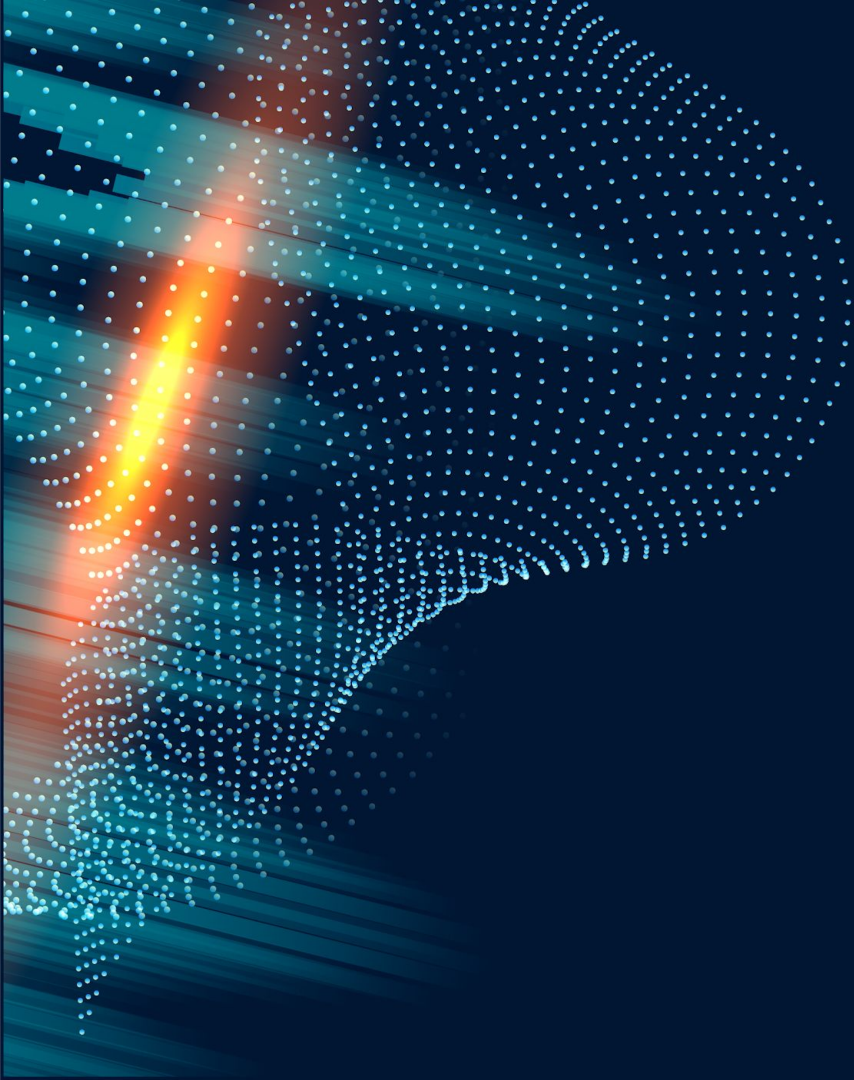
STARCRAFT II

Artificial Intelligence project

Cinvestav Guadalajara

Carlos Cardenas
Emilio Tonix
Julia Abud



Part 3

Alpha-beta Prunning / Minmax
Cost function with Heuristic

TASKS

13 February 2021 - 26 February 2021

- ✓ Implement
    - ✓ Alpha-beta Prunning / Minmax
    - ✓ Cost function with Heuristic
    - ✓ More complicated scenario
- Decisions
    - Data gathering (in progress)

Map: FindAndDefeatZerglings

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

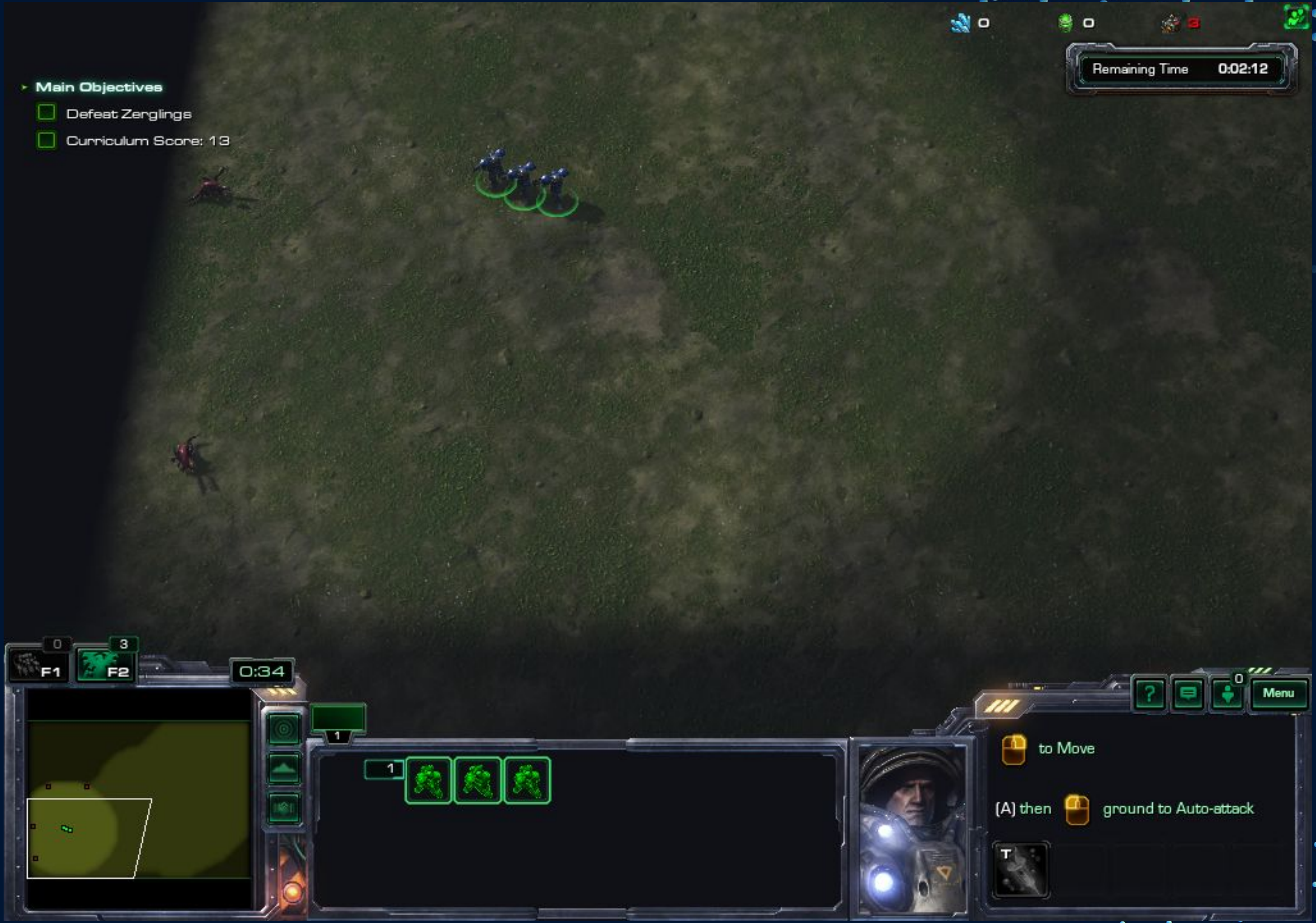
- Time elapsed
- All Marines defeated

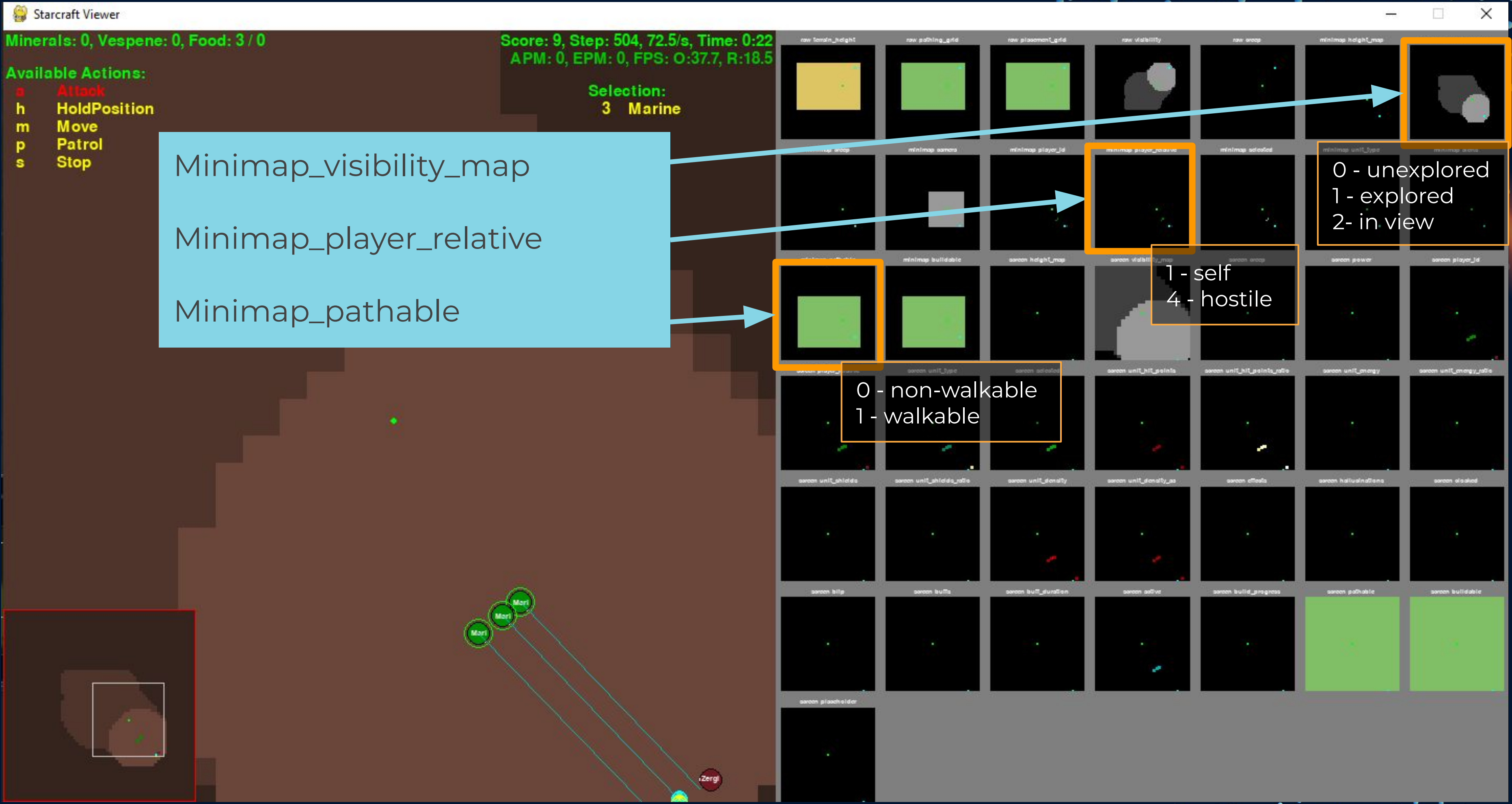
Time Limit

- 180 seconds

Additional Notes

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


Main Objectives
Defeat Zerglings
Curriculum Score: 13
0
0
3
Remaining Time 0:02:12
0
3
F1
F2
0:34
1
1
Menu
to Move
(A) then ground to Auto-attack
T



Minerals: 0, Vespene: 0, Food: 3 / 0

Score: 9, Step: 504, 72.5/s, Time: 0:22
APM: 0, EPM: 0, FPS: O:37.7, R:18.5

Available Actions:
a Attack
h HoldPosition
m Move
p Patrol
s Stop

Selection:
3 Marine

Minimap_visibility_map

Minimap_player_relative

Minimap_pathable

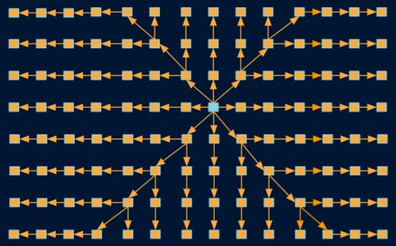
0 - unexplored
1 - explored
2- in view

1 - self
4 - hostile

0 - non-walkable
1 - walkable

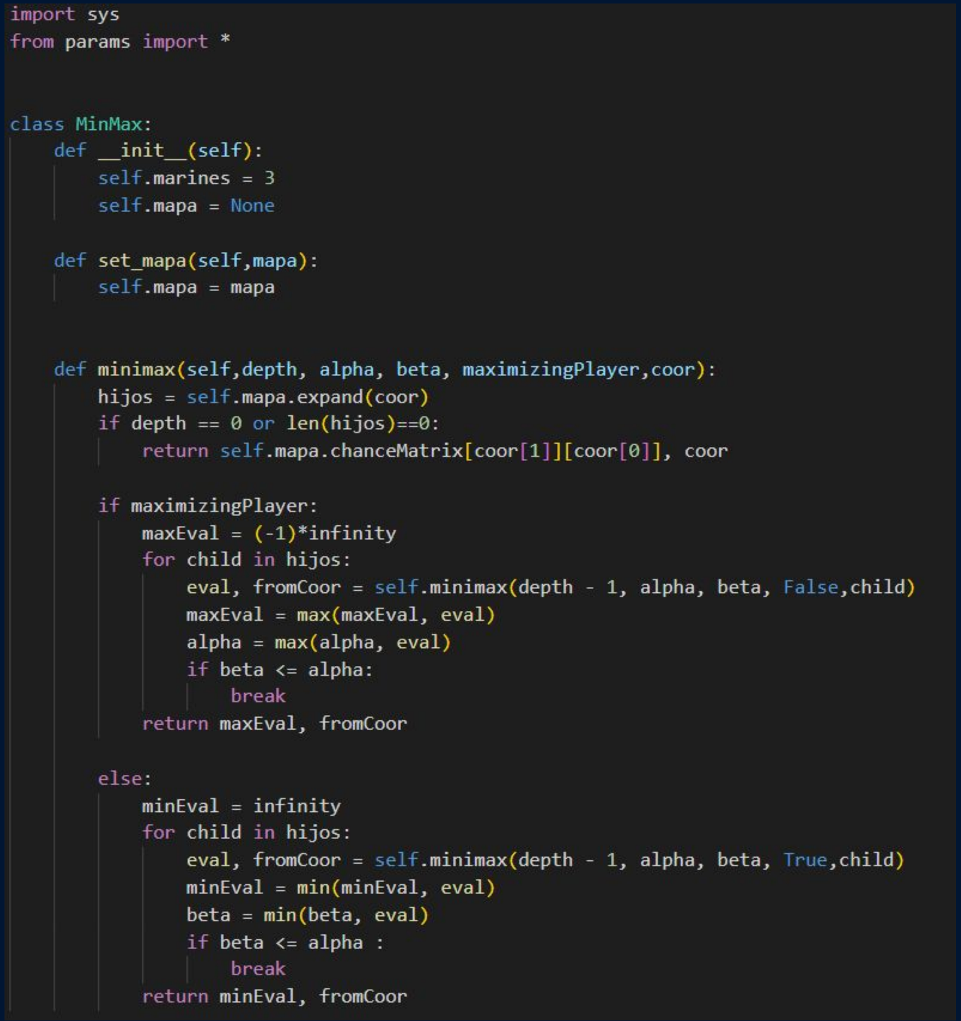
Minmax

Reuse of children matrix of the map



Minmax algorithm with intention of being used to find areas to explore with good chance of having Zergs

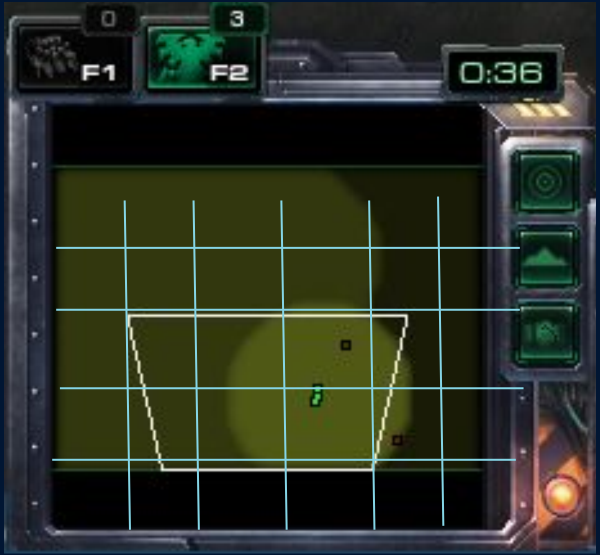
Note: we couldn't wrap our head around the turn based of the minmax

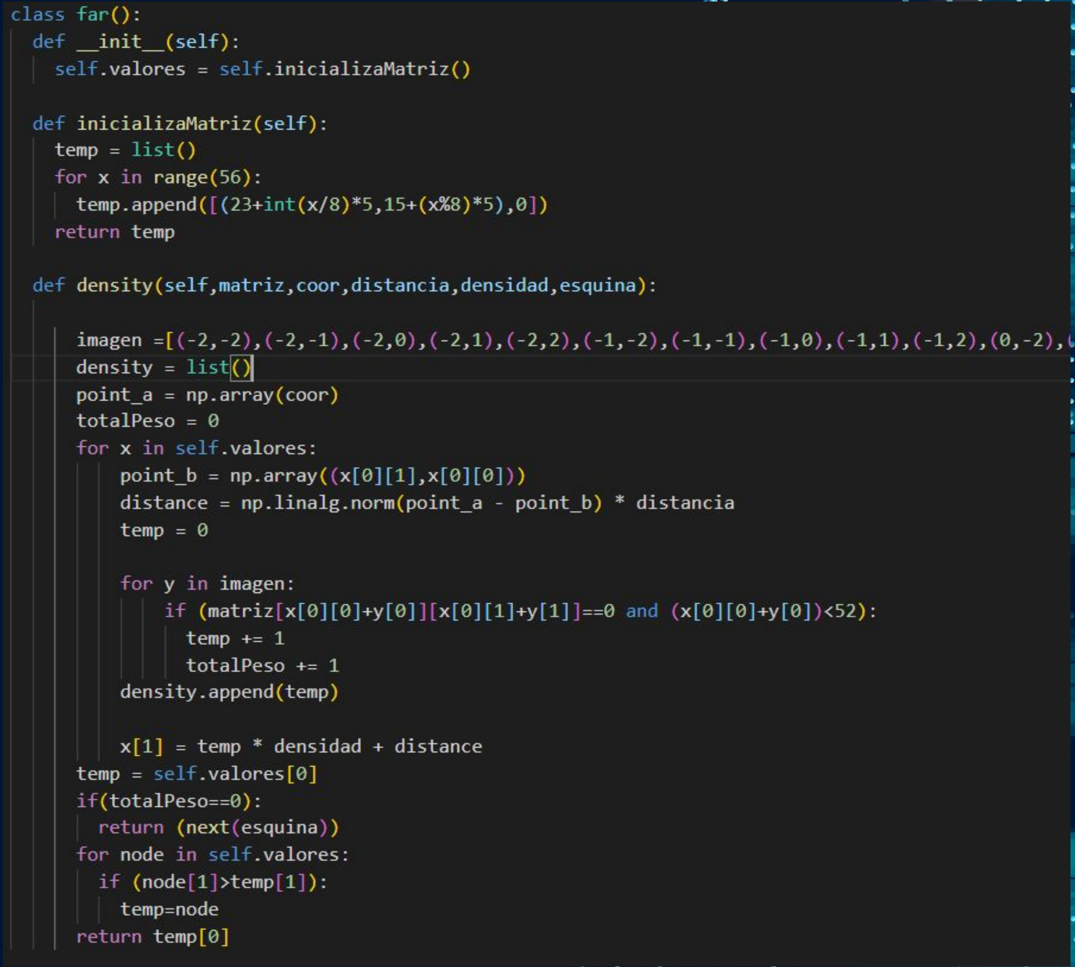


```
import sys
from params import *


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
```

Cost function with Heuristic

Distance
Unexplored areas





```
class far():
    def __init__(self):
        self.valores = self.inicializaMatriz()

    def inicializaMatriz(self):
        temp = list()
        for x in range(56):
            temp.append([(23+int(x/8)*5,15+(x%8)*5),0])
        return temp

    def density(self,matriz,coor,distancia,densidad,esquina):

            imagen =[(-2,-2),(-2,-1),(-2,0),(-2,1),(-2,2),(-1,-2),(-1,-1),(-1,0),(-1,1),(-1,2),(0,-2),(
            density = list()
            point_a = np.array(coor)
            totalPeso = 0
            for x in self.valores:
                    point_b = np.array((x[0][1],x[0][0]))
                    distance = np.linalg.norm(point_a - point_b) * distancia
                    temp = 0

                    for y in imagen:
                            if (matriz[x[0][0]+y[0]][x[0][1]+y[1]]==0 and (x[0][0]+y[0])<52):
                                temp += 1
                                totalPeso += 1
                    density.append(temp)

                    x[1] = temp * densidad + distance
            temp = self.valores[0]
            if(totalPeso==0):
                return (next(esquina))
            for node in self.valores:
                if (node[1]>temp[1]):
                    temp=node
            return temp[0]
```