Software requirements specification for a checkers application computer science

The project is a checkers game, which is implemented in android and contains some of its important features to the user such as single player (vs computer), and two player, save game.

Purpose

The purpose of this document is to describe the behavior of the Checkers game, including non-functional requirements of the checkers game, constraints and other factors which provide a complete and comprehensive description about the game.

Scope

In this document basic overview of the game will be described with all functional and non-functional requirements. Actors of the project will be identified along with the description of the Use Cases performed by these actors describing each of the use case with GUIs for each use case. System Architecture of the game will be discussed with the GUI's related to the actors and use cases.

Overview

Game checkers provides certain facilities (functionalities) to the user, to solve the user requirements (1 player, 2 players).

Product Perspective

In the project of checkers, the product perspective is to provide complete interface where user can play single player and multiplayer game in android envoirment. GUI will be provided to facilitate this purpose.

Product Functions

Single-player and Multi-player option:

The main function of checkers is to provide an interface where user will be able to choose their desired game and play according to their requirements. Auto player option will also be also given where computer will play on one side.

User Characteristics

Special features will be provided to the user to fulfill the requirements of the user. There are three different characteristics that will be completed at the end of this project.

Goals for using system

The goal for using this system is just to get entertained by this game in the new enviorment of android.

Potentials patterns of use

Regarding game of checkers, some users just play a game just for entertainment while other users might play a game occasionally. This game will facilitate all types of users whether they play just for entertainment or play on regular basis.

Demographics

There is no age range and educational background is required to play this game. Every type of users can play a game. This game can be easily played by adults and children. You just need to have knowledge how to play this game.

Constraints

The game will be implemented in Android and Java, and special interactive GUIS are made for the user.

Functional Requirements

System shall be able to play against the single player.

System shall be able to keep check of the valid moves.

System shall be able to keep check of the invalid moves.

System shall be able to tell at the end that which player has won the match.

System shall be able to keep track that currently which player has turn either player 1 or player 2.

System will keep track of the valid kill moves.

System shall be able not to allow the player to take wrong moves.

System shall be able to tell which player has won the game.

Non-Functional RequirementsPlatform:

The game should be implemented in android platform.

Response:

Early response will be given to the user on his/her every action. The user will not have to be wait for a long time for the response from the system.

Reliability:

The system will be reliable as the user is confirmed that no invalid moves will be performed as discussed at the Checkers Rule at the end of the document.

Actors

Following are the actor(s) of the system:

Users

Users:

First of all every type of users of any kind can interact with the game. They can be kids, adults or old people. Their role will be the same in the game.

Use-Cases

This section specifies the use-cases for the system:

One player or Two players

Moves in the game by user

One Player or Two PlayersBrief Description

This use-case captures steps to show how to start new game of checkers game.

Pre-condition(s)

The user should be on menu page .

Main Flow

User clicks on the “ 1 Player” to play single player game.

1. A. User clicks on the “ 2 Player” to play two players game.

System starts a new game either one player or two players.

Post-condition(s)

The user started playing the game.

Normal Moves in the game by the userBrief Description

This usecase shows the normal valid moves made by the user.

Pre-condition(s)

The user is playing game.

Main Flow

User pressed a piece.

User clicked the particular selected piece and then click on the empty box where the user wants to place that particular piece.

The system will check that position where the user has dropped the piece.

If at the new position of the piece there is already a piece then the piece which was dropped will come to its original position.

if at the new position there exist no piece and if the move of the piece is according to the rules of checkers given at the end of the document. Then the piece will be placed at this new position.

If the move of the piece is invalid then the piece will come to its original position.

Alternative Flows

3A. The place where the piece is dropped is not correct according to the rules of checkers given at the end of the document therefore it will be an invalid move.

Therefore the piece will again come at its previous position. And the user has to again jump on to step 1 of the Main Flow.

Post-condition(s)

The user is playing checkers.

GUIs

The main components which are used in making of this GUI are:

Linear Layout

Frame Layout

Image View

Text View

The property of Linear Layout is that when you add child components in Linear layout, they are automatically arranged. Now there is a parent Linear Layout which has 8 child Linear Layouts which are automatically arranged horizontally.

Now in each child Linear Layout 8 Frame Layouts are added.

The box pointed by arrow is one frame layout. Similarly 8 frame layouts are added in one child linear layout. The property of frame layout is that one image present as a child in frame layout can be over covered by other components. This particular property of frame layout is used here. As the box which is shown is basically an image of grey box which is added as first child in frame layout. Similarly, the pieces are also images, as two pieces are used, so black piece image and red piece image are also added in frame layout as its children. Now there is one frame layout having three child images (for images Image View is used), white or grey box (as first child of frame layout), black piece as second child of frame layout and red piece as third child of frame layout. So basically 64 frame layouts are made in the pattern described above.

How the pieces are positioned in the checker board:

As it is discussed in the previous paragraph that each frame layout has three children images, one is box(grey or white) and other two children are red and black pieces. Now there is an function of setvisible. Basically setvisible function is used to visible or invisible the component, if stevisible is passed true value then that particular component will be visible on screen and if setvisible is passed false value then that particular component will be disappeared from the screen. So, initially according to the rules of the checkers intial start positions, at red pieces positions, red pieces images are shown by giving the true value to setvisible function for red pieces, similar is the case with the black pieces.

How the pieces are moved in the checker board:

Each framelayout is given an id, so we can know the framelayout through its id. Whenever user on its turn clicks on its piece, that piece will be dissappeard from that position and when the user clicks on the new position if that particular position is valid position then the piece will shown there as its valid movement otherwise the piece will come back to its original position. These movements are discussed below:

None of the piece is selected. Now i will click on the piece shown by arrow:

When I will click on the piece that particular piece is removed from its original position as shown below by arrow pointing that position:

Now I will place the piece at the right position according to the rules of checkers:

As I have placed the piece at new valid position so that the piece will be placed and shown at the new position.

Kill move of the piece:

Consider the following scenario:

The kill scenario is shown by the three arrows, one arrow is pointing at the red piece which is going to kill the black piece pointed by the arrow and then red piece will move to the new position, pointed by the third arrow.

So by taking this move of red piece is placed at the new position and the black piece is removed from its position.

AI Concept

Min-Max algorithm is used in single player game, where the computer chooses its best possible move, also considering the moves of the user .

The minimax theorem states:

For every two-person, zero-sum game with finite strategies, there exists a value V and a mixed strategy for each player, such that (a) Given player 2's strategy, the best payoff possible for player 1 is V, and (b) Given player 1's strategy, the best payoff possible for player 2 is A? E†aˆ™V.

Example B chooses B1 B chooses B2 B chooses B3

A chooses A1 +3 A? E†aˆ™2 +2

A chooses A2 A? E†aˆ™1 A? a,¬aˆ? 0 +4

A chooses A3 A? E†aˆ™4 A? E†aˆ™3 +1

The following example of a zero-sum game, where A and B make simultaneous moves, illustrates minimax solutions. Suppose each player has three choices and consider the payoff matrix for A displayed at right. Assume the payoff matrix for B is the same matrix with the signs reversed (i. e. if the choices are A1 and B1 then B pays 3 to A). Then, the minimax choice for A is A2 since the worst possible result is then having to pay 1, while the simple minimax choice for B is B2 since the worst possible result is then no payment. However, this solution is not stable, since if B believes A will choose A2 then B will choose B1 to gain 1; then if A believes B will choose B1 then A will choose A1 to gain 3; and then B will choose B2; and eventually both players will realize the difficulty of making a choice. So a more stable strategy is needed.

Some choices are dominated by others and can be eliminated: A will not choose A3 since either A1 or A2 will produce a better result, no matter what B chooses; B will not choose B3 since B2 will produce a better result, no matter what A chooses.

A can avoid having to make an expected payment of more than 1/3 by choosing A1 with probability 1/6 and A2 with probability 5/6, no matter what B chooses. B can ensure an expected gain of at least 1/3 by using a randomized strategy of choosing B1 with probability 1/3 and B2 with probability 2/3, no matter what A chooses. These mixed minimax strategies are now stable and cannot be improved.

Coding Details

XML Code for the basic GUI:

```
android: orientation=” vertical”

android: layout_width=” fill_parent”

android: layout_height=” fill_parent”

>
```

In the following XML code one Linear Layout contains 8 Linear layouts are added. Each linear layout contains 8 child framelayouts and each framelayout contains the background image and two pieces images as children and these images are used by using Image layout.

Min-Max Algorithm(for computer move):

```
public int minMax(int []board)

{

int finalValue= 0;

ArrayList value= new ArrayList ();

ArrayList move= new ArrayList ();

int start=-1, end=-1;

if(currPos>= 1&&currPos <= 8)

{
```

```
start= 1;

end= 8;

}

else if(currPos>= 9&&currPos <= 16)

{

start= 9;

end= 16;

}

else if(currPos>= 17
```