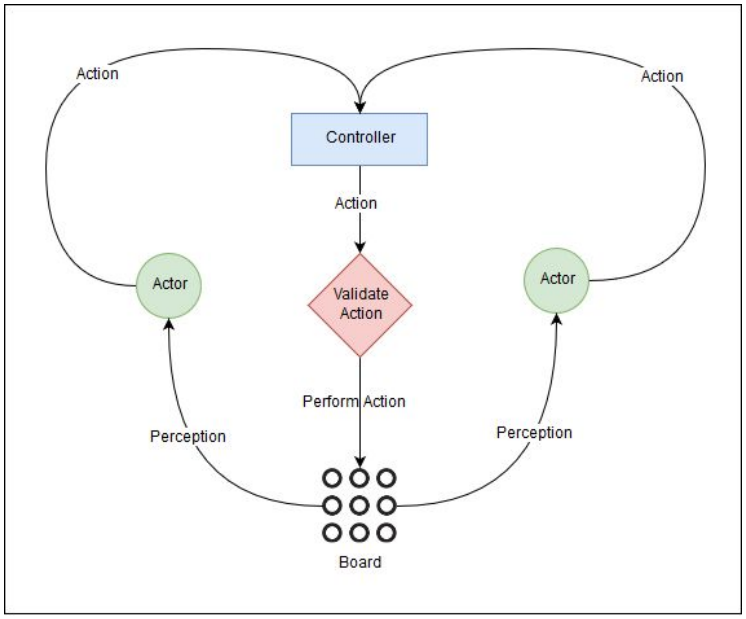
Introduction

This document is a detailed description of my implementation of Reversi game. The implementation is done using pure Python3. It has been run and tested on Ubuntu 16.4. This document includes diagrams that tries to explain the flow of the program, the heuristics, and to compare the efficiency of the heuristics in terms of space, time and the probability of winning.

Flow of control

There are two options to start with, the game can be played either locally or online.

Local match



Local Match

There are 3 main components of in a local match, The Controller, Actor, and the board.

Controller

The jobs of the controller are:

- Maintain the turns of the actors
- Taking the action from each actor, making sure that it's a valid action, and performing the actions if they're valid. Otherwise it asks the actor for another action
- Keeping track of the special cases when an actor doesn't have any valid action to be performed.
- Keeping track of the special cases when neither actor have any valid action to be performed.
- Keeping track of the results of the game.

Actor

The Actor (Agent) is an object that implements “GetAction” function. The controller uses that function whenever an actor has the turn to get the action of that actor. I’ve implemented different types of actors. More on the types of the agents in ‘Agents types’ section.

board

The board can be thought of as a simple array that represents the dices locations in an 8x8 array of integers, where 0 indicates an empty place, 1 indicated a black dice and 2 indicates a white dice.

However, the Board class does more than simply that. It has functions to the count the dices, to print the board in a user-friendly manner, and to print the valid actions, too.

Moreover, the Board class is responsible for posting the board on an online service, this will become handy in the Online Match which will be introduced next.

Online match

In the online match, the used is offered 2 options, either to start a new game, or to join an already existing game.

Starting a new game

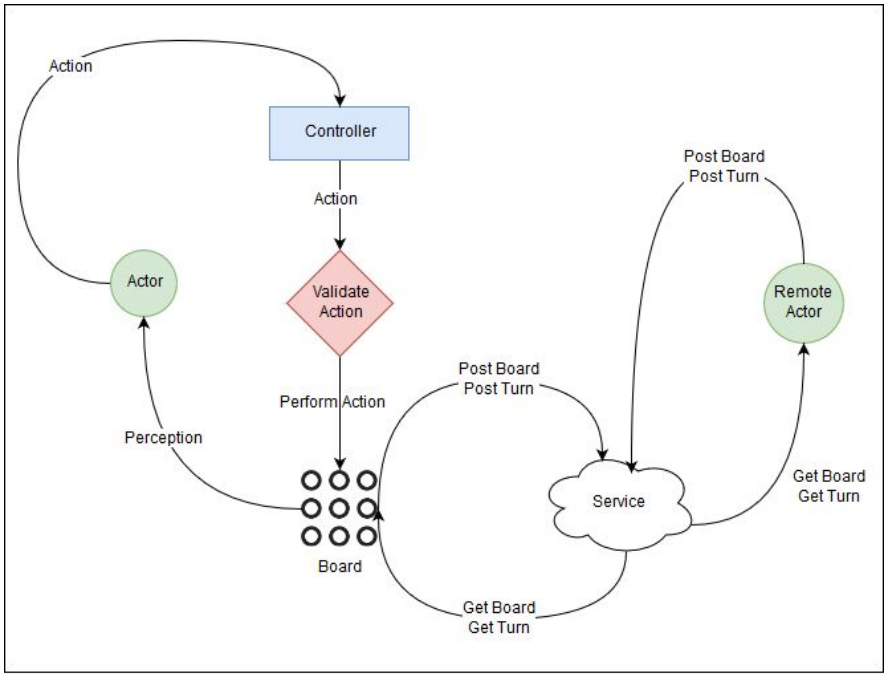
This requires connecting to a web service. I chose MyJSON.com for its simplicity and sufficiency for our purposes. When creating a new game, the local agent posts the board to the web service, which returns an ID number that could be used to access the same data, or modify it.

The agent posts 3 values to the service:

1- The Board array
2- The turn (white/black)
3- The status of the game (True = Active / False = Inactive)

The game is initially inactive, it becomes active when a second agent joins the game and *PUTs* True in the “Status” element of the JSON data.

The flow of the game is described in the following diagram:



Online Match

The Black agent starts a new game, posts the initial board, the Turn as “White”, and the status as “Inactive”, and gets the game ID. The White agent (remote) can use the same ID to *join the game*, changing the game status to Active and getting the board, performing its action, and re-posting the new board, and the turn as “Black” to the same game ID.

Notes:

- I’m assuming that the remote agent is performing valid actions only, for simplicity.
- The agents could be of any time in either end. (More on the agent types in the next section)

Agents types

There are 3 types of agents that I implemented. The common functionality between them is “getAction” which returns a 2-tuple that has the coordinates of the dice that should be placed.

Human Agent

Takes the coordinates of the dice that should be placed from the user. The user is supposed to interact with the console.

Random agent

Generates a random action given the set of all possible valid actions.

Minimax agent

Uses Minimax Algorithm with Alpha-Beta Pruning. It uses one of 3 Heuristics that will be analyzed in the next section.

heuristics

H1 – Dice count

This is the greedy approach for the game, to maximize one's dice count. It works well against a naïve or random player. With a probability of 75% win. More on the nodes count in a later comparison table

H2 – Minimize opponents' next actions

This approach works to minimize the available actions of the opponent.

H3 – mix

This is a mixed approach between the above 2 heuristics. It uses the following formula to evaluate a state:

$$\theta_1 \times H1\,(state) + \theta_2 \times H2\,(state)$$

Where θ_1 , θ_2 are factors. I've tried to test many values of these factors, and the values 1.2, 0.7 respectively seem to have the best performance against a random player, or a player with H1 heuristic or H2 heuristic.

Heuristic	Average Time to take an action (milliseconds)				Nodes Count			
Depth:	1	2	3	4	1	2	3	4
H1	2.3	19.8	33.2	75.8	7	15	59	101
H2	3.5	21.2	58.1	57.6	8	28	114	190
H3	3.6	20.0	85.1	102	7	26	237	157

Run Results (including tournament)

Note : The tournaments done between 2 minimax agents that use non-random heuristics results in the same result no matter what the number of runs is. For the random agent, however, the behaviour changes and the percentage of wins is reported based on 100 runs. The following table shows the winners in matches with different heuristics.

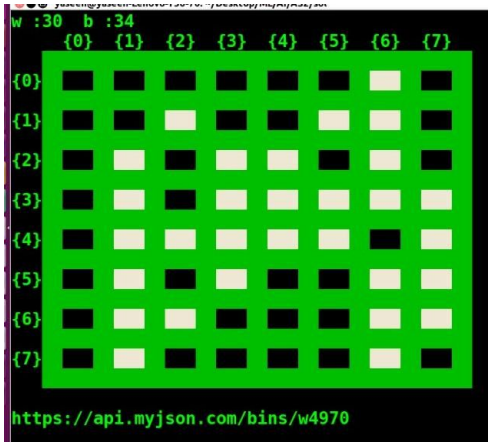
VS: WHITE / BLACK	H1 (Depth:3)	H2 (Depth:3)	H3 (Depth:3)	Random (100 runs)	Darwish's agent (Depth:3)
H1 (Depth:3)	H1 (white)	H2 (white)	H3 (white)	Random (white: 61%)	-
H2 (Depth:3)	H2 (black)	H2 (white)	H3 (white)	H2 (black: 73%)	-
H3 (Depth:3)	H3 (black)	H2 (white)	H3 (black)	H3 (black: 84%)	Darwish
Random	H1 (white)	H2 (white)	H3 (white)	Random (white: 52%)	-

A brief description of my experience programming for and participating in the tournament.

It was an intriguing experience to have hands on experience with the minimax algorithm and the alpha beta pruning. It took rather long time to have a full grasp of the implementation and to tune it to fit Reversi game, but it was such a thrill to see things work after the long nights of work. One more great part of this assignment was the tournament part. Dealing with simple online API to exchange JSON data, and to polish it to serve the purpose of exchanging the board, turn and status of the game, and to see two remote agents work and compete with each other was exceptionally satisfying. The tournament part has required more than the mere programming skills; I had to employ some communication skills to explain how my online interface work, and to agree with the opponent on what representations should we both use to make things work appropriately; writing the interface code to be readable by anyone was a bit tricky. Finishing this assignment has resulted in a deeper understanding of the Game theory, Minimax algorithm, and the abstract idea of an "Agent" which takes a percept and "acts" and an even further passion about AI in general. Overall, I discovered that wouldn't mind spending the rest of my life working on such topics, both theoretically and practically.

I can't thank you enough to have us experiencing such realm.

Some screenshots from the tournament :



Reasoning of the results : Darwish's heuristic takes the location of the action into account; i.e. he defines a weight for each location in the array, which is a very convenient way that is proven to be efficient.

However, we noticed that his Agent takes much more time to come up with the next action.