Requirements Document

SOEN 6011 Tic-Tac-Toe Project

XOGEEKS

Assignment 2

Team - 4

Beerpreet Singh Guliani 27644930

Keerthana Gudavalli 27588569

Neha Sharma 27733240

Ramanjeet Gill 27732902

Ramanjit Dhillon 40010613

Shidokht Hejazi Sepehr 40002808

Sushil Patil 27148917

Submitted To : Prof Nicolangelo Piccirilli

Table of Contents

Table of Figures

Requirements

A. Problem

Developing ‘Tic-Tac-Toe’ a two player game is the main goal of our project. It is an entertaining game which is easy to download, install and play for everyone. The interface of Tic-Tac-Toe we are developing will be user friendly, so it is easy to learn the directions of how to play.

The concept is not logical in depth, hence it relieves the player from any stressful conditions and gives a good exercise to mind. For this two player game a 3*3 grid space is to be developed. So, as a development team of Tic-Tac-Toe game, our project focuses on developing this application using java in the Eclipse environment. Android Studio is the official IDE chosen to develop the Tic-Tac-Toe android application.

The whole team has put efforts to brainstorm the ideas to gather requirements. As in the beginning we don't have a clear idea on developing the game, it could take a lot of time to gather information and then implement them. As we progress stage by stage we’ll gain many skills of how to make an application progressive. With these confidence and experience, we hope to become a good business analysts in the future and build necessary teamwork and programming skills.

B. Background Information

The Tic Tac Toe game uses logics and presents different moves in game. In this game we’ll use Java application of 3*3 grid of buttons for cells. We require two users to play this game at first. Later, one user will be enough to begin a game with the computer if there is no second player. This game will be portable as it can be played on computer as desktop application as well as mobile application on mobile platform. In this game users use O and X signs to play in a matrix which consists of 3 rows and 3 columns or nine squares. The strategy of a game for winning is to place consecutively same three signs in horizontal, vertical and diagonal direction by making alternating moves.

Our team is developing ‘Tic Tac Toe’ by using Java, Junit, android studio. In first deliverable, users can select their sign and they can keep their name according to their desires. They start putting marks on the board by taking alternate turns. In deliverable 2, we are going to show status of game as win or lose or tied based on the game criteria if any player makes straight or diagonal line of consecutively same three signs. In the game, player can play multiple rounds in one game and player’s score will be displayed on system’s screen. In third deliverable user can play with the computer in different difficulty levels.

C. Requirements

Functional and non-functional requirements for each deliverable are included in the following table along with their priorities.

Deliverable 1:

Name	Description	Type	Priority
3*3 Board displayed on the screen for game	After starting the game the screen will display a 3*3 Board on which the game will be played	Functional	Must Have
User can exit the game	The player will be provided the option to close the game at any point of time.	Functional	Must Have
User can reset the board	The player will be able to reset the game whenever he wants.	Functional	Must Have
User can choose mark	By default "Player 1" will be represented by 'X' and "Player 2" will be represented by 'O' on the board. The user has the ability to change the mark he wants to play the game with.	Functional	Must Have
User can set player's name	By default "Player 1" and "Player 2" are set player names, the user can change the name if desired. Also the player named as player 1 will play first.	Functional	Nice to Have
System switches user turn	The game starts with player 1 and auto switching of the players will take place after each move.	Functional	Must Have
User can put dedicated mark in an	A mark will be put on any cell on the board when the user clicks on it only if	Functional	Must Have

empty cell	it is empty.		
Learnability	Help facility would be there at all points of time to guide the user and show them how to play the game.		
Operability	- Messages will be informative and displayed only when an action is necessary regarding the game. - The interface will be user friendly and well-formed so that it's easy to learn and operate it.	Non-Functional	Nice to Have
Modifiability	The user interface should be easily modifiable since new features will have to be added and displayed later.	Non-Functional	Must Have
User error protection	System is prepared for wrong or invalid inputs (names) and selections by the user and will make responses accordingly.	Non-Functional	Nice to Have

Deliverable 2:

Name	**Description**	**Type**	**Priority**
User can start a new game	The player can start a new game at any point of time.	Functional	Must Have
System displays the mark of the player who has the next	The player having the next move will be indicated by displaying their mark and name on the screen.	Functional	Must Have

move.			
System determines win, lose, or tied state	On the basis of the game criteria if three marks of the same kind come in a straight line then the player with that mark will win and the one with the opposite mark will lose else there would be a tie if the board is full.	Functional	Must Have
Multiple rounds in one game	The player can choose the “best of 3” or “best of 5” options before the start of the game. Players will enter the details once at the initial point. Once a round is finished, player may choose to continue to the next round until the 3rd or 5th round (depending on the option selected).	Functional	Must Have
System displays player’s scores	The scoreboard of the game will maintained and updated by the system until the window is closed or a new game is started.	Functional	Must Have
Portability	The game will be compatible on both android devices and desktop computer.	Non-Functional	Must Have
Maintainability	GitHub issue tracker will be used to track issues reported by users and apply fixes as necessary.	Non-Functional	Nice to have
Extensibility	Increasing the scope will be taken into consideration while designing the game to include a computer player	Non Functional	Could have

	and also incorporate features such as maintaining a record of players in a Database and displaying top 10 in later releases.		
Response time	The player turn and selected cell will be updated immediately after he clicks on the cell. The scores of each round are also immediately updated when game is won or tied.	Non-Functional	Must have
User error protection	System is prepared for invalid cell selections by the user and will make responses accordingly.	Non-Functional	Nice to Have

Deliverable 3:

Name	**Description**	**Type**	**Priority**
System provides different levels of game play	Various difficulty levels namely beginner, intermediate and advanced would be available for the players to provide a variation in the game. In the beginner level, there will be random moves keeping in mind the easy understandability for the user. In the difficult level, the computer player is programmed in such a manner that it blocks the path of the user to eliminate its winning chances after each move and increase its own. The intermediate level is the mixture of the two levels.	Functional	Must Have

System would play background music for each game	Different kinds of music would be played in the background depending upon player vs player or player vs computer to maintain the interest.	Functional	Must Have
User can choose the mark	The player has the ability to choose ‘X’ or ‘O’ as its mark	Functional	Must Have
System gives rewards to the winning player	Winning music and special icons will be displayed for player after each win.	Functional	Must Have
User can play in two different modes	The user would have the ability to play with the device itself or in a 2 player mode as earlier.	Functional	Must Have
User is able to select turn	While playing with the system the human player by default gets the first move, but the user has the ability to change that.	Functional	Nice to have
Extensibility	While designing the system extensibility would be kept into consideration so that in the later releases new algorithms could be added for the computer player without introducing defects.	Non-Functional	Could have
Maintainability	GitHub issue tracker will be used to track issues reported by users and apply fixes as necessary.	Non-Functional	Nice to have
Usability (User interface aesthetics)	Displaying attractive icons and music will help provide a nicer experience for the user.	Non-Functional	Must have
Performance	In designing the AI, best practices	Non-Functional	Nice to have

efficiency	will be used to decrease the amount of processing time required to determine the next move.		
Portability	Application can be converted to an android app.	Non-Functional	Nice to have

Use Case Analysis

Deliverable 1

The following are the use case diagram and use case description for deliverable 1

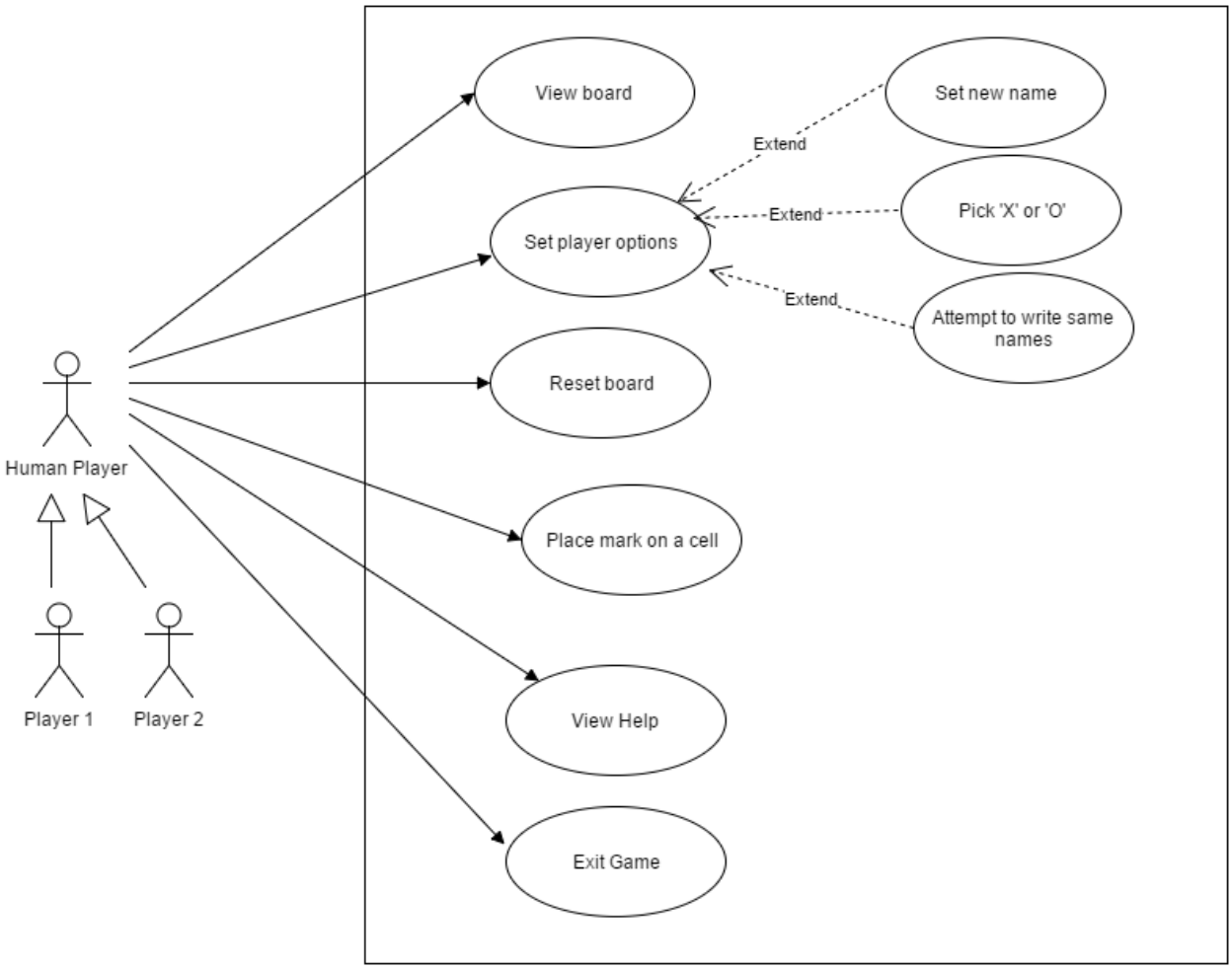

Fig 1: Deliverable 1-Use case Diagram

Use case description-1

<table>
<tr><td>Name</td><td>Set player options</td></tr>
<tr><td>Actor</td><td>Human Player (one or both of the players can perform this use case)</td></tr>
<tr><td>Actor’s goal</td><td>Set player’s name and mark</td></tr>
<tr><td>Summary</td><td>Players are differentiated during the game by their name. The name can be the default set by the system or players can write their own unique name. Also players are assigned ‘X’ or ‘O’ by default. They have the option to change their mark.</td></tr>
<tr><td>Precondition</td><td>-</td></tr>
<tr><td>Post condition</td><td>Both players have unique names and a dedicated mark; ‘X’ or ‘O’</td></tr>
<tr><td>Related use cases</td><td>“Set new name” and “pick ‘X’ or ‘O’” are extensions of this use case. “Attempt to write same names” is the exception (extension) for this use case</td></tr>
<tr><td>Main success scenario</td><td>
<table>
<tr><td>Actor’s actions</td><td>System responses</td></tr>
<tr><td>1- Open the game
4- Confirm values</td><td>2- Assign default values for players’ name and mark
3- Display the menu for choosing player options
5- Display main game interface with set values
6- Set current player to Player 1</td></tr>
</table>
</td></tr>
</table>

Use case description - 2

Name	Reset board
Initiating actor	Human Player
Actor’s goal	Clear all marks from the board

<table>
<tr><td>Summary</td><td>In this use case players can start playing a new game by clearing the board</td></tr>
<tr><td>Precondition</td><td>-</td></tr>
<tr><td>Post condition</td><td>After this use case, game will be initialized i.e. the 3*3 board will be cleared</td></tr>
<tr><td>Related use cases</td><td>-</td></tr>
<tr><td>Main success scenario</td><td><table><tr><td>Actor’s actions</td><td>System responses</td></tr><tr><td>1- Reset game board</td><td>2- Set turn to player 1
3- Clear all marks in the board
4- Display which player’s turn is to play
5- Display empty 3*3 board.</td></tr></table></td></tr>
<tr><td>Alternate scenario</td><td></td></tr>
</table>

Use case description - 3

Name	Exit Game
Initiating actor	Human player
Actor’s goal	To close the game.
Summary	Players can quit the game anytime they want.
Precondition	Application should be open.
Post condition	Game will be closed.
Related use cases	-
Main success scenario	

<table>
<tr><td></td><td><table>
<tr><td>Actor’s actions</td><td>System responses</td></tr>
<tr><td>1- Selects exit game
3- User confirms to quit.</td><td>2- Display message box for confirmation.
4- Application is closed.</td></tr>
</table></td></tr>
<tr><td>Alternate scenario</td><td>Step 3:
3-a: user denies to quit
4-a: Application continues</td></tr>
</table>

Deliverable 2

Following are the use case diagram and use case descriptions for deliverable 2

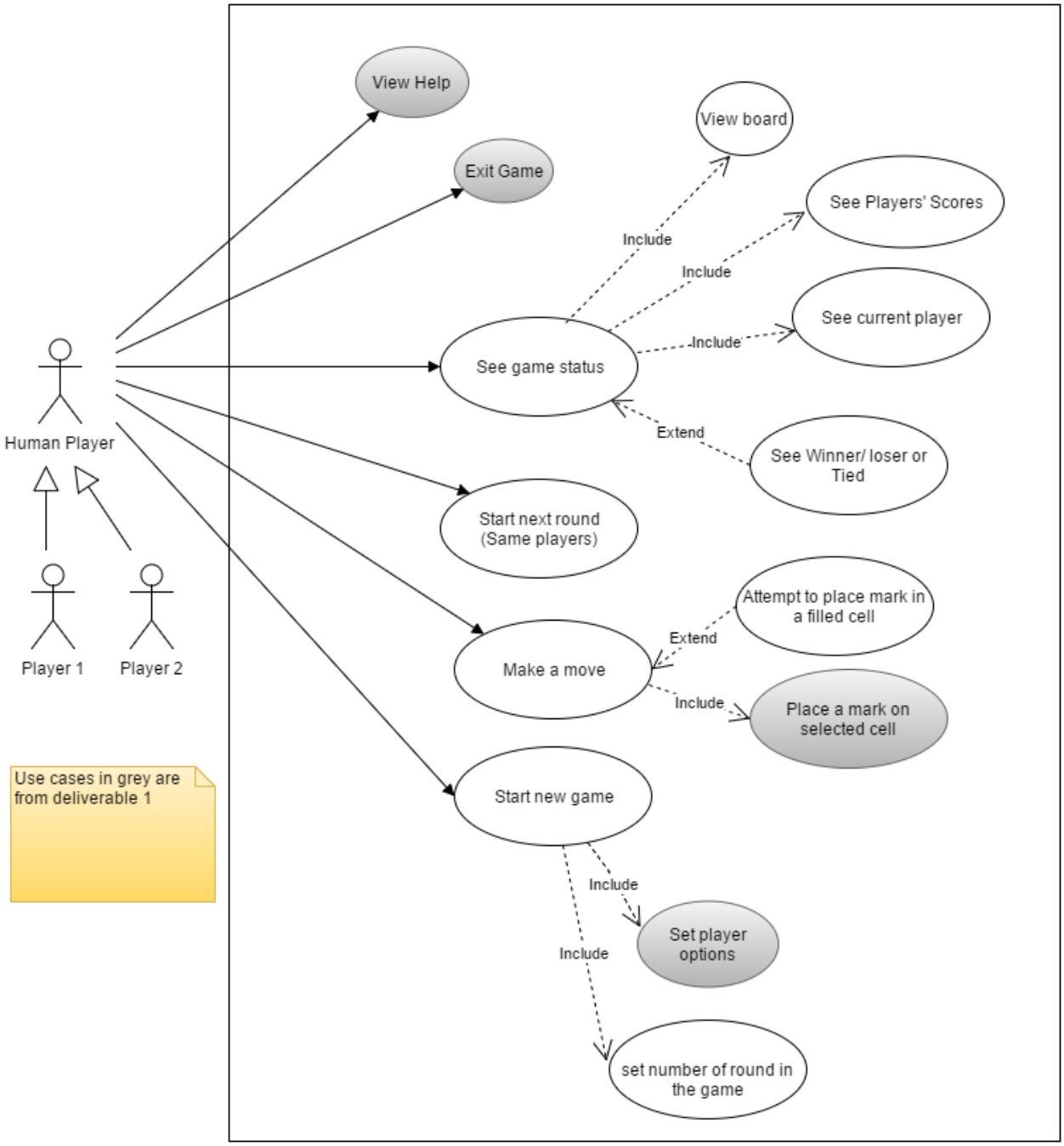

Figure 2: Deliverable 2 Use case Diagram

Use case description - 1

<table>
<tr><td>Name</td><td>Make a move[1]</td></tr>
<tr><td>Initiating actor</td><td>Human Player</td></tr>
<tr><td>Actor’s goal</td><td>Play the game by placing the dedicated mark on the board</td></tr>
<tr><td>Summary</td><td>Players put their mark on the board by clicking on the desired empty cell, when it's their turn. This use case is performed multiple times by the players to play the game.</td></tr>
<tr><td>Precondition</td><td>- Players name and mark must be set
- It should be the player’s turn</td></tr>
<tr><td>Post condition</td><td>- The mark is added to the board</td></tr>
<tr><td>Related use cases</td><td>“Attempt to place mark in a filled cell” is an extension of this use case (exception). This use case includes “Place a mark on selected cell”.</td></tr>
<tr><td>Main success scenario</td><td>
<table>
<tr><td>Actor’s actions</td><td>System responses</td></tr>
<tr><td>1- Click on a cell</td><td>2- Confirm the cell is empty
3- Fill the cell with player’s mark: Include “Place a mark on selected cell” use case
4- Confirm it’s not a win scenario
5- Confirm it’s not a tie scenario
6- Switch Player turn
7- Display whose turn it is</td></tr>
</table>
</td></tr>
<tr><td>Alternate scenario</td><td>Step 4:
4-a: if it’s a win, the round/game will stop
4-b: The game status and scoreboard will be updated.
4-c: If round number is less than total rounds, an option will be provided to the user to continue next round in game
Step 5:</td></tr>
</table>

[1] This is the completed version of “Place a mark on selected cell” from deliverable 1

	5-a: if it's a tie, the round/game will stop 5-b: The game status and scoreboard will be updated. 5-c: If round number is less than total rounds, an option will be provided to the user to continue next round in game

Use case description - 2

Name	Start new game
Initiating actor	Human Player
Actor's goal	To start a new game.
Summary	In this use case players can start playing a new game
Precondition	-
Post condition	After this use case game will be initialized i.e. user will be prompted to input name, mark, and number of rounds.
Related use cases	Includes "Set player options" and "set number of round in the game"
Main success scenario	

Actor's actions	System responses
1- Start new game. 3- Enter game	2- Display player option window - Include Use case "Set player options" - Include Use case "Set number of rounds in the game" 4- Initialize game window 5- Display game board 6- Display score board 7- Display player turn

Use case description - 3

Name	Attempt to place a mark in a filled cell

<table>
<tr><td>Initiating actor</td><td>Human Player</td></tr>
<tr><td>Actor’s goal</td><td>To put his/her mark in a filled cell</td></tr>
<tr><td>Summary</td><td>A case where in order to make a move, the player wants to mark a cell which has already been filled.</td></tr>
<tr><td>Precondition</td><td>- The cell has to be already filled
- It has to be the player’s turn.</td></tr>
<tr><td>Post condition</td><td>Message indicating invalid move is displayed.</td></tr>
<tr><td>Related use cases</td><td>Extension of “Make a move” use case. It is an exception to it.</td></tr>
<tr><td>Main success scenario</td><td><table><tr><td>Actor’s actions</td><td>System responses</td></tr><tr><td>1- Click a filled cell</td><td>2- Determine the cell is filled
3- Display “invalid move” message</td></tr></table></td></tr>
</table>

Use case description - 4

Name	Start Next Round
Initiating actor	Human Player
Actor’s goal	To start the next round of the game between same players
Summary	After the completion of a round the players choose to continue for another round. This is dependent on maximum of rounds in the set not being reached.
Precondition	- The previous round has to be concluded - Next round’s number is less than the total rounds in chosen set
Post condition	Next round is started with a clean board
Related use cases	-

Main success scenario	<table><tr><th>Actor's actions</th><th>System responses</th></tr><tr><td>1- Choose continue playing</td><td>2- Clear the board. 3. Increment round number by one</td></tr></table>

Use case description - 5

Name	Set number of rounds
Initiating actor	Human player
Actor's goal	Choose number of rounds to play
Summary	Player can select 3 or 5 rounds they want to play. Game will be finished after number of rounds selected.
Precondition	-
Post condition	Game has 3 or 5 rounds mode
Related use cases	"Start new game" includes this use case.
Main success scenario	<table><tr><th>Actor's actions</th><th>System responses</th></tr><tr><td>1- Player sets 3 rounds</td><td>2- set current round to one 3- Set total number of round to three</td></tr></table>
Alternate scenario	Step 1: 1-a Player sets 5 rounds 3-a Set total number of round to five

Deliverable 3

Following are the use case diagram and use case descriptions for deliverable 3

Figure 3: Deliverable 3 Use case Diagram

Use case description - 1

<table>
<tr><td>Name</td><td>Generate Possible Game States</td></tr>
<tr><td>Initiating actor</td><td>Computer Player</td></tr>
<tr><td>Actor’s goal</td><td>Generate all possible moves based on current game state</td></tr>
<tr><td>Summary</td><td>Computer must be able to generate all possible moves when it is the turn of the computer player</td></tr>
<tr><td>Precondition</td><td>- The current turn is for the computer player</td></tr>
<tr><td>Post condition</td><td>- Computer player has a list of all possible moves</td></tr>
<tr><td>Related use cases</td><td>-</td></tr>
<tr><td>Main success scenario</td><td>
<table>
<tr><td>Actor’s actions</td><td>System responses</td></tr>
<tr><td>1- Request the current state
3- sequentially generate a list of possible moves
4- Request to store possible moves</td><td>2- Respond with the current state
5- Store generated states</td></tr>
</table>
</td></tr>
<tr><td>Alternate scenario</td><td>-</td></tr>
</table>

Use case description - 2

Name	Calculate Next Best Move
Initiating actor	Computer Player
Actor’s goal	Make a move that will defeat the human player
Summary	Once all states have been generated a score must be assigned to each state based on a heuristic. The next move will be chosen based on the results.

<table>
<tr><td>Precondition</td><td>- Possible Game states generated</td></tr>
<tr><td>Post condition</td><td>- The move with the highest score is selected</td></tr>
<tr><td>Related use cases</td><td>-</td></tr>
<tr><td>Main success scenario</td><td><table><tr><td>Actor’s actions</td><td>System responses</td></tr><tr><td>1- Request possible states
3- Evaluate each state based on a heuristics
4- Sort from highest score to lowest
6- Choose the state with the highest score</td><td>2- Return stored states
5- Return sorted list</td></tr></table></td></tr>
<tr><td>Alternate scenario</td><td>Step 6:
There exists a draw in the highest score
6a- use a random method to pick the winner
Step 3:
3a- Unable to obtain a score for a state
3b - assign default score of zero</td></tr>
</table>

Use case description - 3

Name	Choose difficulty level.
Initiating actor	Human Player.
Actor’s goal	Select difficulty level which player want to play.
Summary	Player can select difficulty level i.e. beginner, intermediate or advanced to play with computer.
Precondition	Player should select 1 player mode.
Post condition	1 player game will be initialized based on difficulty level
Related use cases	-

<table>
<tr><td>Main success scenario</td><td><table><tr><td>Actor’s actions</td><td>System responses</td></tr><tr><td>1- Selects difficulty level.</td><td>2- Choose algorithm according to difficulty.</td></tr></table></td></tr>
<tr><td>Alternate scenario</td><td>-</td></tr>
</table>