CSCI 5511 – (001) Artificial Intelligence Final Project Report

(Dots & Boxes Game)

Group Members:

Member 1: Sree Ganesh Lalitaditya Divakarla (divak014@umn.edu)

Member 2: Rushikesh Zende (zende039@umn.edu)

Description of the Project:

Our main agenda of this project was to recreate our nostalgic childhood game into a new generation AI game challenge. This satisfied our urge to play our favorite old game which we used to play sitting on the back benches of our classrooms to kill time in a boring lecture. On the contrary, now we sat in a classroom the entire semester and paid attention in order to learn how to build AI agents to play such games and win using all our old strategies and some new which we learnt while doing this project.

How is this game played?

Setup:

- Dots and Boxes is played on a paper sheet with a grid of dots. A common starting grid is 6 x 6 dots, but any even-sized square grid can be used.
- Two players (typically Red and Blue) play the game. Each player has a colored pencil or marker.
- The grid starts empty, with just the dots at the intersections of horizontal and vertical lines.

Gameplay:

- Players take turns adding a single horizontal or vertical line between two adjacent dots.
- When a player completes the 4th line in a 1x1 box area, they write their initial (R or B) in the box to claim it.
- After claiming a box, that player takes another turn. So box completion leads to consecutive turns.
- Once a box is claimed with an R or B, no more lines can be added within the box. Only outer lines extending from the box can be played.
- Players keep taking turns drawing lines extending from boxes, trying to create and claim more boxes.
- When no more lines can be added to the grid, the game ends.

Scoring:

- The player who has claimed more boxes wins.
- In case of a tie, the game is declared a draw.

Explaining our approach towards making this game with AI Agents:

To achieve this we used a github code which was to play two human players against each other and we wrote two different agents (players) and played them against each other to see whose player wins.

We used the below -mentioned repositories to get our desired GUI of the game https://github.com/aqeelanwar/Dots-and-Boxes https://github.com/sohailzakir/Ai-Project-Dot-And-Boxes-Game which are open source codes for 2 vs 2 Human players.

Moreover, We also referred to a research paper from the Department of Mathematics, University of Texas, Austin, USA (https://web.ma.utexas.edu/users/allcock/research/dots.pdf) as this was a very straightforward paper talking about some of the best do's and don'ts regarding the game.

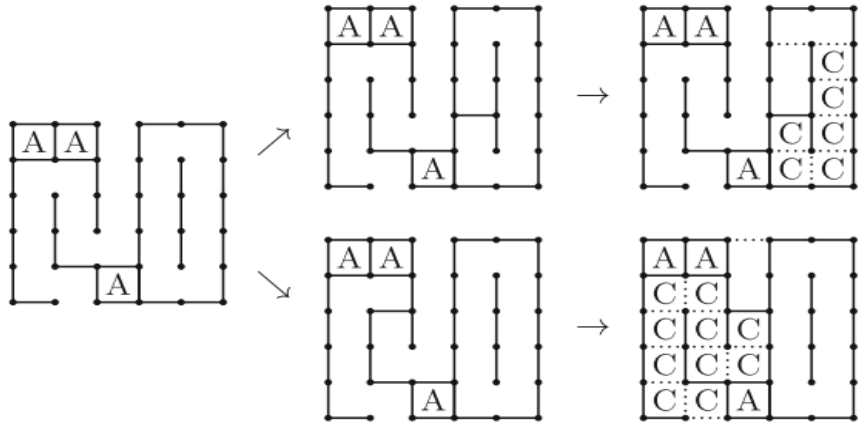


Fig. 1 A Dots and Boxes position consisting of a 12-chain and a 10-loop, and two lines of play demonstrating the hard-hearted handout

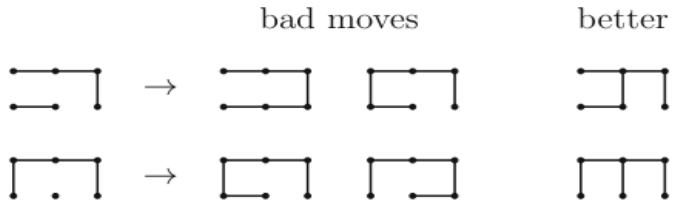


Fig. 2 Opening a 2-chain in one of the "bad" ways allows the opponent to choose between capturing both boxes and then moving again, or replying with the hard-hearted handout. Bisecting the 2-chain as shown on the right removes the opponent's second option

We adapted to couple of strategies to make our heuristics for the game such as:

Rushikesh wrote an AdversarialSearchPlayer using Mini-Max function and extending it with alpha-beta pruning so that the player could search deeper in the min-max tree and make best moves.

For the AdversarialSearchPlayer we used heuristics named below which are working very well when used in the Min-max function with alpha-beta pruning. Also, these prove to be most basic heuristics for the game of Dots & Boxes

- Count of Boxes won or lost
- Chain rule (forming a long chain)
- Win / Loss
- Count the number of long chains & Find adjacent boxes which can build a chain.

Lalit wrote a LocalSearchPlayer as he wanted to see if advanced and detailed implementation of strategies via heuristics could help him win against a min-max function with basic heuristics. So, he used Heuristics like:

- Mobility - High bonus for having more available moves
- Stability - Bonus for having more boxes complete
- Parity - Bonus for having more boxes with parity
- Box won or loss
- Penalty based on move count to find shortest solution

With all these advanced heuristics LocalSearchBot was able to win with a good margin against RandomBot and Human player.

But it was not successful to win against the AdversarialSearchPlayer Min-max with Alpha-Beta pruning having simple heuristics. Though it lost by just 1 point in 13 out of 15 games played.

Instructions to Run the Game:

We used a Python script to run our project. You can extract the zip file in a folder. The folder should consist of AdversarialSearchPlayer.py , Bot.py, GameAction.py, GameState.py, LocalSearchPlayer.py, main.py, RandomBot.py. Program dependency such as

from tkinter import *

import numpy as np

from Bot import Bot

from typing import Optional

from GameState import GameState

from RandomBot import RandomBot

from AdversarialSearchPlayer import AdversarialSearchPlayer

from LocalSearchPlayer import LocalSearchPlayer

from time import time

and then simply:

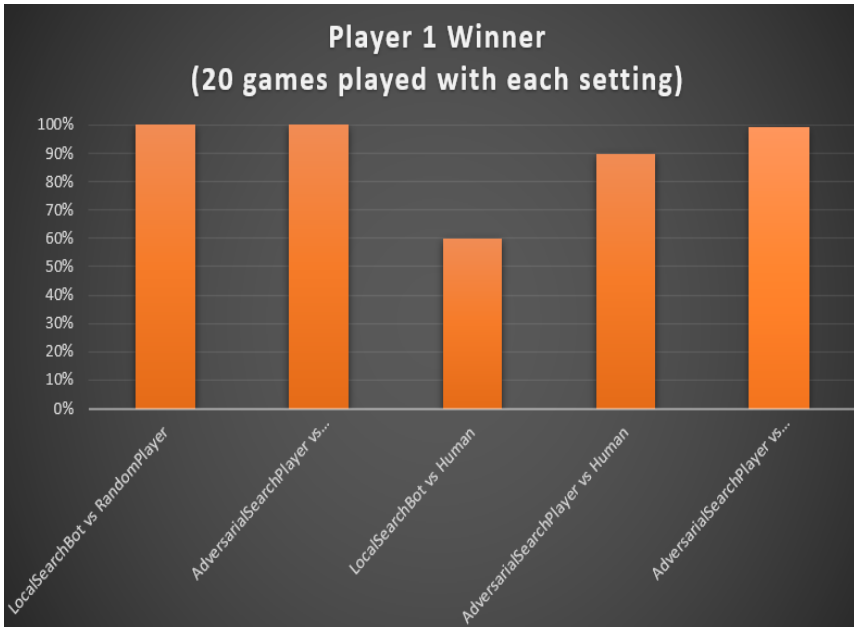
→run py -3 main.py

→ select the number of grids

→game mode.

Enjoy!

Results:



We played 20 games with each setting mentioned below:

- LocalSearchBot vs RandomPlayer
- AdversarialSearchPlayer vs RandomPlayer
- LocalSearchBot vs Human
- AdversarialSearchPlayer vs Human
- AdversarialSearchPlayer vs LocalSearchBot