Artificial Intelligence Agents for StarCraft 2

Carlos Cardenas-Ruiz, Emilio Tonix-Gleason, and Julia Rodriguez-Abud

Artificial Intelligence

Abstract

This document describes a series of projects developed for the 2021 Cinvestav Guadalajara Machine Learning class given by Dr. Andres Mendez Vazquez. The course is comprised of three main parts: searching algorithms, bayesian networks, and multilayer perceptrons. The aim was to develop a StarCraft AI agent using the PySC2 Python framework. In total 5 agents were developed, 3 of which were developed for minimaps (maps with clear and small objectives) and the rest for the simple 64 map (a small map that actually emulates a normal game with all its components). Most agents had good results although they were not perfect implementations, as our time was quite limited and the learning curve for PySC2 was steep. Nevertheless, hopefully our learning process will help others going through the same path by using our documentation and github code as reference material.

Carlos Cardenas-Ruiz is with the Department of Computater Science, Cinvestav, Guadalajara, México, e-mail: carlos.cardenas@cinvestav.mx.
Emilio Tonix-Gleason is with the Department of Computater Science, Cinvestav, Guadalajara, México, e-mail: emilio.tonix@cinvestav.mx.
Julia Rodriguez-Abud is with the Department of Computater Science, Cinvestav, Guadalajara, México, e-mail: julia.rodriguez@cinvestav.mx.

Index Terms

CONTENTS

I. Introduction

A. Installation

The easiest way to get PySC2 is through a pip install command:

python 2.7

```
$ pip install pysc2
```

python 3.X

```
$ pip3 install pysc2
```

For more info, templates and documentation on PySC2, visit the website

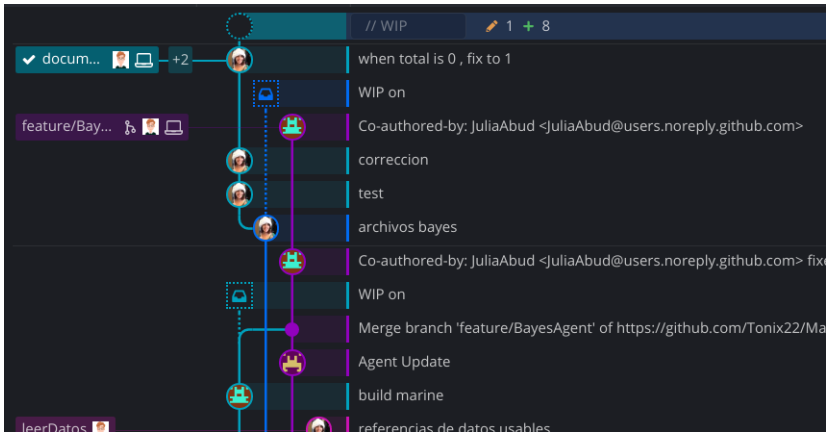
B. About PySC2

PySC2 [1] is a non-official Deepmind product commonly used for machine learning competitive programming. It is a multiplatform python framework that allows reading and writing of StarCraft II data. This allows the implementation of automatic strategies based on learning algorithms.

C. Workflow

To develop and produce our project we used gitflow strategy. This allowed us to split tasks in batches, which are treated as features in the code implementation. In Git we used branching and merging to do parallel work and code review. When each task was completed, a branching merge was done.

Fig. 1. GitFlow



D. Roadmap

We used the DevOps methodology, which focuses on bringing the operations lifecycle into the same agile experience as the development teams. When adopting the DevOps philosophy the team remains responsible for the release during the entire lifecycle of the product. Organization and planning were done using a Kanban board. And the roadmap used for the project was done in an iterative way with the following steps:

1) Choosing StartCraftII map (or minigame)
2) Understanding the problem
3) Planning a solution with the course tools.
4) Designating tasks and features
5) Coding and doing of individual tasks
6) Merging and code reviewing
7) Testing
8) Releasing git code
9) Showing the product and the results for final review and feedback to the teacher

Fig. 2. Kanban board

II. Uninformed Search

A. Beacon Agent

A map with 1 Marine and 1 Beacon. Rewards are earned by moving the marine to the beacon. Whenever the Marine earns a reward for reaching the Beacon, the Beacon is teleported to a random location (at least 5 units away from Marine).

Initial State

1 Marine at random location (unselected) 1 Beacon at random location (at least 4 units away from Marine)

Rewards

Marine reaches Beacon: +1

End Condition

Time elapsed

Time Limit

120 seconds

B. Mesh

Map is shown in a 64x64 array fashion, this is equivalent to 4K elements to compare and analyse for any recursive algorithm. This means it will significantly underperform the user expirience. It was done a quantisationon of bigger grids as a result of doing less iteration of the algorithms. For example, if we originaly had a grid of 64X64 , then we could transform it to 8x8 grid.

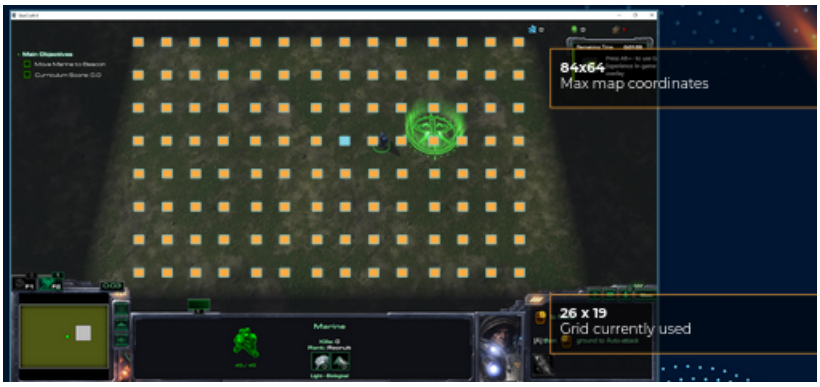


Fig. 3. Grid in the map

C. Iterative Deepening Search (IDS)

Exploration node expansion is limited by the neighborads in an straight forward manner. Only the first exploration check all neightborads. However the idea is to avoid repeating nodes, in this way we could have a faster solution using dynamic programing. In the image bellow, is show the diagram of how expansion is exploring node. Note that mesh help us to reduce work becuase we don't check each pixel of the map, only a subsample steped.

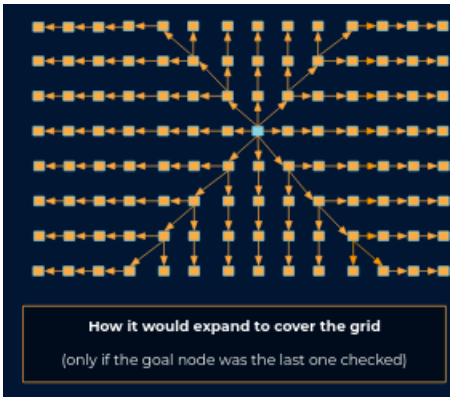


Fig. 4. Mesh exploration

D. Bellman Ford Implicit (BFI)

This Algorithm uses a closed/open logic, to explored the nodes, it is quite similar to A*. The difference is that this method doesn't has an heuristic, we just only used ecludian distance to choose de shortest path. It is a dymanic programing solution, it cuts when the goal is reached.

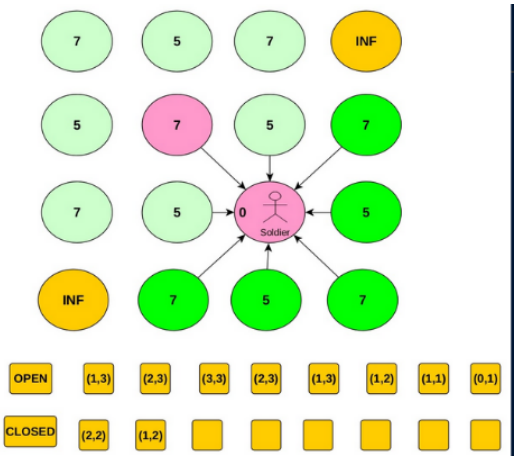


Fig. 5. Mesh Belman ford

E. Conclusion

Both methods were fast in proportion with user expirience. There were not so delay, and the game was totaly functional. In this first stage of the project we had to deal with pysc2. So there was also some time invested in this task. The main idea was to familiriazed with the framework.

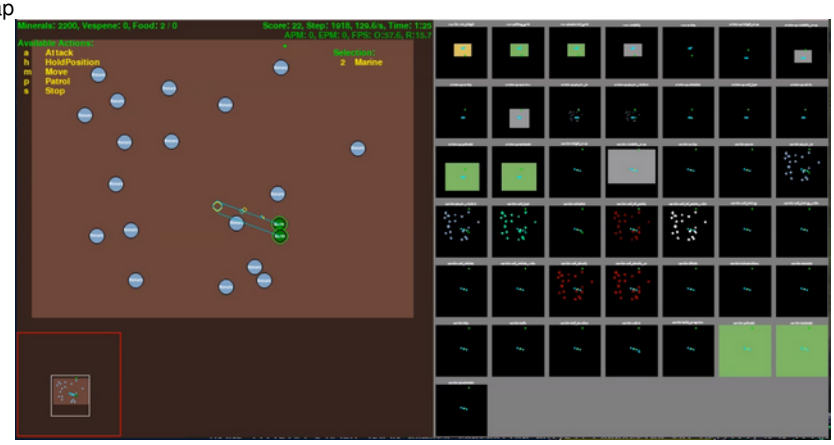
III. Informed Search

A. CollectMineralShards

Description

A map with 2 Marines and an endless supply of Mineral Shards. Rewards are earned by moving the Marines to collect the Mineral Shards, with optimal collection requiring both Marine units to be split up and moved independently. Whenever all 20 Mineral Shards have been collected, a new set of 20 Mineral Shards are spawned at random locations (at least 2 units away from all Marines).

Fig. 6. Collect Minerals Map

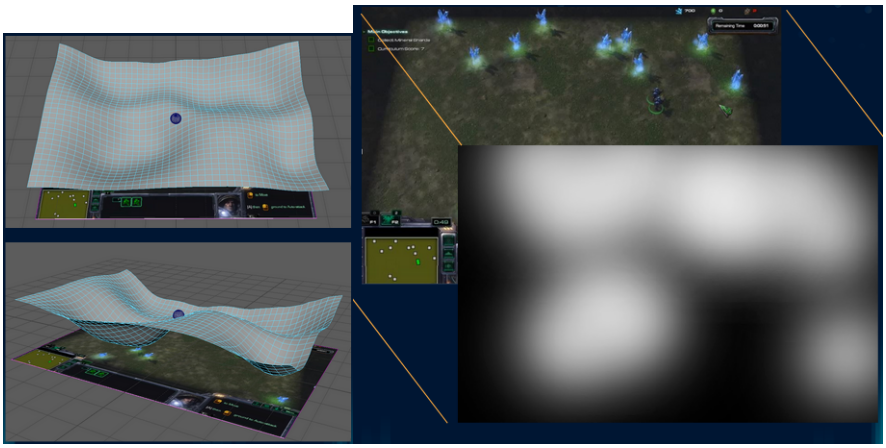


B. Brush

It generates a squared matrix with a Gaussian distribution. We call the output as heightmap. The idea of the heightmap is to generate deep used in the HillClimbing algorithms, as a point on the new mapping is higher, the density of minerals grows too.

- Map - Matrix with the size of our game screen (initialized with 0s)
- We can stamp out our brush
  - Add the values of our brush over a coord in Map
- If we stamp several times we get our map with concentration point

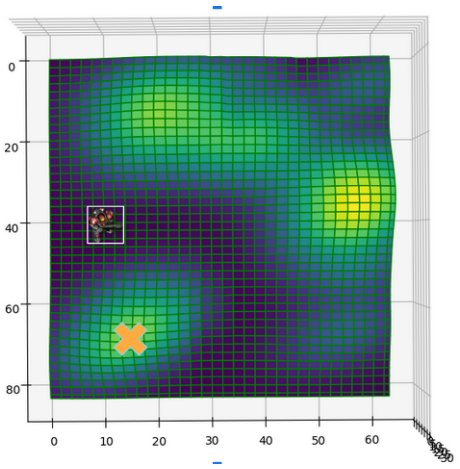
Fig. 7. Height map



C. A*

This version of A* try to find a next first maximum local only using a heuristic based in the value of coordinates of the mesh. The real problem is that needs a lot of calculus when the map is almost empty, because there are a lot of “plain ground” and that make difficult to find the maximum.

Fig. 8. Upper view of heights used in A*



D. HillClimbing and Simulated Annealing

Using the height map, we could procced to find a max height point, as a best option to gather minerals. This is hill climbing algorithm, however a problem is that we could get stuck in a local maxima, so our strategy is to implement a chance given by a probability distribution, in this case we use a temperature boltzmann distribution $e^{\frac{\Delta E}{T}} > rand(0,1)$. The temperature will decrease by an alpha factor inside a loop. For tuning the alpha we did the following equations. $T_0 = inital\,temperature$, $a = decreasing\ factor$, $n = iterations$, $T_f = final\,temperature$

$$T_0 * a^n = Tf$$

$$a = e^{\frac{ln(\frac{T_f}{T_0})}{n}}$$

As shown below there is an application of the simulated annealing, the chance part is designed with the equation above.

Algorithm 1 Simulated Annealing

```
Algo(simulated annealing):
  for Temp=Tmax to Tmin:
    #************CURRENT***********#
    Energy_current = E(Current) # Function cost at given point
    #************NEXT***********#
    N = Next(c) # go for next neighbour
    Energy_Flanders = E(Next) # Function cost at Flanders
    #************DELTA***********#
    Delta_Energy = Energy_Flanders - Energy_current
    #************UPDATE***********#
    if(Delta_Energy >0): # if positive
      Current = Next
    #************CHANCE***********#
    #Delta was negative, lets give another chance and
    #throw a probabilistic shot, maybe we update current
    else if (e^(Delta_Energy/Temp) > rand(0,1)):
      Current = Next
```

E. Conclusion

In this stage it was found that Hill climibing was faster than A*, this could be because the height brush, was helpful to reduce the iterations in the search, get the local optimal point was cheaper than comparing all nodes like A*, maybe A* is more acurate but the trade off is the time perfomance.

IV. Min Max and Probabilistic

A. FindAndDefeatZerglings

1) Description: A map with 3 Marines and an endless supply of stationary Zerglings. Rewards are earned by using the Marines to defeat Zerglings, with the optimal strategy requiring a combination of efficient exploration and combat. Whenever all 25 Zerglings have been defeated, a new set of 25 Zerglings are spawned at random locations (at least 9 units away from all Marines and at least 5 units away from all other Zerglings).

Initial State

- 3 Marines at map center (preselected)
- 2 Zerglings spawned at random locations inside player's vision range (between 7.5 and 9.5 units away from map center and at least 5 units away from all other Zerglings)
- 23 Zerglings spawned at random locations outside player's vision range (at least 10.5 units away from map center and at least 5 units away from all other Zerglings)

Rewards

- Zergling defeated: +1 Marine defeated: -1
- End Conditions Time elapsed All Marines defeated

Time Limit

- 180 seconds

Additional Notes

- Fog of War enabled
- Camera movement required (map is larger than single-screen)

B. Alpha-beta Prunning / Minmax

It was reused the children matrix of the map (Figure 4). But this time it is analysed the cost of each point depending of the not foged view. Estimating the chance of win or lose, it is weigh with 1 win(Zerling death) , 0 tie(Zerling and Marine death) and -1 lose(Marine death). Knowing this information it was built an alpha beta prunning algorithm.

```python
import sys
from params import *


class MinMax:
    def __init__(self):
        self.marines = 3
        self.mapa = None

    def set_mapa(self,mapa):
        self.mapa = mapa


    def minimax(self,depth, alpha, beta, maximizingPlayer,coor):
        hijos = self.mapa.expand(coor)
        if depth == 0 or len(hijos)==0:
            return self.mapa.chanceMatrix[coor[1]][coor[0]], coor

        if maximizingPlayer:
            maxEval = (-1)*infinity
            for child in hijos:
                eval, fromCoor = self.minimax(depth - 1, alpha, beta, False,child)
                maxEval = max(maxEval, eval)
                alpha = max(alpha, eval)
                if beta <= alpha:
                    break
            return maxEval, fromCoor

        else:
            minEval = infinity
            for child in hijos:
                eval, fromCoor = self.minimax(depth - 1, alpha, beta, True,child)
                minEval = min(minEval, eval)
                beta = min(beta, eval)
                if beta <= alpha :
                    break
            return minEval, fromCoor
```

Fig. 9. Alpha beta prunning code

C. Cost function with Heuristic

Given the minimap and the explored ares, there is a shadowed grayscale look of the minimap. The info is provided in the following way : {0: Not seen before, 1: Seen before, but not visible, 2: Visible}. With this 3 states, it was calculated the enemies in state 1 and 2. Knowing this is calculated a density of enemies in relation with each shaded area, in this way it is given weights to each array of cordenates in state 1 and 2. Finally the distance of unexplored areas is taken and added to heuristic in an euclidan way.

Fig. 10. Heristic shaded map

D. Conclusion

In this particular case, alpha beta prunning didn't work because it is a real time game, so the response of the system is not quite acurate, system responds very slow to atacks, and just in some seconds all maries were death. In the other hand the heuristic one, perfoms better than the other, but its main problem was that the marines kill themselves when they almost explored the map. This could be because the set of points of the unexplored area where to close, and they atack themselves.

V. Bayesian Networks

A. Map Description

This map is consist in a map medium map size with four camps that contains minerals and vespene gas. This map is for two players that try to destroy each other. They are two agents against each other.

Initial State

- 1 command center
- 12 SCV

Rewards

- Win and keep your live

End Conditions

- Destroy the enemy

Time Limit

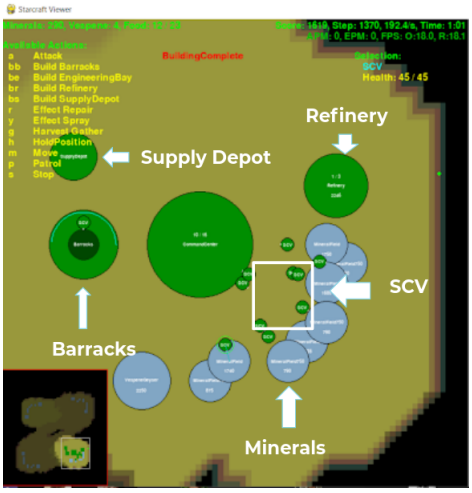
- No

Additional Notes

- We manually activated fog of war And visualize features

B. Map and characters overview

This map is a little more complex than others, because it could have more types of characters. SCV, are used to gather minerals, with this minerals it could be build supply depots, then this ones are used to build barracks, with the barracks the marines are built, and finally with marines attack is possible.



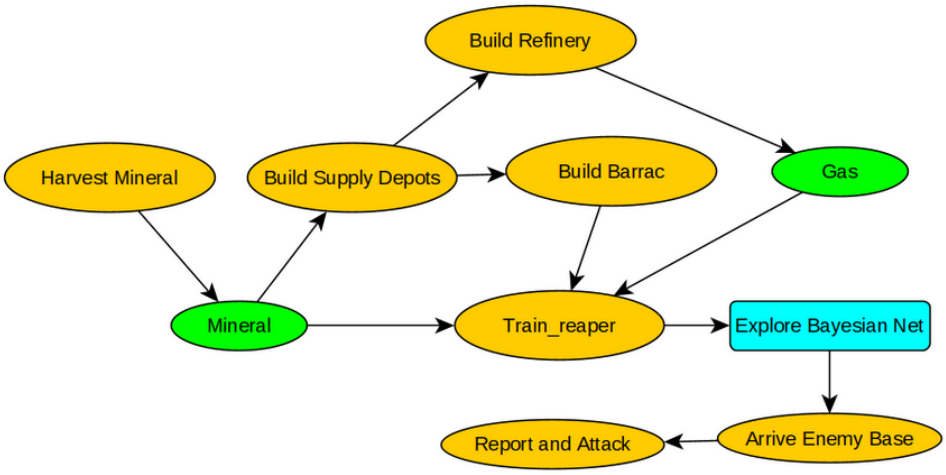


Fig. 11. Map overview simple 64

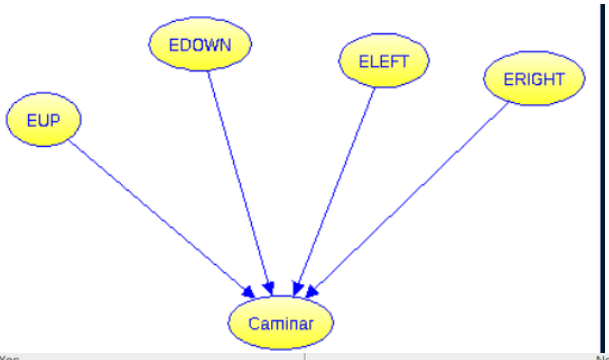
The main idea is to generate more marines as possible, as soon as one of them explore the map and then the others atack the enemy base. However the reaper is an ideal character to explore because could fly and skip mountains. So this character can avoid go around the mountains and hills, so he do a faster work.

Fig. 12. Characters in simple 64

C. Bayesian newtork scouting

The pourpose of the bayesian network is to control the scouting, in others words, explore and find enemy base. This is a strategy that comes from micromanagment as a branch of RST AI workflow. Based on the manhatan distance a scout character will take a desition in only 4 possible ways. To do things simpler this task uses a reaper , that can skip mountains and hills.

Firstly, we emulate our bayesian network in a GUI (**unbbayes-4.22.18**). Secondly we pass our net to the **Pgmpy** library that is in python. Both softwares allow you to do belief propagation depending on the inputs given to the bayesian net. Finally it was done the integration with the simple64 commands and the **Pgmpy**.



ERIGHT	Yes								No							
ELEFT	Yes				No				Yes				No			
EDOWN	Yes		No		Yes		No		Yes		No		Yes		No	
EUP	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No
ERIGHT	0.25	0	0	0	0	0	1	0	1	0.5	0	0.33	0	0	0.33	0.25
ELEFT	0.25	0	0	0	1	0.5	0	0.33	0	0	0.5	0	0	0.5	0.34	0.25
EDOWN	0.25	0	1	0.5	0	0	0	0.33	0	0	0.5	0.33	0.5	0	0.33	0.25
EUP	0.25	1	0	0.5	0	0.5	0	0.34	0	0.5	0	0.34	0.5	0.5	0	0.25

Fig. 13. Scouting bayesian network dag and its probalities

```
class Explorer():
    def __init__(self):
        self.G = BayesianModel([('exploredUp', 'walk'),
                                ('exploredDown', 'walk'),
                                ('exploredLeft', 'walk'),
                                ('exploredRight', 'walk')])



        self.walk_cpd   = TabularCPD(variable = 'walk',
                                     variable_card = 4,
        #explored Right     #                  y                |                  n
        #explored Left   #         y                  n          |         y                n
        #explored Down   #     y         n     |    y         n   |    y         n     |    y         n
        #explored Up  #     y    n     y    n     y    n     y    n    y    n     y    n    y    n     y    n
                values = [[0.25, 0.33, 0.33, 0.5, 0.33, 0.5, 0,    1,   0,    0,   0.5,  0, 0.5,  0.5,  0, 0.25], #walk Right
                          [0.25, 0.33, 0.33, 0.5, 0,    0,   0.33, 0,   0.33, 0.5, 0,    1, 0.5,  0,    0, 0.25], #walk Left
                          [0.25, 0.34, 0,    0,   0.33, 0.5, 0.33, 0,   0.33, 0.5, 0,    0, 0,    0.5,  0, 0.25], #walk Down
                          [0.25, 0,    0.34, 0,   0.34, 0,   0.34, 0,   0.34, 0,   0.5,  0, 0,    0,    1, 0.25]],#walk Up
                        evidence=['exploredRight', 'exploredLeft', 'exploredDown', 'exploredUp'],
                        evidence_card=[2, 2, 2, 2])
        self.eUp_cpd    = TabularCPD(variable = 'exploredUp',
                                     variable_card = 2,
                                     values = [[0.5], [0.5]])
        self.eDown_cpd  = TabularCPD(variable = 'exploredDown',
                                     variable_card = 2,
                                     values = [[0.5], [0.5]])
        self.eLeft_cpd  = TabularCPD(variable = 'exploredLeft',
                                     variable_card = 2,
                                     values = [[0.5], [0.5]])
        self.eRight_cpd = TabularCPD(variable = 'exploredRight',
                                     variable_card = 2,
                                     values = [[0.5], [0.5]])

        self.G.add_cpds(self.eUp_cpd,
                        self.eDown_cpd,
                        self.eLeft_cpd,
                        self.eRight_cpd,
                        self.walk_cpd)
```

Pgmpy library

Fig. 14. Pgmpy python code

D. Reduce overfiting

Map is represented as a minimap in the pysc2 framework, we use a feature of the minimap called pathable. That is the map that characters could used to move around, dark areas are 0 and 1 white ones, white areas are where characters is allow to walk (pathable). However data is presented in a fashion of 64x64 array, so to reduce the calculations it was a subsampled data with a grid, that reduces data size, enconding data in a shorter array of 8X8. This allow us to do less belief prograpagtions in the bayesian network.

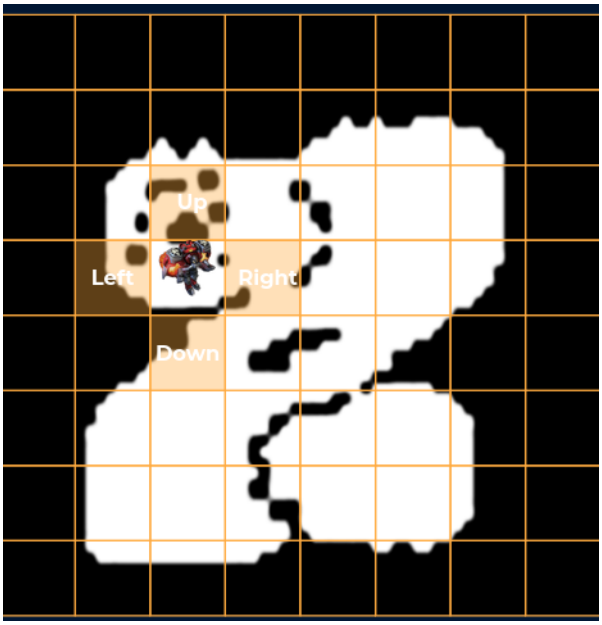


Fig. 15. Grid bayesian network

E. Results

After grid processing and bayesian network integration, the scouter finally reach the desired area.

Fig. 16. Bayesian working

F. Conlusions

Designing Bayesian Networks can be a hard job. Assigning the probabilities is not so intuitive for humans and training is required to help the Bayesian Network to work optimally. Bayesian Networks could be mixed with other strategies, like A* and Alpha Beta pruning, but before you may have a good software planing to do.

VI. Reinforcement learning and pytorch

A. Map Description

A map with 9 Marines on the opposite side from a group of 6 Zerglings and 4 Banelings. Rewards are earned by using the Marines to defeat Zerglings and Banelings. Whenever all Zerglings and Banelings have been defeated, a new group of 6 Zerglings and 4 Banelings is spawned and the player is awarded 4 additional Marines at full health, with all other surviving Marines retaining their existing health (no restore). Whenever new units are spawned, all unit positions are reset to opposite sides of the map.

Initial State

- 9 Marines in a vertical line at a random side of the map (preselected)
- 6 Zerglings and 4 Banelings in a group at the opposite side of the map from the Marines

Rewards

- Zergling defeated: +5
- Baneling defeated: +5
- Marine defeated: -1

End Conditions

- Time elapsed
- All Marines defeated

Time Limit

- 120 seconds

Additional Notes

- Fog of War disabled
- No camera movement required (single-screen)
- This map and DefeatRoaches are currently the only maps in the set that can include an automatic, mid-episode state change for player-controlled units. The Marine units are automatically moved back to a neutral position (at a random side of the map opposite the Roaches) when new units are spawned, which occurs whenever the current set of Zerglings and Banelings is defeated. This is done in order to guarantee that new units do not spawn within combat range of one another.

Fig. 17. Bannelings and zerlings

B. Defining Net

For the net is used pytorch as the main framework. During the game we will train a neuronal net of 3 layers. Input with 6 Neurons, hidden with 64 and ouput with 2. The input neruron came from the state that RAW data of minimap gives. The input layer is defined as follows : {len (self.marines), marines_hp, len (self.zergling), zergling_hp, len (self.baneling), baneling_hp]}, and output layer is : {0 (zerling) and 1(banneling)}, this ones are two type of enemies that has different attack properties.

```python
class LinearDeepQNetwork(nn.Module):
    def __init__(self, lr, n_actions, input_dims):
        super(LinearDeepQNetwork,self).__init__()

        self.fc1 = nn.Linear(input_dims, 64)
        self.fc2 = nn.Linear(64, n_actions)

        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.loss = nn.MSELoss()
        self.device = 'cpu'

        self.to(self.device)
    def forward(self,state):
        layer1 = F.relu(self.fc1(state))
        actions = self.fc2(layer1)

        return actions
```

Fig. 18. Python NN code

C. Defining EPOCHs

Usally the marines will be defeat because zerlings and bannelings respawn, the idea is to survive such as make the high score. Once marines are defeat the system get the score and pass it to the net in order to correct the weights and do better decitions. In the image below it is shown, how the net is query to atack certain enemies, note that there is a time increment in the atacking cycle, this ones one provides sufficent respond time to the control system.

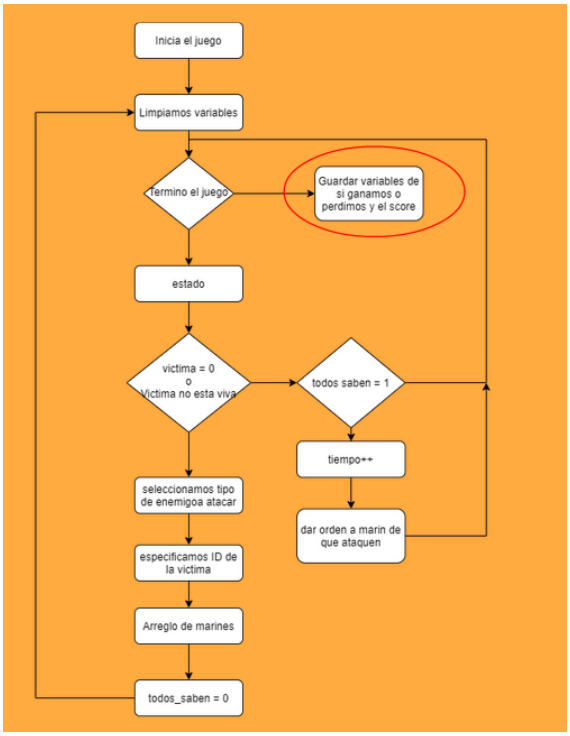


Fig. 19. Epochs cycling

D. Results and Runs

The more epochs have passed, the better the score is. It was tried with different learning rates. And it was found that lr=.05 was the smoother.

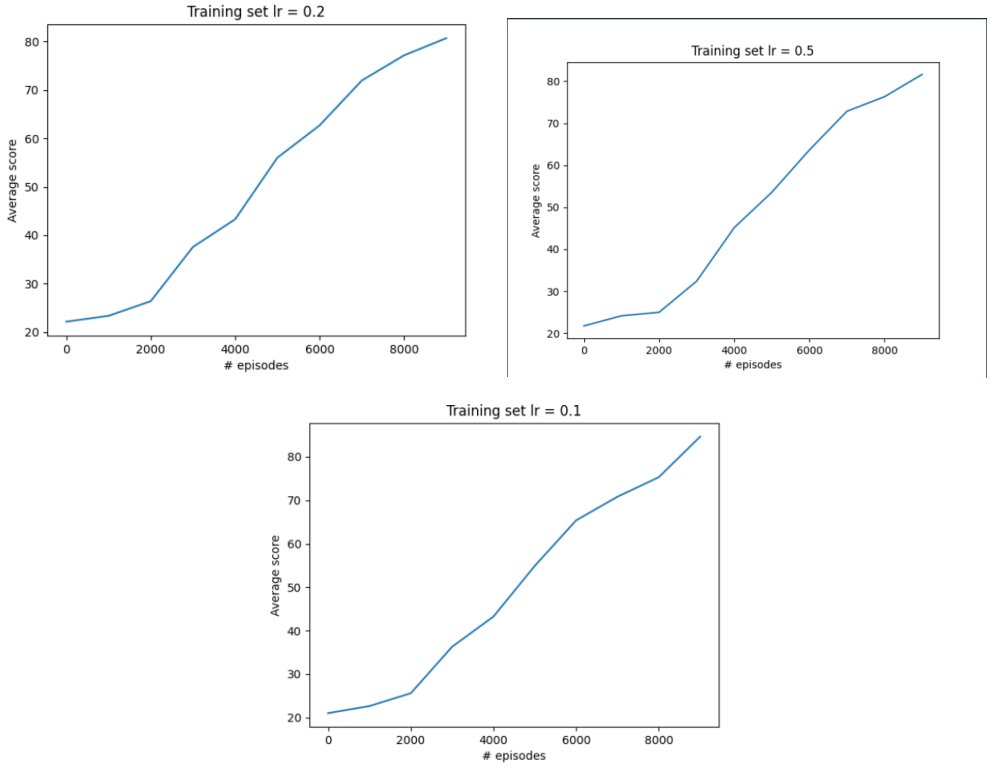


Fig. 20. Learning rate curves

E. Conclusions

The pytorch framework seems to be friendly, as you don't have to worry about some details and is also customizable. The implementation of the network works correctly, and the target for the next release is to integrate this one in the simple64

VII. Final release

A. Map description

B. Strategy and implementation

C. Results

D. Conclusions

VIII. Conclusions

...

ACKNOWLEGMENTS

We thank Dr. Andres Mendez Vazquez for being our mentor for the full length of the development of this project. We also thank M.Sc. Luis Mario Ramírez Solis for helping us visualize the framework to use at the beginning of this project.

REFERENCES

[1] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. KÃŒttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "Starcraft ii: A new challenge for reinforcement learning," 2017.

AUTHORS

Carlos Cardenas-Ruiz is a Computer Systems Engineer. He has worked as a software developer and currently is pursuing his M.Sc. at Cinvestav Guadalajara.

Emilio Tonix-Gleason is a Computer Systems Engineer, He has worked as a software developer and will soon be pursuing his M.Sc. at Cinvestav Guadalajara.

Julia Rodriguez-Abud has a Digital Arts major. She has worked as a software developer and currently is pursuing her M.Sc. at Cinvestav Guadalajara.