Amazons Project

MAT-564 Final Project

Student: Rafael Diaz Cruz <rafael.d@digipen.edu>
Professor: Matt Klassen

Overview

Console application for the Game of the Amazons combinatorial game.
The project was written using features from C++11 and should compile and run across multiple platforms.

Gameplay

https://youtu.be/rwR0VzZkGRo

Extra feature: disjoint regions

User is able to identify disjoint games and list them with useful information.

AI algorithms

Optimal play: Guru

Canonical positions

In order to solve optimally for small games I implemented the concept of a canonical position. A canonical position is a non zero position which has at least one amazon in the left column and top row.

Map canonical positions to unsigned ints

For canonical positions fully contained in a 4x4 region it is trivial to turn them into an unsigned 32-bit integer in a 1-1 way, this is there is a bijective function which can identify every single canonical position. And the most important part, the function is reversible.
The mapping is done using the following algorithm:

1) Pad the region with killed tiles (3) until you get a 4x4 region
2) Remap values: killed tiles=0, blanks=1, left=2, right=2 (only did this so the most common tile in small positions has a zero value and the id was not huge)
3) Write a base-4 number in little endian (i.e. first 4-base digit is the less significant)
4) Turn base-4 number into a normal number. This is the canonical id

Example: canonical position #285

0 2 0
0 3 3

Step 1: pad with 3's

0 2 0 3
0 3 3 3
3 3 3 3
3 3 3 3

Step 2: remap values

1 3 1 0
1 0 0 0
0 0 0 0
0 0 0 0

Step 3: read in little-endian

0000000000010131 (base-4)

Step 4: turn into number

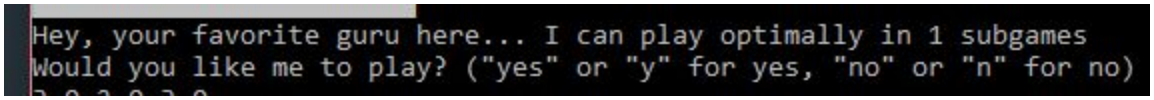
0000000000010131 (base-4) = 100011101 (base-4) = 285 (base-10)

Map any position to canonical positions

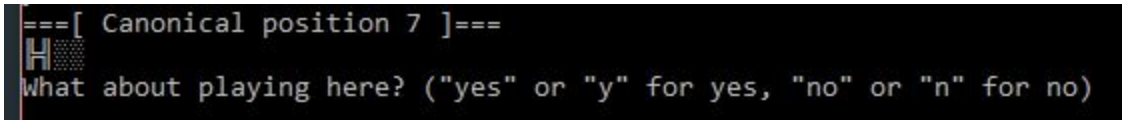
If a connected position is fully contained in a 4x4 region, then it can be mapped to a canonical position. This is very powerful as it might be possible that we already know how to optimally solve the contained region.

Guru entity

There is a guru entity who is looking for canonical regions that they are able to solve optimally. As soon as the guru finds a position whose respective canonical position they already know how to solve, you will be prompted.



If you decide to get help from the guru, they will ask you if you want them to play in every position they know how to solve.



Guru database

The guru uses dynamic programming and precomputed results to be able to solve positions.

Training guru

During standard gameplay the guru cannot expand their database. It is via an training process that the guru can learn and expand its database. The process is partially automated and allows manual intervention for solving positions that the automation logic behind the guru cannot unscramble.
Here is a video of how the guru learns: https://youtu.be/SfotZdwuyb0

Next steps

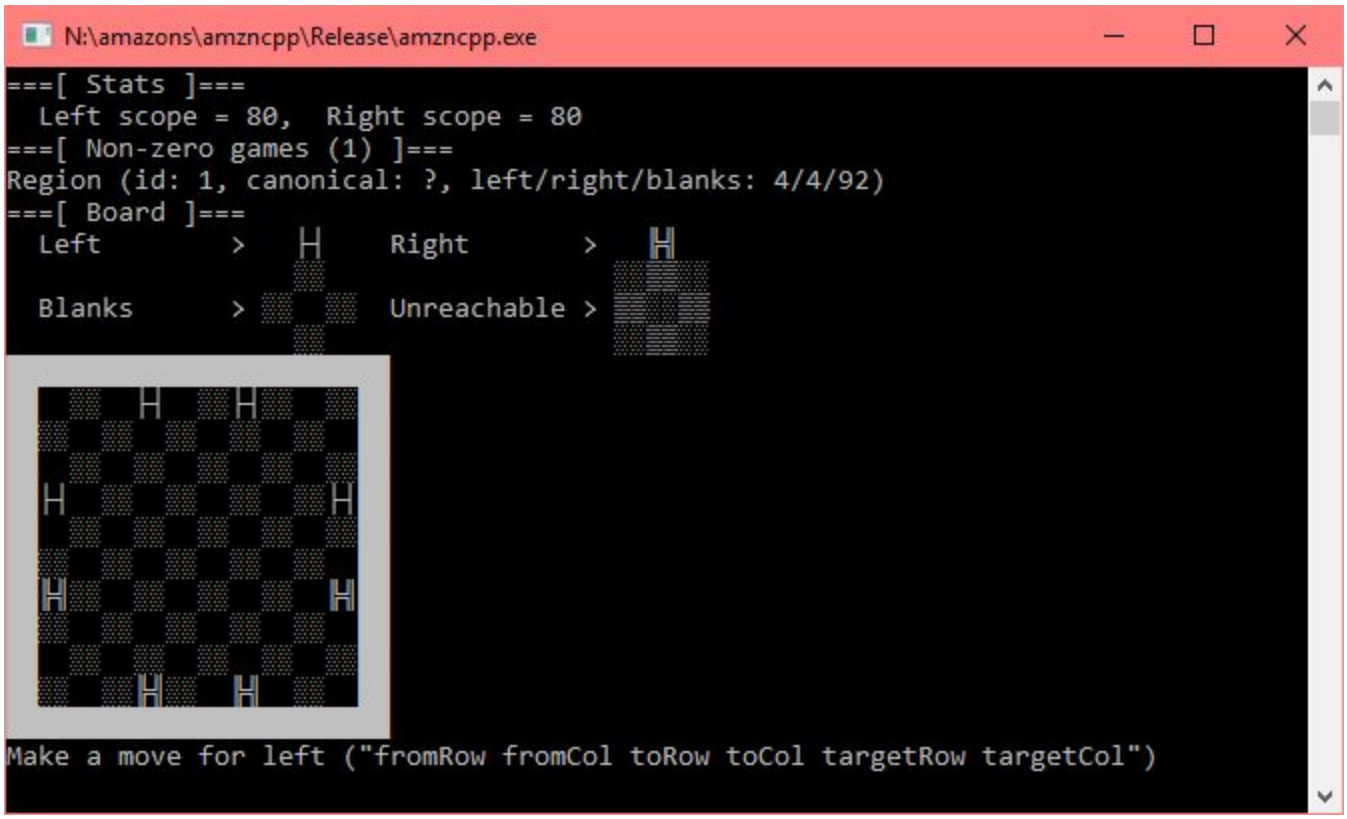
Most of the automated canonical games I was able to persist were zeros or trivial games. I created the concept of safe number, which is a value the guru is sure that matches the actual game number of a position. It is easy to be confident about the zero games, and using this it is possible to find in an automated way, ones, and minus ones. This is the next big step, which will iteratively increase the size and complexity of the database. This is as for expanding automated scenarios. As for expanding the 4x4 grid that constraints the supported canonical positions, if we use an arbitrary length integer we will be able to go to larger canonical boards.

Max/Min & Min/Max

I implemented the Max/Min and Min/Max algorithms for automating the decision making of the computer player.

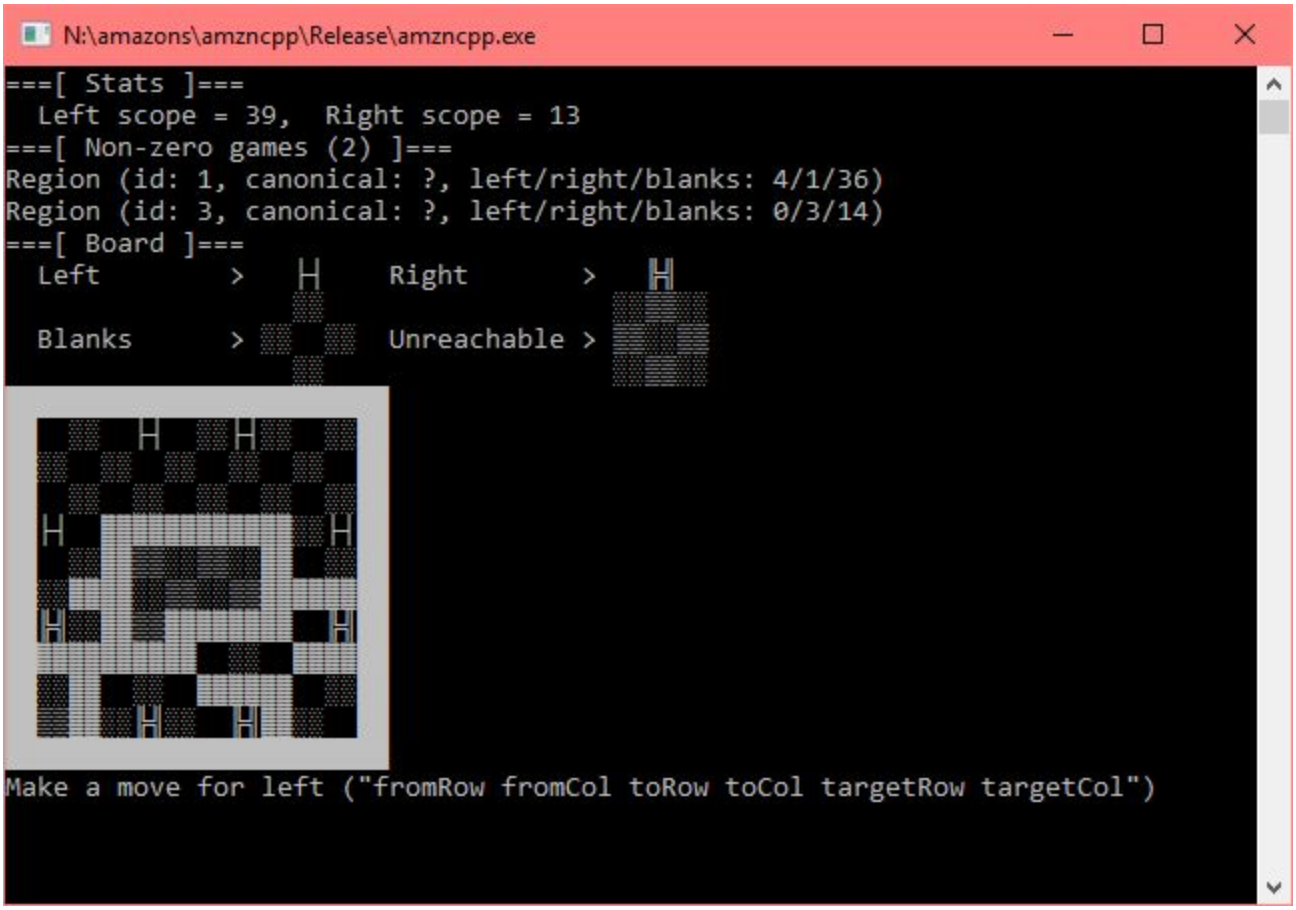
UI/UX

I used distinc enough symbols for every element in the board so it is easy to distinguish them. Also, the board blank tiles use alternating colors to help visually follow rows and columns. I also included useful information for the user so they can make good decisions when playing.



Unreachable regions

Regions with no amazon pieces will gray out visually indicating they are non playable regions.



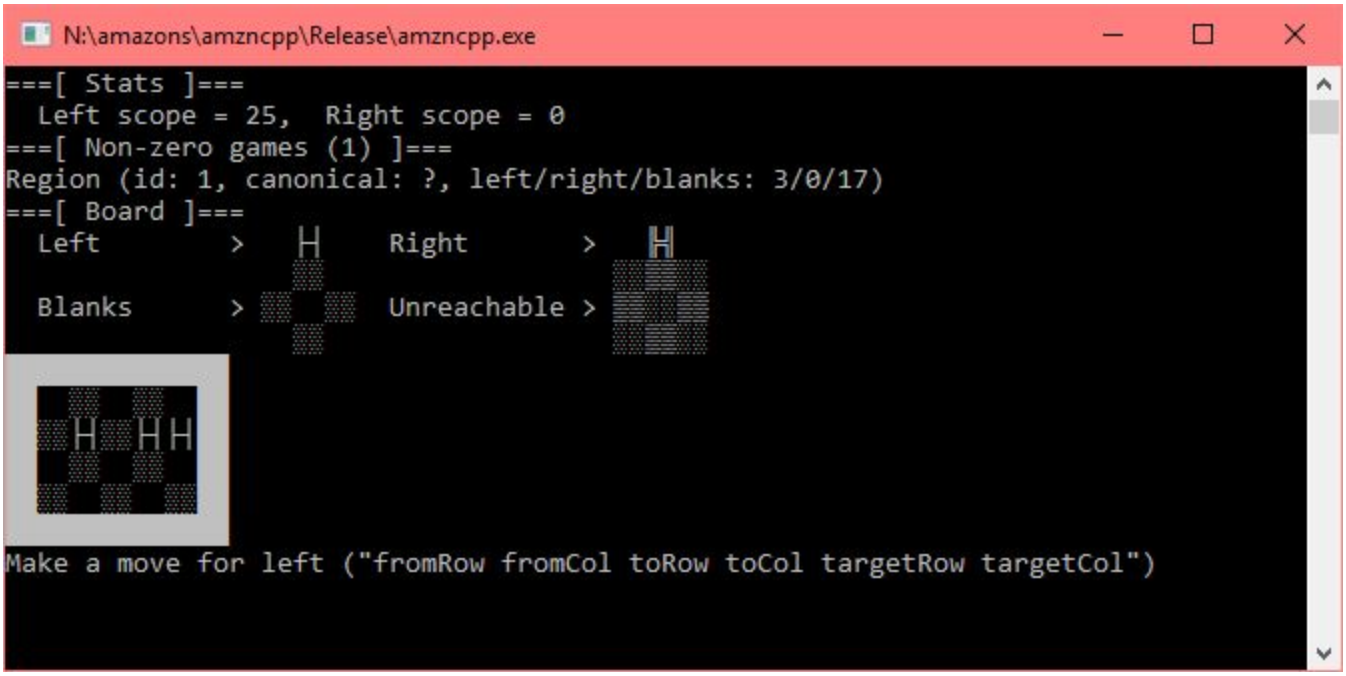
Other features

User can play as right

Because the game is not hardcoded in any way, and because there is a TurnManager component encapsulating the turn taking, it was really easy to give the player the chance of playing as the right player.

Dynamic board size

The board is generic and not constrained to 10x10 or even to square regions, you can use any number of rows and columns and all the functionality should stay the same.
Important to mention that this was implemented without creating any compatibility issues for 10x10 boards that other systems will use. In order to indicate you want to use other dimensions start your file with: “-1 {rows} {cols}”, replacing the values inside braces with your desired values. The braces are not required.



Design features

I spent a good amount of time focusing on keeping good design for the project. So it is easy to extend. Among the main features: it uses a game engine which controls the flow of the game, it has an IO system that allows you to easily ask questions and persist files, the system for playing optimally is encapsulated in a Guru entity, which uses dynamic programming and precomputed results in order to be able to determine the best moves.