**Advanced Artificial Intelligence Assessment Item 2 Resit**

**Rakesh Kumar Elumalai**

[**25954263@students.lincoln.ac.uk**](mailto:25954263@students.lincoln.ac.uk)

**Abstract:**

Gaming involves interaction with a user interface to generate visual feedback. Arcade games, as a genre, are truly hybrids that combine a variety of gaming styles. Short levels with easy and intuitive controls are common in arcade games, which swiftly rise in difficulty. To make this business model successful, the game's difficulty must be high enough to cause players to reach a game-over state while yet being entertaining or addicting enough to keep them playing [1].

In this assessment a python code is given that implements the Mean arena game where the tallon moves and provides information on location of pits, bonus stations and so. Here the tallon’s movement is non deterministic and they move in direction they wish to. The task is to train the tallon using reinforcement learning so that the tallon can survive as long as it can in the game environment and to provide a control code to navigate in fully and partial observable condition.

**Introduction:**

Arcade video games were introduced in the 1970s, with Pong being the first commercially successful game. Electronic or computerized circuitry takes input from the player and displays it on an electronic display device, such as a monitor or television set [2]. The gamer's performance in playing a game is usually defined unambiguously by the game's score or outcome, which is specified by an explicit set of rules. From single-player puzzles to two-player board games to massively multiplayer video games, games are extremely versatile and vary tremendously in complexity.

We have a python code for the game "Mean Arena". The main file is "game.py," and we must include our method in the make move function of the "Tallon" class in the tallon.py file. We only have one Arena, resulting in a 10x10 playboard matrix. We must use Reinforcement Learning to train the tallon so that it can survive in the Field for as long as possible. In this task we are going to make tallon to move in a way that it avoids meanies and pits to collect maximum number of rewards with the help of Markov decision process.

**Related Works:**

**PAC-MAN**

Pac-Man was first released in May 1980 and has since become one of the most iconic games in the gaming industry, with a massive selling units in its first year. The game was originally known as 'Puck Man' in Japan before being renamed for the foreign market. Pac-Man VR was first launched in 1996. Although the game didn't have a significant impact on the industry at the time, it does show how the iconic franchise has strived to modernize gaming throughout its growth. Pac-Man was still available to play despite the evolution of video game devices [3].



**Figure: PAC-MAN (Arcade Game)**

While many of the genre's titles, such as Pac-Man, have made the transition from famous arcade machines to current platforms, their stylistic effect on broader audiences has risen. Aside from console expansions, numerous online casinos have included vintage arcade features into their latest releases. Along with games like Arcade Bomb, casino party's famed melon madness blends free spins with conventional arcade graphics to offer prospective players a throwback progressive jackpot title. Furthermore, because the platform is compatible with both Android and iOS devices, the arcade genre has never been more accessible.

**Solution A**

In this solution the tallon is made to survive more in the arena to make the tallon move in arena by avoiding meanies and the pits in fully observable condition and the code is developed.

**Markov Decision Process(MDP):**

Here we use different Markov decision making process which implicit the process in easy way [4]. In this case, the action A influences the probability of the process advancing to a new state Sa. MDP's solution or policy is expressed as. However, we'd need the best policy possible.



After that, the policy's utility should be calculated. The predicted utility value is used to compare policies.

To solve this problem, we must consider the states and actions. There are four possible actions: Up, Down, Right, and Left, and 100 possible states for the 10x10 Matrix.

0.8 percent chance of moving in the action's direction, and 0.1 percent chance of moving in each of the action's perpendicular directions for the motion model in python code. Policy iteration, value iteration, and Q-Learning are all approaches that can be used to solve this problem.

**Policy:**

A policy is a way of dealing with an MDP (s). It gives an action for each state S. The purpose of an MDP is to find the optimum policy that maximises expected utility. Here's an illustration of a policy**.**

**Fully observability:**

In fully observability the system is fully visible and the obstacles can be avoided easily and it makes tallon’s to collect bonus points easily.



**Fig: Code for fully observable condition where partial visibility is false.**

**Value iteration:**

Value iteration is a technique for determining the best MDP and policy values[8]. It calculates the utility of each state, which is defined as the sum of discounted rewards from that point forward.



**Fig: Value iteration value**

**Policy Iteration:**

Policy iteration (PI) is a recursive policy evaluation and improvement approach for solving an optimal decision-making/control problem, often known as a reinforcement learning (RL) problem [5]. Every time a new policy is generated in the improvement section, and every time a new policy is created in the policy assessment section, an initial policy is created that is used in the improvement. If no further progress is made, the iteration will end.

**Q-Learning:**

Q-Learning is a method of active reinforcement learning that does not require the usage of a model [9]. The tallon moves in four different directions which is decided based on the movement of the monsters and also depends on the number of pits involved in the environment.

In Q-learning, the learning environment is modelled as a state machine, and value iteration is used to find the optimum policy. It stores a value of the predicted total (current and future) reward, represented by Q, for each pair of (state, action) in a table. For each action completed in a certain condition, a reward will be given, and the

Q- value will be updated according to the following rule:

Q(st, at) ← (1−αs,a)Q(st, at)+αs,a(r+γ max(Q(st+1, at+1))

**Bellman equation:**

If the state space and action space are both continuous, the optimum criterion can be found by solving the Bellman equation.



Here we are applying ADP: applying value iteration, it is Adaptive dynamic programming. A list of states S‚ Each state has a utility estimate associated with its U(s). ‚ Each state has an action associated with it, π(s).  
Each state-action pair has a probability distribution: P(S’|s, π(s)). over the states’ S’ that it gets to from s by doing π(s).

**Solution B**

When partial visibility is true, tallon can only detect up to the provided limits in the entire arena; for example, if we give six restrictions, tallon can only smell bonus up to six states of metrics; otherwise, it won't be able to receive anything from the arena and it can only view to the certain level.

There are two types of approaches, one is deterministic policy and non- deterministic approach or policy.

**Deterministic policy:**

A deterministic policy relates states and actions using a function. The best deterministic strategy is that which maximizes the expected discounted sum of rewards for the agent if they follow it. There is no uncertainty.

**Non-deterministic policy:**

Non-deterministic policies can be defined as functions transforming each state S into a non-empty collection of actions represented by S(a)[7]. When the MDP is in state S, the agent can choose to perform any action a ∈ Π(s). For this Mean arena game, I gave a code, and my tallon was able to locate the bonus in the partial visibility option as well.

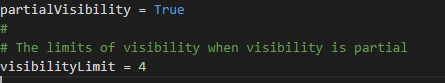
A hidden Markov model (HMM) is a Markov chain augmentation that includes observations. These observations can be partial in the sense that several states can map to the same observation. They can also be noisy, meaning that the same state can stochastically map to various other states at different times.

**Partial Observability:**

A partially observable Markov decision process (POMDP) is a Markov decision process that is more general (MDP). A POMDP is a model of an agent decision process in which the system dynamics are thought to be defined by an MDP, but the agent is unable to perceive the underlying state directly. Instead, it must keep track of the underlying MDP and a sensor model (the probability distribution of distinct observations given the underlying state). POMDP's policy is a mapping from the observations (or belief states) to the actions, unlike MDP's policy function, which maps the underlying states to the actions. The POMDP framework can be used to simulate a wide range of real-world sequential decision processes.

Utility values and policies are found in way that the tallon could move towards bonus points by one by one step ahead.





**Solution C**

We can assess the code and the tallon in this section by making various modifications, such as increasing and decreasing the Arena size, and then I've done an analysis that offers all the policies and values for any size. We're basically training our agent.

In addition, I attempted to use all of the methods available in the "Mdptoolbox," such as value iteration, policy iteration, and Q-learning.

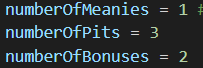
I also modified my bonus price, verified it, and made changes to the reward array's values. It produces appropriate outcomes in terms of policies and values in each manner.When we change the array or arena size, the major effect is on our processing time; when the size is small, it responds quickly, but when the size is larger, it takes longer to deliver iteration values.

Another observation is that when we perform small size arenas, we have no trouble determining the bonus position, but when we do large arrays, we occasionally have trouble getting the exact route of the bonus and also used several algorithms and just assessed the values and policies; value iteration takes longer, but policy iteration little quicker.

I tested with 10x10, 7x7, 13x13 arrays with 3 pits and 1 beginning meanies for a total of 2 bonuses; in this instance, my method is fast and accurate as I used 1D as the grid matrix. And the output displayed as quick as possible.

**Arena size 10x10:**





**Fig: Number of pits and meanies.**

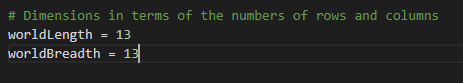
As shown in the above figure the size of arena and the no of meanies can be changed by increasing or decreasing in the configuration section.

**Arena size 7x7:**



In this case the size of arena is reduced to 7x7 matrix where it is difficult for the tallon to move and avoid the meanies also there’s no enough place for the tallon to move in the arena.

**Arena size 13x13:**



In this case the size of matrix is considered as 13x13 matrix where the tallon moves freely in the arena and as it has more space for the taloon, meanies and pits comparatively. Tallon won't be affected by increasing the number of pits or the speed of gameplay as the size of arena is huge.

In increase in size makes it difficult for the tallon to locate the bonus and also the time required to collect the bonus increases as it can be seen in 13x13 matrix whereas the case is not same in the 7x7 and 10x10 matrix as the same of arena is small. Since the tallon can only see certain grids in the partially observable true condition, the difficulty level rises as the number of pits increases; however, in the false condition, the condition is the opposite.

The change in size of mean arena, iteration values and visibility outputs are given in appendix of the report.

**Conclusion:**

In this task we have used different types of Markov Decision process for decision making. We have built a program to move tallon’s towards reward point avoiding obstacles and pit. The learning outcomes are about how effective the MDP method is to find the solution and decision making. After learning and using AI methods it’s an example that we can gain a precise knowledge that the next viable action in Reinforcement Learning is dependent on the prior action and its values, and we can achieve the best and optimal response with it.  The tallon outputs are taken for different arenas like

10x10,7x7, 13x13 and hence the outputs are shown.

**References:**

[1] DiMarzio, J.F., 2012. What Is an Arcade Game?. In *Android Arcade Game App* (pp. 7-11). Apress, Berkeley, CA.

[2] Rohlfshagen, P., Liu, J., Perez-Liebana, D. and Lucas, S.M., 2017. Pac-man conquers academia: Two decades of research using a classic arcade game. *IEEE Transactions on Games*, *10*(3), pp.233-256.

[3] Newman, J., 2016. Mazes, monsters and multicursality. Mastering Pac-Man 1980–2016. *Cogent Arts & Humanities*, *3*(1), p.1190439.

[4] Sigaud, O. and Buffet, O. eds., 2013. *Markov decision processes in artificial intelligence*. John Wiley & Sonns.

[5] Koller, D. and Parr, R., 2013. Policy iteration for factored MDPs. *arXiv preprint arXiv:1301.3869*.

[6] Dai, P. and Goldsmith, J., 2007, January. Topological Value Iteration Algorithm for Markov Decision Processes. In *IJCAI*(pp. 1860-1865).

[7] Pineau, M. M. (2011, may 27). ncbi.nlm.nih.gov. Retrieved Feb 2022, from ncbi.nlm.nih.gov: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3103230/>

[8] Shen, Y. (2021, Dec 28). pythonrepo. Retrieved Jan 2022, from pythonrepo: https://pythonrepo.com/repo/SparkShen02-MDP-with-Value-Iteration-and-Policy- Iteration

[9] Carmona, R., Laurière, M. and Tan, Z., 2019. Model-free mean-field reinforcement learning: mean-field MDP and mean-field Q-learning. *arXiv preprint arXiv:1910.12802*.

[10] Rachelson, E., Garcia, F. and Fabiani, P., 2008. Extending the Bellman equation for MDPs to continuous actions and continuous time in the discounted case.

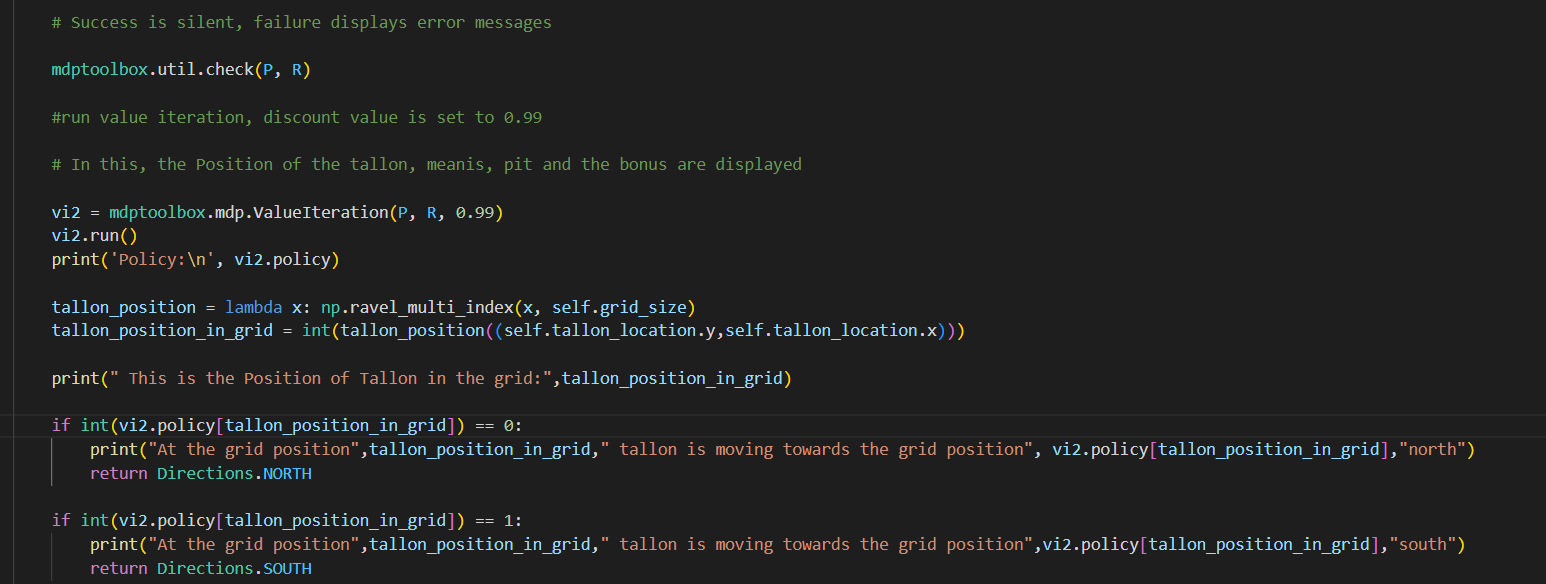
**APPENDICES**

**TASK 1:**

****

**Fig: Tallon can’t locate the bonus**

As the tallon moves away from the bonus, it loses its partial observability and is unable to find the bonus. Tallon goes towards the bonus when it approaches the bonus coordinates are known.

****

**Fig: Code for Value, Policy iteration and Utility value**

The code for Utility values is seen above, and it is used for Value and Policy Iteration. These show the tallon, Bonus, Pits, and Meanies' coordinates.

****

**Figure : Coordinates of bonus, meanies, pit and policy values**

The policy value is derived in the above figure using Value iteration and utility, as well as the nearest coordinates of meanies, pits, and bonuses.

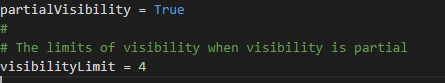


**Figure : 1D Conversion**

The code to convert grid to 1D is shown in the diagram above. This accelerates the procedure.

**TASK 2:**

In in this the partial visibility is set true so that it is changed from fully observable to partial observable. And the output for this condition is shown below. The 10\*10 matrix is considered for this condition.







**Fig: Setting partial visibility and visibility level**

****

**Fig : 10X10 Matrix for partial observable.**

****

**TASK 3:**

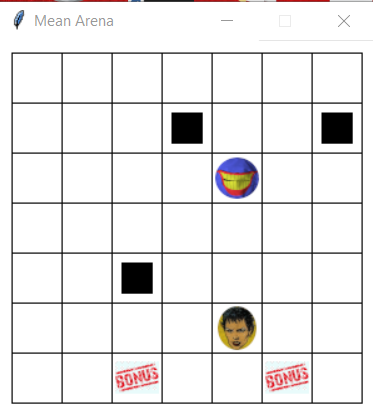
For task 3 the arena size is increased and the number of pits and bonus points are also increased. The arena size taken are 10\*10, 7\*7, 13\*13. Hence the outputs are shown below.

****

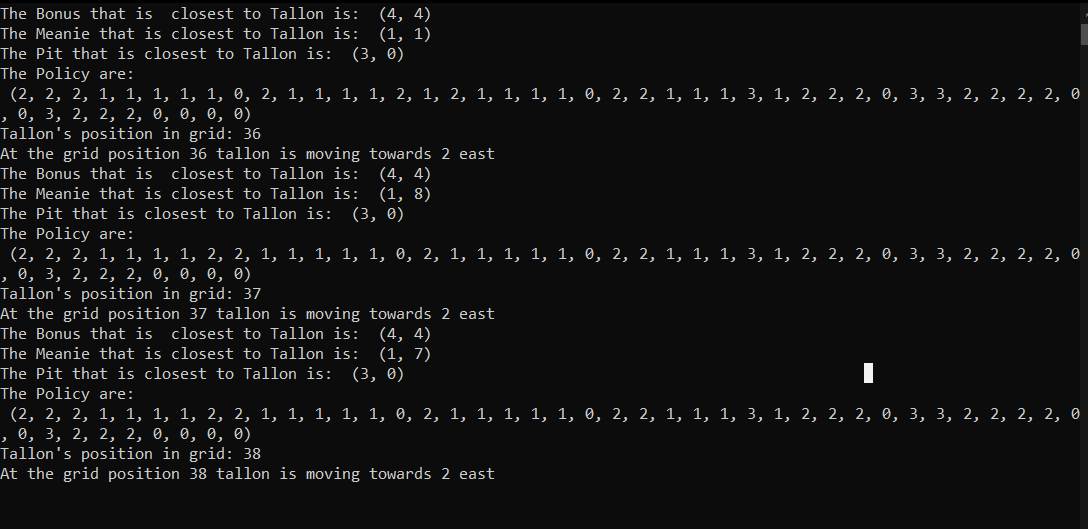
**Fig: 10\*10 Matrix**

****

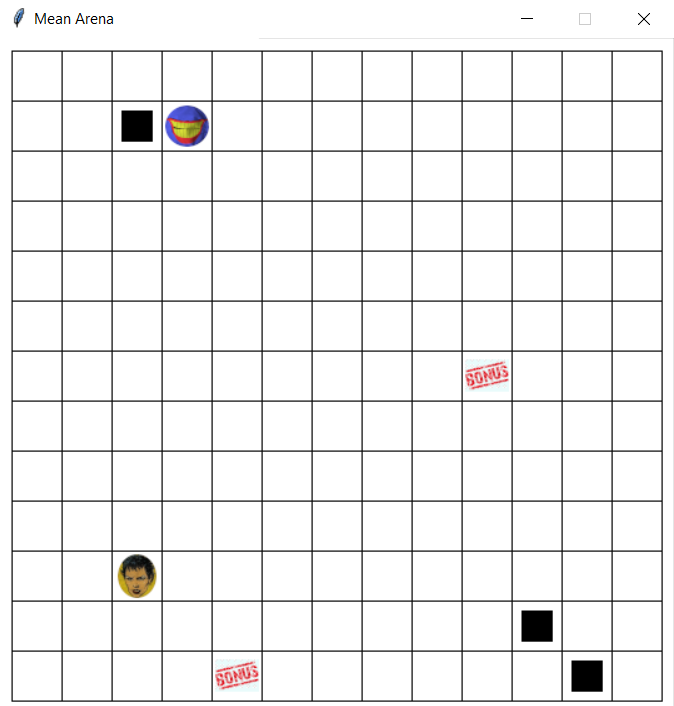
**Fig: Iteration value for 10\*10 matrix**

****

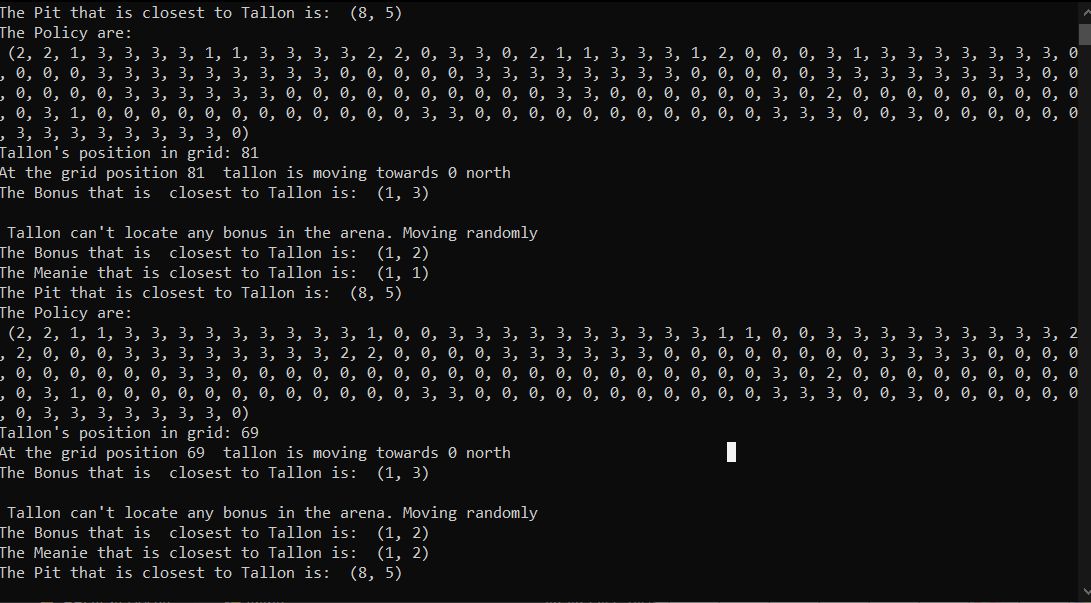
**Fig: 7\*7 Matrix**

****

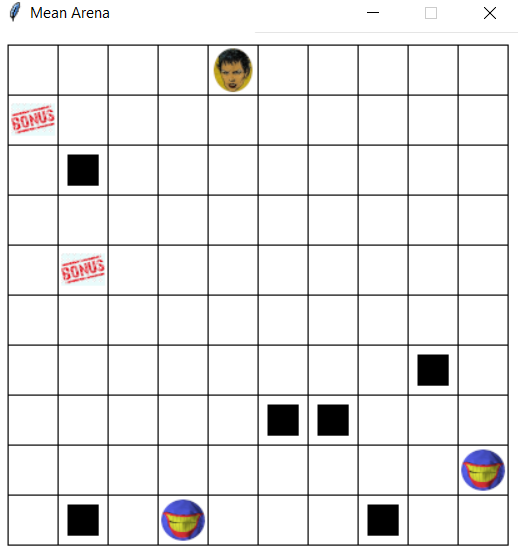
**Fig: Iteration values for 7\*7 matrix**

****

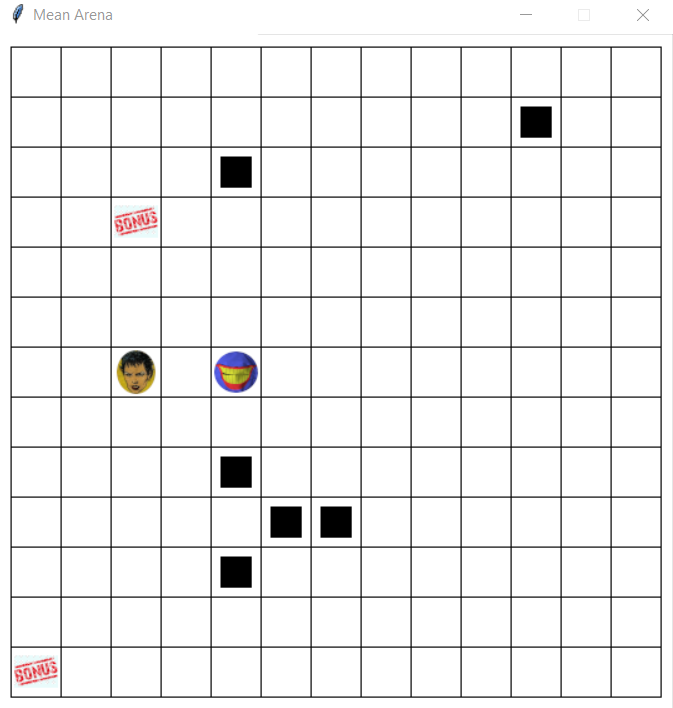
**Fig: 13\*13 Matrix**

****

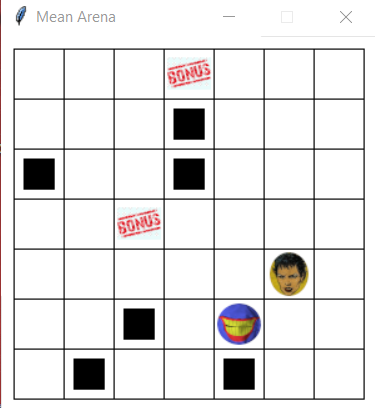
**Fig: Iteration values for 13\*13 matrix**

****

**Fig: Increase in number of pits for 10\*10 matrix**

****

**Fig: Increase in number of pits for 13\*13 matrix**

****

**Fig: Increase in number of pits for 7\*7 matrix**