# CSC 503: Computational Applied Logic
# Project Report

## Building a Strategy Game using ASP solver

Harshit Patel(hpatel24) & Sagar Bajaj(sbajaj)

**Overview**: We built a Strategy Game using ASP and Python, because we wanted to solve a complex problem in an easy way. That problem here was Map Generation, and ASP can make a very good map in few lines of code and besides, a game is a very good application of using that map.

Our work uses ASP for map generation, which is not a standard way in current practice. We find Clingo to be very useful in this particular application. With our current code (only of 60 lines) we can generate as large of a map as we want.

We were aiming to answer the research question that whether we can use ASP to create any full fledged strategy game. Based on our project we conclude that, given enough time one can easily create a full fledged game when we combine ASP with other tools like Python. We also found that Clingo's python API is a little weak when it comes to generating random models. If that feature is added than we could do random maps very easily.

We had to remove a feature that we proposed to add in the game:

- We were not able to add multiple units for each player, because it already required a lot of text processing and file manipulation to work with one unit per player. Hence, we concluded that given the time constraint, we would either be able to add this feature or add other features of the game and we choose not to do this feature.

**Related work:** We were inspired by strategy games like Civilization and Polytopia to work on this project. None of these games have mentioned the use of ASP solvers to build the game. Moreover, with our approach we ended up making a unique game with the aim to prove that ASP solvers in combination with other tools like python can be used to build such games in future.

**Approach:** We ended up with the following gameplay:

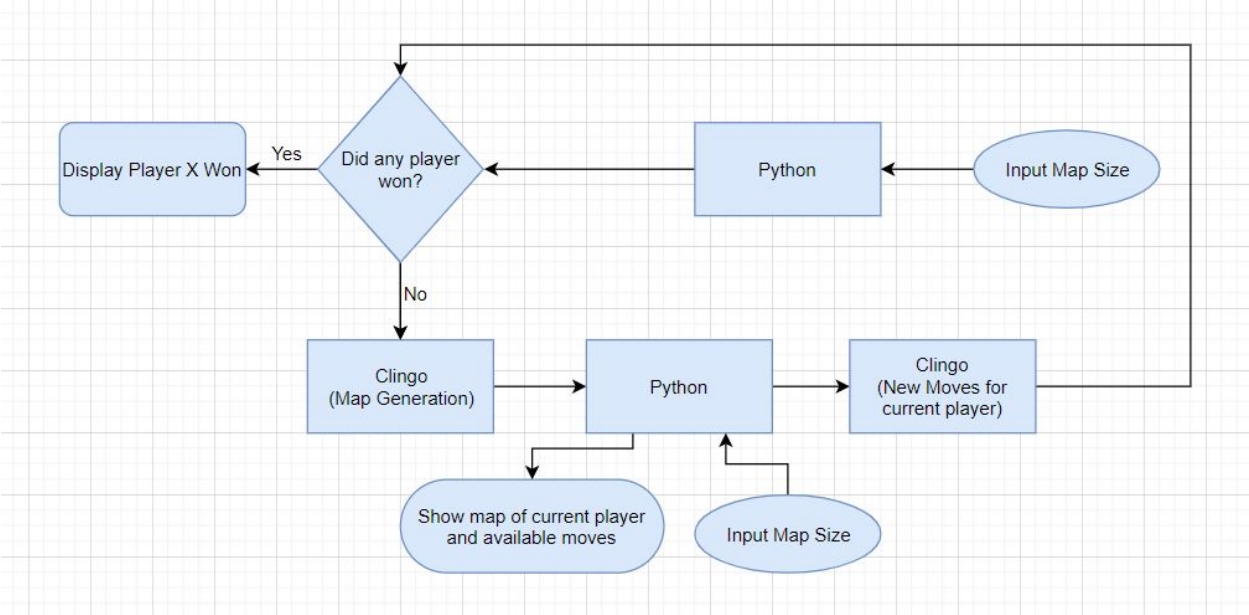
- Two players
- User defined map size
- Slightly random maps
- Players goal is to find the resource or kill the other player.
- Three types of terrain: Flatland, Mountain, Water
- Player can move into any three of them, but would always start on a flatland
- Two players would never spawn close to each other nor would a resource spawn near any player
- When on a flatland a player can see 8 tiles around them and the area that they already explored

- When on a mountain a player can see 24 tiles around them and the area that they already explored
- When in water a player is immune to other players attack
- A player can shoot(attack) other player if the player is in their sight, the other player would die with a probability of 20% if shot at
- If a player chooses to attack then they have to lose their option to move and do a diplomatic deal
- Players can interact with each other using diplomacy
- Other players can accept or decline proposal that the current player sends
- The resource changes its position every once in a while (depending upon the size of map)

We used the following solver, programming languages and frameworks:

- Python
- Pottasco Clingo
- Clingo API for Python
- Numpy
- Astropy
- Random
- Ast
- Ipython

Architecture:




**False starts, pitfalls, failures:** We were supposed to add multiple units for each player, but decided to limit it to one. The question that arose was how the new players will be added and how will that affect the game. For creating a new player for, say team red, new pawn can be spawned by the player themself after accumulating specific number of points. But to accommodate that, a point system had to be in place which we didn't do from the start and time interfered. Also, clingo was generating moves for specific color for a player and not pawn number. Clingo file would have needed a lot of changes to successfully depict more players on the map. We unfortunately ran out of time before attending this as removing current bugs was more important than trying new things. In short our clingo logic was not scalable for more pawns in the game at the moment. We decided to change the gameplay and include features like hiding in water and shooting longer distances from mountain to make up for it.

**Outcome and evaluation:**

- Our project was a game and we tested our code by playing the game. We checked the amount of randomness in map, and how interesting was the game to play. After these tests we found out that, clingo is very bad at randomizing, basically it changes the grounded atoms one by one to generate different models, so each map that it generates is different from its previous map by just one atom. Coming to subjective test which was how interesting the game is, we would say it is interesting but could be made more interesting by adding features to it. The results fall short of our expectations, first of all random map generation was very important feature for the game and second we expected ourselves to build a lot of features but we ended up adding a few.
- The project was certainly more difficult than we expected. First of all randomizing map was challenging, secondly adding accommodating multiple units of a player was a difficult task and last of all, representing the map without a GUI was a troublesome task as well. In order to randomize the map, we generated a random number of models, from those random number of models we choose one model randomly. This added good enough randomness to the map. It was very hard to make a single unit game and so adding multiple units would have been even harder, therefore, given the time constraints we dropped that idea. Finally, we saved our map to a csv file and represented them as a table on the python console.
- This project was a very good platform to test the skills that we learned in this class, and we learnt a lot from it. We learned how to use an ASP in an application. We learned that, clingo API could be made better by giving a way to randomize models.**
- We learned to code in clingo better. We also learned a lot of tricks in python to achieve needed goals. Our understanding of logic in building a map or generating new move set will help us solve more problems like this in the future using clingo and have inspired us to use it for various similar problems in our future works.
- We believed that we put a lot of time and effort in this game and consistently tried to make it better, therefore, we would grade ourselves A+.

NOTE :

- The code will only run if you have anaconda installed because clingo API needs it.
- Please run the code again if it throws error, sometimes the API doesn't work while map generation.