Yarmouk University

Faculty of Information Technology and Computer Science

Department of Computer Science

Graduation project

"Siga Game"

Supervisor:

Dr. Faisal Alkhateeb

Table of content:

Chapter one: Project Initiation and Planning	
1.1 Project Description & Objective	**3**
1.2 Problem Statement	**4**
1.3 Project Scope Statement	**4**
1.4 General Rules	**5**
Chapter two: System Analysis	
2.1 System Functional Requirement	**6**
2.2 Dataflow Diagram	**8**
2.3 Use Case	**11**
Chapter Three: System Design	
3.1 Algorithmic Design	**13**
3.2 Data Flowchart	**16**
3.3 Class Diagram	**17**

Table of figures:

Chapter One:	
Figure 1.1 Project Scope Statement	**4**
Chapter two:	
Figure2.1 Context Dataflow Diagram	**8**
Figure 2.2 Dataflow Diagram Levels 0	**10**
Chapter three	
Figure 3.1 Data Flowchart	**16**
Figure 3.2 Class Siga	**17**
Figure 3.3 Class ArrayListType	**17**

Chapter One:

Project Initiation and Planning

1.1 Project Description and objectives

1.1.1 Project description

"Siga Game" is a matching competitive game between two players, the system developed for the game is a 2D array of a size 5x5, containing 25 squares to be filled with either X or O,X is symbol controlled by the first player while O is controlled by the second player which may be another human or the machine, the user is allowed to choose the number of the players, in case of choosing one player the user needs to determine the difficulty of the game (easy or difficult), choosing one player means that the player is going to challenge the machine, choosing easy level of difficulty means that the machine will take its moves randomly, whereas difficult level means that the machine will follow a specific heuristic algorithm when taking moves, choosing two players means that the player is going to play against another human.

The game starts with placing the stones for each player one after another, then the moving process, each player moves a stone one square at a time, the move could be up, down, left, right, the location must be available i.e. (empty and adjacent), 15 moves are allowed for each player in an attempt to set the 5 stones in a sequence and win.

The game ends when a player creates a winning combination of their stones or when the 15 moves end. The winning combination is defined as 5 horizontally, vertically or diagonally adjacent symbols.

1.1.2 Objectives

The project objective is to replace the paper-based game with a more developed computerized system built depending on essential AI concepts, especially the heuristic algorithm. Also, this game aims to challenge the human brain in an entertaining and fun way.

1.2 Problem statement

The paper-based version of the game lacks graphical features, whereas the computerized version is more fun and catchier for the users because of the graphics.

1.3 Project scope statement

shown in the figure 1.1 including general information about the project: Project Name (Siga Game), Sponsor (Dr Faisal Alkhateeb), project manager represents the names of the workers on the project, business benefits mention the benefits of the new system, system deliverables describe documentation for project and estimated project duration represent the time required to complete project.

Project scope statement
Project General Information*:* – **Project Name:** Siga Game – **Sponsor:** Dr Faisal Alkhateeb – **Project Manager**: Fatima Obeidat, Shorouq Ababneh, Hadeel Al-Omari
Business Benefits: – Reduce paper used in the manual game. – Attract more users for a computerized system. – Financial benefits for the developers of the game.
System Deliverables: – Computerized version of the game. – Documentation for specification. – Documentation for design. – Documentation for implementation.
Estimated Project Duration: – Project duration is 3 months starting on 15/02/2015.

Figure 1.1 project scope statements

1.4 General Rules

-Moving

1. Four directions are allowed for moving, up, down, left, right.
2. Diagonal moves are not allowed.
3. Moves should be taken to only available locations, which are empty and adjacent locations.
4. One move is allowed in each turn, no more.
5. “moves” is the number of the moves that are allowed for each player.

-Placing:

1. The players start the game by placing their stones, which are 5 stones for each, in any empty location.
2. Each player has a turn for placing and moving as well.
3. Each player has their own symbol to play with; it's not allowed to try and move the stones of the other player.

Chapter two:

System Analysis

2.1 System Functional Requirements:

Input:

Main Requirements:

1. The system should view the game when the user clicks on the application.
2. The system should start the game once the user has finished determining the number of players, by drawing the array.
3. The system should provide a restart button, allowing the player to start the game from the beginning anytime.
4. The system should allow the user to terminate the game by an exit button.

Processing:

This game system should do the following:

1. The system should display the game by (view the game) process when the user clicks on the game icon.
2. The system should allow the user to choose the number of the players, the available options are one or two players, choosing one player means that the player will play against the machine, two players option means that the player will play against another human.
3. The system must allow the user to determine the difficulty of the game; this option is only available for the (one player) option from the previous step; the difficulty could be easy or difficult.
4. The system should provide (start the game) process, which draws the array allowing the players to launch their game.
5. The system should allow the players to place their stones; which are 5 stones for each of them; in any allowed location.
6. The system should check for any possible winning combination after each stone placed and after each move taken as well.
7. The system should have a counter for the number of stones placed, to make sure each player has placed the 5 stones.

8. The system should allow the user to choose the location that he wants to move the stone from.
9. The system should check the location selected by the player in the previous step, if the location is not allowed; which means that the stone is not one of the player's; the system should get the player choose a location again, choosing an allowed location leads for the next step.
10. The system should allow the player to choose a location to move the stone to.
11. The system should check the location determined by the player to move the stone to, the location must be empty and adjacent to the location which the stone was moved from, if the location is not valid, the system should get the player to choose another location.
12. The system should apply the move determined by the player and increase the moving counter by one; the system should check for any winning combination after this step.
13. The system should check the moving counter, if the counter value is less than 30; the system moves forward to the next step, otherwise the system applies process 15.
14. The system should change the role to the other player if the counter hasn't reached the value of 30 yet. After changing the role, the system should let the player take the same steps to move their stone.
15. The system should show the game result in case of any winning combination existed or if the moving counter reached the value 30.
16. The system should end the game as a complementary process for showing the game's results.
17. The system should allow the user to start the game from the beginning with a restart button.

<u>Output:</u>

Siga Game

2.2 Dataflow Diagram

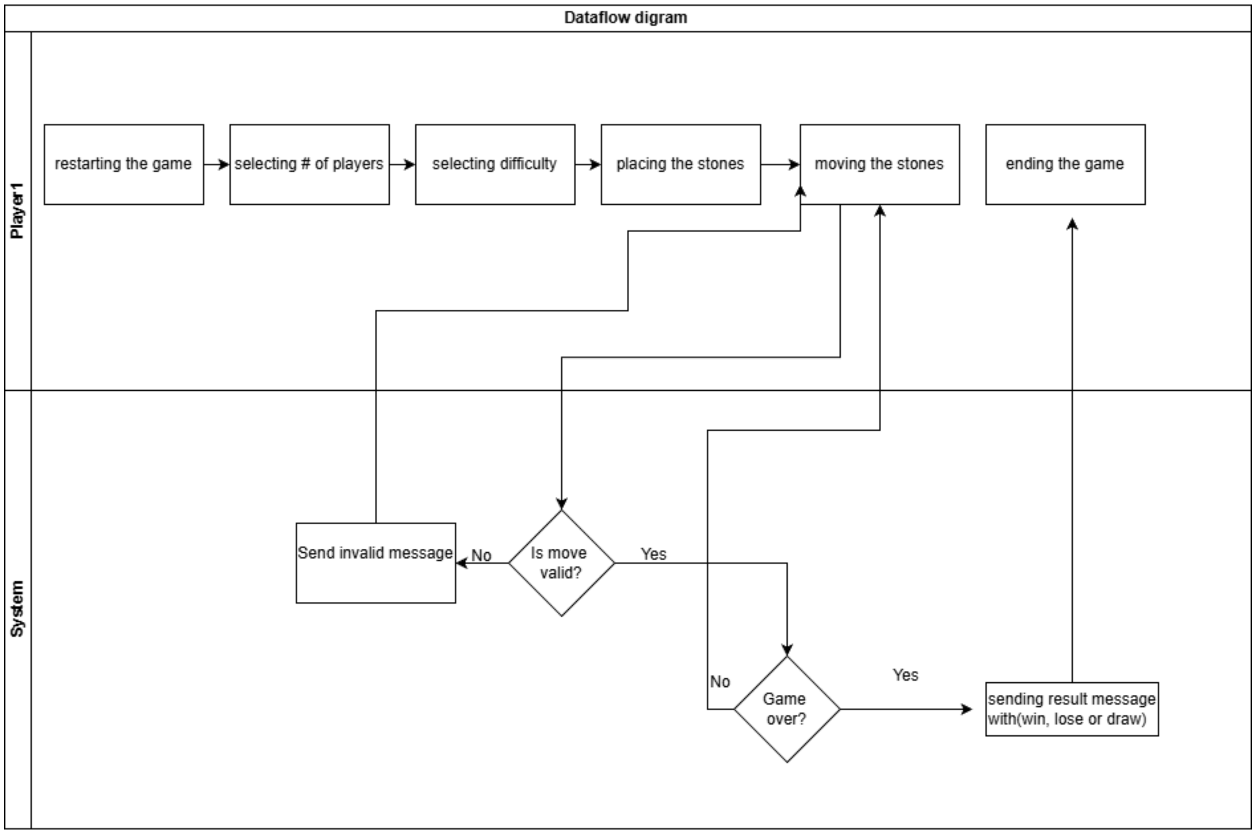


Figure 2.1: Context Dataflow Diagram

2.2.1 Context diagram as shown in figure 2.1 including two main entities represented as rectangular; player1 as source and system as sink also include six dataflow: number of players selecting, selecting difficulty (easy, difficult), restarting the game, terminating the game for completely exiting from the game, moving the stones, checking the location for validity.

2.2.2 Dataflow Diagram Level 0 as shown in figure 2.2 including seventeen processes represented as circles, (1.0 view the game), (2.0 select the number of players), 1 or 2 players, in case of choosing one player the game shows the process (3.0 choose difficulty) with two options (easy, difficult), then process (4.0 start the game) process, which choosing two players is led to immediately, process (5.0 place the stone and increase the placing counter by one) the stone placed is determined according to the player turn, process (6.0 check for winning combination) is applied after each stone placed ,if no combination was found, the system apply process (7.0

check the placing counter) which assure to have 5 stones placed for each player, if the counter's value is less than 10, process 5.0 is applied again, when the counter has the value 10, process (8.0 select a location to move from) followed by process (9.0 check the location for validation) the location should be filled with a stone and it should be a stone for the player whom turn is, if the location is not valid, process 8.0 is required again, then process (10.0 select a location to move to) and process (11.0 check the location for validation) which should be empty and adjacent to the location moved from ,if these two conditions are verified process (12.0 apply the move) is applied, otherwise another location is selected by process 10.0 .

After applying each move, process(6.0 check for winning combination) is necessary to check for any winning combination, if no combination was found, process (13.0 check the moving counter) is necessary to check if each player has 15 move, as long as the counter is less than 30 ,the system apply process (14.0 change the player) followed by process 8.0 again to give the other player a chance to take their move.

Process (15.0 show the game result) is applied if any winning combination is found, or if the moving counter reached the value 30, in this process the system should give a message showing the game final result, (16.0 end the game) the system then end the current game, allowing the players to play again at any time.

Process (17.0 restart the game) is available at any time through the game in case the player wanted to try and play from the beginning.

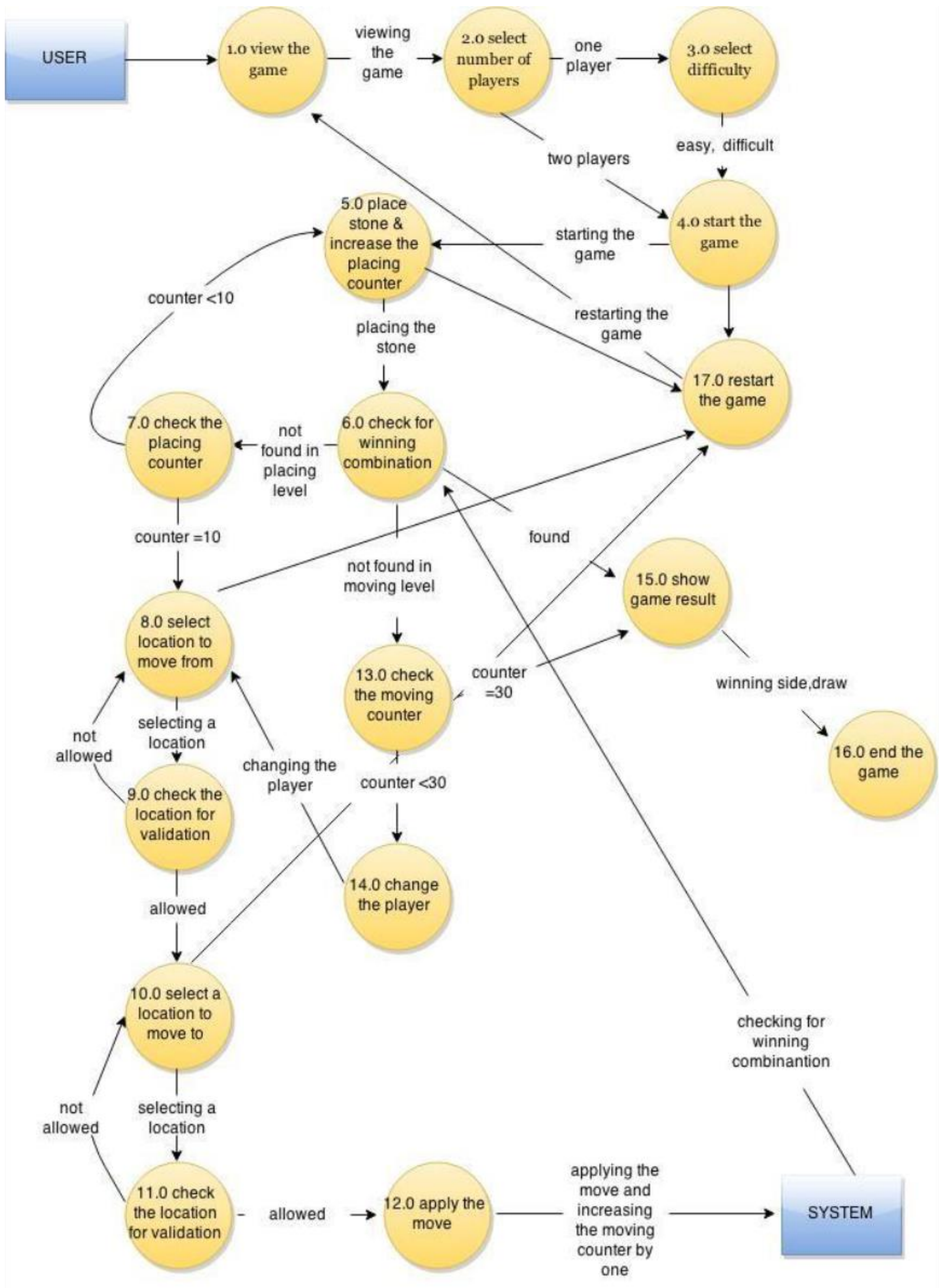

USER
1.0 view the game
viewing the game
2.0 select number of players
one player
3.0 select difficulty
easy, difficult
two players
4.0 start the game
starting the game
5.0 place stone & increase the placing counter
restarting the game
17.0 restart the game
counter <10
placing the stone
6.0 check for winning combination
not found in placing level
7.0 check the placing counter
counter =10
found
not found in moving level
15.0 show game result
8.0 select location to move from
13.0 check the moving counter
counter =30
winning side,draw
16.0 end the game
not allowed
selecting a location
changing the player
counter <30
9.0 check the location for validation
14.0 change the player
allowed
10.0 select a location to move to
checking for winning combinantion
not allowed
selecting a location
11.0 check the location for validation
allowed
12.0 apply the move
applying the move and increasing the moving counter by one
SYSTEM

Figure 2.2: Dataflow level 0

2.3 Use Cases

Case #1: winning O

The player views the game, selects the number of the players (2 players), player1 goes with X for their symbol and player2 with O, then the game starts, each player places their stones in any available location, the placing is in an alternate order, the figure 2.3 shows this step, then the moving process, with 15 moves for each player with alternate order, player 1 starts with moving the stone in location (3,5) down to location (4,5), player 2 moves their stone in (3,1) to (4,1), player1 moves their stone in (2,5) down to (3,5), player2 moves their stone in (3,2) left to (3,1), player1 moves their stone in (2,4) right to (2,5), player2 moves the stone in (3,3) left to (3,2), player1 moves their stone in (4,5) down to (5,5), player2 moves the stone in (4,1) down to (5,1), then player1moves the stone in (3,5) down to (4,5), player2 moves their stone in (3,1) down to (4,1), player1 moves the stone in (2,5) down to (3,5), player2 moves the stone in (3,2) left to (3,1), winning this game by forming one column of their stones in less than 15 moves, player1 with symbol X loses the game, winning combination is shown in figure 2.4

	1	2	3	4	5
1	O			X	X
2	O			X	X
3	O	O	O		X
4					
5					

figure 2.3: placing the stones.

	1	2	3	4	5
1	O			X	X
2	O			X	X
3	O	O	O		X
4					
5					

Figure 2.4: Winning form for O

Case #2: Draw

In this case none of the players wins, each of them tries to form a winning combination with the 15 moves they have, but both of them went out of moves before any of them could win.
The game starts with selecting the number of the players, in this case one player is selected, this means that the player is going to play against the machine, which leads to determining the difficulty of the game, the player selects the (difficult) option,
after that, the players places their stones in the locations of their choice, showed in figure 2.5.

Player1 starts with moving their stone in location (2,4) up to location (1,4), the machine moves its stone in location (4,1) to (3,1), player1 moves their stone in (3,3) up to (2,3), the machine moves the stone in (3,1) right to (3,2), the player moves the stone in (2,3) up to (1,3), the machine moves the stone in (3,2) right to (3,3), the player tries to move the stone in (2,1) to (1,2) in a diagonal move, but he gets an invalid message telling him that the move is not allowed, so he goes with moving the same stone (2,1) down to (3,1), the machine moves the stone in (3,4) down to (4,4), the player moves the stone in (3,1) right to (3,2), the machine moves the stone in (2,5) left to (2,4), the player moves the stone in (3,5) left to (3,4), the machine moves the stone in (2,4) left to (2,3), the player moves the stone in (3,2) to (4,2), the machine moves the stone in (4,4) down to (5,4), the player tries to fill a location needed by the machine to win in a diagonal order, so the player moves the stone in (3,4) down to (4,4), the machine moves the stone in (5,4) left to (5,3), the player moves the stone in (4,2) right to (4,3), the machine moves the stone in location (5,3) left to (5,2), the player moves the stone in (4,4) right to (4,5), the machine moves the stone in (5,2) left to (5,1), the player moves the stone in (4,3) right to (4,4), the machine moves the stone in (3,3) left to (3,2), the player moves the stone in (4,5) down to (5,5), the machine moves the stone in location (3,2) left to (3,1), the player moves the stone in (4,4) right to (4,5), the machine moves the stone in (3,1) down to (4,1), the player moves the stone in (1,4) down to (2,4), the machine moves the stone in (2,2) left to (2,1), the player moves the stone in (2,4) down to (3,4), the machine moves the stone in (2,1) down to (2,2), each of the players had 15 moves, no more moves are allowed for both of them and no one achieved a winning combination, so it's a draw, the final form is shown in figure 2.6

	1	2	3	4	5
1	O				X
2	X	O		X	O
3			X	O	X
4	O				
5					

figure 2.5: placing the stones.

	1	2	3	4	5
1	O				X
2	O		X		
3			O	X	
4	O				X
5	O				X

figure 2.6: draw form

Chapter Three:

System Design

3.1 Algorithmic Design.

Process 1.0: View the game.

1. The user clicks on the application.
2. The system displays the game window.

Process 2.0: Select the number of players.

1. The user determines the number of players; choices are 1 r 2 players.
2. if (number of players =1)
   select difficulty
3. Return number of players.

Process 3.0: Select difficulty.

1. Input: number of players.
2. The user determines the difficulty level to play against the machine (1 player), choices are easy or difficult.
3. if (difficulty level = easy)
   play randomly, the machine will pick a random location from the possible locations.
   else play according to a heuristic algorithm, the machine will pick the location using the algorithm specified.
4. return difficulty level.

Process 4.0: Start the game.

Input: 1. 2D array with a size of 5x5.
2. Draw the array.

Process 5.0: Place the 5 stones for each player.

```
1. Input: location (x, y).
2. if (location is empty) && (stone selected = the player's stone) && (player turn is true).
           then place the stone & increase the placing counter by 1
           & switch the turn of the players & return true.
   else return false & choose another location to place the stone.
```

Process 6.0: Check for winning combination.

```
1. Input: array .
2. if (stones in: (i,1) = (i,2) = (i,3) = (i,4) = (i,5)) ||
   (stones in: (1,n) = (2,n) = (3,n) = (4,n) = (5,n)) ||
   (stones in: (1,1) = (2,2) = (3,3) = (4,4) = (5,5)).
           return true, call process 15.0
   else return false,
   if (level==placing) call process 7.0.
   if (level==moving) call process 13.0.
```

Process7 .0: check the placing counter.

```
1. Input: placing counter.
2. if (placing counter < 10)
           Then iterate process 5.0 to place the next stone.
   else proceed to process 8.0 .
```

Process 8.0: select a location to move form.

1. The player selects a stone to move.
2. Process 9.0 is called.

Process 9.0: Check the location to move from.

```
1. Input: location (x, y) .
2. if (location is not empty)&&(stone selected = the player's stone)
   && (player turn = true).
           then return true, proceed to process 10.0 .
   else return false, call back process 8.0 .
```

Process 10.0: Select a location to move to.

1. The player selects a location to move to.

```
2. Process 11.0 is called.
```

Process 11.0: Check the location to move to.

```
1. input: location1 (x1, y1) & location2 (x2, y2)
2. if (location2 is empty) && ($|x2-x1| + |y2-y1| = 1$)
   \\an equation to check adjacency.
           then return true,
   else return false, back to process 10.0 .
```

Process 12.0: Apply move.

```
1. Input: location1 (x1, y1) & location2 (x2, y2) &
2. Move the stone from location1 to location2.
3. Increase the moving counter
4. call process 6.0
```

Process 13.0: Check the moving counter

```
1. Input: moving counter.
2. If (moving counter <30)
           call process 14.0
   else if(moving counter=30)
           call process
```

Process 14.0: Change the player

```
1. input (player turn)
2. if (player turn= user)
           switch to the machine or the other player .
   else switch back to the user turn.
3. call process 8.0.
```

Process15.0: Show the game result

```
1. if (moving counter= 30) || ( check for winning combination= true)
           then show game result ("winning" ,"draw").
```

Process 16.0: End the game.

```
1. Process 15.0 is executed
2. Terminate the game.
```

Process 17.0: Restart the game.

```
1. Click on (restart the game) button.
2. Start the game from process 1.0 (view the game).
```

3.2 Data Flowchart

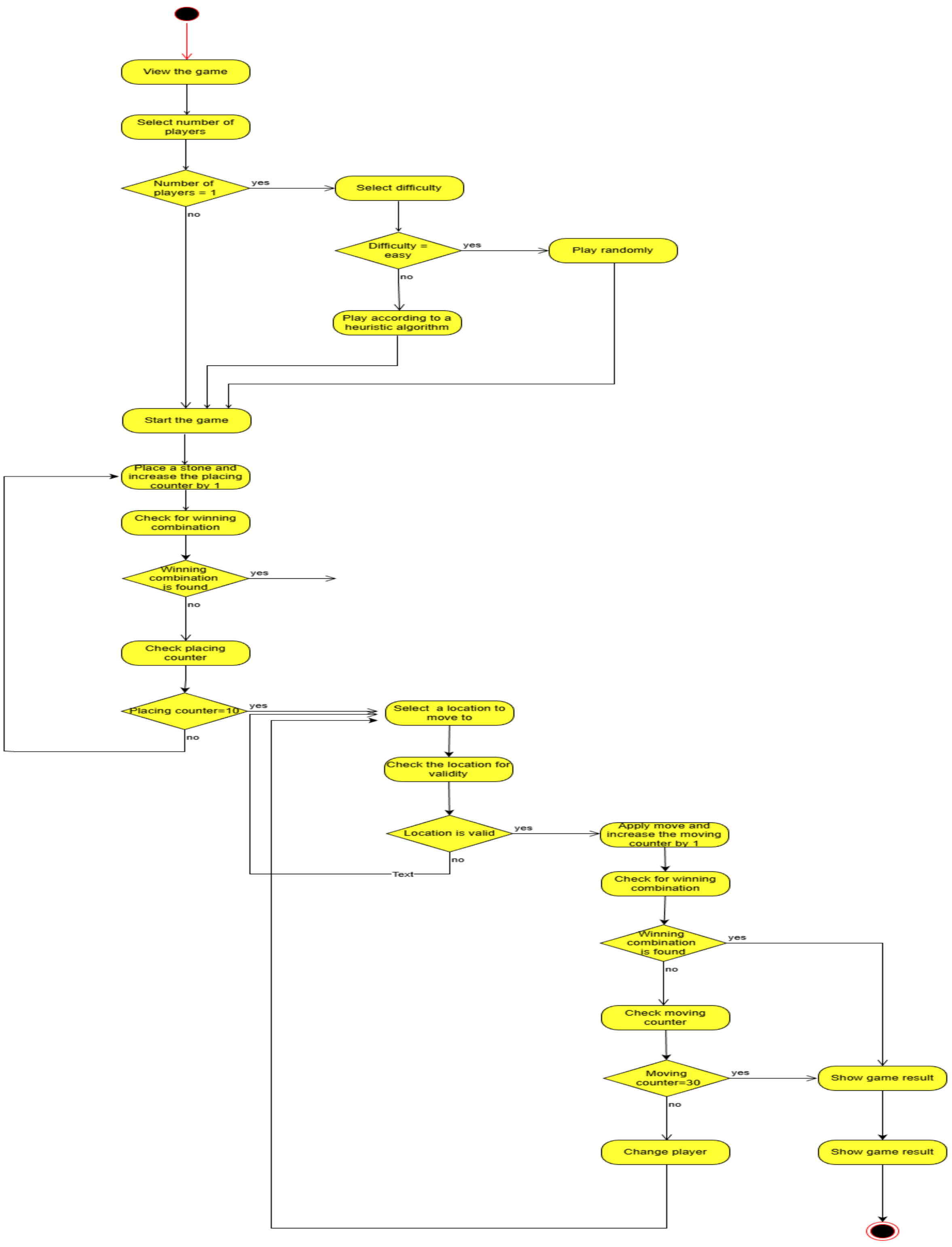
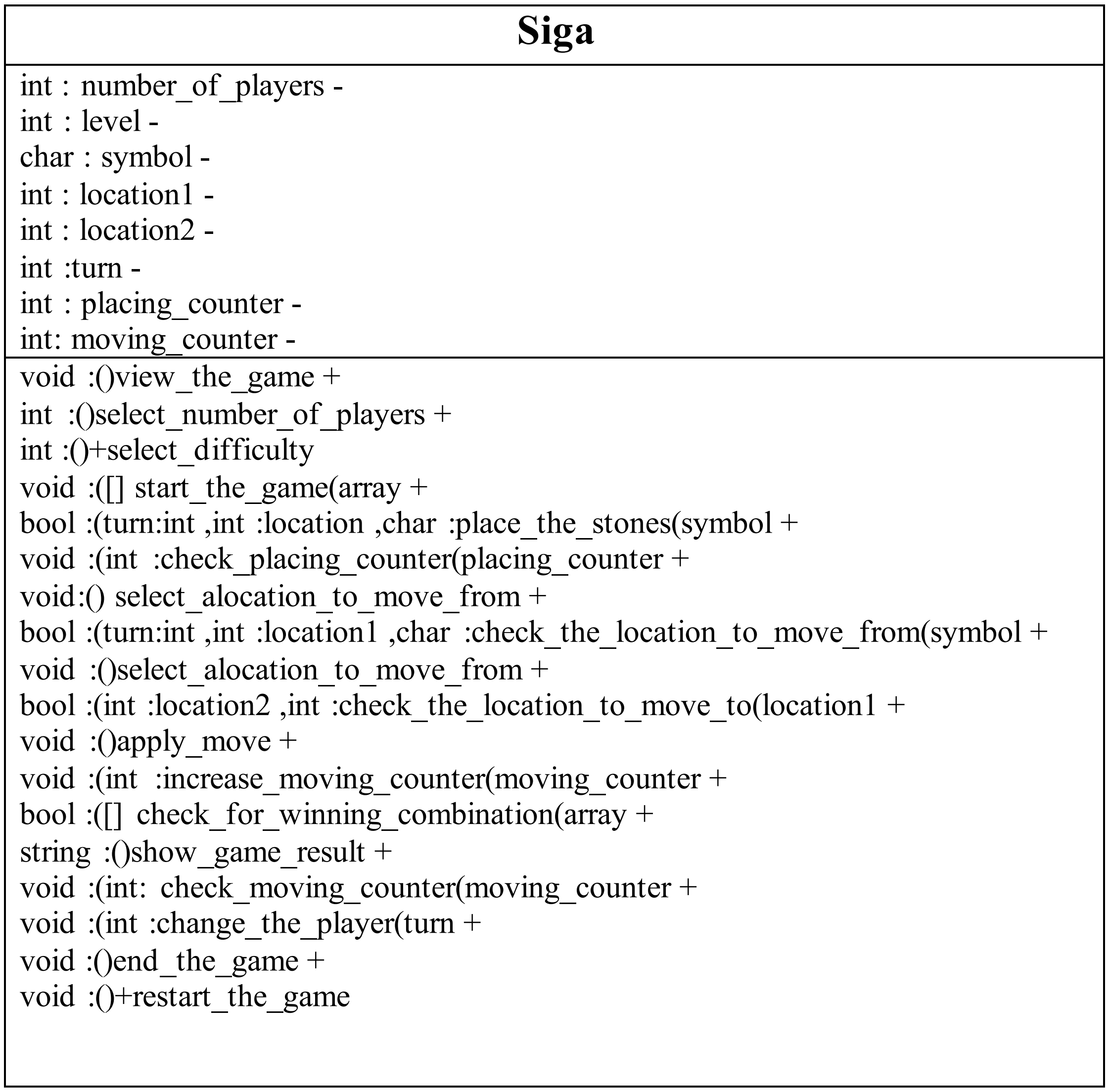

View the game
Select number of players
Number of players = 1
yes
no
Select difficulty
Difficulty = easy
yes
no
Play randomly
Play according to a heuristic algorithm
Start the game
Place a stone and increase the placing counter by 1
Check for winning combination
Winning combination is found
yes
no
Check placing counter
Placing counter=10
yes
no
Select a location to move to
Check the location for validity
Location is valid
yes
no
Text
Apply move and increase the moving counter by 1
Check for winning combination
Winning combination is found
yes
no
Check moving counter
Moving counter=30
yes
no
Change player
Show game result
Show game result

Figure 3.1 Data Flowchart

3.3 Class Diagram

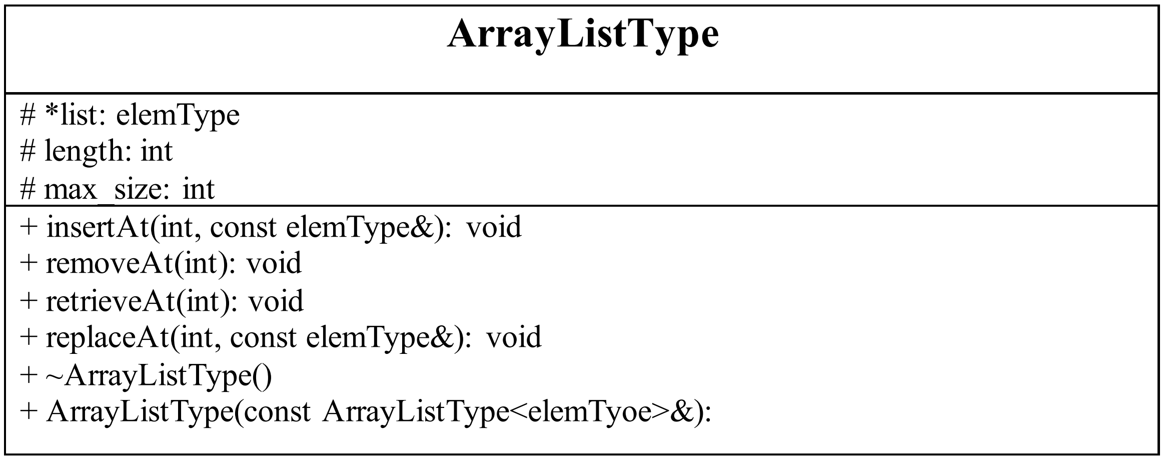


Siga

int : number_of_players -
int : level -
char : symbol -
int : location1 -
int : location2 -
int :turn -
int : placing_counter -
int: moving_counter -

void :()view_the_game +
int :()select_number_of_players +
int :()+select_difficulty
void :([] start_the_game(array +
bool :(turn:int ,int :location ,char :place_the_stones(symbol +
void :(int :check_placing_counter(placing_counter +
void:() select_alocation_to_move_from +
bool :(turn:int ,int :location1 ,char :check_the_location_to_move_from(symbol +
void :()select_alocation_to_move_from +
bool :(int :location2 ,int :check_the_location_to_move_to(location1 +
void :()apply_move +
void :(int :increase_moving_counter(moving_counter +
bool :([] check_for_winning_combination(array +
string :()show_game_result +
void :(int: check_moving_counter(moving_counter +
void :(int :change_the_player(turn +
void :()end_the_game +
void :()+restart_the_game

Figure 3.2 Class Siga



ArrayListType

# *list: elemType
# length: int
# max_size: int

+ insertAt(int, const elemType&): void
+ removeAt(int): void
+ retrieveAt(int): void
+ replaceAt(int, const elemType&): void
+ ~ArrayListType()
+ ArrayListType(const ArrayListType<elemTyoe>&):

Figure 3.2 Class ArrayListType