**Intelligent**

**Systems**

**Assignment 3: Adversary Search and CSPs**

# Members:

|  |  |
| --- | --- |
| **Registration Id** | A01746886 |
| **Registration Id** | A01746887 |

**Answers**

**PROBLEM 1: Games and Adversarial Search**

# Game: Blob

The main questions that must be answered during the development of this activity are:

1.- What are the main features of the game that can make a player of any type win or lose?

* Firstly, based on the game found in the internet link, the positions of random blobs are critical since the initial state is not equitable in both players for example in an initial state could be the situation that most green blobs are next to the red ones, as red player moves first the green blobs are going to be eaten and is a clear advantage for the first player.
* Another feature to consider depend on the player’s ability to predict some moves in advance because there are some strategic moves although they may look bad but are better than expected.
* Avoid as possible do moves where blobs are eliminated due to the border.
* The player with the most number of blobs should do moves for eating as many blobs as possible, while the player with the least number of blobs should avoid doing offensive moves (attack opponent blobs).
* Always look for the number of blobs is greater or equal to that of the opponent, in case there are movements in which the same number of bubbles remain, follow the play that leaves the least number of bubbles and equal both red and green.
* An important parameter to feed the strategy game is the number of green and red blobs, if after a movement the number of blobs red and green are equal, the strategy to follow consist in

2.- How easy is it to implement those features in computer systems?

All of them can be implemented by programming. At first glance are just needed if else sentences.

3.-Are they worth considering according to the computational cost of their implementation?

Of course, in fact the computational cost is low.

4.-Is it possible to include strategies within an evaluation function? How?

Strategies can be implemented with a weighted linear function

5.-By what means is it possible to adjust the weights assigned to the features in the evaluation function?

With a function by parts.

**Evaluation function**

The evaluation function was developed to maximize the values of the red player, however in any given case you want to maximize the plays of the green side also designed the corresponding function.

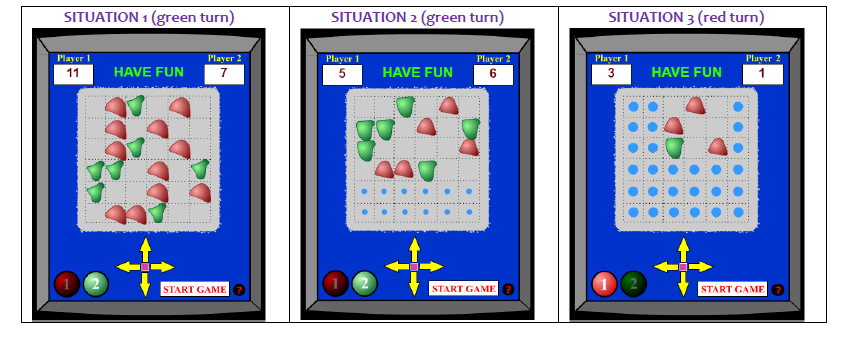
For red color

For green color

the function has the following strategies:

* When the game is started, it is given priority to eat the largest number of blobs of the opponent, this to have some advantage throughout the game, this is achieved by making the difference of the own blobs and the opponent's. In case there are plays where the difference is the same, it is easy to tie off them by a second factor added which gives priority to moves played leading to a position where there are fewer blobs in the board.
* Later in a half game where there are less than 6 blobs is similar to the previous strategy, only now it is not wanted that movement leaves less number of blobs.
* As an intermediate strategy in case it is lost, is to look for a tie by getting the moves to have the same number of blobs.

**Descripition of the following situations with the evaluation function.**



**Situation 1, Green turn.**

The green movement must be optimized on base with the following function:

where g is the green blobs left and r is the red ones after the move is applied.

For movement to Left, the function is:

For movement to Right, the function is:

For movement to Up, the function is:

For movement to Down, the function is:

**The best movement for MIN is Up**

**Situation 2, Green turn.**

The green movement must be optimized on base with the following function:

where g is the green blobs left and r is the red ones after the move is applied.

For movement to Left, the function is:

For movement to Right, the function is:

For movement to Up, the function is:

For movement to Down, the function is:

**The best movement for MIN is Down**

**Situation 3, Red turn.**

The green movement must be optimized on base with the following function:

where g is the green blobs left and r is the red ones after the move is applied. As r < 6 the second function is used.

For movement to Left, the function is:

For movement to Right, the function is:

For movement to Up, the function is:

For movement to Down, the function is:

**The best movement is Down**

**PROBLEM 2: Constraint Satisfaction Problem (CSP)**

# Puzzle: Pic-a-Pix

**Part 1: Problem Documentation as CSP**

1. Describe how these types of problems can be formulated as a CSP, that is, describe the variables, their domains, and the constraints they must met.

First we need to define the board, as an array of variables and list them in order to have them identified. Then you need to define the domain for each variable ie what kind of values (colors) they may have. Among the restrictions that must be formulated are if two are found with the same color together, they must be separated by a space. Define which colors can not have according to the rules imposed.

1. Illustrate this formulation for the 2 challenges illustrated below.

**Challenge 1:**

**Variables:**

**V1, V2, V3 ……,V24, V25**

**Domains:**

**Red, Yellow, White, Green**

**Constraint:**

* **The default value is white**
* **If you have two equal colors in the same row or column (constraint) we need to separate by at least one white or other color variable.**
* **If you have a constraint in row or column with a number > 1, we need to put the number that constraint says into together variables, in the row or column specified.**

**RULE Match:**

**For ELEMENTS could take a COLORS, if in the correspondent row constraint has a least the same color.**

1. **ELEMENTS: v1, v6, v11, v16, v21, v26 COLORS: Red**
2. **ELEMENTS: v2, v7, v12, v17, v22 COLORS: Yellow, Red, Green**
3. **ELEMENTS: v3, v8, v13, v18, v23 COLORS: Red**
4. **ELEMENTS: v4, v9, v14, v19, v24 COLORS: Red, Green**
5. **ELEMENTS: v5, v10, v15, v20, v25 COLORS: Red**

**Challenge 2:**

**Variables:**

**V1, V2, V3 ……,V24, V25**

**Domains:**

**Orange, Yellow, White, Black**

**Constraint:**

* **The default value is white**
* **If you have two equal colors in the same row or column (constraint) we need to separate by at least one white or other color variable.**
* **If you have a constraint in row or column with a number > 1, we need to put the number that constraint says into together variables, in the row or column specified.**

**RULE Match:**

**For ELEMENTS could take a COLORS, if in the correspondent row constraint has a least the same color.**

1. **ELEMENTS: v1, v6, v11, v16, v21, v26 COLORS: Yellow**
2. **ELEMENTS: v2, v7, v12, v17, v22 COLORS: Black, Orange**
3. **ELEMENTS: v3, v8, v13, v18, v23 COLORS: Yellow**
4. **ELEMENTS: v4, v9, v14, v19, v24 COLORS: Orange, Black**
5. **ELEMENTS: v5, v10, v15, v20, v25 COLORS: Orange, Yellow**

**Part 2: Scheduling the solution**

To program Pic-a-Pix, you must create a subclass of the CSP class whose definition is in the **csp.py** file of the online code provided as a resource of the AIMA textbook in the Python language.

**Evaluation Criteria (50% of activity):**

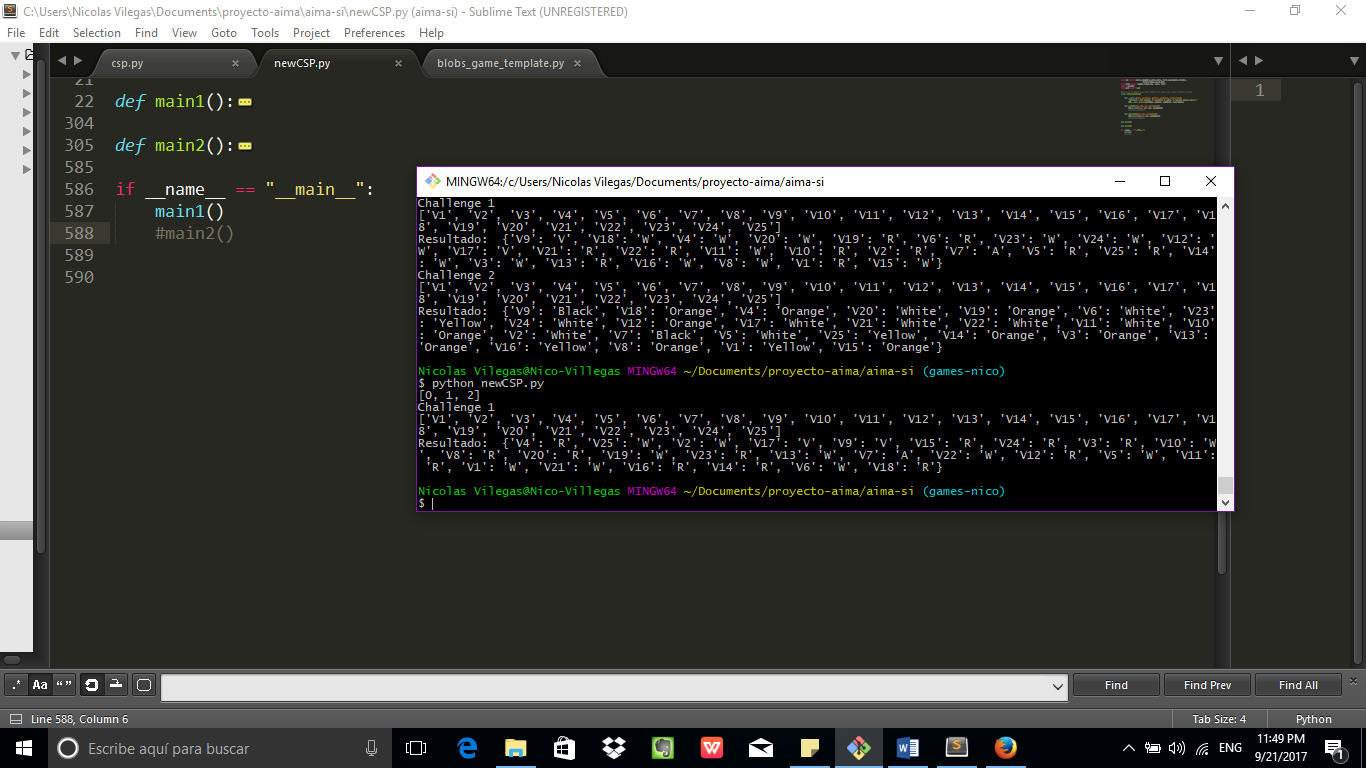
Word or Pdf document that includes (40 points):

* The general formulation as a CSP (60%)
* Specific formulations of challenge problems (40%)

Python source program and its execution (60 points):

* If the program does not run but a "readable" code is given for all required functions (50%).
* If the program runs, but does not solve the problems and gives a code "readable" and that makes sense according to the formulation of the challenges (75%).
* If the program runs, the code matches the formulation of the problems and gets the correct answers for the challenges (100%).

Run for challenge 1:



Run for challenge 2:

