**Name:Chanchal Sonawane**

**Class :BE comp (B).**

**ML: Mini project No. 3**

**Title:** Console Based Ludo Game.

**Prerequisite:**

-Basic of Python, OpenMP ,MPI and CUDA

**Problem Statement**

Design a program which demonstrate Ludo-Game with search space execution on multiple file using OpenMP directives in C.

**Software Requirements:**

1. GNU Complier
2. OpenMP
3. C language

**Hardware Requirements:**

1. Multi Core Processor
2. Ubuntu 16.04

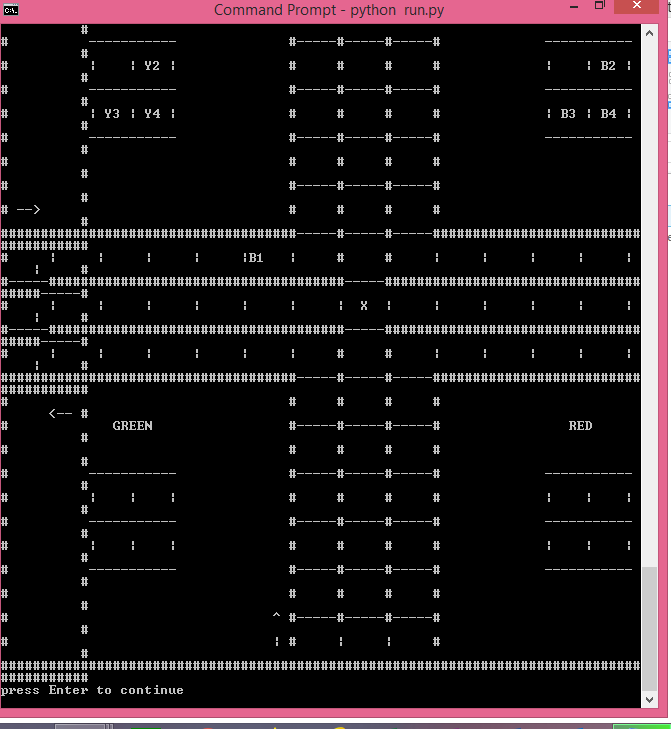
**Theory:**

Race games [1] are board games with the objective of being the first player to get one’s pieces around a linear track and into a designated winning location, usually in accordance with die rolls. These range from the simple games such as snakes and ladders in which game-play depends entirely on chance, to the more complex ones such as backgammon which involves the use of several strategies. Ludo [2], a derivative of Pachisi [3], is a non-deterministic race game with 2-4 players. Each player is represented by one of the colors Red, Green, Blue and Yellow, and has four pieces. The first player to circumnavigate all four pieces around the board and into the home area is the winner. Obstacles to this objective include shared paths with opponent pieces, unlucky die rolls, getting knocked off by opponent pieces and piece doubling. Several variants of the game exist, for example Parcheesi (United States), Ludo (Britain), Parques (Columbia), Parchis (Spain) and Ludo ´ (South Asia) [3]. Temporal Difference Learning [4] is one of the successful methodologies for developing game-playing agents in board games. Tesauro’s successful application of the basic TD(λ) algorithm to produce TD-Gammon is the best known example [5]. However, application of the basic TD(λ) algorithm for other board games such as Chess [6] and Othello [7], has produced mixed results. Variations in learning modes such as learning by playing against an expert player and learning by observation have produced improved game-play [8], in contrast to the self-play learning approach used by Tesauro. Another promising approach for improved game-play in board games has been evolutionary computation [9], with the incorporation of domain-specific knowledge at various stages in game-play resulting in enhanced performance in checkers [10].

**Implementation Highlights**

State-space complexity is defined as the number of legal game states reachable from the initial state of the game [13]. Estimated state-space complexities of some of the popular board games are: Chess (1050) [13], Othello (1028) [13], Backgammon (1020) [14] and Checkers (1018) [15]. The primary application of state-space complexity is to determine whether a perfect evaluation function can be constructed using table look-up by listing and evaluating every possible state of a game [13]. However, this reasoning may not be applicable to games for which a perfect algorithmic strategy has been discovered. The complexity class of generalization of a game is also used as an indication of game complexity, however this is dependent on the constructions used for generalization, for example in n × n chess [16]. It can be argued that a high value of state space complexity is not a definitive indicator of game complexity. For example, although the game of Nim (with n piles having n marbles each) has state-space complexity of the order of 4 n/ √ n which is huge for n = 140 [17], yet a simple polynomial-time perfect strategy exists for the game. However, a smaller state-space complexity is a definitive indicator that a game may be solved using enumeration. As an example, although an extended generalization of tictac-toe is PSPACE-Complete [18] , yet a perfect evaluation function may be constructed for a 3 × 3 tic-tac-toe game since its state space complexity is bounded above by 103 . At present no attempts have been made to compute the complexity of a generalized version of Ludo; several variants of the game may provide some hints in that direction. In the absence of such a generalization, obtaining a reliable value of the state space complexity for Ludo is one way to rule out the possibility that Ludo can be easily solved by enumerating every possible state of the game.

**System Architecture**

****

**Future Scope:**

The future work could be aimed at extending this approach to a GPGPU computing environment using CUDA to give us high performance .

**Conclusion:**

## Hence ,We have studied and practically implemented consolw based Ludo Game

## with search space execution on multiple file using OpenMP directives in C.