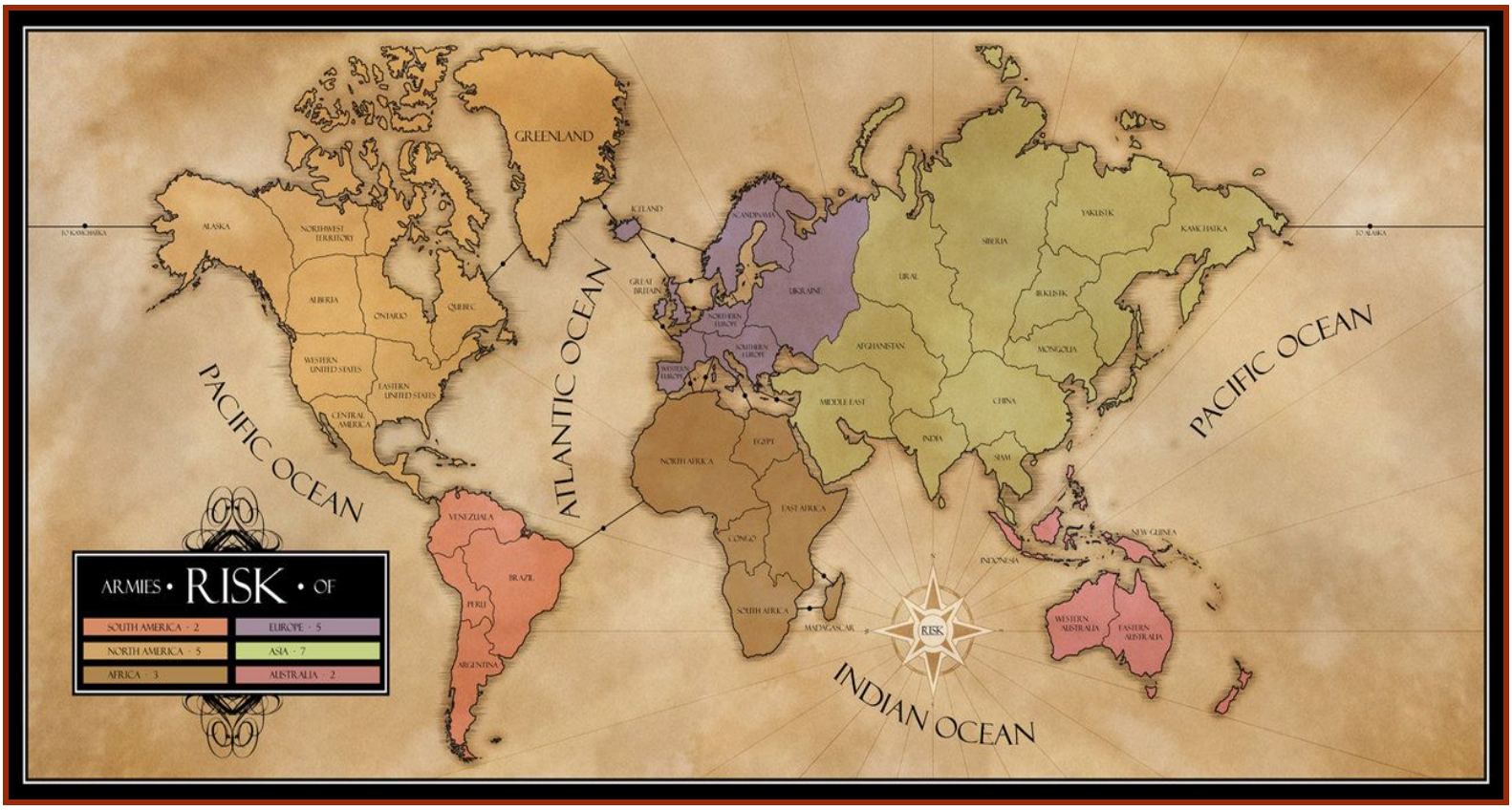


RISK

CC482 : Artificial Intelligence

Assignment 2

Simple agent and environment simulator for game of RISK

8 /12/2018

TEAM

Dahlia Chehata 27 **Omar Shawky 43** **Fares Mehanna 52**

TABLE OF CONTENTS

Overview

Java implementation of a simple environment simulator of RISK game that generates instances of a search problem, runs agent programs, and evaluates their performance according to a simple performance measure.

Data Structures and Algorithms

The game design follows MVC design pattern where :

1. Model

- **Agents**

  the player move consists of 3 main steps : place armies, attack if possible and end turn:

  Data structures: player_id : int
  enemmy_id : int
  game : Risk_game object

  - **non-IA Agents**
    - **Aggressive Agent**

```
make_move(){
       place_armies() {
           • get the player countries
           • get the strongest country with most number of
             armies
           • place reinforcement + bonus armies from previous
             turn in the chosen country
                }
      attack_if_possible(){
           • get the attackable countries
           • if the list is empty : no possible attack exists
           • if the user have continent/s, then try to attack
             any country to lose a continent, pick biggest
             first.
           • if can't do such attack, then attack the country
             with the most soldiers.
          }
     end_turn(){
           • if no bonus armies and no reinforcement armies
             : switch players and return true
           •  else return false
          }
}
```

- **Passive Agent**

```
make_move(){
       place_armies() {
           •  get the player countries
           •  get the weakest country with least number of
              armies
           •  place reinforcement + bonus armies from previous
              turn in the chosen country
                 }
         end_turn(){
           •  if no bonus armies and no reinforcement armies
              : switch players and return true
           •   else return false
         }
}
```

- **Nearly Pacifist Agent**

```
make_move(){
       place_armies() {
           •  get the player countries
           •  get the strongest country with most number of
              armies
           •  place reinforcement + bonus armies from previous
              turn in the chosen country
            }
     attack_if_possible(){
           •  get the attackable countries
           •  if the list is empty : no possible attack exists
           •  pick the attack with the least amount of enemy
              soldiers
           •  execute the best attack possible
         }
    end_turn(){
           •  if no bonus armies and no reinforcement armies
              : switch players and return true
           •   else return false
         }
}
```

- **Human Agent**

handled in the GUI

- **IA Agents**
  - **Greedy Agent**
    - heuristic chosen : number of enemy 's countries

Algorithm:

```
make_move(){
        greedy_step() {
            • get the player countries
            • for each country of the greedy player
                    ○ clone a new game
                    ○ set current player soldiers
                    ○ get the attackable countries
                    ○ for every possible attackable country
                            ■ compute the game score which is
                              the difference between player and
                              enemy soldiers
                            ■ for k=1 to score
                                • clone new game
                                • attack the enemy with old
                                  number of soldiers k and
                                  new number score-k
                                • evaluate those moves
                                  according to the chosen
                                  heuristic , if it was
                                  better than the best yet,
                                  then select it.
            • move new army to the selected country
            • execute the best attack if possible
         }
         end_turn(){
            • if no bonus armies and no reinforcement armies
              : switch players and return true
            •  else return false
           }
}
```

  - **A* Agent**
    - ***class A* game score:***

      It is a helper class for the A* agent

      Data structures:

```
public final AStarGameScore parent_;
private final RiskGame game_;
public String action_;
```

      - the chosen heuristic is

        $(number\ of\ enemy's\ countries)^2$

      - used to compare between A* game's different scores

- ***class A* agent:***
  - additional DS:
    - `private List<AStarGameScore> wining_moves_;`
    - `Queue<AStarGameScore> game_states_pq = new PriorityQueue<>();`
    - `AStarGameScore wining_state = null;`

```
make_move(){
    • if first step:
        A_Star_initialization(){
          while (game_states_pq not empty(){
                • best_game = game_states_pq.poll();
                • curr_game = best_game.get_game_obj();
                • check if it's a winning state and break;
                • check if it's a lose state and continue;
                • check if the current state is visited:
                  continue
                • check who is the current player
                    ○ if it is my agent, then check what
                      move it is:
                        ■ if state=0,then move the army
                        ■ if state=1,then do the attack
                    ○ if it is the enemy then play as a
                      passive agent
              }
              if it is not a winning state: all moves are lost
              else add it to the winnig_moves list
          }catch(clone not supported exception){
              random moves=true
          }
    • if random moves:
          ○ move tha army to the country I (the current
            agent)have
          ○ do any attack if possible
    • else
          ○ get the moves from the list of optimal moves
          ○ get the set soldiers move
          ○ check if there is an attack step
    end_turn(){
          • if no bonus armies and no reinforcement armies  :
            switch players and return true
          •  else return false
        }
}
```

■ Real Time A* Agent

- additional DS:
  - `private List<AStarGameScore> wining_moves_;`
  - `Queue<AStarGameScore> game_states_pq = new PriorityQueue<>();`
  - `AStarGameScore wining_state = null;`

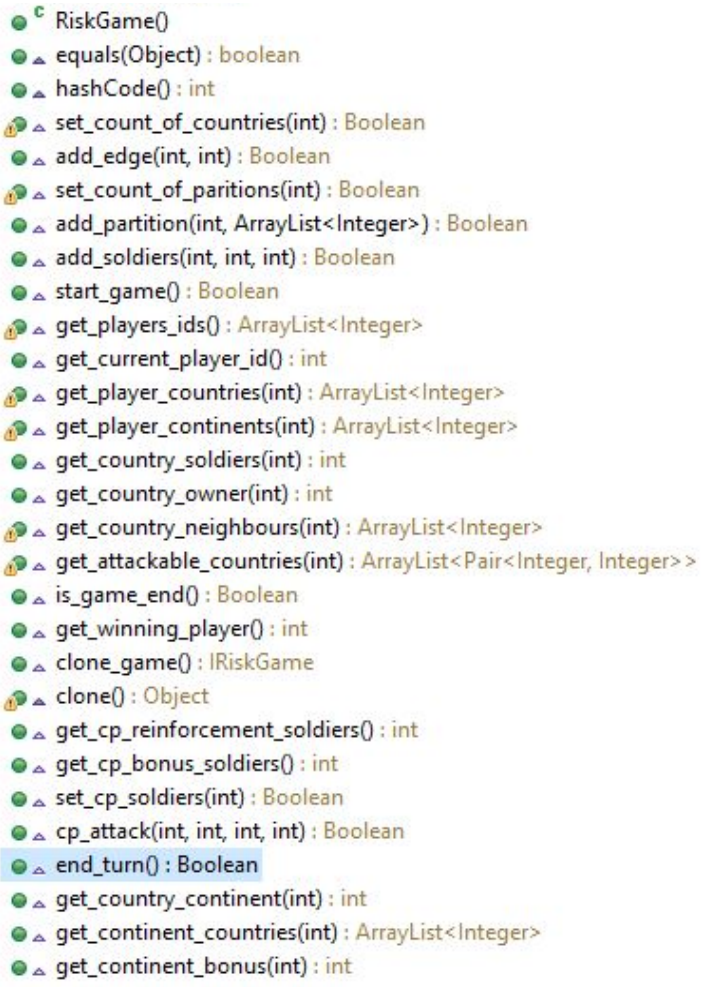
```
make_move(){
```

```
    • if first step:
       A_Star_initialization(int millis_limit){
         while (game_states_pq not empty(){
                • best_game = game_states_pq.poll();
                • curr_game = best_game.get_game_obj();
                • check if it's a winning state and break;
                • check if it's a lose state and continue;
                • check if the current state is visited:
                  continue
                • if (current time - starting time>
                  millis_limit) break
                • check who is the current player
                    ○ if it is my agent, then check what
                      move it is:
                        ■ if state=0,then move the army
                        ■ if state=1,then do the attack
                    ○ if it is the enemy then play as a
                      passive agent
             }
             if it is not a winning state: all moves are lost
             else add it to the winnig_moves list
          }catch(clone not supported exception){
              random moves=true
          }
    • if random moves:
          ○ move tha army to the country I (the current
            agent)have
          ○ do any attack if possible
    • else
          ○ get the moves from the list of optimal moves
          ○ get the set soldiers move
          ○ check if there is an attack step
    end_turn(){
          • if no bonus armies and no reinforcement armies  :
            switch players and return true
          •  else return false
         }
}
```

- **Game simulation**
  - the class is responsible for the 2 players selection and set the game object, simulation for single step for 1 and 2 agents
  - function simulate represents the main play

```
public int Simulate() {
       while(true) {
           //make player1 move
           player1.make_move();
           if(game.is_game_end())
               return game.get_winning_player();
           //make player2 move
           player2.make_move();
           if(game.is_game_end())
               return game.get_winning_player();
       } }
```

- **RISK Game**
    - this class is responsible for the main game and all information concerning players, armies, countries, continents, reinforcement and bonus armies.
    - Its main functions are:



2. View

3 main classes handle the GUI

- GraphMenuController
- ResultMenuController
- StartMenuController

3. Controller

- the class is responsible for reading graph and players information from file

Performance evaluation

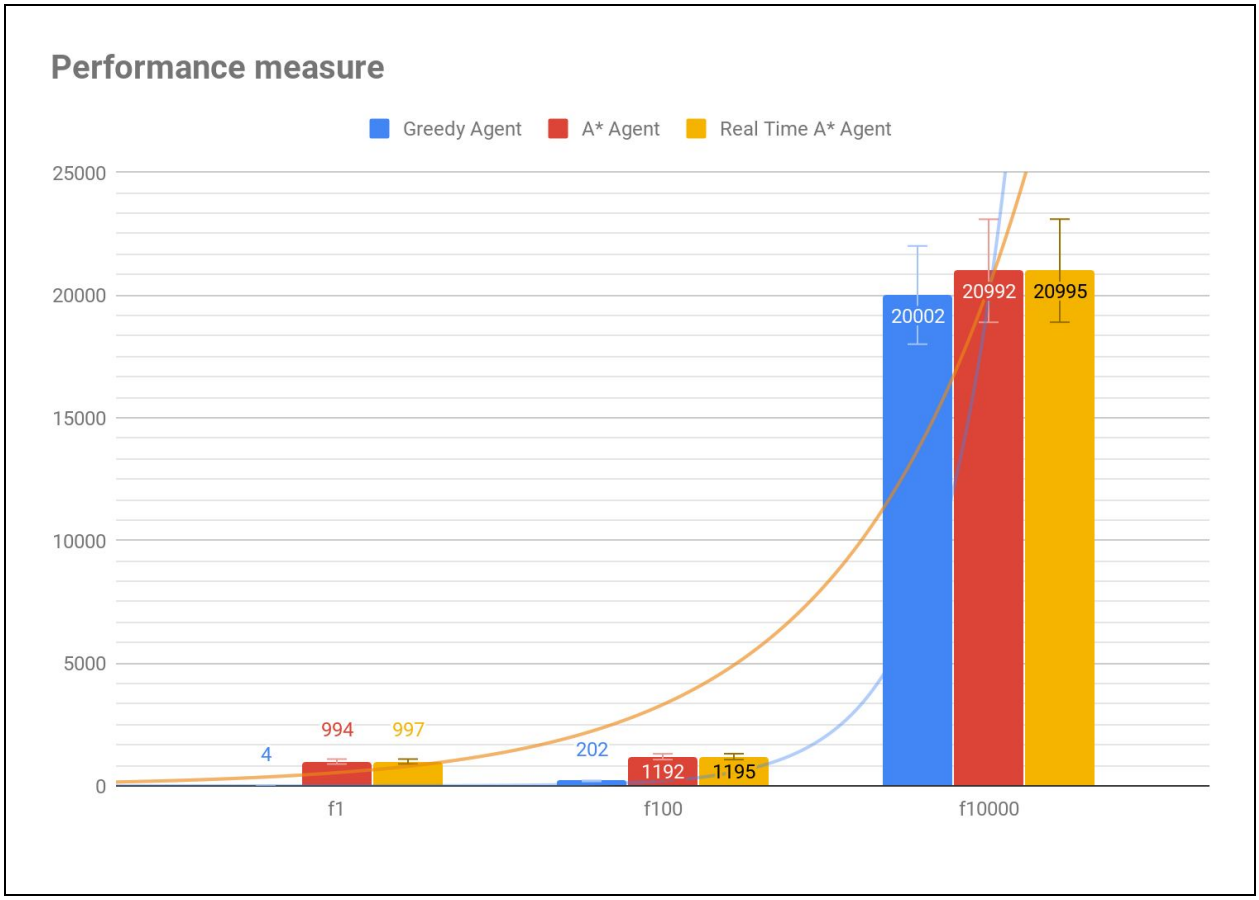
$$P = f * L + T$$

where P is the agent's performance

L is the number of turns it takes the agent to win the game

T is the number of search expansion steps performed by the search algorithm

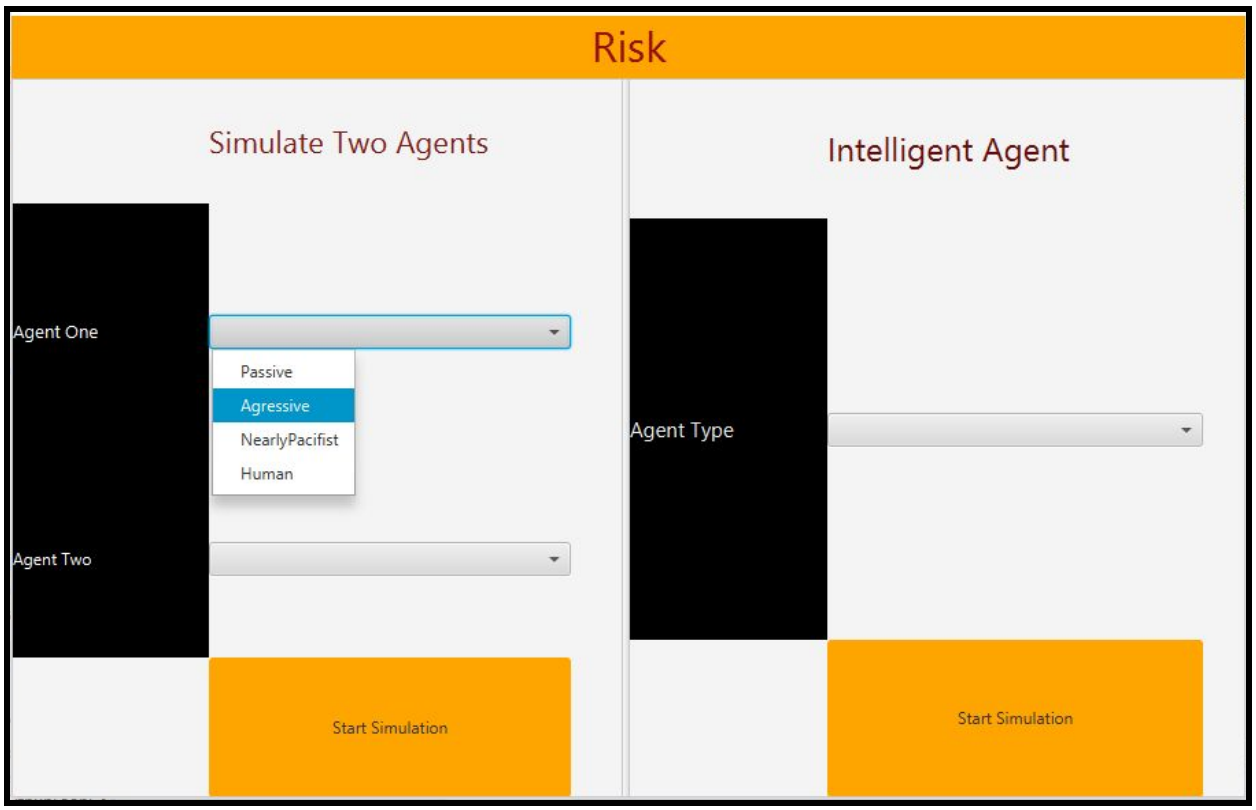
f is the weight taking values = 1, 100, 10000

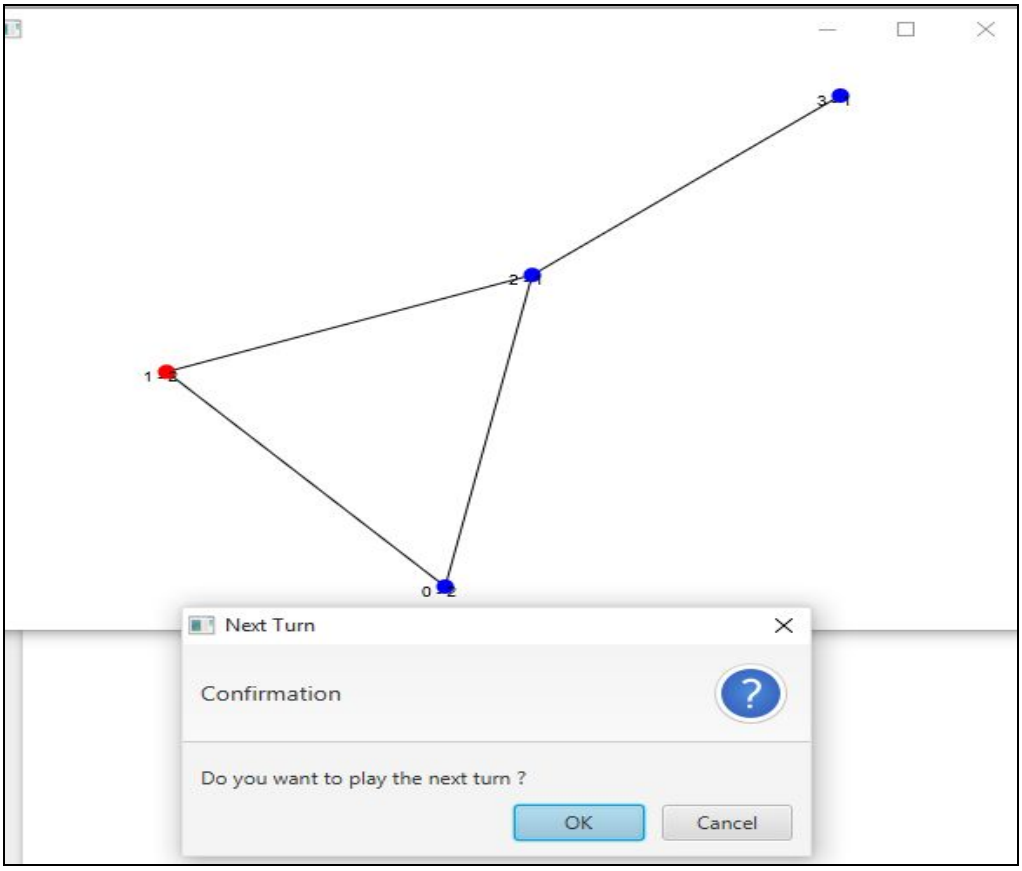


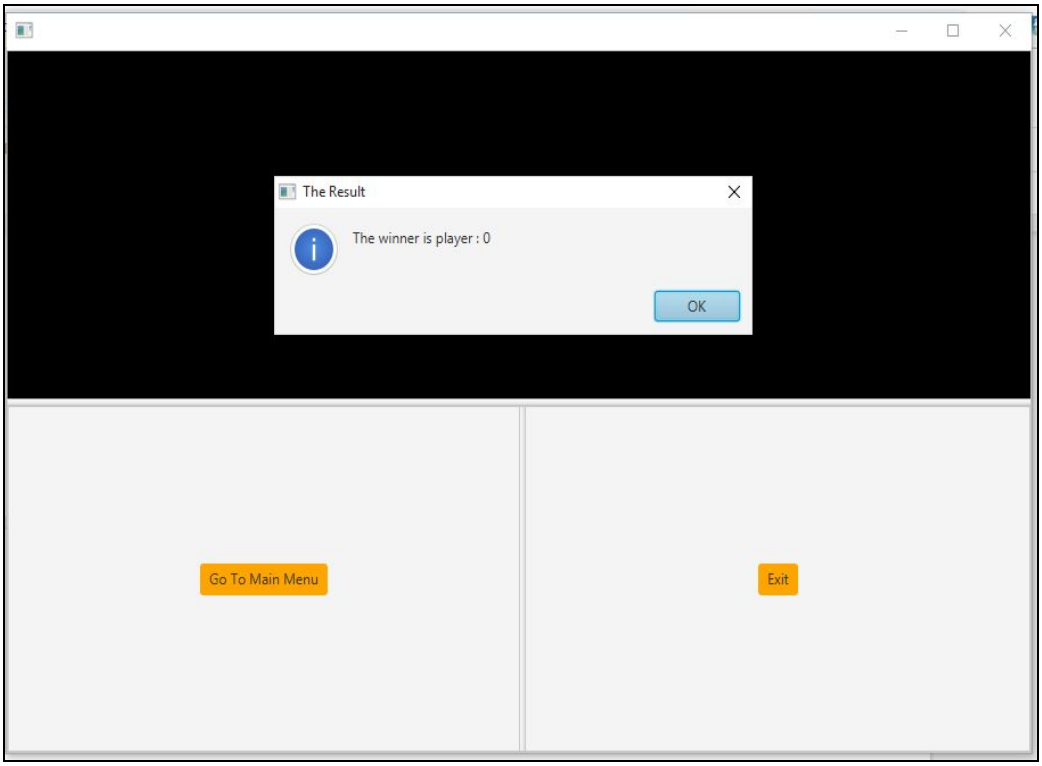
Assumptions

Every player is given a reinforcement army in each turn

Sample runs




3
2
1
0
Next Turn
Confirmation
Do you want to play the next turn ?
OK
Cancel


The Result
The winner is player : 0
OK
Go To Main Menu
Exit