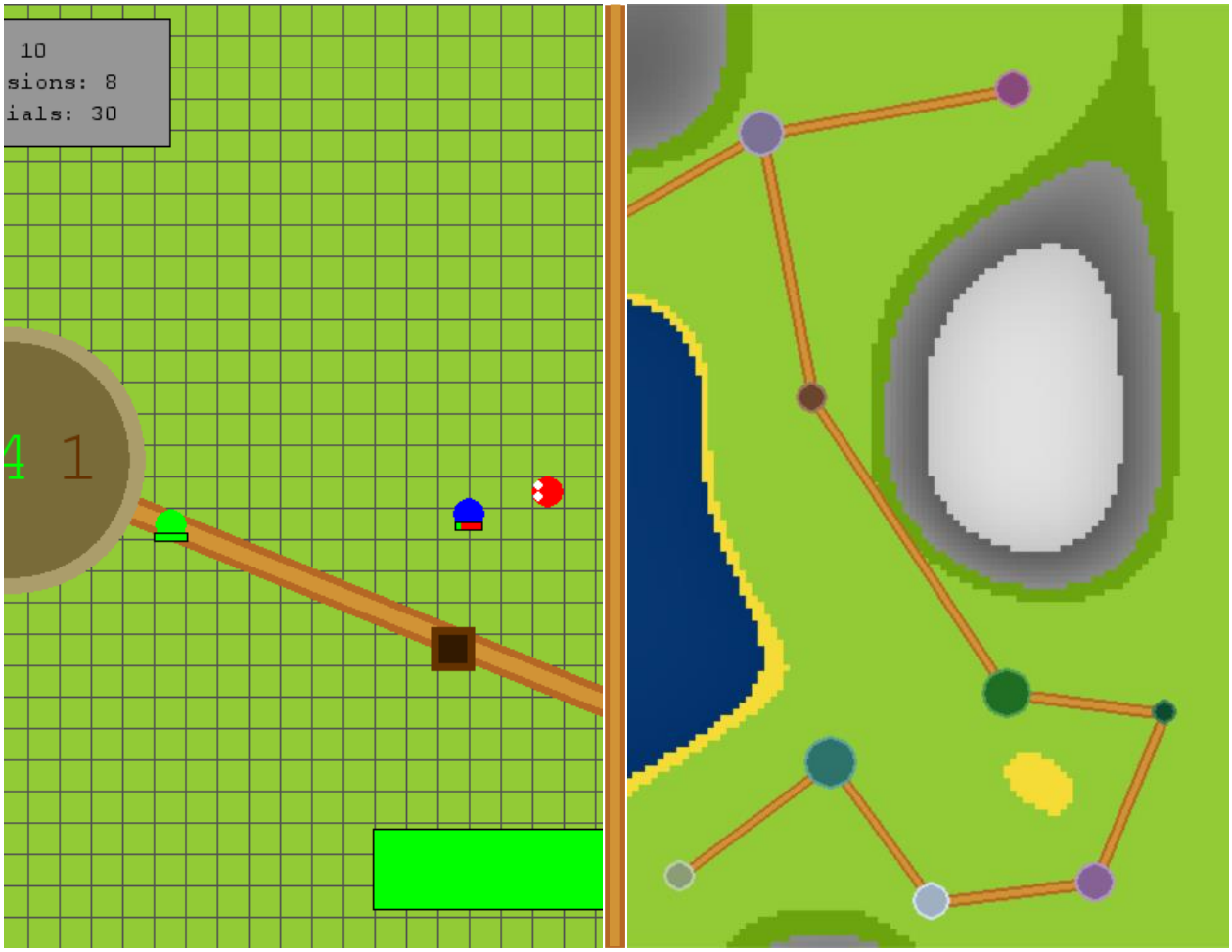
Highwayman

Jonas Mariager Jakobsen



Programmerings synopsis

Projektperiode:
21-3-2019 til 9-5-2019

Indhold

Alle figurer henvist til i afsnittet kan ses nederst i afsnittet hvis andet ikke er angivet.

Indledning

Mit projekt har bestået af at opbygge et spil. Spillet skal selvfølgelig være underholdende, så jeg valgte et lidt alternativt mål for spilleren. Mit spillet går nemlig på at man er en landevejsrøver og for at overleve og samle points er det nødvendigt at plyndre de forbipasserende handelsmænd, som oftest er beskyttet af nogle vagter. Man kan enten angribe dem direkte eller sætte fæller rundt på de veje som de bevæger sig på. Jeg har lavet mit spil sådan at hvert spil er forskelligt. Hver gang man starter et nyt spil, bliver en tilfældig verden genereret med tilfældigt placerede byer. Vejene til gængæld er placeret således at de ikke krydser hverken bjerge eller vand. Samtidig bliver vejen placeret således at det er de korteste mulige.

Programmets opbygning

Programmet består af 8 klasser samt en hovedfil ved navn ”main” som opretter ”Game”-klassen hvorfra resten af klasserne er afhængige. Den eneste undtagelse er ”Pygame_texintput”-klassen som kun bliver brugt i ”main”-filen til at indskrive navnet til den highscore man laver, mere til det senere. Hele sammenhænget kan ses i bilaget under Diagrammer.

Da mit spil indeholder en del klasser og funktioner har jeg udvalgt nogle specifikke at tale om.

Verden generation

Verdenen bliver genereret i ”World_map” klassen og bliver altså derfor refereret til gennem det objekt der bliver oprettet af ”Game”-klasse.

For at generere den tilfældige verden anvendte jeg 2D ”Perlin noise” til at generere sudo-tilfældigt terræn. Måden det bliver genereret på sikrer at der ikke er nogle ”skarpe kanter” altså at værdierne ikke ændrer sig drastisk. Specifik genererer ”perlin noise” biblioteket en 2D liste, altså en liste af lister, af tal mellem -1 og 1. For at sikre at banen ikke var fuldkommen tilfældig hvor der er mulighed for at der ikke kommer nogen formationer som søer (eller floder) eller bjerge. For at sikre at der altid fremkom enten både vand eller bjerge tog jeg brug af ”tangens hyperbolsk” eller ”tanh” som jeg fik at vide om fra en klassekammerat. Funktionen som er vist i Fig. 1 Giver det sammenhæng som er vist i Fig. 3. Funktionen tager variablen ”size_val” i brug til at holde værdierne inden for de ønskede grænser, biblioteket ”Numpy” har en god funktion til dette, nemlig funktionen ”unravel_index” som giver de 2 indlejrede listers indekser for, først, hvor i listen den største, og næst, hvor den mindste værdi er. De 2 linjer i Fig. 3 viser netop denne grænse og udseendet af funktionen ændrer sig ud fra denne værdi, og sikrer dermed variation i den genererede verden.

Hver værdi bliver til et felt i verdenen med specifikke karakteristika. Disse værdier bliver tildelt efter nogle arbitrære grænser som vist i Fig. 4. Den specifikke værdi kan lave en af 6 forskellige felter, også skrevet som kommentarer i Fig. 4. De værdier der bliver tildelt er: Type, farve og bevægelseshastighed på feltet.

Til sidst bliver feltet sat på en ny 2D liste ved navn ”self.tiles” altså den bliver en del af ”World_map” klassen.

Derefter bliver der generet byer og veje. Byerne er placeret tilfældigt bare ikke på andet end ”Grasslands” (Fig. 4). Koden skrevet til generering af vejende er dog baseret på idéen bag et ”minimum spanning tree”, samt at de heller ikke må krydse andet end ”Grasslands” og ”Highlands”. Hovedidéen bag et ”minimum spanning tree” at lave den korteste mulige net af veje, hvilket lykkedes mig (Fig. 5).

Jeg fandt biblioteket der laver perlin noise på StackOverflow fra brugeren ”tgirod” på en af hans opslag.

```
tile = tanh(p[x][y]/(0.90*size_val)) * 0.5 + 0.5
```

Fig. 1 Normaliserende funktion (kode)

```
i,j = np.unravel_index(p.argmin(), p.shape)
min_val = p[i,j]
i,j = np.unravel_index(p.argmax(), p.shape)
max_val = p[i,j]

size_val = max_val
if size_val < abs(min_val):
    size_val = abs(min_val)
```

Fig. 2 Finder numerisk største værdi

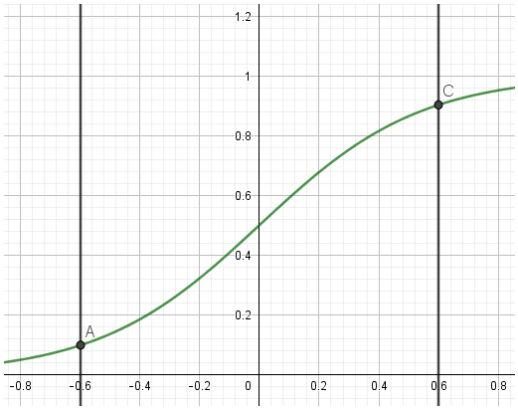


Fig. 3 Normalisering GeoGebra repræsentation

```
for x in range(w):
    self.tiles.append([])
    for y in range(h):
        tile = tanh(p[x][y]/(0.90*size_val)) * 0.5 + 0.5

        if tile < 0.2:
            tile_info = (0, (13 - tile * 54, 61 - tile * 57, 120 - tile * 61), 0.2) #Water
        elif tile < 0.22:
            tile_info = (1, (246, 220, 55), 0.75) #Beach
        elif tile < 0.65:
            tile_info = (2, (146, 203, 54), 1.25) #Grassland
        elif tile < 0.7:
            tile_info = (3, (107, 164, 15), 1) #Highlands
        elif tile < 0.8:
            tile_info = (4, (-45 * (tile-0.7)*10 + 140, -45 * (tile-0.7)*10 + 140, -45 * (tile-0.7)*10 + 140), 0.6) #Mountain
        else:
            tile_info = (5, (55 * (tile-0.8)*5 + 200, 55 * (tile-0.8)*5 + 200, 55 * (tile-0.8)*5 + 200), 0.45) #Mountain_top_snow

        self.tiles[x].append(tile_info)
```

Fig. 4 Tildeling af felt karakteristika

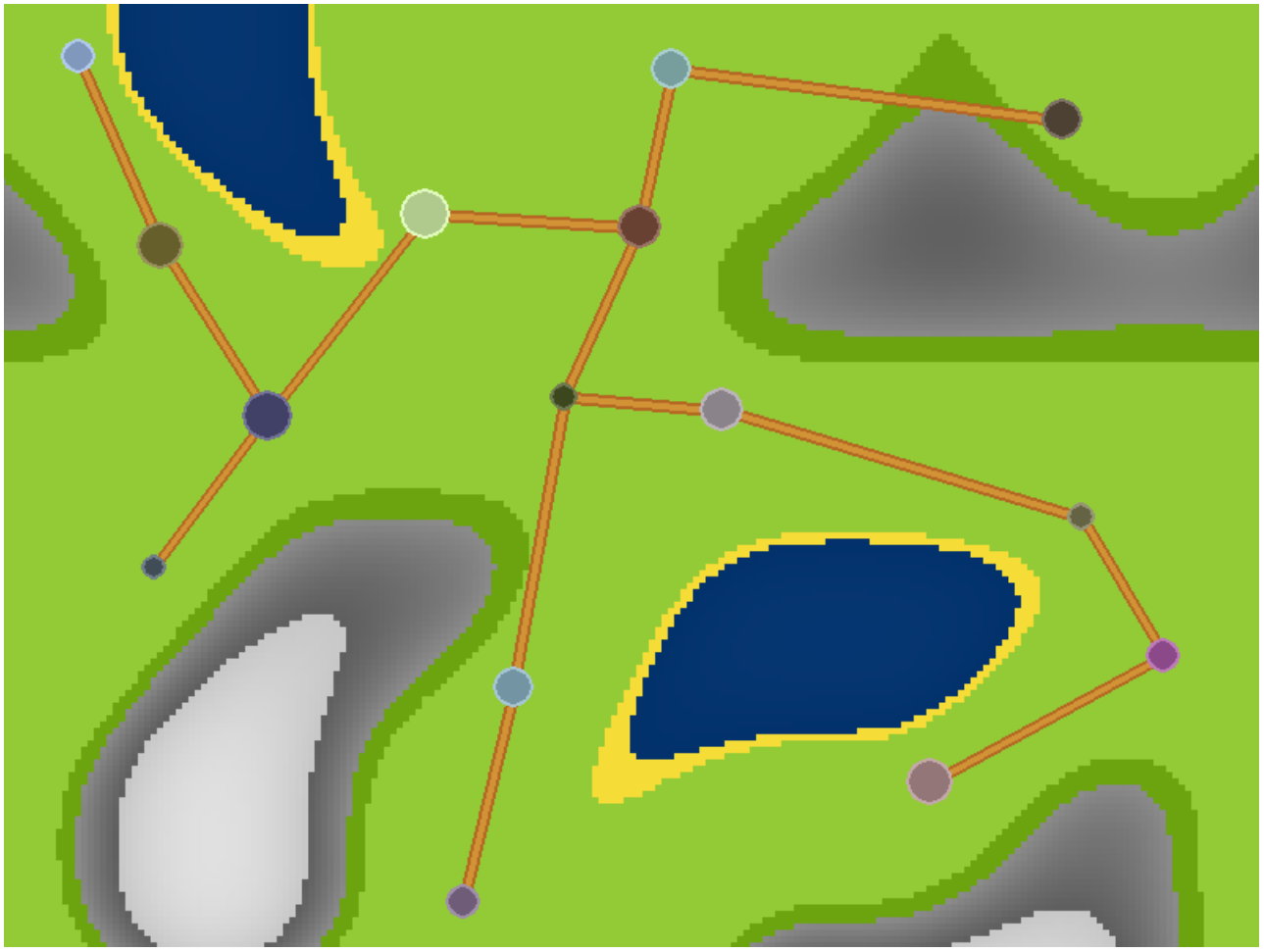


Fig. 5 Alle veje og byer holder sig inde for specifikationer

Eksempler på game-klasse funktionalitet

”Game”-klassen har en funktion ved navn ”tick” denne funktion er til for at håndtere alle spillets funktioner og logikken bag alt den logik der forgår.

Bevægelse

Et eksempel er spillerens bevægelse. Hvert ”tick” bliver op, ned, venstre og højre knappen undersøgt for om der bliver trykket på dem, hvis det er tilfældet, bliver der plusset en enhedsvektor til vektoren p_vel i den retning som man trykket Fig. 6, på den måde hvis man trykker på f.eks. både op og venstre vil p_vel have de endelige koordinater (-1,-1). Grunden til at de begge to er negative er fordi billedet starter oppe i venstre hjørne. Derefter bliver vektoren normaliseret igen altså gjort til en enhedsvektor.
”speed_modifier”-variablen er baseret på spillerens position i den genererede verden og eller rettere sagt hvilken type felt spilleren står på. Der bliver refereret til det andet indeks i listen hvor bevægelseshastighed er gemt. De 2 variabler bliver brugt til at køre funktionen ”move” på player (Fig. 7). self.player er et Player objekt.

```
# Controls input
# Player
p_vel = vect(0, 0)
if pressed[pg.K_UP]:
    p_vel += vect(0, -1)
if pressed[pg.K_DOWN]:
    p_vel += vect(0, 1)
if pressed[pg.K_LEFT]:
    p_vel += vect(-1, 0)
if pressed[pg.K_RIGHT]:
    p_vel += vect(1, 0)
```

Fig. 6 Input håndtering

```
# Movement
# Player
p_vel = Normalize(p_vel)


p_pos = vect(self.player.pos.x, self.player.pos.y)

speed_modifier = self.world_map.tiles[int(p_pos.x)][int(p_pos.y)][2]

next_pos = p_pos + p_vel * speed_modifier * self.player.speed

# Stops player from moving outside the world
if (0 < next_pos.x < self.world_map.width) and (0 < next_pos.y < self.world_map.height):
    self.player.move(p_vel, speed_modifier)
```

Fig. 7 Aktuel bevægelse af spilleren

Fælder

Det er muligt for spilleren af placere fælder i spillet via koden i Fig. 8. For at placere en fælde skal man trykke på t, dette bliver håndteret på samme måde, og tidspunkt, som bevægelse af spilleren. At trykke på ”t” sætter variablen ”place_trap” til True, mens ikke at trykke sætter den til False. På den måde kan man holde ”t” inde for at blive ved med at placere fælder. For at sikre at der på den måde ikke bliver placeret en fælde hvert tick. Er der sat timing op som vist i Fig. 9. Disse variabler bliver sat til Unix tid (mængden af sekunder siden starten af 1970). Og kan så refereres til og forskellen mellem det gemte variabel og den nye tid som bliver målt hver gang koden i Fig. 8. Det er dermed hvis forskellen mellem de 2 tider er større end 0.2 sekunder, at spilleren kan placere en fælde. Det kræver dog også at spilleren har nogle materialer de kan anvende. Når tiden så overstiger 0.2 sekunder bliver ”self.place_trap_ref” igen sat til Unix tid, og timeren bliver på den måde genstartet. På denne måde bliver alle de andre timede events også håndteret: Spillets samlede tid gået, at spilleren spiser fra sine rationer, at spilleren regenererer liv de har mistet, at spilleren angriber og at vandrende salgsmænd bevæger sig ud fra byerne.

```
# Player actions
if place_trap:
    if time() - self.place_trap_ref > 0.2:
        if self.player.materials >= 5:
            self.place_trap_ref = time()
            self.player.materials -= 5
            self.player_traps.append(Trap((int(self.player.pos[0]), int(self.player.pos[1]))))
```

Fig. 8 Placering af fælder

```
# Reference times
self.game_ref = time()
self.eat_ref = time()
self.regen_ref = time()
self.attack_ref = time()
self.place_trap_ref = time()
self.merchant_spawn_ref = time()
```

Fig. 9 Variabler der holder styr på tid gået

Trade_unit-klassen - Handelsmænd og vagter

Et Trade_unit objekt definerer en handelsmand som kan have nogle vagter af Guard-klassen (Fig. 10). Mængden af vagter en handelsmand får med sig afhænger af hvor stor byen handelsmanden kommer fra er, og hvor mange ressourcer handelsmanden har med. Måden hvorpå vagterne styres, altså måden de bevæger sig på, afhænger af om spilleren er i nærheden. Hvis spilleren ikke er i nærheden, er vagternes bevægelse den sammen som handelsmanden. Måden dette opnås er ved at vagternes position er defineret ud fra handelsmandens position, altså deres position er relativ (Fig. 11). Dette kan ses på måden vagterne bliver tegnet og på deres angrebs logik. Når vagterne bliver tegnet, tegnes de ud fra handelsmandens position plusset med deres relative position til handelsmanden (Fig. 12, Fig. 16). Grunden til at deres position er relativ er at der dermed ikke er behov for at ændre på vagternes position individuelt, det eneste der skal udregnes, er handelsmandens bevægelse. Dette er kun tilfældet når der ikke er en spiller til stede inde for rækkevidde af handelsmandens synsfelt. Ellers hvis spilleren er tæt nok på bevæger vagterne og handelsmanden sig anderledes. Mens vagterne render efter spilleren, bliver handelsmanden stående og venter på vagterne. Dette kan ses gennem ”move” funktionen for Trade_unit-klassen (Fig. 13). Hvis der er en spiller, returnerer koden: `if player is not None:` True. Grunden til dette ligger bag koden i ”Game”-klassen hvor enten Player objektet eller None bliver sendt afhængigt af om afstanden mellem handelsmanden og spilleren er mindre end den afstand handelsmanden kan se spilleren fra (Fig. 14).

Handelsmændene bevæger sig fra by til by. Måden objektet bliver oprettet på er gennem den timede event ”spawn_trade_unit” (Fig. 18).
Som det første i ”spawn_trade_unit” bliver en liste lavet. Denne liste indeholder alle byerne x antal gange afhængigt af deres ”weight” variabel (Fig. 19). Dette variabel har bestemt både den visuelle størrelse af byen og den samlede ressource mængde. Det er et variabel der beskriver værdi af byen.
Næst bliver et tilfældigt indeks i listen valgt som startbyen for handelsmanden, siden der er flere indekser af de byer med store værdier, har de større chance for at blive valgt end de mindre værdifulde byer. Og giver dermed en realistisk sammenhæng mellem størrelsen af byen og frekvensen der kommer en handelsmand ud af byen. Endebyen bliver valgt tilfældigt mellem de byer startbyen har veje til (Fig. 20).
Den sidste del af ”spawn_trade_unit” Bestemmer mængden af varer og hvilke varer handelsmanden får

baseret på startbyens ressourcer på samme måde som startbyen blev valgt. Mængden af varer bliver bestemt af byens værdi. Mængden af vagter bliver bestemt både ud fra byens værdi, men også ud fra mængden af varer (Fig. 21).

”dist”-funktionen jeg har anvendt til at finde afstanden, tager 2 punkter og udregner afstanden mellem dem ud fra Pythagoras, altså afstandsformlen (Fig. 15).

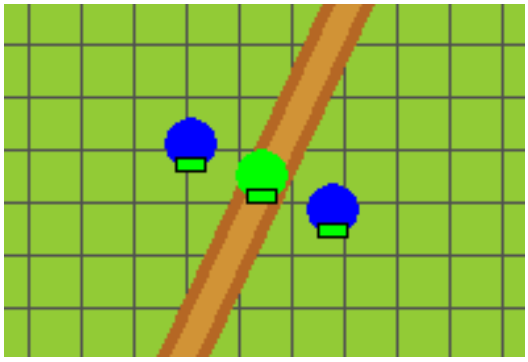


Fig. 10 En handelsmand med 2 vagter. Vagterne er blå

```
class Guard:
    def __init__(self, rel_pos, time):
        x, y = rel_pos.x, rel_pos.y
        self.rel_pos = vect(x,y)
        self.original_rel_pos = vect(x,y)
```

Fig. 11 Guard-klassen bliver oprettet med relativ og original position

```
for unit in game.trade_units:
    unit_pos = world_to_screen_no_grid(unit.pos)
    pygame.draw.circle(screen, (0, 255, 0), unit_pos, 2 * S, 0)
    health_bar((unit_pos[0], unit_pos[1] + 8), (unit.max_hp, 4), unit.hit_points, unit.max_hp)
    for guard in unit.guards:
        guard_pos = world_to_screen_no_grid(guard.rel_pos + unit.pos)
        pygame.draw.circle(screen, (0, 0, 255), guard_pos, 2 * S, 0)
        health_bar((guard_pos[0], guard_pos[1] + 8), (unit.max_hp, 4), guard.hit_points, guard.max_hp)
```

Fig. 12 Udklip fra "Main" Her tegnes handelsmænd og vagter

```python
def move(self, player):
    if player is not None:
        if len(self.guards) > 0:
            for guard in self.guards:
                guard.move(self.pos, player)
        else:
            self.pos += self.vel * self.speed * 1.2
    else:
        origin = True
        for guard in self.guards:
            if dist(guard.original_rel_pos, guard.rel_pos) > 0.15:
                origin = False
        if origin == True:
            self.pos += self.vel * self.speed
        else:
            for guard in self.guards:
                guard.move(self.pos)
```

Fig. 13 move funktionen for Trade_unit-klassen

```python
player = None
if dist((int(p_pos.x), int(p_pos.y)), unit.pos) < unit.detect_dist:
    player = self.player
unit.move(player)
```

Fig. 14 Udklip der kører move funktionen for Trade_unit-objektet og sender enten Player objektet eller None

Dist.py

```python
from math import sqrt

def dist(P1, P2):
    # Returns distance between 2 points
    return sqrt(((P1[0] - P2[0])**2) + ((P1[1] - P2[1])**2))

```

Fig. 15 Funktion som er blevet placeret i sin egen fil for at kunne blive anvendte af flere forskellige filer

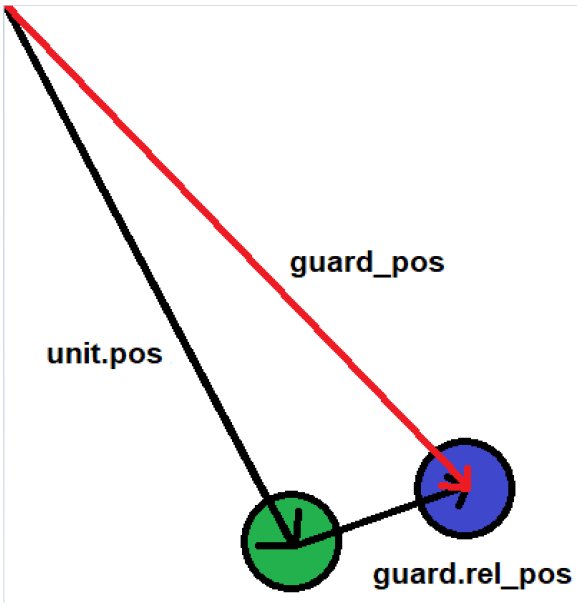


Fig. 16 Vektor udregning for en vagt

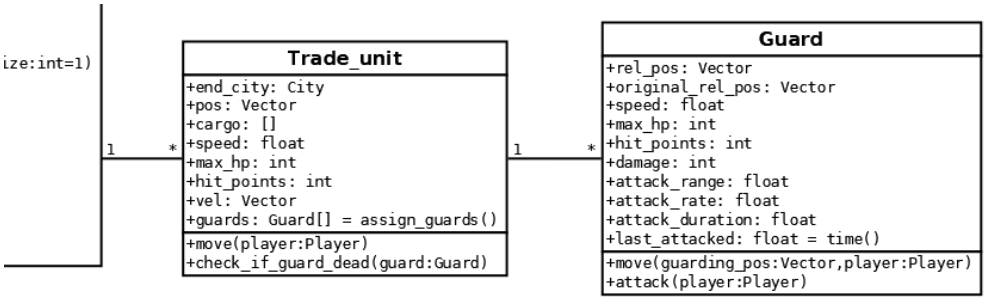


Fig. 17 Sammenhæng mellem Game-, Trade_unit- og Guard-klassen

```
# Cities
if time() - self.merchant_spawn_ref > 3: # Merchant spawning
    self.spawn_trade_unit()
    self.merchant_spawn_ref = time()
```

Fig. 18 spawn_trade_unit-funktionen bliver kørt hvert 3. sekund

```
def spawn_trade_unit(self):
    city_list = []
    for city in self.world_map.cities:
        for i in range(ceil(city.weight/2)):
            if (len(city.roads) > 0) and (city.weight > 0):
                city_list.append(city)
```

Fig. 19 Del 1 af spawn_trade_unit

```
    start_city = city_list[randint(0,len(city_list) - 1)]
    end_city = start_city.roads[randint(0,len(start_city.roads) - 1)]
```

Fig. 20 Del 2 af spawn_trade_unit

```
    cargo_size = randint(1, ceil(start_city.weight/2))
    cargo = [0,0,0]
    for i in range(cargo_size):
        item = start_city.resources[randint(0,len(start_city.resources) - 1)]
        cargo[item] += 1

    guards = randint(0, cargo_size - 1) + randint(0, ceil(start_city.weight/2) - 1)

    self.trade_units.append(Trade_unit(start_city, end_city, cargo, time(), guards))
```

Fig. 21 Del 3 af spawn_trade_unit

Andet

Det er værd at nævne at jeg har anvendt en del kode fra et tidligere projekt, dette projekt var også et spil, det hed ”Astroid”. Både menuerne og highscores som jeg ikke har nævnt kommer fra dette spil.
Kort beskrevet bestemmer gamestates hvilken kode der bliver kørt både i ”tick” (fra Game-klassen) og i ”draw_game” (Fra ”Main”-filen)

Evaluering efter projektets færdiggørelse

Test

Jeg har optaget en lille video af spillet kørende. Of testet alle de funktionaliteter som jeg har programmeret. Det tog utroligt lang tid at få skærmen til at opføre sig som den skulle og at få vejene til at blive genereret ordentligt. Der opstår f.eks. en meget mærkelig fejl utroligt sjældent hvor der på en eller anden måde ikke bliver lavet forbindelser til nogle af de byer som tydeligvis kan laves forbindelser til (Fig. 22).
Der er stadig ting som spillet mangler, f.eks. bevæger hverken handelsmænd eller vagter sig med andre hastigheder på forskelligt terræn lige som spilleren gør. En anden ting er at vagterne bare forsvinder når handelsmanden dør. Selv under skrivning af denne synopsis har jeg ændret i koden for at den virkede bedre eller bare så bedre ud.

Bilag

Andre billeder

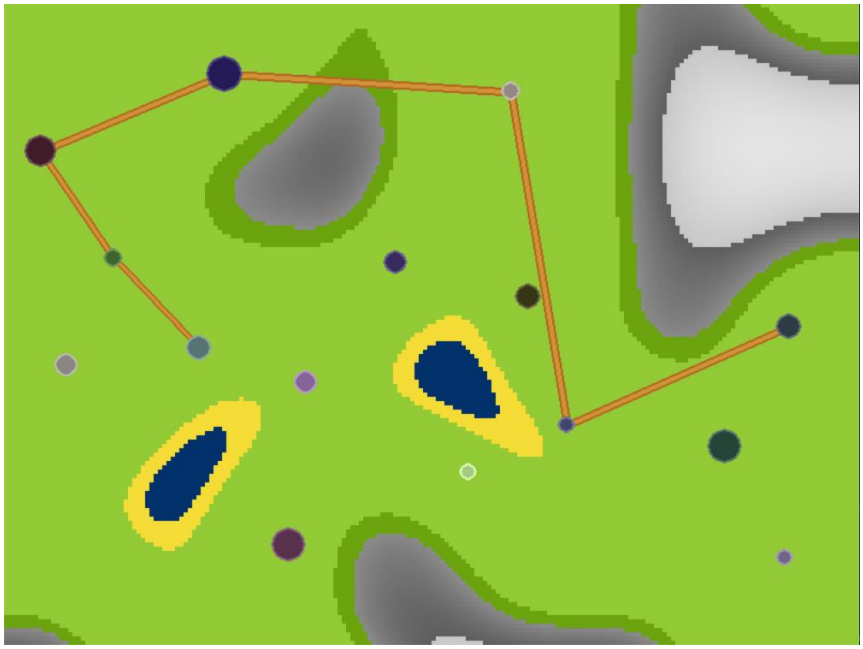


Fig. 22 Mærkelig bug

Diagrammer

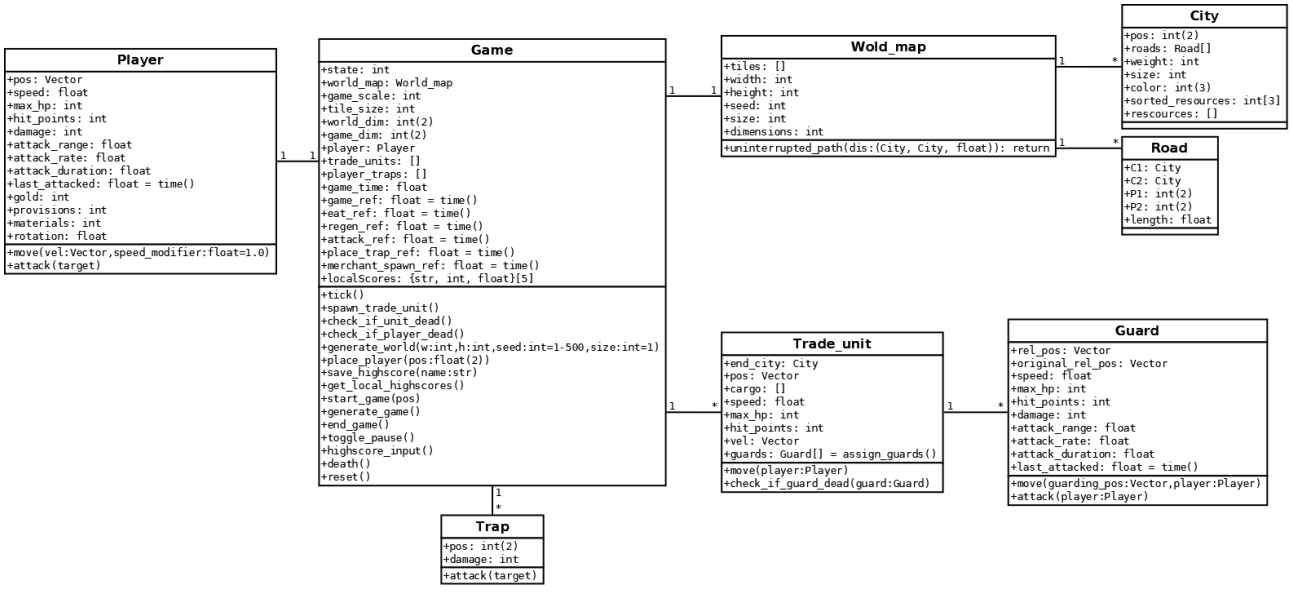


Fig. 23 Samlet klassediagrammet

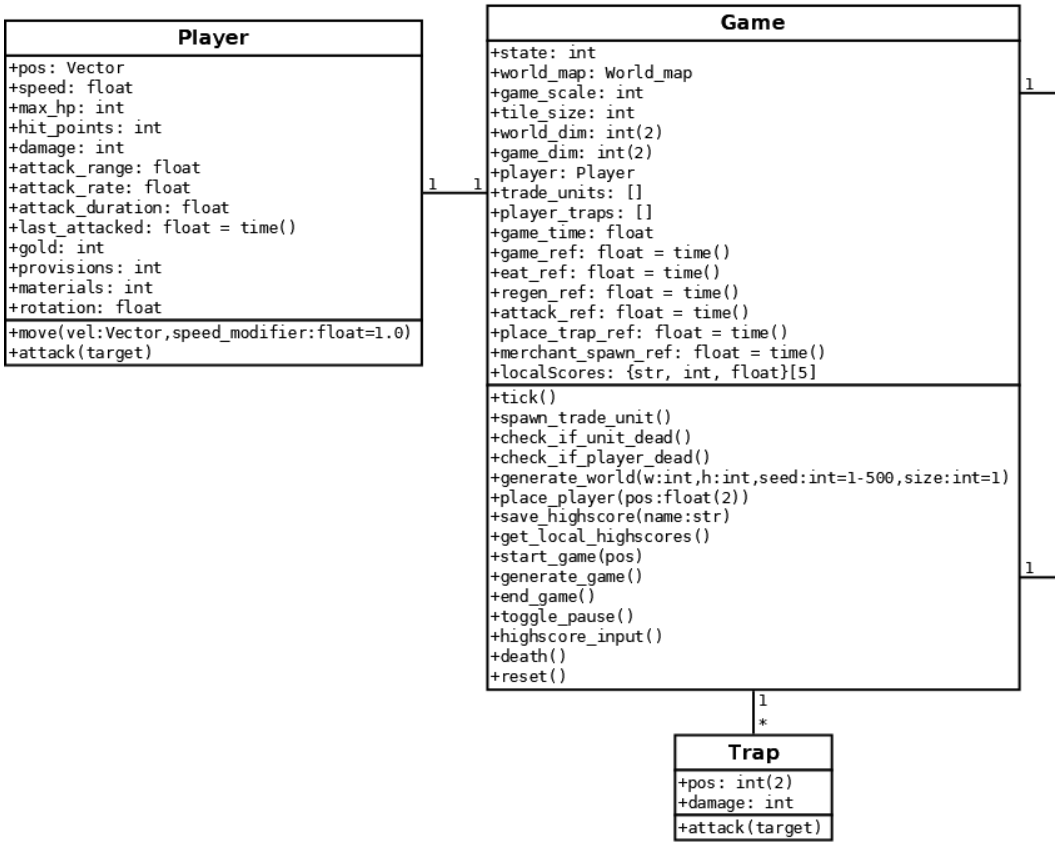


Fig. 24 Venstre side af klassediagrammet

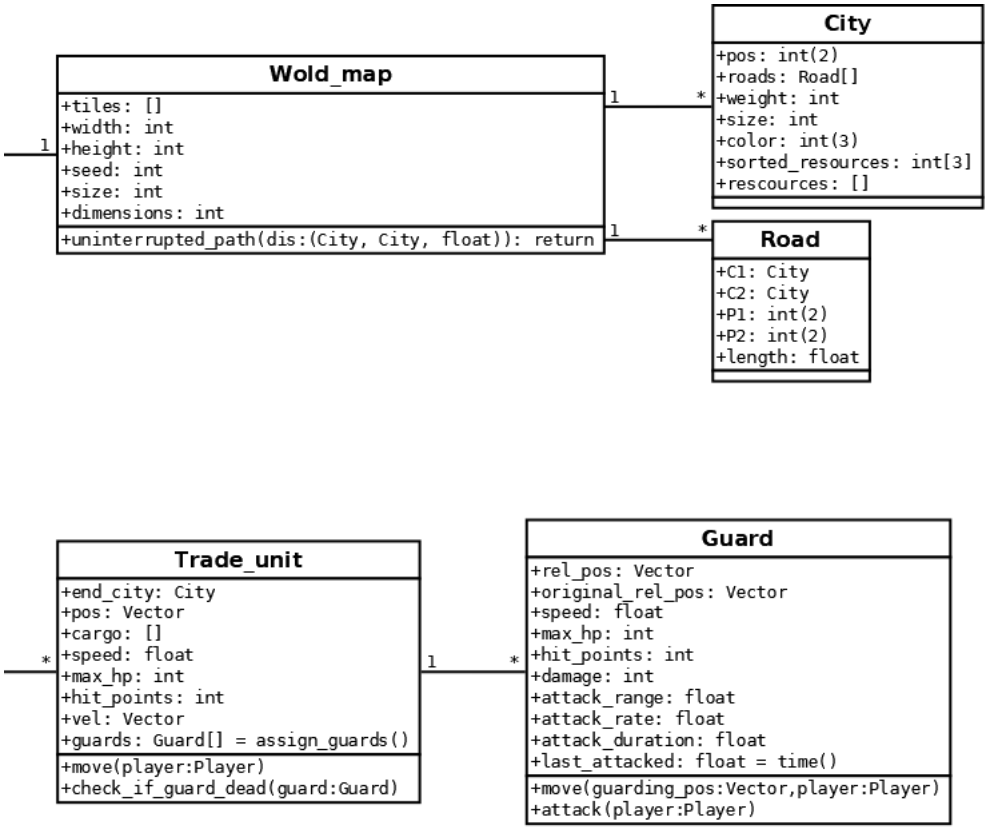


Fig. 25 Højre side af klassediagrammet

Video

Kort test af spillet: http://youtu.be/M4AzjC7Kj6k

Kode

Link til github: https://github.com/jonas8217/Highwayman

Main

```
1.  import pygame
2.  import pygame_textinput
3.  from math import sqrt, cos, sin, pi
4.  from Dist import dist
5.  from Game import Game
6.
7.
8.
9.  pygame.init()
10. """
11. icon = ""
12. icon = io.BytesIO(base64.b64decode(icon))
13. icon = pygame.image.load(icon)
14. pygame.display.set_icon(icon)
15. """
16. pygame.display.set_caption('highwayman')
17. screen = pygame.display.set_mode((800, 600))
18. # Initialize font; must be called after 'pygame.init()' to avoid 'Font not Initialized' error
19. small_font = pygame.font.SysFont("monospace", 15)
20. big_font = pygame.font.SysFont("monospace", 45)
21.
22. running = True
23.
24. game = Game(pygame.display.Info())
25.
26. textinput = pygame_textinput.TextInput()
27.
28. clock = pygame.time.Clock()
29.
30.
31. def draw_game():
32.     screen_info = pygame.display.Info()
33.     w, h = screen_info.current_w, screen_info.current_h
34.     Big_size = big_font.size(' ')
35.     Small_size = small_font.size(' ')
36.
37.     if game.state == 0:
38.
39.
40.         pygame.draw.rect(screen, (0, 0, 0), pygame.Rect(0, 0, w, h))
41.
42.         pygame.draw.rect(screen, (30, 30, 30), pygame.Rect(w//2 - 40, 50, 80, 40))
43.         txt_size = small_font.size("MENU")
44.         screen.blit(small_font.render("MENU", 1, (255, 255, 255)), (w//2 - txt_size[0]//2, 70 - txt_size[
    1]//2))
45.
46.     elif game.state == 0.5:
47.
48.         map = game.world_map # World_map object
49.         ts = game.tile_size # size of an individual tile in pixels
50.
51.         screen.fill((255, 255, 255))
52.         for x in range(map.width):
53.             for y in range(map.height):
54.                 pygame.draw.rect(screen, map.tiles[x][y][1], pygame.Rect(x * ts, y * ts, ts, ts))
```

```
55.         for road in map.roads:
56.             pygame.draw.line(screen, (181, 103, 36), (road.P1[0] * ts + ts/2, road.P1[1] * ts + ts/2), (r
    oad.P2[0] * ts + ts/2, road.P2[1] * ts + ts/2), int(ts * 2))
57.             pygame.draw.line(screen, (209, 147, 54), (road.P1[0] * ts + ts/2, road.P1[1] * ts + ts/2), (r
    oad.P2[0] * ts + ts/2, road.P2[1] * ts + ts/2), ts)
58.         for city in map.cities:
59.             pygame.draw.circle(screen, city.color, (city.pos[0] * ts + ts//2 , city.pos[1] * ts + ts//2),
     city.size, 0)
60.             pygame.draw.circle(screen, (city.color[0]-50, city.color[1]-50, city.color[2]-
    50), (city.pos[0] * ts + ts//2 , city.pos[1] * ts + ts//2), city.size-2, 0)
61.
62.
63.
64.     elif game.state == 1:
65.         screen.fill((100, 100, 100))
66.
67.         # Declaring shorter variabels for later use
68.         player = game.player    # Player object
69.         p_pos = player.pos      # Position of player
70.         map = game.world_map    # World_map object
71.         ts = game.tile_size     # Size of an individual tile in pixels
72.         dims = game.game_dim    # Game_dim[0],game_dim[1] = game view size in tiles
73.         S = game.game_scale     # Difference in scale between world and view size
74.         p_pos_x, p_pos_y = p_pos.x - int(p_pos.x), p_pos.y - int(p_pos.y)
75.         world_to_screen_grid =    lambda pos : (int(((pos[0] - int(p_pos.x) + dims[0]//2 + 1/2) * ts) * S
    ), int(((pos[1] - int(p_pos.y) + dims[1]//2 + 1/2) * ts) * S))
76.         world_to_screen_no_grid = lambda pos : (int(((pos[0] - p_pos.x + dims[0]//2 + p_pos_x + 1/2) * ts
    ) * S), int(((pos[1] - p_pos.y + dims[1]//2 + p_pos_y + 1/2) * ts) * S))
77.
78.         lB = 0 # LeftBoundary
79.         rB = 0 # RightBoundary
80.         tB = 0 # TopBoundary
81.         bB = 0 # BottomBoundary
82.
83.         # Veiw boundaries
84.         if int(p_pos.x) - dims[0]//2 < 0:
85.             lB = dims[0]//2 - int(p_pos.x)
86.         if int(p_pos.x) + dims[0]//2 > map.width:
87.             rB = int(p_pos.x) + dims[0]//2 - map.width
88.         if int(p_pos.y) - dims[1]//2 < 0:
89.             tB = dims[1]//2 - int(p_pos.y)
90.         if int(p_pos.y) + dims[1]//2 > map.height:
91.             bB = int(p_pos.y) + dims[1]//2 - map.height
92.
93.
94.         # Rendering
95.         # Map
96.         for x in range(lB, w//(ts*S) - rB):
97.             for y in range(tB, h//(ts*S) - bB):
98.                 tile_color = map.tiles[int(p_pos.x) - dims[0]//2 + x][int(p_pos.y) - dims[1]//2 + y][1]
99.                 pygame.draw.rect(screen, tile_color, pygame.Rect(x * ts * S, y * ts * S, ts * S, ts * S))
100.
101.         # Lines
102.         for x in range(w//(ts*S)):
103.             pygame.draw.line(screen, (80, 80, 80), (x * ts * S, 0), (x * ts * S, h))
104.         for y in range(h//(ts*S)):
105.             pygame.draw.line(screen, (80, 80, 80), (0, y * ts * S), (w, y * ts * S))
106.
107.         # Roads
108.         for road in map.roads:
109.             P1, P2 = road.P1, road.P2
110.             roadlen = dist(P1, P2)
```

```
111.             PMid = ((P1[0] + P2[0])/2, (P1[1] + P2[1])/2)
112.             if dist(PMid, (p_pos.x, p_pos.y)) < sqrt((dims[0]//2)**2 + (dims[1]//2)**2) + roadlen/2:
113.                 RP1_pos,RP2_pos  = world_to_screen_grid(road.P1), world_to_screen_grid(road.P2)
114.                 pygame.draw.line(screen, (181, 103, 36), RP1_pos, RP2_pos, ts * S)
115.                 pygame.draw.line(screen, (209, 147, 54), RP1_pos, RP2_pos, int(ts * S/2))
116.
117.
118.         for city in map.cities:
119.             if dist(city.pos, (p_pos.x, p_pos.y)) < sqrt((dims[0]//2)**2 + (dims[1]//2)**2) + city.size:

120.                 darker_col = (city.color[0]-50, city.color[1]-50, city.color[2]-50)
121.                 c_pos = world_to_screen_grid(city.pos)
122.                 pygame.draw.circle(screen, city.color, c_pos, city.size * S, 0)
123.                 pygame.draw.circle(screen, darker_col, c_pos, (city.size-2) * S, 0)
124.                 screen.blit(big_font.render(str(city.sorted_resources[0]), 1, (255, 255, 0)), (c_pos[0] -
     int(S * city.size/2) - Big_size[0]/2, c_pos[1] - Big_size[1]/2))
125.                 screen.blit(big_font.render(str(city.sorted_resources[1]), 1, (  0, 255, 0)), (c_pos[0] -
     Big_size[0]/2, c_pos[1] - Big_size[1]/2))
126.                 screen.blit(big_font.render(str(city.sorted_resources[2]), 1, (100,  50, 0)), (c_pos[0] +
     int(S * city.size/2) - Big_size[0]/2, c_pos[1] - Big_size[1]/2))
127.
128.         for unit in game.trade_units:
129.             unit_pos = world_to_screen_no_grid(unit.pos)
130.             pygame.draw.circle(screen, (0, 255, 0), unit_pos, 2 * S, 0)
131.             health_bar((unit_pos[0], unit_pos[1] + 8), (unit.max_hp, 4), unit.hit_points, unit.max_hp)
132.             for guard in unit.guards:
133.                 guard_pos = world_to_screen_no_grid(guard.rel_pos + unit.pos)
134.                 pygame.draw.circle(screen, (0, 0, 255), guard_pos, 2 * S, 0)
135.                 health_bar((guard_pos[0], guard_pos[1] + 8), (unit.max_hp, 4), guard.hit_points, guard.ma
    x_hp)
136.
137.         for trap in game.player_traps:
138.             trap_pos = world_to_screen_grid(trap.pos)
139.             pygame.draw.rect(screen, (100,  50,  0), pygame.Rect(trap_pos[0] - 14, trap_pos[1] - 14, 28,
    28))
140.             pygame.draw.rect(screen, (50,  25,  0), pygame.Rect(trap_pos[0] - 9, trap_pos[1] - 9, 18, 18)
    )
141.
142.         # Player
143.         pygame.draw.circle(screen, (255, 0, 0), (w//2 + S * ts//2, h//2 + S * ts//2), S * ts//2, 0)
144.         p_rot = player.rotation
145.         pygame.draw.circle(screen, (255, 255, 255), (int(w//2 + S * ts//2 + cos(p_rot - pi/6) * S * ts//3
    ), int(h//2 + S * ts//2 + sin(p_rot - pi/6) * S * ts//3)), int(S * ts//6), 0)
146.         pygame.draw.circle(screen, (255, 255, 255), (int(w//2 + S * ts//2 + cos(p_rot + pi/6) * S * ts//3
    ), int(h//2 + S * ts//2 + sin(p_rot + pi/6) * S * ts//3)), int(S * ts//6), 0)
147.
148.
149.         # Hud
150.         # Info
151.
152.         screen.blit(small_font.render("FPS:{}".format(str(int(clock.get_fps()))), 1, (0, 0, 0)), (w - Sma
    ll_size[0] * 6 - 1, int(S * ts * 1/2 - Small_size[1]//2)))
153.
154.         # Rescources
155.         pygame.draw.rect(screen, (  0,   0,   0), pygame.Rect(S * ts//2 - 1, S * ts//2 - 1, S * ts * 8 +
    2, S * ts * 4 + 2))
156.         pygame.draw.rect(screen, (150, 150, 150), pygame.Rect(S * ts//2    , S * ts//2    , S * ts * 8
     , S * ts * 4    ))
157.         screen.blit(small_font.render("Gold: {}".format(game.player.gold),             1, (0, 0, 0)), (in
    t(S * ts * 3/2 - Small_size[0]), int(S * ts * 3/2 - Small_size[1]//2)))
158.         screen.blit(small_font.render("Provisions: {}".format(game.player.provisions), 1, (0, 0, 0)), (in
    t(S * ts * 3/2 - Small_size[0]), int(S * ts * 5/2 - Small_size[1]//2)))
```

```
159.        screen.blit(small_font.render("Materials: {}".format(game.player.materials),   1, (0, 0, 0)), (in
    t(S * ts * 3/2 - Small_size[0]), int(S * ts * 7/2 - Small_size[1]//2)))
160.
161.        # Health
162.        width, height, x_pos, y_pos = 200, 50, w//2, h- 50
163.        health_bar((x_pos,y_pos), (width,height), player.hit_points, player.max_hp)
164.
165.    elif game.state == 2:
166.        pygame.draw.rect(screen, (30, 30, 30), pygame.Rect(w//2 - 40, h//2 - 20, 80, 40))
167.        screen.blit(small_font.render("PAUSE", 1, (255, 255, 255)), (377, 291))
168.
169.    if game.state == 2 or game.state == 0:
170.
171.        controls = ["Movement: WASD", "Attack: Spacebar","Place trap: t" , "Pause: p", "Exit Game/New Gam
    e: ESC", "Sumbmit Score: Enter"]
172.        pygame.draw.rect(screen, (30, 30, 30), pygame.Rect(w//2 - 150, 400, 300, (len(controls) + 1) * (S
    mall_size[1] + 3) + 3))
173.        screen.blit(small_font.render("Controls:", 1, (255, 255, 255)), (270, 403))
174.        for i, text in enumerate(controls):
175.            screen.blit(small_font.render(text, 1, (255, 255, 255)), (275, 403 + (i + 1) * (Small_size[1]
     + 3)))
176.
177.
178.        pygame.draw.rect(screen, (30, 30, 30), pygame.Rect(w//2 - 175, 120, 350, 3 + Small_size[1] + 3 +
    (Small_size[1] + 3) * len(game.localScores)))
179.        screen.blit(small_font.render("Highscores:", 1, (255, 255, 255)), (w//2 - 175 + 5, 120 + 3))
180.        for i, score in enumerate(game.localScores):
181.            if len(score['Name']) > 0:
182.                screen.blit(small_font.render(str(score['Name']) + ' - Gold: ' + str(score['Gold']) + ' T
    ime: ' + str(int(score['Time']/60)) + ':' + str(int(score['Time'] % 60)), 1, (255, 255, 255)), (w//2 - 17
    5 + 5, 120 + (i + 1) * (Small_size[1] + 3)))
183.
184.
185.    elif game.state == 3:
186.
187.        screen.fill((225, 225, 225))
188.        global textinput
189.        screen.blit(textinput.get_surface(), (10, 10))
190.        if textinput.update(events) and len(textinput.get_text()) > 0:
191.            game.save_highscore(textinput.get_text())
192.            textinput = pygame_textinput.TextInput()
193.
194.
195.
196.
197.def screen_to_world(pos):
198.    return (int(pos[0]/game.tile_size), int(pos[1]/game.tile_size))
199.
200.def health_bar (pos, size, health, max_health):
201.    pygame.draw.rect(screen, (  0,   0,   0), pygame.Rect(pos[0] - size[0]//2 - 1, pos[1] - size[1]//2 -
    1, size[0] + 2, size[1] + 2))
202.    pygame.draw.rect(screen, (255,   0,   0), pygame.Rect(pos[0] - size[0]//2    , pos[1] - size[1]//2
     , size[0]    , size[1]))
203.    pygame.draw.rect(screen, (  0, 255,   0), pygame.Rect(pos[0] - size[0]//2    , pos[1] - size[1]//2
     , int(size[0] * health/max_health), size[1]))
204.
205.
206.while running:
207.    events = pygame.event.get()
208.    for event in events:
209.        if event.type == pygame.QUIT:
210.            running = False
211.        if event.type == pygame.KEYDOWN and event.key == pygame.K_p:
```

```
212.             if game.state == 1 or game.state == 2:
213.                 game.toggle_pause()
214.         if event.type == pygame.MOUSEBUTTONDOWN:
215.             if game.state == 0.5:
216.                 pos = pygame.mouse.get_pos()
217.                 game.start_game(screen_to_world(pos))
218.         if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
219.             if game.state != 0:
220.                 game.end_game()
221.             else:
222.                 game.generate_game()
223.
224.     pressed = pygame.key.get_pressed()
225.
226.     game.tick(pygame, pressed)
227.
228.     draw_game()
229.     pygame.display.flip()
230.     clock.tick(60)
```

Game

```
1.  from math import ceil, pi
2.  from random import randint
3.  from Worldgen import World_map
4.  from Player import Player
5.  from Trade_unit import Trade_unit
6.  from Trap import Trap
7.  from Vector import Normalize, Vector as vect
8.  from Vector_math import vect_to_angle, vectors_to_angle, angle_to_vector
9.  from Dist import dist
10. import pickle
11. from time import time
12.
13.
14.
15. class Game:
16.     def __init__(self, screen_info):
17.         w, h = screen_info.current_w, screen_info.current_h
18.         # Gamestate
19.         self.state = 0
20.
21.         # Worldmap variabels
22.         self.world_map = None
23.
24.         self.game_scale = 5
25.         self.tile_size = 4
26.         self.world_dim = (w//self.tile_size,h//self.tile_size)
27.         self.game_dim = (self.world_dim[0]//self.game_scale, self.world_dim[1]//self.game_scale)
28.
29.         # Game dependant variabels
30.         self.player = None
31.
32.         self.trade_units = []
33.
34.         self.player_traps = []
35.
36.         self.game_time = 0
37.
38.         # Highscores
```

```
39.         self.localScores = self.get_local_highscores()[:5]
40.
41.
42.     def tick(self, pg, pressed):
43.         if self.state == 0.5:
44.             if pressed[pg.K_r]:
45.                 self.generate_world(self.world_dim[0],self.world_dim[1],randint(1,500),3)
46.
47.             # Reference times
48.             self.game_ref = time()
49.             self.eat_ref = time()
50.             self.regen_ref = time()
51.             self.attack_ref = time()
52.             self.place_trap_ref = time()
53.             self.merchant_spawn_ref = time()
54.
55.         if self.state == 1:
56.
57.             # Controls input
58.             # Player
59.             p_vel = vect(0, 0)
60.             if pressed[pg.K_w]:
61.                 p_vel += vect(0, -1)
62.             if pressed[pg.K_s]:
63.                 p_vel += vect(0, 1)
64.             if pressed[pg.K_a]:
65.                 p_vel += vect(-1, 0)
66.             if pressed[pg.K_d]:
67.                 p_vel += vect(1, 0)
68.             if pressed[pg.K_SPACE]:
69.                 attack = True
70.             else:
71.                 attack = False
72.             if pressed[pg.K_t]:
73.                 place_trap = True
74.             else:
75.                 place_trap = False
76.
77.
78.             # Player actions
79.             if place_trap:
80.                 if time() - self.place_trap_ref > 0.2:
81.                     if self.player.materials >= 5:
82.                         self.place_trap_ref = time()
83.                         self.player.materials -= 5
84.                         self.player_traps.append(Trap((int(self.player.pos[0]), int(self.player.pos[1])))
    )
85.
86.             # Movement
87.             # Player
88.             p_vel = Normalize(p_vel)
89.
90.
91.             p_pos = vect(self.player.pos.x, self.player.pos.y)
92.
93.             speed_modifier = self.world_map.tiles[int(p_pos.x)][int(p_pos.y)][2]
94.
95.             next_pos = p_pos + p_vel * speed_modifier * self.player.speed
96.
97.             # Stops player from moving outside the world
98.             if (0 < next_pos.x < self.world_map.width) and (0 < next_pos.y < self.world_map.height):
99.                 self.player.move(p_vel, speed_modifier)
100.
```

```
101.             # Trade_units
102.             to_pop = []
103.             for unit in self.trade_units:
104.                 if dist(unit.pos, unit.end_city.pos) < 1 * unit.end_city.size / self.tile_size:
105.                     to_pop.append(unit)
106.                 else:
107.                     player = None
108.                     if dist((int(p_pos.x), int(p_pos.y)), unit.pos) < unit.detect_dist:
109.                         player = self.player
110.                     unit.move(player)
111.
112.             for unit in to_pop[::-1]:
113.                 self.trade_units.remove(unit)
114.
115.             # Attacks
116.             # Player
117.
118.             if attack:
119.                 if time() - self.player.last_attacked > self.player.attack_rate:
120.                     self.player.last_attacked = time()
121.                     for unit in self.trade_units:
122.                         if dist(unit.pos, self.player.pos) < self.player.attack_range:
123.                             if abs(vectors_to_angle(unit.pos - self.player.pos, angle_to_vector(self.play
    er.rotation))) < pi/3:
124.                                 self.player.attack(unit)
125.                                 self.check_if_unit_dead(unit)
126.                         for guard in unit.guards:
127.                             if dist(guard.rel_pos + unit.pos, self.player.pos) < self.player.attack_range
    :
128.                                 if abs(vectors_to_angle((guard.rel_pos + unit.pos) - self.player.pos, ang
    le_to_vector(self.player.rotation))) < pi/3:
129.                                     self.player.attack(guard)
130.                                     unit.check_if_guard_dead(guard)
131.
132.
133.             # Guards
134.             for unit in self.trade_units:
135.                 for guard in unit.guards:
136.                     guard_pos = unit.pos + guard.rel_pos
137.                     if dist(guard_pos, p_pos) < guard.attack_range:
138.                         if time() - guard.last_attacked > guard.attack_rate:
139.                             guard.last_attacked = time()
140.                             guard.attack(self.player)
141.                             self.check_if_player_dead()
142.
143.             # Traps
144.             for trap in self.player_traps:
145.                 for unit in self.trade_units:
146.                     if dist(trap.pos, unit.pos) < 1:
147.                         trap.attack(unit)
148.                         self.player_traps.remove(trap)
149.                         self.check_if_unit_dead(unit)
150.                         break
151.                     for guard in unit.guards:
152.                         if dist(trap.pos, (guard.rel_pos + unit.pos)) < 1:
153.                             trap.attack(guard)
154.                             self.player_traps.remove(trap)
155.                             unit.check_if_guard_dead(guard)
156.                             break
157.                     else:
158.                         continue
159.                     break
160.
```

```
161.
162.             # Time Stuff
163.
164.             self.game_time = time() - self.game_ref
165.
166.             # Timed events
167.             # Player
168.             if time() - self.eat_ref > 25 - (self.player.max_hp - self.player.hit_points): # Eating
169.                 if self.player.provisions > 0:
170.                     self.player.provisions -= 1
171.                 else:
172.                     self.death()
173.                 self.eat_ref = time()
174.
175.             if time() - self.regen_ref > 1.75: # Regenration
176.                 if self.player.hit_points < self.player.max_hp:
177.                     self.player.hit_points += 1
178.                     self.regen_ref = time()
179.
180.             # Cities
181.             if time() - self.merchant_spawn_ref > 3: # Merchant spawning
182.                 self.spawn_trade_unit()
183.                 self.merchant_spawn_ref = time()
184.
185.
186.     def spawn_trade_unit(self):
187.         city_list = []
188.         for city in self.world_map.cities:
189.             for i in range(ceil(city.weight/2)):
190.                 if (len(city.roads) > 0) and (city.weight > 0):
191.                     city_list.append(city)
192.
193.         start_city = city_list[randint(0,len(city_list) - 1)]
194.         end_city = start_city.roads[randint(0,len(start_city.roads) - 1)]
195.
196.         cargo_size = randint(1, ceil(start_city.weight/2))
197.         cargo = [0,0,0]
198.         for i in range(cargo_size):
199.             item = start_city.resources[randint(0,len(start_city.resources) - 1)]
200.             cargo[item] += 1
201.
202.         guards = randint(0, cargo_size - 1) + randint(0, ceil(start_city.weight/2) - 1)
203.
204.         self.trade_units.append(Trade_unit(start_city, end_city, cargo, time(), guards))
205.
206.     def check_if_unit_dead(self, unit):
207.         if unit.hit_points <= 0:
208.             self.player.gold += unit.cargo[0] * randint(5,10)
209.             self.player.provisions += unit.cargo[1] * randint(5,10)
210.             self.player.materials += unit.cargo[2] * randint(5,10)
211.             self.trade_units.remove(unit)
212.
213.     def check_if_player_dead(self):
214.         if self.player.hit_points <= 0:
215.             self.death()
216.
217.
218.     def generate_world(self, w, h, seed=randint(1, 500), size=1):
219.         self.world_map = World_map(w, h, seed, size)
220.
221.
222.     def place_player(self, pos):
223.         self.player = Player(pos[0], pos[1], time())
```

```
224.
225.
226.    def save_highscore(self, name):
227.        #Pickle database
228.
229.        try:
230.            with open('highscore.txt', 'rb') as f:
231.                scores = pickle.load(f)  #score = {'Name':'','Gold':0,'Time':0} layout of stored indexes
232.        except:
233.            print('No Scorefile, creating score file')
234.            score = {'Name': '', 'Gold': 0, 'Time': 0}
235.            scores = []
236.            for i in range(5):
237.                scores.append(score)
238.            with open('highscore.txt', 'wb') as f:
239.                pickle.dump(scores, f)
240.        for i in range(len(scores)):
241.            if self.player.gold > scores[i]['Gold']:
242.                newHigh = {'Name': str(name), 'Gold': self.player.gold, 'Time': self.game_time}
243.                scores.insert(i, newHigh)
244.                break
245.        scores = scores[:5]
246.        self.localScores = scores[:5]
247.        with open('highscore.txt', 'wb') as f:
248.            print('saving scorefile')
249.            pickle.dump(scores, f)
250.        # Might implement later, depends on somthing i don't have control over
251.        """
252.        #online database
253.        if self.player.gold > 0:
254.            self.logger.post_score('Highwayman', self.player.gold, str(name), self.game_time)
255.
256.        scores = []
257.        try:
258.            for s in self.logger.get_scores('Highwayman'):
259.                scores.append({'Name': s['Opt1'], 'Gold': s['Gold'], 'Time': s['Opt2']})
260.            scores = sorted(scores, key=lambda scores: scores['Gold'], reverse=True)
261.        except:
262.            print('server database error')
263.        """
264.        self.reset()
265.        self.end_game()
266.
267.    """
268.    def get_highscores(self):
269.        scores = []
270.        try:
271.            for s in self.logger.get_scores('Highwayman'):
272.                scores.append({'Name': s['Opt1'], 'Gold': s['Gold'], 'Time': s['Opt2']})
273.            return sorted(scores, key=lambda scores: scores['Gold'], reverse=True)
274.        except:
275.            print('server database error')
276.            return []
277.    """
278.    def get_local_highscores(self):
279.        try:
280.            with open('highscore.txt', 'rb') as f:
281.                scores = pickle.load(f)  #score = {'name':'','score':0,'Time':0}
282.        except:
283.            print('No Scorefile, creating score file')
284.            score = {'Name': '', 'Gold': 0, 'Time': 0}
285.            scores = []
```

```
286.             for i in range(5):
287.                 scores.append(score)
288.             with open('highscore.txt', 'wb') as f:
289.                 pickle.dump(scores, f)
290.         return scores
291.
292.     def start_game(self, pos):
293.         if self.state == 0.5:
294.             self.place_player(pos)
295.             self.state = 1
296.
297.     def generate_game(self):
298.         if self.state == 0:
299.             self.state = 0.5
300.
301.         self.generate_world(self.world_dim[0],self.world_dim[1],randint(1,500),3)
302.
303.
304.     def end_game(self):
305.         if self.state > 0:
306.             self.state = 0
307.
308.     def toggle_pause(self):
309.         if self.state == 1:
310.             self.state = 2
311.         elif self.state == 2:
312.             self.state = 1
313.
314.     def highscore_input(self):
315.         if self.state == 1:
316.             self.state = 3
317.
318.     def death(self):
319.         if self.state == 1:
320.             self.highscore_input()
321.
322.     def reset(self):
323.         self.player = None
324.         self.trade_units[:] = []
325.         self.player_traps[:] = []
326.         self.localScores = self.get_local_highscores()[:5]
```

Worldgen

```
1.  from Perlin import perlingrid as pGrid
2.  from City import City
3.  from Road import Road
4.  from math import ceil,tanh,sqrt
5.  from Dist import dist
6.  from Vector import Length,Normalize,Vector as vect
7.  from random import randint
8.  import numpy as np
9.
10.
11.
12. class World_map:
13.     def __init__(self, w, h, seed, size):
14.         self.tiles = []
15.         self.width = w
16.         self.height = h
```

```
17.         self.seed = seed
18.         self.size = size
19.
20.         self.cities = []
21.         self.roads = []
22.
23.         #Confine variabels
24.         if size < 1:
25.             size = 1
26.         elif size > 10:
27.             size = 10
28.
29.         self.dimensions = w
30.         if w < h:
31.             self.dimensions = h
32.
33.         #Generate map
34.
35.         p = pGrid(size, self.dimensions, self.seed)
36.
37.         #Find extremedies
38.         i,j = np.unravel_index(p.argmin(), p.shape)
39.         min_val = p[i,j]
40.         i,j = np.unravel_index(p.argmax(), p.shape)
41.         max_val = p[i,j]
42.
43.         size_val = max_val
44.         if size_val < abs(min_val):
45.             size_val = abs(min_val)
46.
47.         for x in range(w):
48.             self.tiles.append([])
49.             for y in range(h):
50.                 tile = tanh(p[x][y]/(0.90*size_val)) * 0.5 + 0.5
51.
52.                 if tile < 0.2:
53.                     tile_info = (0, (13 - tile * 54, 61 - tile * 57, 120 - tile * 61), 0.2) #
    Water
54.                 elif tile < 0.22:
55.                     tile_info = (1, (246, 220, 55), 0.75) #Beach
56.                 elif tile < 0.65:
57.                     tile_info = (2, (146, 203, 54), 1.25) #Grassland
58.                 elif tile < 0.7:
59.                     tile_info = (3, (107, 164, 15), 1) #Highlands
60.                 elif tile < 0.8:
61.                     tile_info = (4, (-45 * (tile-0.7)*10 + 140, -45 * (tile-0.7)*10 + 140, -
    45 * (tile-0.7)*10 + 140), 0.6) #Mountain
62.                 else:
63.                     tile_info = (5, (55 * (tile-0.8)*5 + 200, 55 * (tile-
    0.8)*5 + 200, 55 * (tile-0.8)*5 + 200), 0.45) #Mountain_top_snow
64.
65.                 self.tiles[x].append(tile_info)
66.
67.         # City generation
68.         for i in range(100):
69.             pos = (randint(8,w-8), randint(8,h-8))
70.             if self.tiles[pos[0]][pos[1]][0] == 2: # Checks if city bouandaries are okay
71.                 if self.tiles[pos[0]][pos[1]-4][0] == self.tiles[pos[0]-
    4][pos[1]][0] == self.tiles[pos[0]+4][pos[1]][0] == self.tiles[pos[0]][pos[1]+4][0] == 2:
72.                     good_pos = True
73.                     for city in self.cities:
74.                         if dist(pos, city.pos) < 25:
75.                             good_pos = False
```

```
76.                     if good_pos:
77.                         self.cities.append(City(pos))
78.
79.         # Road generation
80.         dists = []
81.         for c1 in self.cities:                                          # Generating list of all
    possible city connections and the length thereof
82.             for c2 in self.cities:                                      #
83.                 if c1 is not c2:                                        #
84.                     dists.append((c1, c2, dist(c1.pos, c2.pos)))    # Create touple of 2 citi
    es and distance: (c1, c2, float: 'distance') and adds them to 'dists' list
85.
86.         connections = [[]] # Connected cites
87.         unfound = self.cities.copy() # Unconnected cities
88.
89.         for un in unfound:
90.             s_dist = shortets_dist(dists)
91.             if self.uninterrupted_path(s_dist):
92.                 connections[-1].append(s_dist)  # Append to new connection to connections
93.                 unfound.remove(s_dist[0])       # and
94.                 unfound.remove(s_dist[1])       # Remove now found cities
95.                 break
96.
97.             else:
98.                 unfound.pop(unfound.index(un))
99.
100.
101.
102.         while len(unfound) != 0: #Keeps going until there are no more unfound citites (all ci
    ties are found)
103.
104.             # Finds the shortest connection which has both a connection to 'connected' and to
     'unfound' insuring a correct connection
105.             cons = []
106.
107.             for con in connections[-1]:
108.                 for dis in dists:
109.                     if (con[0] in dis[:2] or con[1] in dis[:2]) and (dis[0] in unfound or dis
    [1] in unfound):
110.                         if self.uninterrupted_path(dis):  # Check for landscape violation (if
     the road crosses something besides Grass- or Highlands)
111.                             cons.append(dis)
112.
113.             s_dist = shortets_dist(cons)
114.             if s_dist is not None:
115.                 connections[-1].append(s_dist)  # Add new connection
116.
117.                 i = 0                           # Delete city from unfound
118.                 if s_dist[1] in unfound:        #
119.                     i = 1                       #
120.                 unfound.remove(s_dist[i])       #
121.
122.
123.             elif len(unfound) > 1:
124.                 cons = []
125.                 for dis in dists:
126.                     if dis[0] in unfound and dis[1] in unfound:
127.                         if self.uninterrupted_path(dis):
128.                             cons.append(dis)
129.
130.                 s_dist = shortets_dist(cons)
131.                 if s_dist is not None:
132.
```

```
                    connections[-1].append(s_dist)  # Add to new connection to connections
                    unfound.remove(s_dist[0])       # and
                    unfound.remove(s_dist[1])       # Remove now found cities

                else:
                    unfound[:] = []

            else:
                break


        for con in connections:                                 # Create road objects and add to self.roads
            for c in con:                                       #
                self.roads.append(Road(c[0], c[1], c[2]))   #
                self.cities[self.cities.index(c[0])].roads.append(c[1]) # Give cities connected cities as "roads"
                self.cities[self.cities.index(c[1])].roads.append(c[0]) #

    def uninterrupted_path(self, dis):
        c1pos, c2pos = dis[0].pos, dis[1].pos
        V = vect(c2pos[0] - c1pos[0], c2pos[1] - c1pos[1])
        length = Length(V)
        V = Normalize(V)
        pos_V = vect(c1pos[0], c1pos[1])
        uninterrupted = True
        for i in range(int(length)):
            Pos = pos_V + V * i
            tile = self.tiles[int(Pos[0])][int(Pos[1])]
            if not (2 <= tile[0] <= 3):
                uninterrupted = False
        return uninterrupted


def shortets_dist(dists):
    # Finds the 'dist' touple with the smallest distance value
    shortest = None
    for d in dists:
        if shortest == None:
            shortest = d
        elif d[2] < shortest[2]:
            shortest = d
    return shortest
```

Perlin (Bibliotek) - perlingrid (egen funktion)

```
import numpy as np

def perlin(x,y,seed=0):
    # permutation table
    np.random.seed(seed)
    p = np.arange(256,dtype=int)
    np.random.shuffle(p)
    p = np.stack([p,p]).flatten()
    # coordinates of the top-left
    xi = x.astype(int)
    yi = y.astype(int)
    # internal coordinates
    xf = x - xi
    yf = y - yi
```

```
    # fade factors
    u = fade(xf)
    v = fade(yf)
    # noise components
    n00 = gradient(p[p[xi]+yi],xf,yf)
    n01 = gradient(p[p[xi]+yi+1],xf,yf-1)
    n11 = gradient(p[p[xi+1]+yi+1],xf-1,yf-1)
    n10 = gradient(p[p[xi+1]+yi],xf-1,yf)
    # combine noises
    x1 = lerp(n00,n10,u)
    x2 = lerp(n01,n11,u)
    return lerp(x1,x2,v)

def lerp(a,b,x):
    "linear interpolation"
    return a + x * (b-a)

def fade(t):
    "6t^5 - 15t^4 + 10t^3"
    return 6 * t**5 - 15 * t**4 + 10 * t**3

def gradient(h,x,y):
    "grad converts h to the right gradient vector and return the dot product with (x,y)"
    vectors = np.array([[0,1],[0,-1],[1,0],[-1,0]])
    g = vectors[h%4]
    return g[:,:,0] * x + g[:,:,1] * y

def perlingrid(b, res, seed):

    lin = np.linspace(0,b,res,endpoint=False)
    x,y = np.meshgrid(lin,lin)

    return perlin(x,y,seed)
```

City

```
from random import randint
from math import ceil

class City:
    def __init__(self, pos):
        self.pos = pos
        self.roads = []
        self.weight = randint(0,9)
        self.size = self.weight + 8
        start_color = randint(100,215)
        self.color = (start_color + randint(0,80) - 40, start_color + randint(0,80) - 40, start_color + r
andint(0,80) - 40)
        self.sorted_resources = [0,0,0]
        for i in range(ceil(self.weight/2)):
            self.sorted_resources[randint(0,2)] += 1
        self.resources = []
        for i in range(3):
            for j in range(self.sorted_resources[i]):
                self.resources.append(i)


    def __repr__(self): #For debugging purposes
        return 'City: pos(' + str(self.pos[0]) + ',' + str(self.pos[1]) + ')'
```

Road

```
1.  class Road:
2.      def __init__(self, C1, C2, length):
3.          self.C1 = C1
4.          self.C2 = C2
5.          self.P1 = C1.pos
6.          self.P2 = C2.pos
7.          self.length = length
```

Player

```
1.  from Vector import Vector as vect
2.  from Vector_math import vect_to_angle
3.
4.  class Player:
5.      def __init__(self, x, y, time):
6.          self.pos = vect(x, y)
7.          self.speed = 0.15
8.          self.max_hp = 30
9.          self.hit_points = self.max_hp
10.         self.damage = 4
11.         self.attack_range = 2.5
12.         self.attack_rate = 0.5
13.         self.attack_duration = 0.15
14.         self.last_attacked = time
15.         self.gold = 0
16.         self.provisions = 10
17.         self.materials = 10
18.         self.rotation = 0
19.
20.
21.     def move(self, vel, speed_modifier = 1):
22.         self.pos += vel * self.speed * speed_modifier
23.         if not (vel[0] == 0 and vel[1] == 0):
24.             self.rotation = vect_to_angle(vel)
25.
26.     def attack(self, target):
27.             target.hit_points -= self.damage
```

Trade_unit

```
1.  from Vector import Normalize, Vector as vect
2.  from math import cos,sin,pi
3.  from Dist import dist
4.
5.  class Trade_unit():
6.      def __init__(self, start_city, end_city, cargo, time, guards = 0):
7.          s_pos = vect(start_city.pos[0], start_city.pos[1])
8.          e_pos = vect(end_city.pos[0], end_city.pos[1])
9.          self.end_city = end_city
10.         self.pos = vect(s_pos.x, s_pos.y)
11.         self.cargo = cargo
12.         self.speed = 0.1
```

```
13.         self.max_hp = 10
14.         self.hit_points = self.max_hp
15.         self.detect_dist = 12
16.
17.         self.vel = Normalize(e_pos - s_pos)
18.         self.guards = []
19.         if guards > 0:
20.             self.guards = assign_guards(guards, self.vel, time)
21.
22.
23.     def move(self, player):
24.         if player is not None:
25.             if len(self.guards) > 0:
26.                 for guard in self.guards:
27.                     guard.move(self.pos, player)
28.             else:
29.                 self.pos += self.vel * self.speed * 1.2
30.         else:
31.             origin = True
32.             for guard in self.guards:
33.                 if dist(guard.original_rel_pos, guard.rel_pos) > 0.15:
34.                     origin = False
35.             if origin == True:
36.                 self.pos += self.vel * self.speed
37.             else:
38.                 for guard in self.guards:
39.                     guard.move(self.pos)
40.
41.     def check_if_guard_dead(self, guard):
42.         if guard.hit_points <= 0:
43.             self.guards.remove(guard)
44.
45.
46.
47. def assign_guards(num, direc, time):
48.     guards = []
49.     rotation = (2*pi)/num
50.     offset = 0
51.     if num % 2 == 0:
52.         offset = rotation/2
53.     x,y = direc.x, direc.y
54.     for i in range(num):
55.         phi = offset + rotation * i
56.         x_pos, y_pos= x * cos(phi) - y * sin(phi), x * sin(phi) + y * cos(phi)
57.         g_pos = vect(x_pos * 1.5, y_pos * 1.5)
58.         guards.append(Guard(g_pos, time))
59.     return guards
60.
61. class Guard:
62.     def __init__(self, rel_pos, time):
63.         x, y = rel_pos.x, rel_pos.y
64.         self.rel_pos = vect(x,y)
65.         self.original_rel_pos = vect(x,y)
66.         self.speed = 0.1
67.         self.max_hp = 15
68.         self.hit_points = self.max_hp
69.         self.damage = 3
70.         self.attack_range = 1.5
71.         self.attack_rate = 2.5
72.         self.attack_duration = 0.15
73.         self.last_attacked = time
74.
75.     def move(self, guarding_pos, player = None):
```

```
76.         pos = self.rel_pos + guarding_pos
77.         if player is not None:
78.             p_pos = player.pos
79.             if dist(pos, p_pos) > self.attack_range:
80.                 vel = Normalize(p_pos - pos)
81.                 self.rel_pos += vel * self.speed
82.         else:
83.             vel = Normalize(self.original_rel_pos - self.rel_pos)
84.             self.rel_pos += vel * self.speed
85.
86.     def attack(self, target):
87.         target.hit_points -= self.damage
```

Trap

```
1.  class Trap:
2.      def __init__(self,pos):
3.          self.pos = pos
4.          self.damage = 12
5.
6.      def attack(self, target):
7.          target.hit_points -= self.damage
```

Dist

```
1.  from math import sqrt
2.
3.  def dist(P1, P2):
4.      # Returns distance between 2 points
5.      return sqrt(((P1[0] - P2[0])**2) + ((P1[1] - P2[1])**2))
```

Vector (Bibliotek) (ændret i linje 76)

```
1.  import math
2.  class Vector:
3.      'Represents a 2D vector.'
4.      def __init__(self, x = 0, y = 0):
5.          self.x = float(x)
6.          self.y = float(y)
7.
8.      def __add__(self, val):
9.          return Point( self[0] + val[0], self[1] + val[1] )
10.
11.     def __sub__(self,val):
12.         return Point( self[0] - val[0], self[1] - val[1] )
13.
14.     def __iadd__(self, val):
15.         self.x = val[0] + self.x
16.         self.y = val[1] + self.y
17.         return self
18.
19.     def __isub__(self, val):
20.         self.x = self.x - val[0]
21.         self.y = self.y - val[1]
```

```
22.         return self
23.
24.     def __div__(self, val):
25.         return Point( self[0] / val, self[1] / val )
26.
27.     def __mul__(self, val):
28.         return Point( self[0] * val, self[1] * val )
29.
30.     def __idiv__(self, val):
31.         self[0] = self[0] / val
32.         self[1] = self[1] / val
33.         return self
34.
35.     def __imul__(self, val):
36.         self[0] = self[0] * val
37.         self[1] = self[1] * val
38.         return self
39.
40.     def __getitem__(self, key):
41.         if( key == 0):
42.             return self.x
43.         elif( key == 1):
44.             return self.y
45.         else:
46.             raise Exception("Invalid key to Point")
47.
48.     def __setitem__(self, key, value):
49.         if( key == 0):
50.             self.x = value
51.         elif( key == 1):
52.             self.y = value
53.         else:
54.             raise Exception("Invalid key to Point")
55.
56.     def __str__(self):
57.         return "(" + str(self.x) + "," + str(self.y) + ")"
58. Point = Vector
59.
60. def DistanceSqrd( point1, point2 ):
61.     'Returns the distance between two points squared. Marginally faster than Distance()'
62.     return ( (point1[0]-point2[0])**2 + (point1[1]-point2[1])**2)
63. def Distance( point1, point2 ):
64.     'Returns the distance between two points'
65.     return math.sqrt( DistanceSqrd(point1,point2) )
66. def LengthSqrd( vec ):
67.     'Returns the length of a vector sqaured. Faster than Length(), but only marginally'
68.     return vec[0]**2 + vec[1]**2
69. def Length( vec ):
70.     'Returns the length of a vector'
71.     return math.sqrt( LengthSqrd(vec) )
72. def Normalize( vec ):
73.     'Returns a new vector that has the same direction as vec, but has a length of one.'
74.     if( vec[0] == 0. and vec[1] == 0. ):
75.         return Vector(0.,0.)
76.     return vec * (1/Length(vec))
77. def Dot( a,b ):
78.     'Computes the dot product of a and b'
79.     return a[0]*b[0] + a[1]*b[1]
80. def ProjectOnto( w,v ):
81.     'Projects w onto v.'
82.     return v * Dot(w,v) / LengthSqrd(v)
```

Vector_math (Eget bibliotek. Samarbejder med Vector bibilioteket)

```
from math import atan2, pi, acos, sqrt, cos, sin
from Vector import Vector, Dot, LengthSqrd

def vect_to_angle(v):
    x,y = v[0],v[1]

    if x == 0 and y == 0:
        return 0
    else:
        return atan2(y,x)

def vectors_to_angle(v1, v2):
    return acos(Dot(v1,v2)/(sqrt(LengthSqrd(v1)*LengthSqrd(v2))))

def angle_to_vector(a):
    return Vector(cos(a), sin(a))
```

pygame_textinput (bibliotek)

```
"""
Copyright 2017, Silas Gyger, silasgyger@gmail.com, All rights reserved.

Borrowed from https://github.com/Nearoo/pygame-text-input under the MIT license.
"""

import os.path

import pygame
import pygame.locals as pl

pygame.font.init()


class TextInput:
    """
    This class lets the user input a piece of text, e.g. a name or a message.
    This class let's the user input a short, one-lines piece of text at a blinking cursor
    that can be moved using the arrow-keys. Delete, home and end work as well.
    """
    def __init__(
            self,
            initial_string="",
            font_family="monospace",
            font_size=35,
            antialias=True,
            text_color=(0, 0, 0),
            cursor_color=(0, 0, 1),
            repeat_keys_initial_ms=400,
            repeat_keys_interval_ms=35):
        """
        :param initial_string: Initial text to be displayed
        :param font_family: name or list of names for font (see pygame.font.match_font for precise format
)
        :param font_size:  Size of font in pixels
        :param antialias: Determines if antialias is applied to font (uses more processing power)
        :param text_color: Color of text (duh)
        :param cursor_color: Color of cursor
        :param repeat_keys_initial_ms: Time in ms before keys are repeated when held
        :param repeat_keys_interval_ms: Interval between key press repetition when helpd
        """
```

```
41.
42.         # Text related vars:
43.         self.antialias = antialias
44.         self.text_color = text_color
45.         self.font_size = font_size
46.         self.input_string = initial_string  # Inputted text
47.
48.         if not os.path.isfile(font_family):
49.             font_family = pygame.font.match_font(font_family)
50.
51.         self.font_object = pygame.font.Font(font_family, font_size)
52.
53.         # Text-surface will be created during the first update call:
54.         self.surface = pygame.Surface((1, 1))
55.         self.surface.set_alpha(0)
56.
57.         # Vars to make keydowns repeat after user pressed a key for some time:
58.         self.keyrepeat_counters = {}  # {event.key: (counter_int, event.unicode)} (look for "***")
59.         self.keyrepeat_intial_interval_ms = repeat_keys_initial_ms
60.         self.keyrepeat_interval_ms = repeat_keys_interval_ms
61.
62.         # Things cursor:
63.         self.cursor_surface = pygame.Surface((int(self.font_size / 20 + 1), self.font_size))
64.         self.cursor_surface.fill(cursor_color)
65.         self.cursor_position = len(initial_string)  # Inside text
66.         self.cursor_visible = True  # Switches every self.cursor_switch_ms ms
67.         self.cursor_switch_ms = 500  # /|\
68.         self.cursor_ms_counter = 0
69.
70.         self.clock = pygame.time.Clock()
71.
72.     def update(self, events):
73.         for event in events:
74.             if event.type == pygame.KEYDOWN:
75.                 self.cursor_visible = True  # So the user sees where he writes
76.
77.                 # If none exist, create counter for that key:
78.                 if event.key not in self.keyrepeat_counters:
79.                     self.keyrepeat_counters[event.key] = [0, event.unicode]
80.
81.                 if event.key == pl.K_BACKSPACE:
82.                     self.input_string = (
83.                         self.input_string[:max(self.cursor_position - 1, 0)]
84.                         + self.input_string[self.cursor_position:]
85.                     )
86.
87.                     # Subtract one from cursor_pos, but do not go below zero:
88.                     self.cursor_position = max(self.cursor_position - 1, 0)
89.                 elif event.key == pl.K_DELETE:
90.                     self.input_string = (
91.                         self.input_string[:self.cursor_position]
92.                         + self.input_string[self.cursor_position + 1:]
93.                     )
94.
95.                 elif event.key == pl.K_RETURN:
96.                     return True
97.
98.                 elif event.key == pl.K_RIGHT:
99.                     # Add one to cursor_pos, but do not exceed len(input_string)
100.                    self.cursor_position = min(self.cursor_position + 1, len(self.input_string))
101.
102.                elif event.key == pl.K_LEFT:
103.                    # Subtract one from cursor_pos, but do not go below zero:
```

```
104.                     self.cursor_position = max(self.cursor_position - 1, 0)
105.
106.                 elif event.key == pl.K_END:
107.                     self.cursor_position = len(self.input_string)
108.
109.                 elif event.key == pl.K_HOME:
110.                     self.cursor_position = 0
111.
112.                 else:
113.                     # If no special key is pressed, add unicode of key to input_string
114.                     self.input_string = (
115.                         self.input_string[:self.cursor_position]
116.                         + event.unicode
117.                         + self.input_string[self.cursor_position:]
118.                     )
119.                     self.cursor_position += len(event.unicode)  # Some are empty, e.g. K_UP
120.
121.             elif event.type == pl.KEYUP:
122.                 # *** Because KEYUP doesn't include event.unicode, this dict is stored in such a weird wa
    y
123.                 if event.key in self.keyrepeat_counters:
124.                     del self.keyrepeat_counters[event.key]
125.
126.         # Update key counters:
127.         for key in self.keyrepeat_counters:
128.             self.keyrepeat_counters[key][0] += self.clock.get_time()  # Update clock
129.
130.             # Generate new key events if enough time has passed:
131.             if self.keyrepeat_counters[key][0] >= self.keyrepeat_intial_interval_ms:
132.                 self.keyrepeat_counters[key][0] = (
133.                     self.keyrepeat_intial_interval_ms
134.                     - self.keyrepeat_interval_ms
135.                 )
136.
137.                 event_key, event_unicode = key, self.keyrepeat_counters[key][1]
138.                 pygame.event.post(pygame.event.Event(pl.KEYDOWN, key=event_key, unicode=event_unicode))
139.
140.         # Re-render text surface:
141.         self.surface = self.font_object.render(self.input_string, self.antialias, self.text_color)
142.
143.         # Update self.cursor_visible
144.         self.cursor_ms_counter += self.clock.get_time()
145.         if self.cursor_ms_counter >= self.cursor_switch_ms:
146.             self.cursor_ms_counter %= self.cursor_switch_ms
147.             self.cursor_visible = not self.cursor_visible
148.
149.         if self.cursor_visible:
150.             cursor_y_pos = self.font_object.size(self.input_string[:self.cursor_position])[0]
151.             # Without this, the cursor is invisible when self.cursor_position > 0:
152.             if self.cursor_position > 0:
153.                 cursor_y_pos -= self.cursor_surface.get_width()
154.             self.surface.blit(self.cursor_surface, (cursor_y_pos, 0))
155.
156.         self.clock.tick()
157.         return False
158.
159.     def get_surface(self):
160.         return self.surface
161.
162.     def get_text(self):
163.         return self.input_string
164.
165.     def get_cursor_position(self):
```

```
166.         return self.cursor_position
167.
168.     def set_text_color(self, color):
169.         self.text_color = color
170.
171.     def set_cursor_color(self, color):
172.         self.cursor_surface.fill(color)
173.
174.     def clear_text(self):
175.         self.input_string = ""
176.         self.cursor_position = 0
```