Sjoerd van der Heijden
Patrick Schilder
Alix Dodu



# Design Document
# Rush Hour

We gaan ervan uit dat een auto bewegen telt als een stap, hoe groot die stap ook is.

```
class Car():
	def __init__((x,y),(x,y),(x,y)):
		self.position

		 # code voor direction uitrekenen
		self.direction

	def getPos():
		return position as list of tuples
	def getDirection():
		return “h” or “v”
	def getLength():
		return int

	def moveCar(int):
		# int is pos voor een richting en neg voor andere
		changes self.position

	isAt((x,y)):
		returns True als de auto daar zit

class RedCar(Car):
	def moveCar(int):
		Car.moveCar(int)
		if pos at exit:
			return TADAAA

class Parking():
	def __init__(length, width, (x_exit, y_exit), cars):
		# (x_exit, y_exit) is het vakje voor de uitgang
		# cars is de lijst met auto's, als Car objects

		self.length
		self.width

		# code om een Array/Dictionnary representation te maken met alle auto's erin. Het is waarschijnlijk handig om dit te maken, omdat het programma dan snel kan checken of ergens een auto staat. Dit is sneller dan de hele lijst met auto's af te gaan om te checken of een van de auto's op de plek staat.
		# Hier wordt ook iets geschreven om errors te handelen die ontstaan als er geprobeerd wordt auto's buiten het veld te plaatsen.
```

self.parkArray() *of* self.parkDict()
*waarschijnlijk array, met booleans*
*We weten nog niet of dit de beste manier is om de posities van auto's te kunnen checken, en of het handiger is om cijfers te gebruiken die overeenkomen met auto's in plaats van booleans.*
def getParking():
# Maakt representatie van parking, voor debugging of mooie plaatjes.
return array met cijfertjes die overeenkomen met de verschillende auto's

def BruteForceSimulation(Parking,
# recursieve functie die alle opties afgaat, opslaat welke een oplossing zijn, en degene met het kleinste aantal stappen uitkiest.