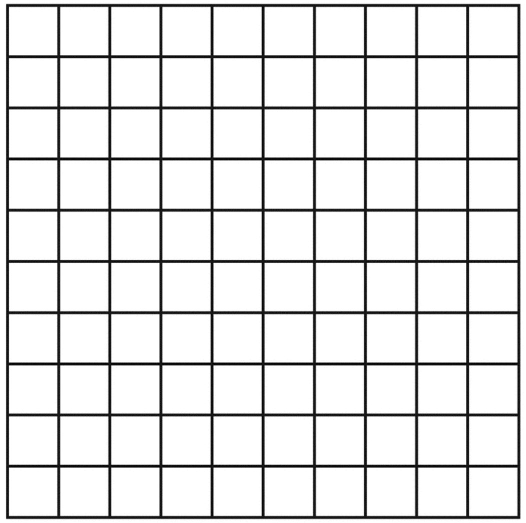
Coursework

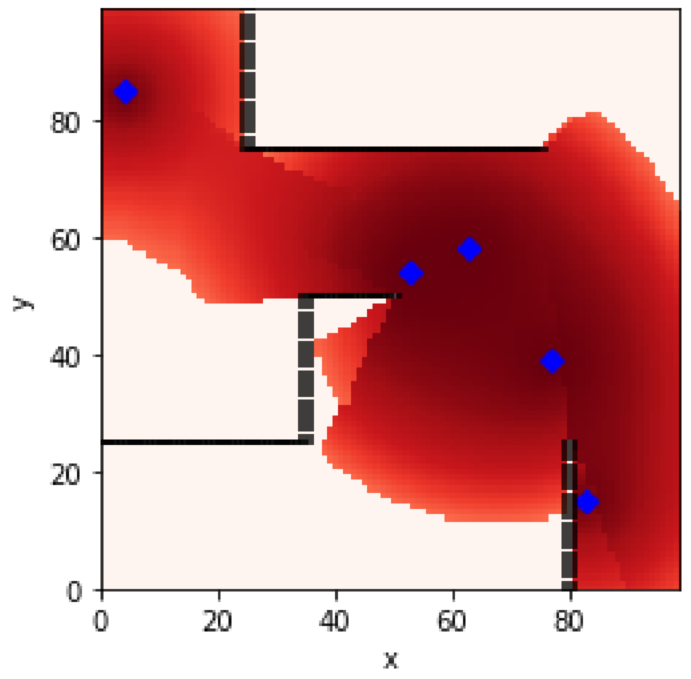
Museum Protection Problem

Museum Protection problem!



- A gallery is defined by a square grid of cells – each cell can hold a single camera.
- Each cells might also contain a wall
- **Goal: place the cameras so that the maximum number of cells are covered**
- You are given 3 different ‘instances’:
  - Each one has a floor layout. Cameras can’t see through or around walls!
  - You must place exactly the number of cameras specified – not more, not fewer!
  - Cameras can’t sit on walls!
  - Only one camera per cell

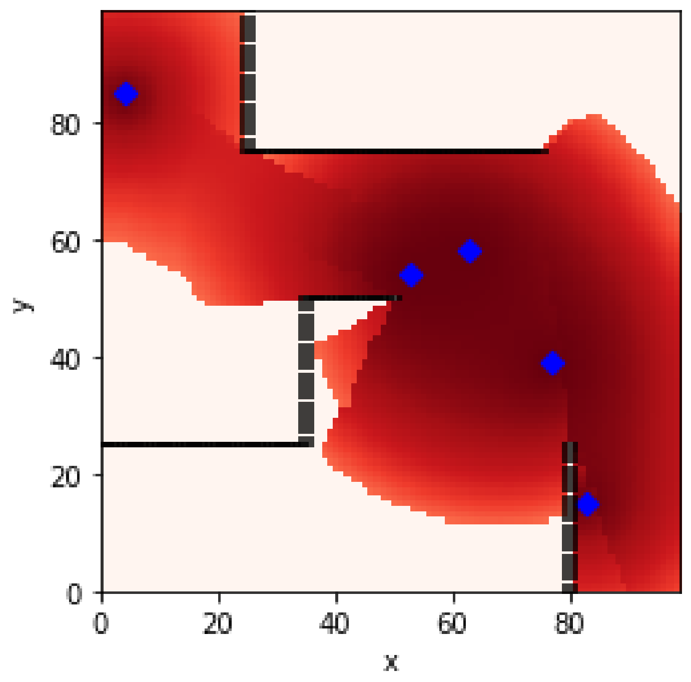
Musuem Protection problem!

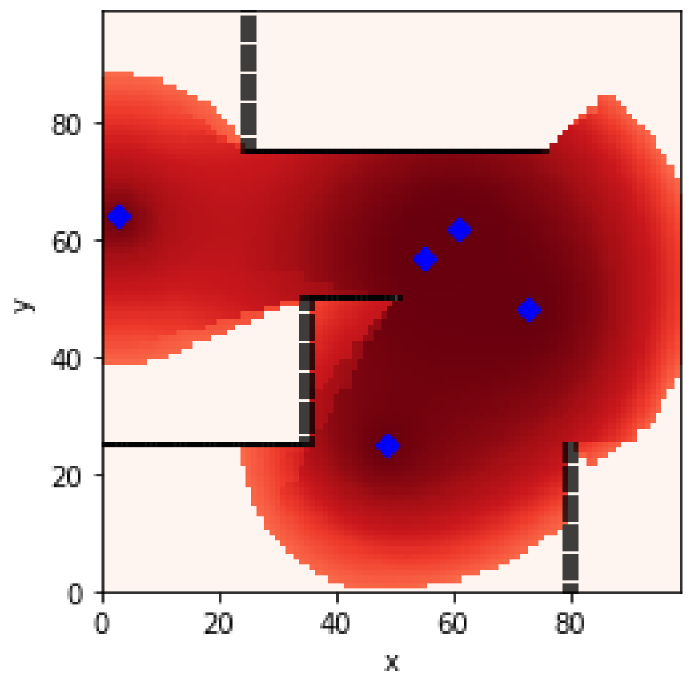


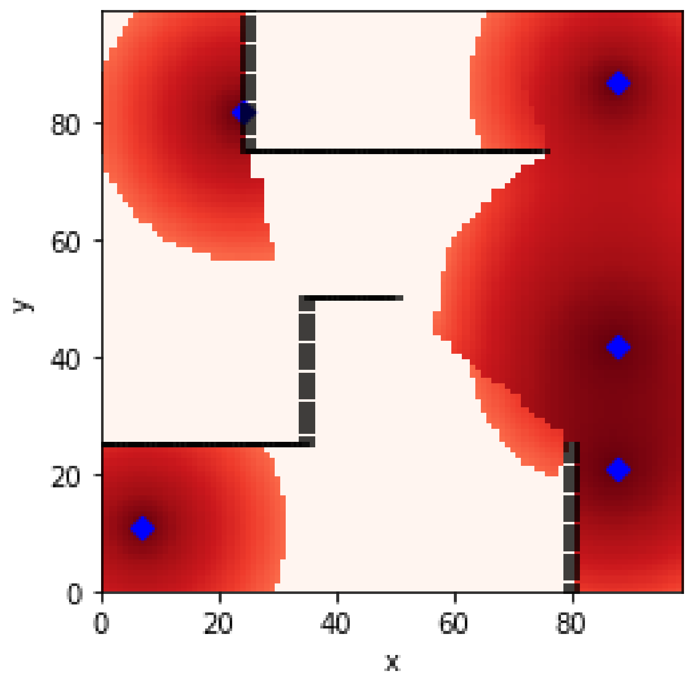
A solution

- Goal: place the cameras so that the maximum number of cells are covered
- You are given 3 different 'instances':
  - Each one has a floor layout. Cameras can't see through or around walls!
  - You must place exactly the number of cameras specified – not more, not fewer!
  - Cameras can't sit on walls!

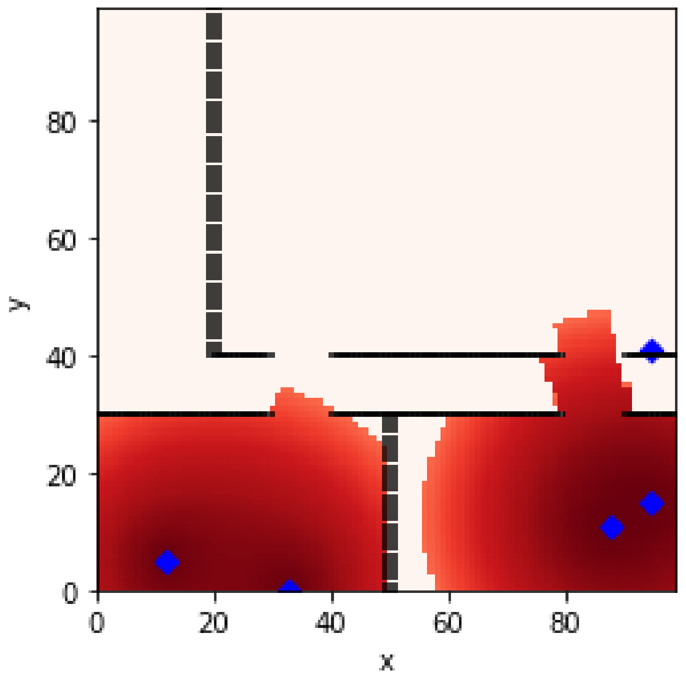
Example solutions to the same instance

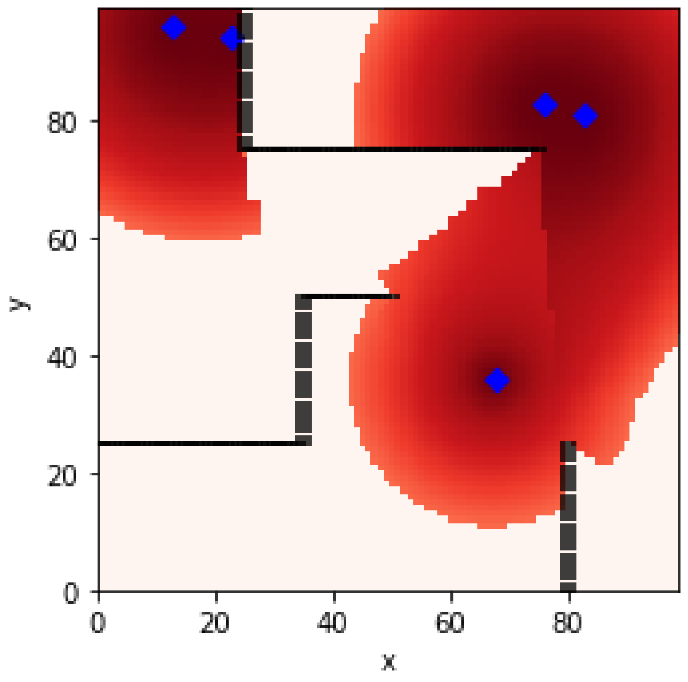


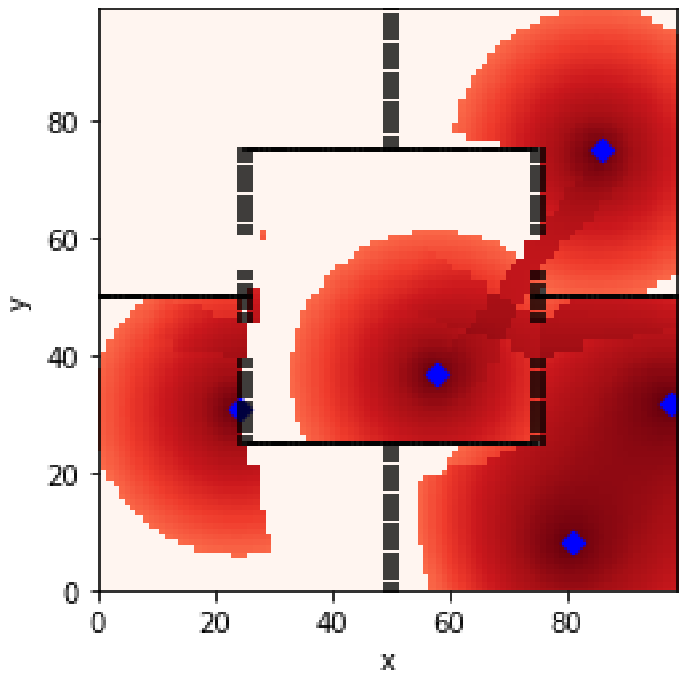




Instances have different wall layouts







Each instance have a size specified by a single number - e.g. Size =100 defines a space 100x100
Each instances has a specified number of cameras
Each instance has a different a layout specified in a csv file

What do you have to do?

- Write a **stochastic search algorithm** that can solve any instance given to it
- You can use any of the classes of algorithm from the module or the literature but they must fall into the category of stochastic search methods:
  - Hill Climber
  - Simulated Annealing
  - Evolutionary Algorithm
  - Etc.
- If you want to use something from the literature not covered in the module, please check with Emma or Sarah first

What to do:

- You have to provide the results from your ‘best’ algorithm on the set of instances given – and also some results on a set of ‘unseen’ instances
- **Most of the marks are NOT for these results – but instead, are for what you did to determine the best approach**

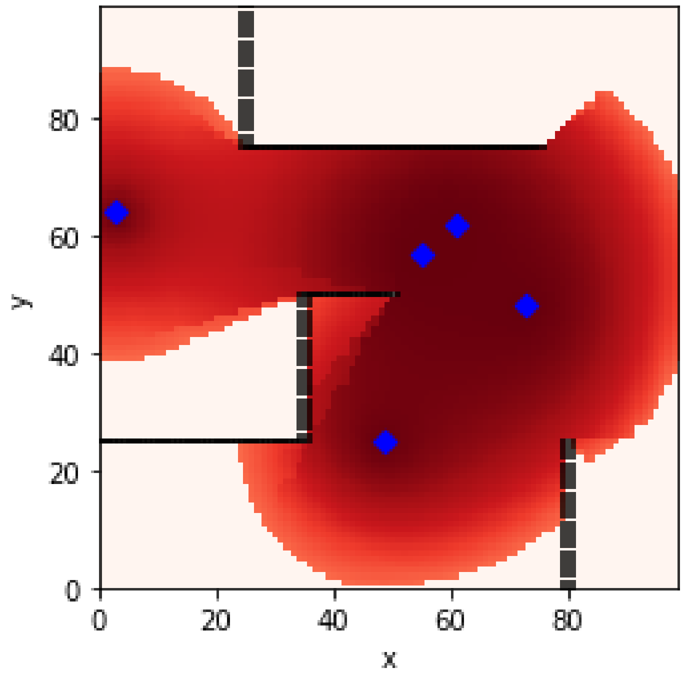
- **Experiment, experiment, experiment!**
  - Parameter tuning
  - Representation
  - Operator design
  - Modifying the evaluation function to deal with constraints
  - Repairing invalid solutions
  - Etc.

Objective function

- A ‘**solution**’ defines the position of each camera
- The quality of a solution is defined as *how much of the space is not covered*
- It is designed as a minimization problem – it returns the number of cells not covered, so the optimal is 0
- The quality of a solution is determined by calling an external executable from your code – you can’t modify this
- You can only call this function with a ***valid*** layout

Valid solutions

- Have exactly the number of cameras specified by the instance
- Don’t place any cameras on a wall
- Don’t have cells with more than one camera
- You **CANNOT** call the objective function with an invalid solution – instead you have to assign a fitness that indicates it is invalid…. or fix it…..**[more on this in week 8]**



So how can this work

- You write an **eval_function()** that is passed an *individual*, just like all the examples you have seen so far
- It needs to check if a solution passed to it is valid – and if so, pass the solution to the external objective_function
- **If it is not valid:**
  - you have to choose how to assign a fitness to it
  - Or you can choose to repair it to make it valid

```
def eval_function(individual):
  if actual_cameras != num_cameras:
          fitness =.....
  elif cameras_on_walls > 0:
          fitness =....
  else
          fitness=
external_objective_function(solution)
```

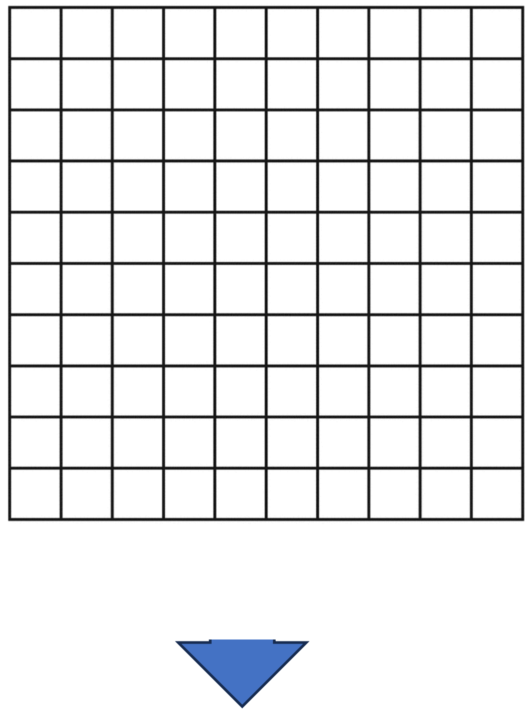
Code provided

Code provided

Code provided

Representing a solution for the external objective function

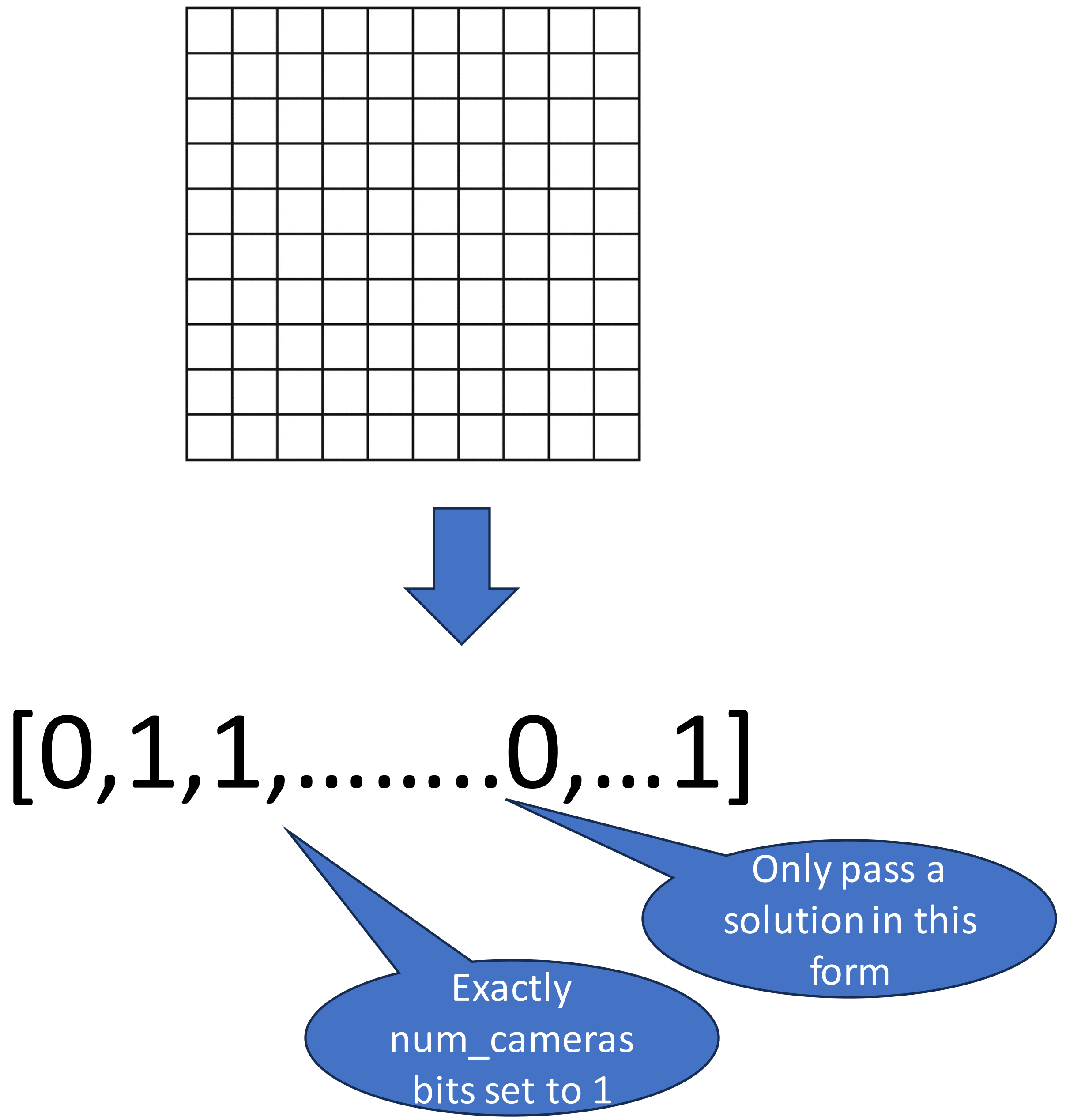
- Think of the solution as placing *n* cameras into the cells of a discretised grid – the solution specifies the locations
- A solution can take many forms: for example we can map this grid to a 1d array and then just evolve a bitstring (like many of the examples in class)
- But there are many other representations and therefore search-spaces possible……



[0,1,1,……..0,…1]

Calling the objective_function with valid solution

- Function MUST be passed a list with n values, where n=number of cells in the grid – regardless of how you choose to represent solutions
- Each value in the list is 1 or 0 indicating whether there is a camera in the cell or not, e.g.


[0,1,1,........0,...1]
Exactly num_cameras bits set to 1
Only pass a solution in this form

IMPORTANT!!

Your EA doen’t need to represent a solution as a bitstring of length n…. your algorithm can search in a different space

BUT – before you send an individual to the external_objective_function, you must convert it into the bitstring format on the previous slide

Marking extremes

Basic pass

- An algorithm that evolves **valid** solutions (but maybe not that good) for any instance
- Some basic tuning (e.g. numerical parameters such as population size)
- Repeated runs, basic statistics (e.g. mean, std)

Excellent mark

- Thoughtful design of algorithm(s) to fit problem (e.g. representation, operator design, constraint handling etc) showing understanding of EC and problem
- Well designed experimental methodology, e.g. comparing algorithms/tunings/operators
- Excellent use of statistics for comparisons, use of appropriate plots etc
- Discussion that shows insight into results (why it did/didn't work)

See the marking rubric for the inbetween states!

Feedback on your plans

- Week 9/10 - please prepare an outline of what you plan to do in note form – I will discuss with everyone individually during the practical and provide direct feedback

Technical stuff

- A notebook is provided which:
  - Provides boiler plate code for reading instance files and setting up variables etc
  - Defines a basic EA that represents a solution as a bitstring (this will produce very poor results as most solutions will be invalid)
  - Examples of how to visualize a solution
- Binaries for linux (Colab), Windows and Mac OSX for calling the objective_function
- Binaries for visualizing solutions (not required to solve the problem but might help with understanding)

Technical Stuff

- You don’t need to run it as a notebook – feel free to run just as a Python program
- If you really want to do it in another language – OK – but it will take more time - check with Emma/Sarah first