Project Report

Hexxagon

by

Sunny Goyal	06005010
Prathmesh Prabhu	06005002

The Game:

The game of Hexxagon is the derivative of an old famous board game called *attax.* The basic idea is to fill the maximum number of cells by one's gems as per the rules given below:

1) Both player have some number of gems placed on the board at the begenning of the game.

2) A player can move any of his gem in only two ways:
   a) By *copying* the peg from a cell to an adjacent cell.
   b) By shifting the existing gem to a cell one layer away from the current cell of the gem

The player who has the maximum number of gems at the end of the game or who forces the opponent into a situation where no move is possible is decalred the wiiner.

This is a game easy to learn and yet difficult to master!

The implementation:

The implementation and the code for the game has the following general contour:

- **The board class**: The board for the game is represented as a 2D vector stored as a free variable in the board class (function name: make-board) The board class has the following components:
  - ***variables***:
    - The board: every cell has the information regarding the occupency.
    - The move-board: an alias board that is preprocessd at the loading of the game to contain the information regarding move generation during run time.
    - The lists other than board that store the user's and the computer's gem positions.
  - ***board implementaion***: the class also contains different procedures that have an access to

the board values such as setting values, reading values, finding game states, copying board vectors etc.

- finally, the class acts also as a link between the three major parts of the program:
    - the board
    - the game thread that moves the game ahead and the thinking part of the game, the AI!
    - the graphics that takes care of the GUI.

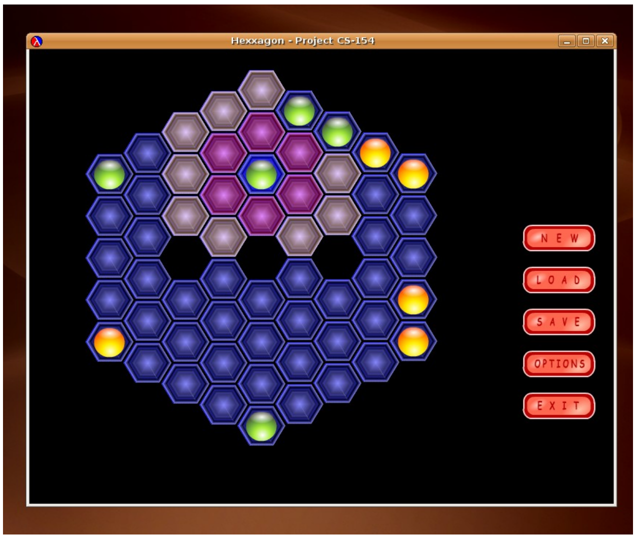
- **The thinking cap of the game**:

  The thinking part of the game is handled by the procedure comp-logic and is the core of the whole AI. The AI, as usual consists of the following three components:
    - ***The evaluater:*** The evaluation of any game state is an easy task as it is directly a function of the number of gems of both players on the board. the functions of the board class are evoked for this task.
    - ***Move generation:*** a record of all possible types and position of moves is processed and stored by the board class at the start up itself and the AI uses this listing (a 2D vector, in fact) to find the possible moves for a given game state.
    - ***The search***: Simple minimax search can be used to look up the effect of all possible moves at any given game state. In order to speed up the process a bit, the program uses alpha-beta pruning going to a static depth chosen by the user while deciding the level of play. The alpha-beta approach of course does not give the best possible move at any state but the results are reasonably good at a depth of 3 ply(ply here having the meaning of one move by any one of the players) at an appreciable speed.
        - Besides, the procedure checking whether a side has won also plays a non trivial role in the game play. It does so by looking up the two ways of a game-end(cf above) whenever a player is left with no possible moves. (The check-win procedure does not come into play otherwise.)

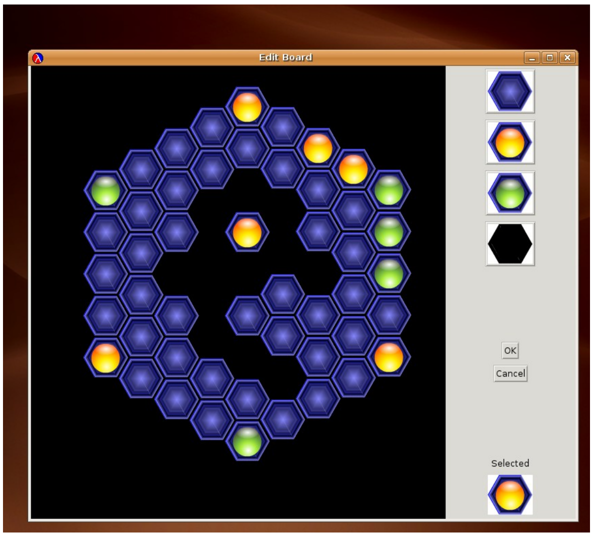
- **The GUI**

  The graphics part of the program is broadly divided into following:
    - Main game window and peripherels: The game window is created at the outer most scope and the related callback functions are active at this level. The game window sees to the following: display and gui on the game board, providing menu frames, and making way for board-edit and peripheral windows.
    - The menu control procedure has the menu as a free variable and all callbacks to the menu are local to this class.
    - The edit board superimposes and clears the main board contents. The callbacks are defined seperatly.
    - XpixMap images are used to represent important gui components in the game and other windows and xoring of images is used to achieve transperency in these images.

Some Screenshots of the game:



The board showing possible moves of a gem



Edit board: a veiw

Installation and running:

Your system must have drscheme installed for the program to run.

To install, on the terminal, change the current directory to the project parent directory and enter the command “. *compile*”

The outpot binary file is put in the bin directory and the associated library files are put in the lib directory

To run the output file enter: *“./bin/hexxagon”*

Note: The untarred folder already contains the compiled version having the folders bin and lib. Compiling the program again simply overwrites both these folders.

Scope and Improvment:

Like any other undertaking we realise that the project has a large scope for improvement. A few points we feel compelled to note:

- There is a huge scope for the improvment of speed and the search of depth of the AI part by implementation of iterative deepening instead of the normal alpha-beta applied.
- Also, a part of project that we wanted to do but had to give up is to modify the program as to accomodate variable board shapes. This can be done by having alternate board class. The AI part and its intigration with the class has been done keeping this point in mind. The thinking part of the game has no idea whatsoever of the type of board on which the game is being playd. It only has the basic game-rules and the game state independent of the board shape.
- Another part we had to compromise on was to include lan gaming facility on the game. We actually ran out of time and lost spirit after a night's long work on this one. We hope to be able to add this one feature just for the cause in the holidays!!!