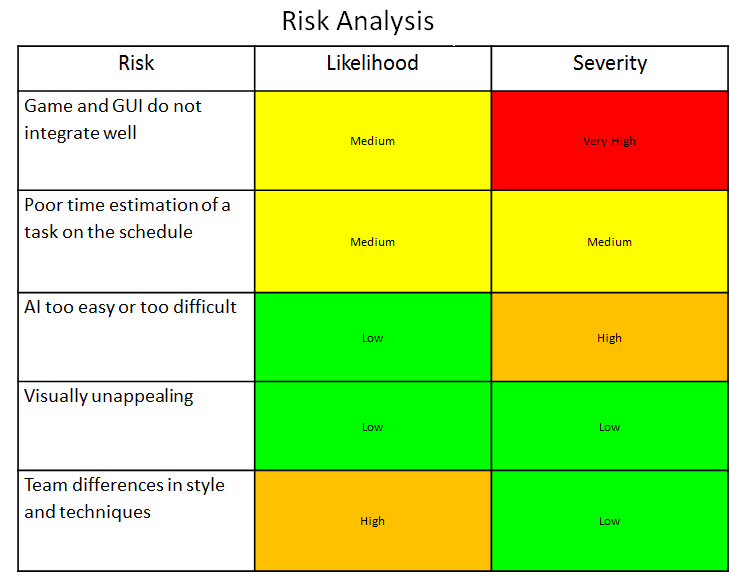


In summary, Project FlyingMoose offers a virtual checkers game experience, congruent with the standard US rules. It allows the user to play the classic game of checkers against the computer, which is powered by artificial intelligence, or against another human player. Project FlyingMoose maintains the user’s win and loss statistics separately for the two playing modes, which the user has the ability to reset if desired. This software is written in Java, and can be played on any system with a Java Virtual Machine. The game of checkers is easy to learn and fun to play—try out Project FlyingMoose today!

Chapter 3

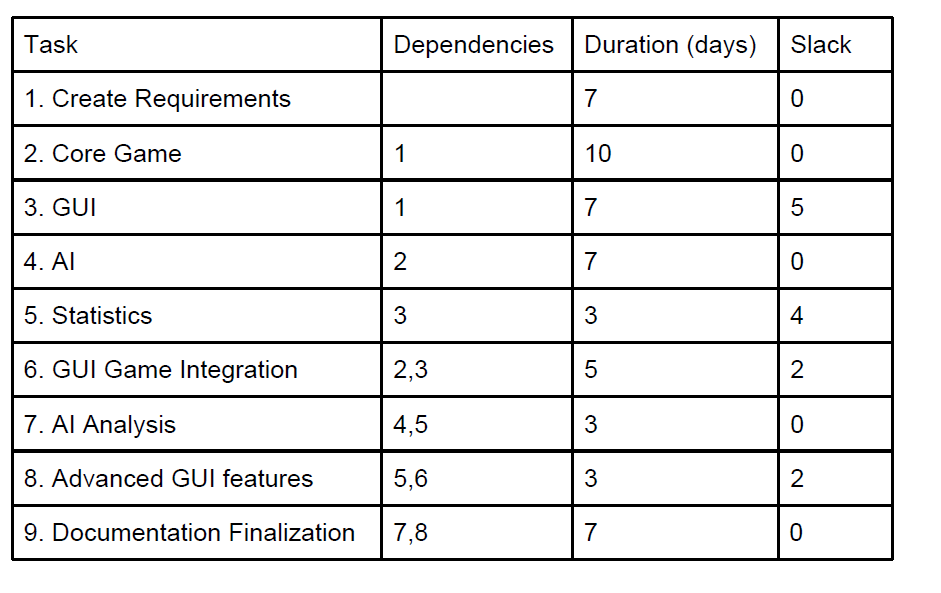
Skill Matrix and Risks

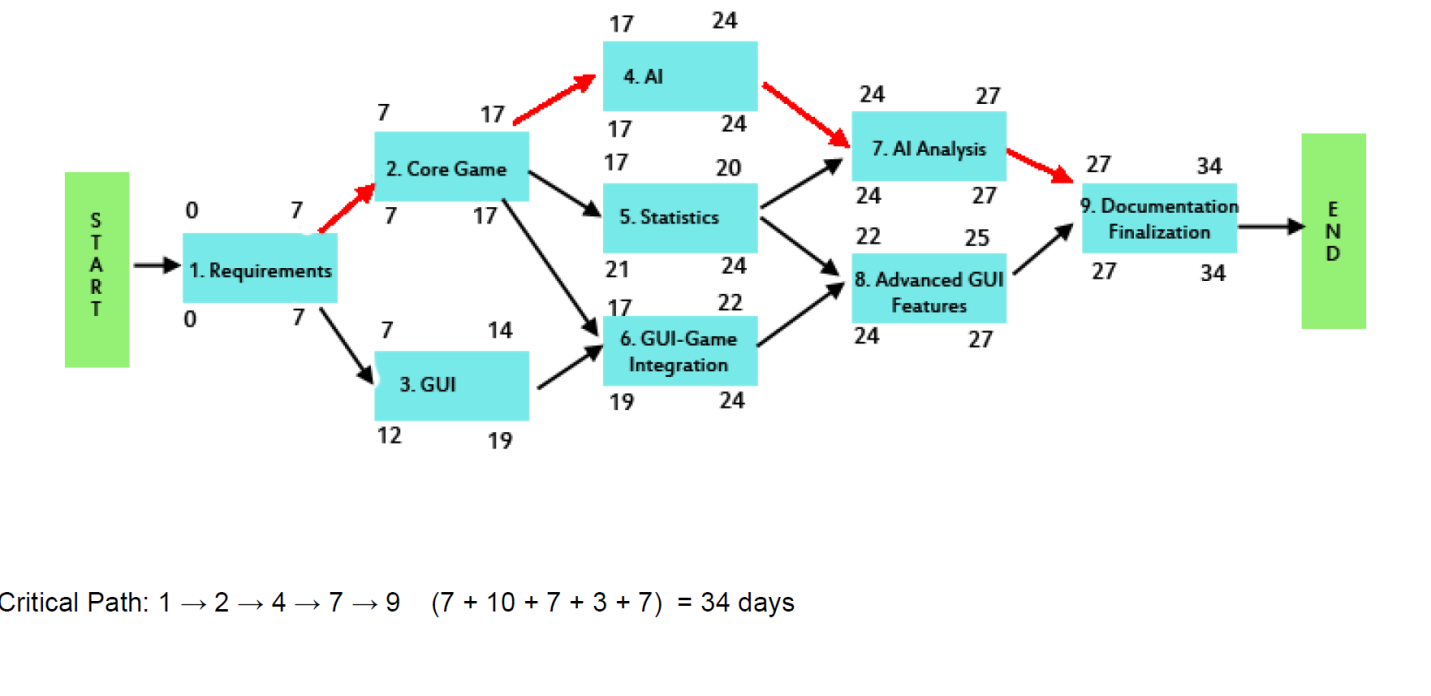




Chapter 4

Schedule





sdsdsd

Chapter 5

Requirements

Class Piesdsd

Chapter 6

Implementation Plan

Class Piece

* + Public int getColor() - Returns the color of the piece
  + Public void setColor(int color) – Sets the color value for this piece
  + Public boolean isKing() – Returns true if the piece is a king
  + setKing(Boolean king) – Sets the king property to true or false

Class coordinate

* + public int getX() – Returns the x coordinate
  + public int getY() – Returns the y coordinate
  + public int setX(int x) – Sets the x coordinate
  + public int setY(int y) – Sets the y coordinate
  + Public String toString() – Returns a formatted move
  + Public boolean equals(Object obj) – Returns true if the two objects are equal

Class Move

* + Public coordinate getSrc() – Gets the source coordinate
  + Public void setSrc(coordinate src) – Sets the srouce coordinate
  + Public coordinate getDst () – Gets the destination coordinate
  + Public void setDst(coordinate dst) – Sets the destination coordinate
  + Public String toString() – Returns a formatted move
  + Public boolean equals(Object obj) – Returns true if the two objects are equal

Class Board

* + Public void kingPiece(coordinate c) – Will king a piece at this location in the board
  + Public Piece getPiece(coordinate c) – Will get the piece at the location c in the board
  + Public boolean remove(int i, int j) – Will remove a piece from this coordinate, return true if a piece was here, false else
  + Public boolean move(Move m) – Will try and move a piece corresponding to move m, returns true if possible, false else
  + Public boolean redHere(int i, int j) – Will return true if this location contains a red piece, else false
  + Public boolean blackHere(int i, int j) – Will return true if this location contains a red piece, else false
  + Public boolean kingHere(int i, int j) – Will return true if this location contains a king piece, else false
  + Public String toString() – Returns a formatted board

Class Checkers

* + Public int getTurn() – Returns the player turn
  + Public void setTurn(int turn) – Sets the turn
  + Public int determineTurn() – Automatically determines the turn
  + Public boolean CPUturn() – will play a computer turn
  + Public boolean move(Move m, Move[] force) – Will attempt to play move m, true if possible, else false.
  + Public Move[] calcPossibleMoves() – Will return an array of all possible moves given the current board configuration and turn
  + Public Move[] calcForcedMoves() – Will return an array of all possible forced moves
  + Public boolean validMove(Move [] possible, Move m) – Will return true if the move is valid, else false. Will not play the move.
  + Public String toString() – Returns a formatted Checkers string

Class MoveList

* + Public boolean addMove(Move m) – Will attempt to add Move m to the current MoveList, will return true if m is not null, else false

Class AI

* + Public int Score(Board b, int color) – calculates and returns the score of the board in terms of pieces and piece locations with respect to the player (color) passed in
  + Public int BoardScore(board b) – Calculates the total score combing the individual player scores while also taking into account the number of available moves for each player. A higher value will represent a better score for black while a lower value will represent a better score for red.
  + Public int getLikely(Checkers c, Move m, int turn) – This function will calculate all of the possible playouts of a passed move and return the score of the most likely result of this move taking into account all possible forcejump scnerios. Will return the most likely scenario score.

Class Checker (GUI)

* + Public int getPosX() – Returns the x position of the piece
  + Public int getPosY() – Returns the y position of the piece
  + Public void setCoor(int x, int y) – Sets the new coordinate and checks if it’s a king
  + Public void paintComponent(Graphics page) – Creates the string that will be drawn on the screen
  + Public void kingMe() – Kings the checker piece
  + Public void select() – Selects which icon to use for the checker piece
  + Public toString() – Displays a string representing this piece

Class CheckerBoard (GUI)

* + Public void paintComponent(Graphics page) – Draws the background checkerboard

Class GameUI (GUI)

* + Public void boardUpdate() – Updates the board and redraws all of the pieces in the grid

-Class Clicklistner implements MouseListener (GUI, subclass of GameUI)

* + Public void mouseClicked(MouseEvent click) – Will respond to ever click and determine which coordinate was clicked. If a move is entered by clicks, it will be played out and the board will update with the current information. If a single piece is selected, that piece will be visible selected with a blue highlight

-Class ExitListner implements ActionListener (GUI, subclass of GameUI)

* + Public void actionPerformed(ActionEvent event) – Will exit when the exit button is clicked

Class MainScreen extends javax.swing.JFrame (GUI)

* + Public int getPosX() – Returns the x position of the piece
  + Public void initComponents – Initializes all of the components of the MainScreen
  + Public void newgame\_buttonMouseEntered(MouseEvent evt) – Highlights the box showing that the user has clicked it
  + Private void newgame\_buttonMouseClicked(MouseEvent evt) -- Opens a menu allowing the user to decide player vs player or player vs computer
  + private void pvpActionPerformed(java.awt.event.ActionEvent evt) – Opens and runs an instance of a player vs player game
  + private void pvcActionPerformed(java.awt.event.ActionEvent evt) -- Opens and runs an instance of a player vs computer game
  + private void newgame\_buttonMouseExited(java.awt.event.MouseEvent evt) – Updates the button graphics
  + private void loadgame\_buttonMouseEntered(java.awt.event.MouseEvent evt) – Updates the button graphics
  + private void loadgame\_buttonMouseExited(java.awt.event.MouseEvent evt) – Updates the button graphics
  + private void viewstats\_buttonMouseEntered(java.awt.event.MouseEvent evt) – Updates the button graphics
  + private void viewstats\_buttonMouseExited(java.awt.event.MouseEvent evt) – Updates the button graphics
  + private void pvpMouseExited(java.awt.event.MouseEvent evt) – Updates the button graphics
  + private void pvpMouseEntered(java.awt.event.MouseEvent evt – Updates the button graphics
  + private void pvcMouseEntered(java.awt.event.MouseEvent evt) – Updates the button graphics
  + private void pvcMouseExited(java.awt.event.MouseEvent evt) – Updates the button graphics
  + private void loadgame\_buttonMouseClicked(java.awt.event.MouseEvent evt)
  + private void pvpMouseClicked(java.awt.event.MouseEvent evt)
  + private void pvcMouseClicked(java.awt.event.MouseEvent evt
  + private void viewstats\_buttonMouseClicked(java.awt.event.MouseEvent evt) – Opens the stats screen
  + public int getUserChoice() –reads what the user clicked
  + public void closeMainScreen() –exits the main screen entirely
  + public static void main(String args[]) – The main function for the entire project

Chapter 7

Testing

1. Game Presentation
   1. Performance Tests
      1. Ensure that when the game begins, the grid is 8x8
         1. Ensure that spaces on the grid are viewable at all times
         2. Ensure that the grid colors are visually distinct
         3. Ensure that the grid is viewable from 1 to 2 feet away
      2. Ensure that the two players’ pieces are two different colors
         1. Ensure the pieces are visually distinct from each other
         2. Ensure that the pieces are visually distinct from the grid
         3. Ensure that the regular pieces are visually distinct from the kings
2. Game Mechanics
   1. Functional Tests
      1. Ensure that at the beginning of the game all of the pieces are in the proper location
      2. Ensure that the black player gets the first move
      3. Ensure that there is a surrender option
      4. Ensure that after surrendering the game ends and a loss is counted
      5. Ensure that jumping a piece results in its removal
      6. Ensure that forced jumps are properly handled
      7. Ensure that multi-jumping works correctly
      8. Ensure that pieces are kinged when they reach the other side of the board
      9. Ensure that kings can properly move forward and backward
      10. Ensure that only valid moves are permitted by the game
      11. Ensure that the game lets the user know if an attempted move is invalid
      12. Ensure that the end of the game is properly determined
      13. Ensure that the player can only move on their turn
   2. Performance Tests
      1. Ensure that turn-taking determined by the computer in an unnoticeable amount of time
      2. Ensure that an invalid move is determined in an unnoticeable amount of time
      3. Ensure that losses are determined in an unnoticeable amount of time
      4. Ensure that the game timer keeps accurate time
      5. Ensure that the player can play a new game upon the completion of a game
      6. Ensure that the game does not ever crash or freeze
3. GUI
   1. Functional Tests
      1. New Game
         1. Ensure the user has the option to play against the computer or another user
         2. Ensure the game board is 8x8
         3. Ensure pieces are in their proper starting position
         4. Ensure no pieces are in the center of the board
         5. Ensure all 24 pieces are on the board
      2. Save Game
         1. Ensure that the user is prompted for a save when the exit a game
      3. Load Game
         1. Ensure that the game correctly loads a saved game
      4. Surrender Game
         1. Ensure that the surrender option functions correctly
         2. Ensure surrendering results in a recorded loss
         3. Ensure that on surrender the user is prompted to exit the game or start a new game
         4. Ensure that if the user exits all statistics are saved
      5. Ability to Exit Game
         1. Ensure that the user is prompted for a save when they exit
         2. Ensure that the user is notified that an unfinished game is counted as a loss
         3. Ensure that the game then reprompts for a save
         4. Ensure that statistics are properly saved
      6. Options Menu
         1. Ensure that the Options Menu contains New Game, Load Game, Surrender Game, Exit Game, Show Statistics, and Help.
         2. Ensure that selecting each option results in the proper action
      7. Mouse Interaction
         1. Ensure that the mouse moves according to the user mouse
         2. Ensure that clicking the board results in the intended action
      8. Display Player’s Turn
         1. Ensure that the current turn is displayed
   2. Performance Tests
      1. Platform Independence
         1. Ensure that the game can be played on Windows XP, Windows 7, GL Linux Servers
         2. Ensure that the look of the game remains the same on all platforms
      2. Steady Frame Rate
         1. Ensure that the game moves fast enough to not hinder gameplay
         2. Ensure that game updates do not appear instantaneous
      3. Pieces move to correct locations
         1. Ensure that pieces move according to the player’s clicks
4. AI
   1. Performance Tests
      1. The AI shall have a response time of less than, or equal to one second.
      2. The AI shall limit the amount of memory used for calculating moves in order to perform efficiently.
      3. The call to the AI shall be simple and require little preparation.
      4. The AI shall always produce a "difficulty appropriate" move under most circumstances.
5. Statistics
   1. Functional Tests
      1. Ensure that the statistics are recorded properly
      2. Ensure that the statistics can be viewed properly
      3. Ensure that the fastest game time can be viewed by the user
      4. Ensure that statistics for AI and Human are maintained separately
      5. Ensure that a loss is recorded on the event of a game exit or surrender
   2. Performance Tests
      1. Ensure that the statistics file is located in the project directory and is smaller than 1 MB
      2. Ensure that the statistics are loaded and saved in under 2 seconds

Appendix

A: Full Set of Requirements…..…………………………………………………….1

B: Visual tools…..……………………………………………………………………..….1

C: Anything Else….……………………………………………………………………….1

Appendix A

Full Set of Requirements

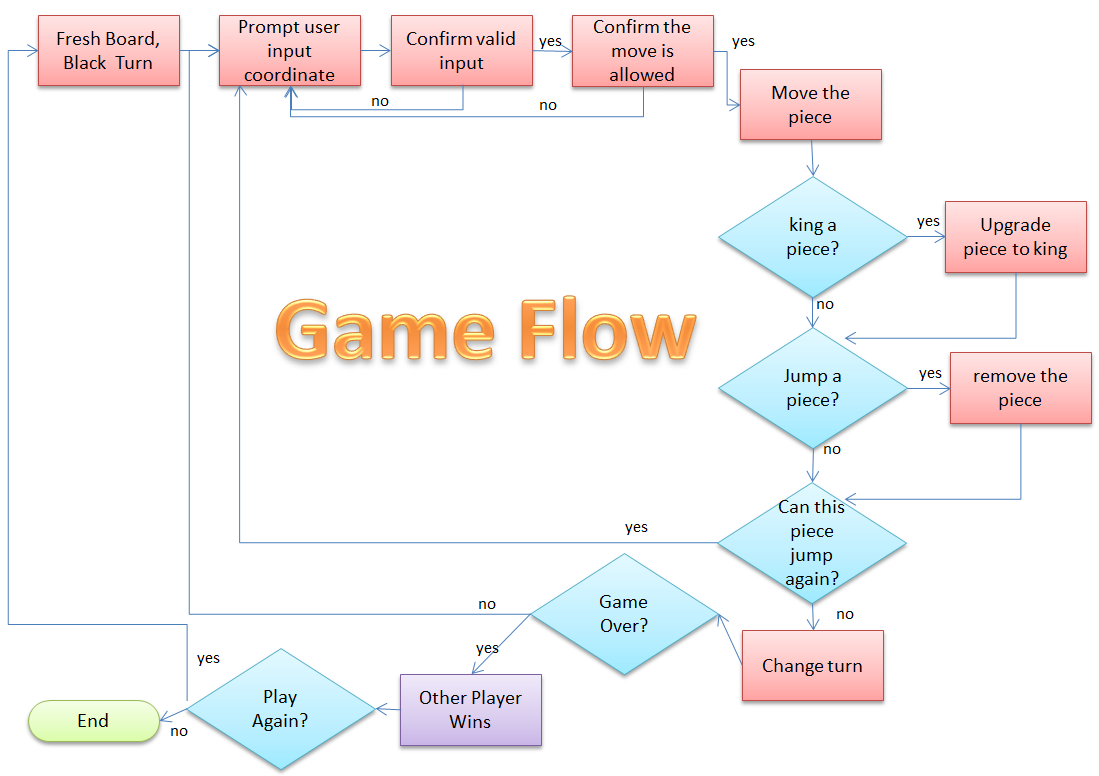
Project Requirements

1. Game Presentation requirements
   1. Performance Requirements
      1. Display 8x8 Grid
         1. All spaces on the grid shall be visible to the player at all times.
         2. The grid colors shall contrast such that they do not blend together.
         3. The grid shall be large enough to identify blank spaces and pieces from a reasonable viewing distance (approximately 1-2 feet from the screen)
      2. Provide two different piece colors (e.g. Red, Black)
         1. Piece colors shall be different enough so that the player can differentiate between their pieces and their opponent’s pieces.
         2. Pieces shall be colored or decorated such that they do not blend into the board.
         3. Kings and regular pieces shall be identifiably different in appearance.
2. Game Mechanics
   1. Functional Requirements
      1. The game shall begin with all pieces in the proper location.
      2. The black player shall begin with the first move.
      3. The human player shall have the option to surrender.
      4. Surrendering shall result in a loss and consequently end the current game.
      5. When a piece is jumped it shall be removed from play and be replaced by an empty block.
      6. The player shall take any forced jump if possible.
      7. The game shall permit and enforce multi-jumping, which forces a player to jump any additional pieces after its first jump so long as an available jump still exists.
      8. The pieces shall turn into a king when they reach their opponents side of the board.
      9. The kings shall be able to move both forwards and backwards, including any combination of the two during a multi-jump.
      10. The program shall check move validity, and ignore invalid moves
      11. The program shall indicate that a move is invalid.
      12. The game shall check end game conditions, and end the game when either player has no available legal moves and determine the winner.
      13. The player shall only be allowed to move his own pieces, on his own turn.
   2. Performance
      1. The game shall determine whose turn it is after a move in an unnoticeable amount of time (UAT).
      2. The game shall determine if a move is valid in an UAT.
      3. The game shall determine if a player has no available moves and therefore loses in an UAT.
      4. The game timer shall keep accurate time.
      5. At the conclusion of a game, the program shall not exit, but permit successive play and navigation of the options menu.
      6. The game shall not crash or lock up.
3. GUI Requirements
   1. Functional
      1. New Game
         1. The game shall ask the user the whether to play single player (against AI) or two player (against another human player) and the level of difficulty shall be asked if the single player option is picked.
         2. Game Board shall be 8 by 8 with alternating color blocks.
         3. The top three rows shall contain pieces of the same color, while the bottom three rows shall contain pieces of the other color, and placed on every other square of a specific color.
         4. Two center rows shall be empty.
         5. All 24 pieces shall be present (12 of one color and 12 of the other color).
      2. Save Game
         1. When user exits during an unfinished game, the game shall ask the user if he wants to save.
      3. Load Game
         1. The game shall load the last save game configurations and statistics.
      4. Surrender Game
         1. Player shall be able to surrender the game at any time.
         2. The game shall be counted as a loss if surrender is chosen.
         3. The game shall ask if the player wants to start a new game or exit the application.
         4. If the user exits the application, the statistics shall be saved.
      5. Ability to Exit Game
         1. The game shall ask if the user wants to save, if the user exits without saving.
         2. The game shall tell user that the game will be counted as a loss, if user does not save.
         3. The game shall re-prompt the user if he wants to save.
         4. The statistics shall be saved.
      6. Options Menu
         1. The Options Menu shall contain New Game, Load Game, Surrender Game, Exit Game, Show Statistics, and Help.
         2. Each option in the option menu shall change the board configuration according to which option was selected.
      7. Mouse Interaction
         1. The mouse shall move according to what the user wants.
         2. The mouse shall be able to change the board according to what the user indicated.
      8. Display Player’s Turn
         1. The player’s turn shall be displayed on the physical interface and not hidden, making it easy to be seen.
   2. Performance
      1. Platform Independence
         1. Game application shall be portable and work properly on all 32-bit platforms, (Windows XP, Windows 7, GL Linux Servers).
         2. Game application shall look the same on any platform.
      2. Steady Frame Rate
         1. Speed shall be fast enough that it does not slow down the game progression.
         2. Speed shall also not be so fast such that board configuration updates appear instantaneous.
      3. Pieces move to correct locations
         1. The Correct piece shall be moved to the designated location on the game board when the player makes a move.
4. AI Requirements
   1. Performance Requirements
      1. The AI shall have a response time of less than, or equal to one second.
      2. The AI shall limit the amount of memory used for calculating moves in order to perform efficiently.
      3. The call to the AI shall be simple and require little preparation.
      4. The AI shall always produce a "difficulty appropriate" move under most circumstances.
5. Statistics Requirements
   1. Functional Requirements
      1. Statistics shall be able to log the number of wins, losses, and ties
      2. At the end of each game, record wins, losses, and ties so that the user shall be able to go back to view statistics from total games played.
      3. The fastest game time achieved by the user shall be recorded and retrievable by the user
      4. Statistics for playing against the AI and playing against a human player shall be maintained separately.
      5. A loss shall be counted and updated to the file in the event of surrender or unexpected exit.
   2. Performance Requirements
      1. The statistics file shall be light-weight and stored in the program directory.
      2. Statistics shall be loaded and saved within a reasonable amount of time.

Appendix B

Visual Tools





Appendix C