CHAIN REACTION PROJECT REPORT

CS 154 – ABSTRACTIONS AND PARADIGMS IN PROGRAMMING

Group Members:

- Shivam Goel (180050098)
- Neel Aryan Gupta (180050067)
- Karan Agarwalla (180050045)

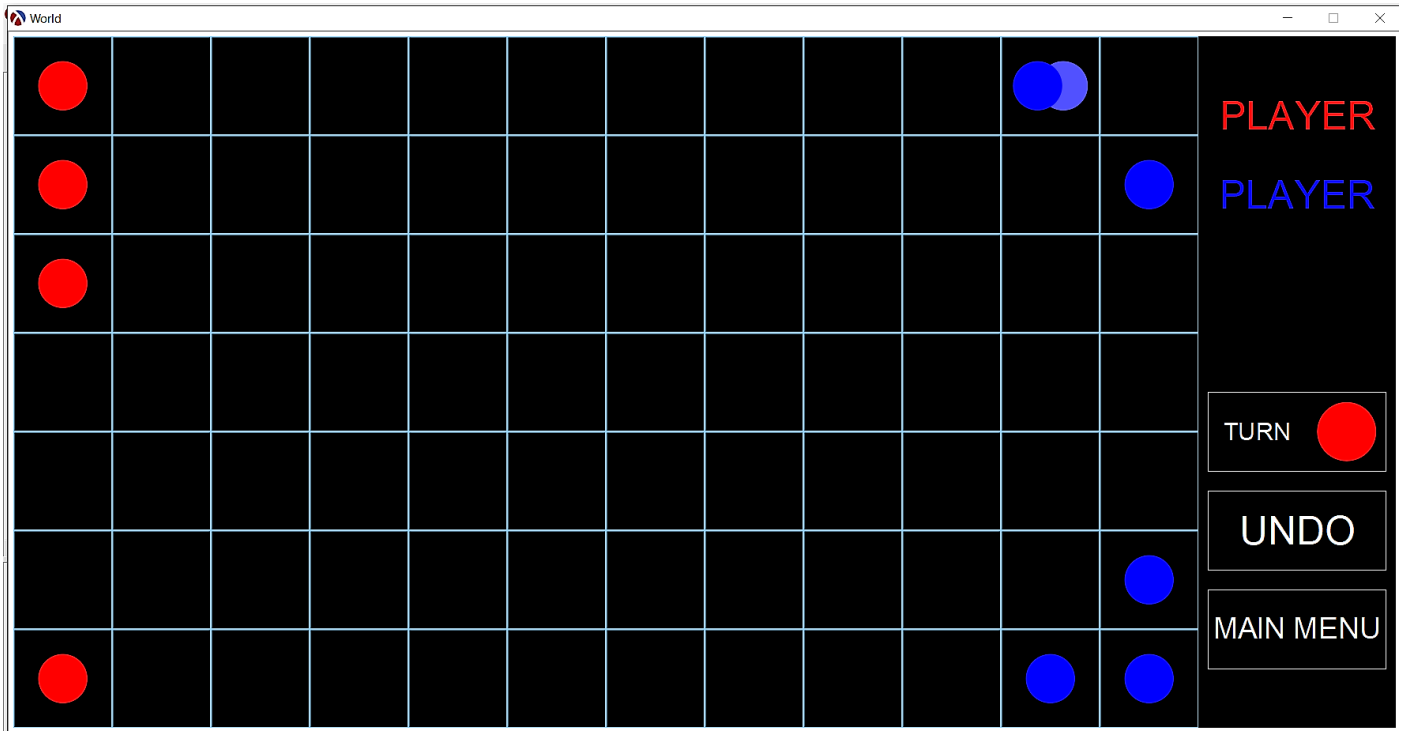
Description of the problem

The aim of our project is to recreate the game “Chain Reaction” which is based on the strategic puzzle single-screen multiplayer game of the same name “Chain Reaction”. The game will consist of Single Player mode (against AI) and Multiplayer mode (up to 4 players) with grid size up to 12×7.

Players take it in turns to place their orbs in a cell. Once a cell has reached critical number of orbs, they explode into the surrounding cells adding an extra orb and claiming the cell for the player. Players may only place their orbs in a blank cell or a cell that contains orbs of their own colour. As soon as a player loses all his orbs is out of the game.

The most interesting thing is how unpredictable the game seems to be in the end, at least when you play it with your human friends. The obvious heuristic that tells us you're better off at the moment by having as many orbs as possible turns out to be very wrong. While it so seems to everyone, that say, red will win, blue suddenly takes over.

A visual of the interface shows all the buttons with the players still having orbs in the game.



Libraries

We have used 2htdp/universe for handling of events and 2htdp/image for various graphical interfaces and display of grid. We have used Microsoft Paint to deliver proper interfaces.

Individual Files description

- orb-models.rkt implements orbs of all colours with all sizes to be displayed.
- AI.rkt is the main AI code tasked with giving the best moves given a vector state and colour.
- main.rkt has images of grids and deals with all event-handling operations.
- settings.rkt contains all the parameters which can be changed by the user.
- Three image files which are interfaces for main-menu, choose players and instructions-guide.

Design of the solution

Input is taken by mouse-click.

Multiplayer mode

- We represent the grid with a 2D vector with up to 12×7 elements. Each element has three members: Colour of orb (‘empty if none), number of orbs and maximum number of orbs allowed.
- The GRIDS variable stores the grids till now helping to implement UNDO feature. The variable COLOR-LST stores the colours whose orbs are still present in the grid. The first element of it represents the current turn.
- The ‘grid state represents the game currently in progress, the ‘grid-transition represents the bursting of orbs in progress and ‘stop represents the completion of game.
- The variable COLOR-LISTS stores the variable COLOR-LST at each ‘grid state.
- The catch was to represent multiple scenes from a single mouse-click. This has been achieved by tick-handler. It has a helper function which checks if a point (X, Y) has orbs beyond critical value upon which it recursively calls its surrounding cells with it.
- The function FILTERS remove all the colours with no presence.
- In short recursive calls to the surrounding cells helps to solve the problem for multiple players.

AI mode

AI basically tries to maximize proper value (which is proportional to chances of winning) which is written based on certain rules. It also assumes that the other player tries to minimize this value and it places it’s turn considering that the other player makes an ideal move.

Now the file constitutes three major functions –

- Board-value - This function calculates board value based on certain colour which it takes as it’s argument. It is calculated using certain guidelines commented at the start of the file.

- Quiescent-state - It takes colour and vector (which gives us the orbs placed on the board) as it’s argument. It returns a value that if certain critical orb is broken then at max how much critical enemy orbs break down. The use of this function is that if the value retuned is small then board value will not change significantly on playing the move. So, it takes proper value to

be the board value of that state. But if the value returned is larger than the function further sees the next player move and sets the proper value accordingly.

- Decision – It computes proper value for each legal turn and compares all values and return that position with greatest proper value. It takes colour and vector as it's arguments. It also includes a time limit which can be changed to increase or decrease the difficulty of the game. If the time limit is small then AI will have less time to compute the required position so returned position may not be the best possible. If time limit is increased then the position will reach as close as possible to the best position.

References

- https://brilliant.org/wiki/chain-reaction-game/
  The link helped us to calculate board-value for a given grid.

- Visual simulation: https://www.youtube.com/watch?v=L15TaZbLUo0

Limitations and bugs

- The player inputs (while playing against AI) will be queued if subsequent clicks are registered while the AI is computing its next move.
- Only 4 colours (Red, Blue, Green and Yellow) are available for playing.
- In one-player, Player always plays the first move.

Clever programming

- The AI has been easily mimicked by mouse-handler function helping us to eradicate repetition of code.
- Multiple world scenes for a single move has been displayed by tick-handle by careful coding without which it is not possible.
- The window can be scaled to any scale-ratio by changing it in orb-models.rkt. However, the ratio of width to height remains same. Moreover, many other changes are also allowed as elaborated in settings.rkt .