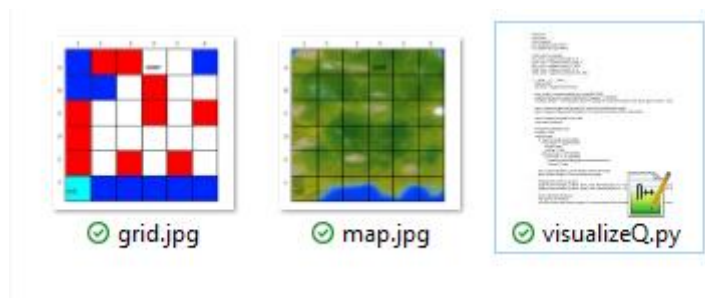


Oppsett av visualiseringskode

I tillegg til dette dokumentet har du fått tre filer: to bakgrunnsbilder og en kodefil:



1. Denne visualiseringen er basert på **pygame**. Installer **pygame** i miljøet til oppgaven. Anaconda: <https://anaconda.org/conda-forge/pygame>
2. Kopier bildene og visualizeQ.py til oppgavemappen din. Husk å bruke «git add -A» slik at de nye filene blir registrert i versjonskontrollen!

Navn	Endringsdato	Type	Størrelse
.git	11.10.2023 19:11	Filmappe	
.vs	13.10.2023 12:16	Filmappe	
__pycache__	13.10.2023 10:37	Filmappe	
DTE-2602 - Karactersatt oppgave 1 - Q-le...	11.10.2023 12:04	Adobe Acrobat D...	448 kB
grid.jpg	11.10.2023 14:38	JPG-fil	56 kB
LICENSE	11.10.2023 12:04	Fil	35 kB
map.jpg	13.10.2023 10:15	JPG-fil	228 kB
qlearning.py	13.10.2023 10:37	PY-fil	5 kB
visualizeQ.py	13.10.2023 12:10	PY-fil	3 kB

3. For at visualiseringen skal fungere må du gjøre noen tilpasninger i Robot-klassen i qlearning.py. Pygame må vite hvor på skjermen den skal tegne agenten, dette løser vi ved å legge til funksjonene get_x() og get_y() som returnerer **kolonnen** og **raden** som roboten står i:

```
class Robot:
    def __init__(self):
        pass

    def get_x(self):
        pass # Return the current column of the robot (0-5).

    def get_y(self):
        pass # Return the current row of the robot (0-5).
```

Lag en metode som heter «one_step_q_learning()» der du gjør én iterasjon av Q-learning-algoritmen (uten while!):

```
def one_step_q_learning(self):
    # Get action based on policy.
    # Get the next state based on the action.
    # Get the reward for going to this state.
    # Update the Q-matrix.
    # Go to the next state.
    pass
```

Lag en metode «has_reached_goal()» der du sjekker om roboten står i måltilstanden:

```
def has_reached_goal(self):  
    """True if the robot is in the goal state."""  
    pass
```

Og til slutt en metode «reset_random()» der du plasserer roboten i en tilfeldig valgt tilstand:

```
def reset_random0(self):  
    """Places the robot in a random state."""  
    pass
```

Disse metodene er allerede i bruk i visualizeQ.py, og må være på plass før du kan kjøre visualiseringen.

4. Pygame bruker en while True-løkke som kontinuerlig tegner opp vinduet så lenge programmet kjører. Vi kan bruke denne for å velge handling og flytte roboten fra tilstand til tilstand helt til den finner målet, i stedet for while-løkken som brukes i vanlig Q-learning. På denne måten synkroniserer vi bevegelsene til roboten med opptegningen. I visualizeQ.py:

```
running = True  
while running:  
    for event in pygame.event.get():  
        if event.type == pygame.QUIT:  
            pygame.quit()  
            running = False  
            break  
        elif event.type == KEYDOWN:  
            if event.key == K_ESCAPE:  
                pygame.event.post(pygame.event.Event(QUIT))  
                running = False  
                break  
  
    play_surface.fill(white_color)  
    play_surface.blit(grid, (0, 0))  
  
    pygame.draw.rect(play_surface, black_color, Rect(robot.get_x() * 70 + 69, robot.get_y() * 70 + 69, 70, 70))  
    pygame.draw.rect(play_surface, green_color, Rect(robot.get_x() * 70 + 70, robot.get_y() * 70 + 70, 70, 70))  
  
    if robot.has_reached_goal():  
        robot.reset_random()  
    else:  
        robot.one_step_q_learning()  
  
    pygame.display.flip()  
    fps_clock.tick(simulator_speed)
```

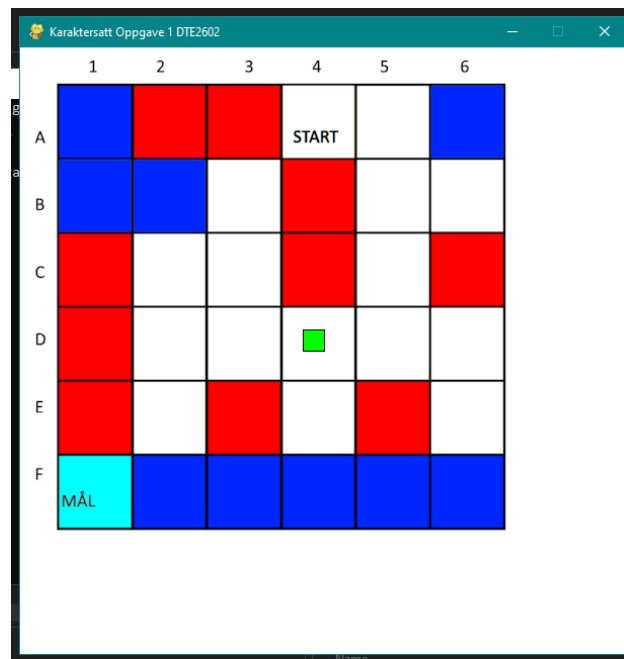
Vi bruker pygame sin while

Tegner basert på x og y

Vi sjekker om roboten har funnet målet og nullstiller den

Hvis roboten ikke har funnet målet kjører vi én iterasjon med Q-learning

- 5.
6. Andre relevante ting er beskrevet med kommentarer i visualizeQ.py. Hvis alt fungerer skal du få opp et vindu når du kjører filen visualizeQ.py der den grønne firkanten er selve roboten:



Til slutt: Visualiseringen er frivillig og ikke en del av oppgaven, og teller derfor ikke mot karakter. Prøv gjerne ut forskjellige ting i visualiseringen og ha det gøy, men ikke gå deg fast og bruk for mye tid på dette!