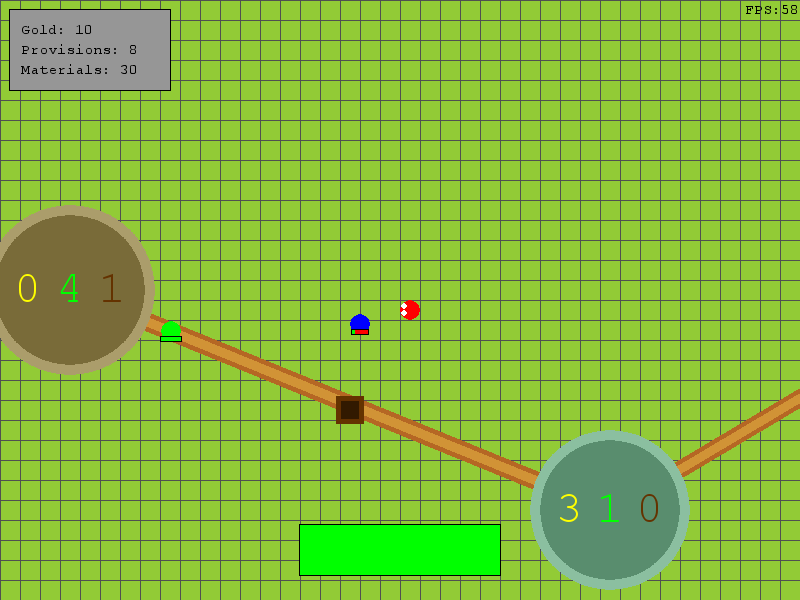
Highwayman

Jonas Mariager Jakobsen

Programmerings synopsis

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

Indhold

[Indledning 3](#_Toc8420293)

[Programmets opbygning 3](#_Toc8420294)

[Verden generation 3](#_Toc8420295)

[Eksempler på game-klasse funktionalitet 5](#_Toc8420296)

[Bevægelse 5](#_Toc8420297)

[Fælder 6](#_Toc8420298)

[Trade\_unit-klassen - Handelsmænd og vagter 7](#_Toc8420299)

[Andet 10](#_Toc8420300)

[Evaluering efter projektets færdiggørelse 11](#_Toc8420301)

[Test 11](#_Toc8420302)

[Bilag 12](#_Toc8420303)

[Andre billeder 12](#_Toc8420304)

[Diagrammer 12](#_Toc8420305)

[Video 14](#_Toc8420306)

[Kode 15](#_Toc8420307)

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](https://stackoverflow.com/users/4303737/tgirod)” på en af hans [opslag](https://stackoverflow.com/questions/42147776/producing-2d-perlin-noise-with-numpy).



Fig. 1 Normaliserende funktion (kode)

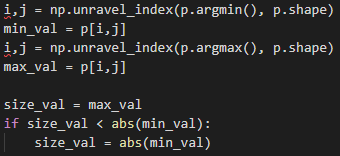


Fig. 2 Finder numerisk største værdi

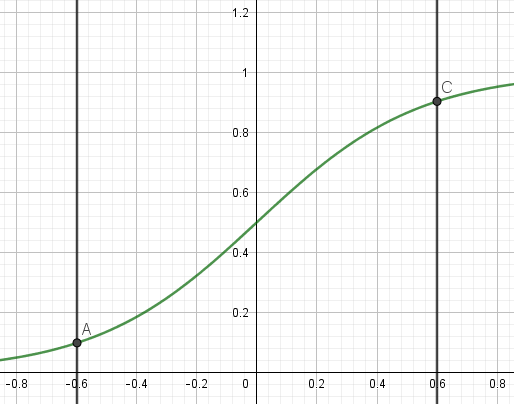


Fig. 3 Normalisering GeoGebra repræsentation

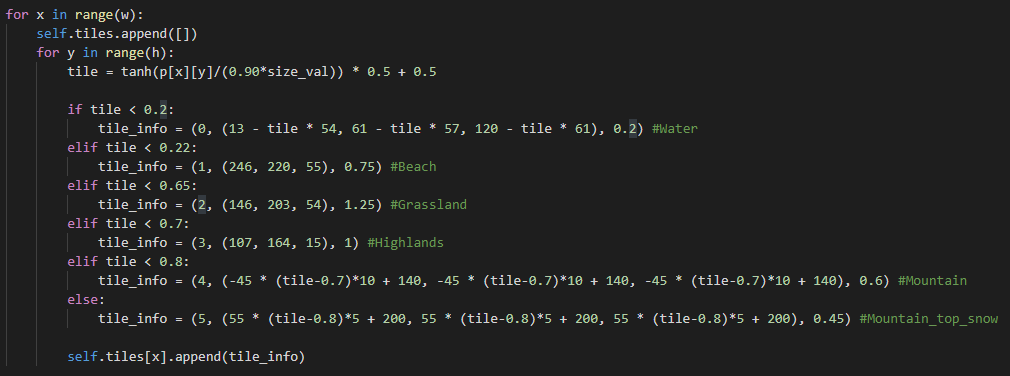


Fig. 4 Tildeling af felt karakteristika



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.

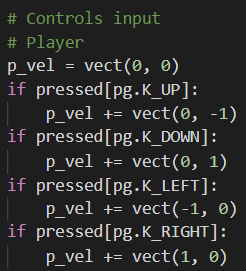


Fig. 6 Input håndtering

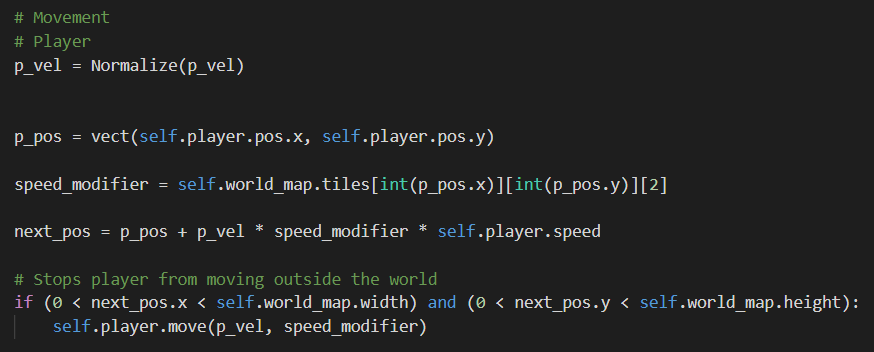


Fig. 7 Aktuel bevægelse af spilleren

### 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.

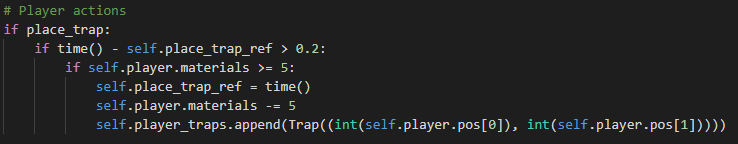


Fig. 8 Placering af fælder

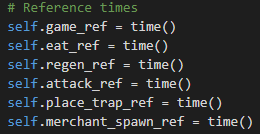


Fig. 9 Variabler der holder styr på tid gået

## 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).

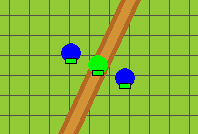


Fig. 10 En handelsmand med 2 vagter. Vagterne er blå

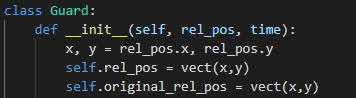


Fig. 11 Guard-klassen bliver oprettet med relativ og original position

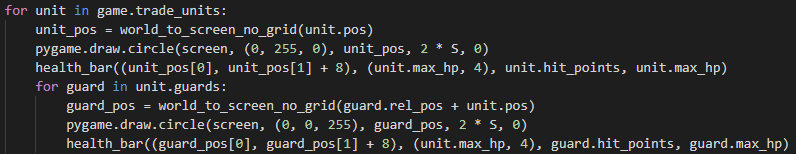


Fig. 12 Udklip fra "Main" Her tegnes handelsmænd og vagter

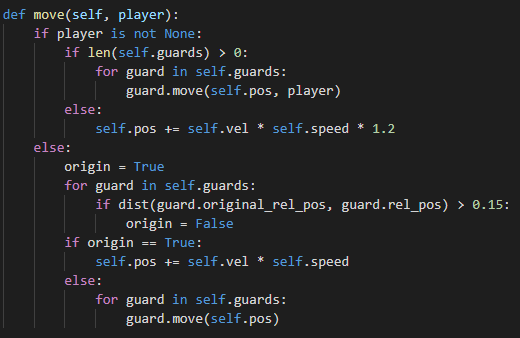


Fig. 13 move funktionen for Trade\_unit-klassen

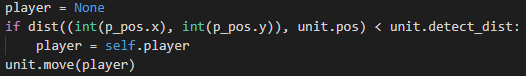


Fig. 14 Udklip der kører move funktionen for Trade\_unit-objektet og sender enten Player objektet eller None

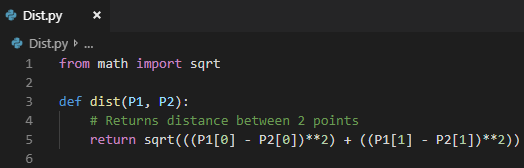


Fig. 15 Funktion som er blevet placeret i sin egen fil for at kunne blive anvendte af flere forskellige filer

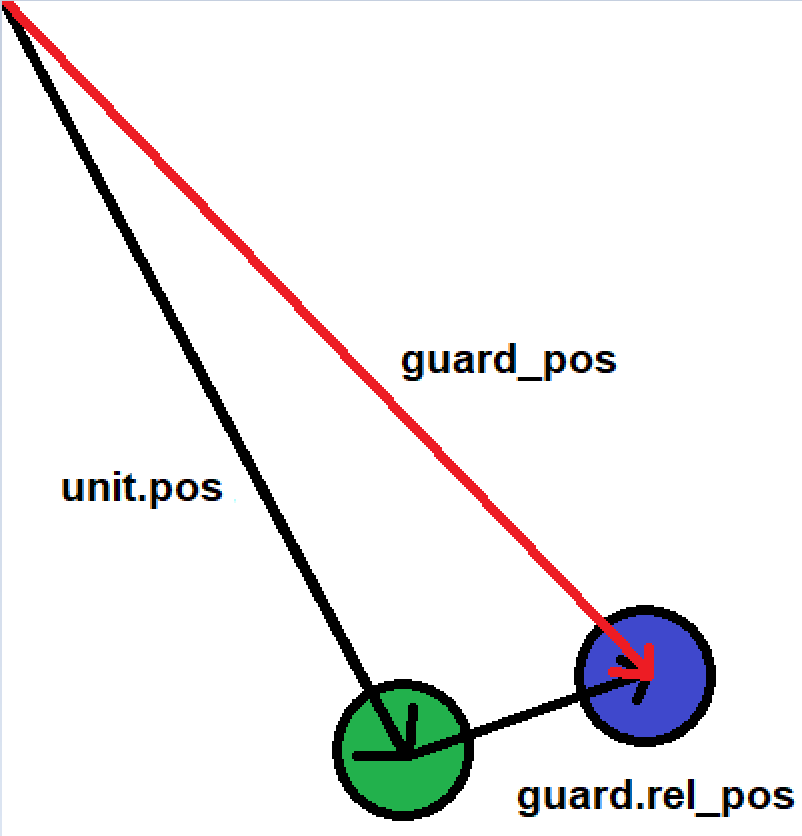


Fig. 16 Vektor udregning for en vagt

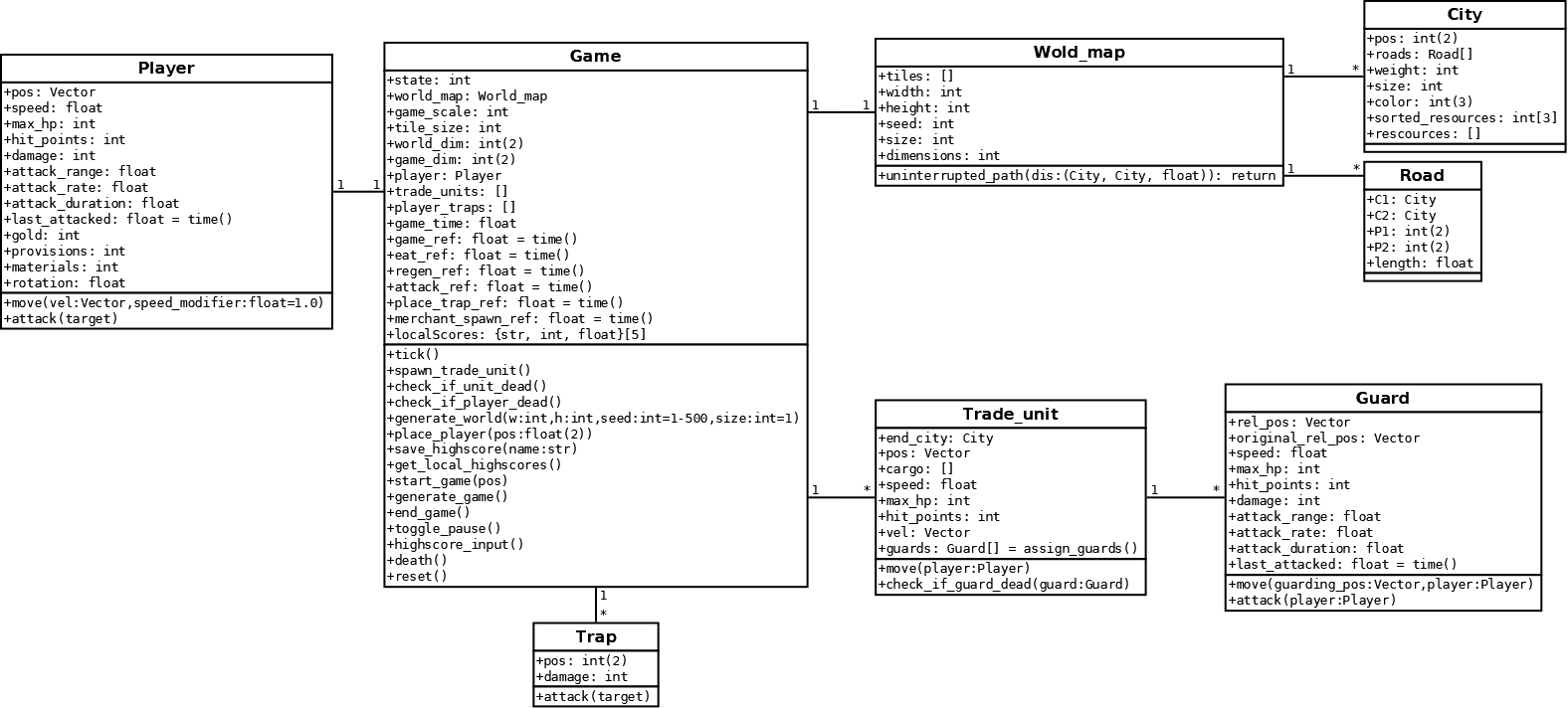


Fig. 17 Sammenhæng mellem Game-, Trade\_unit- og Guard-klassen

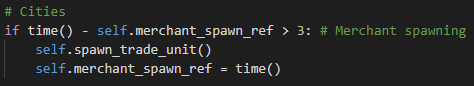


Fig. 18 spawn\_trade\_unit-funktionen bliver kørt hvert 3. sekund

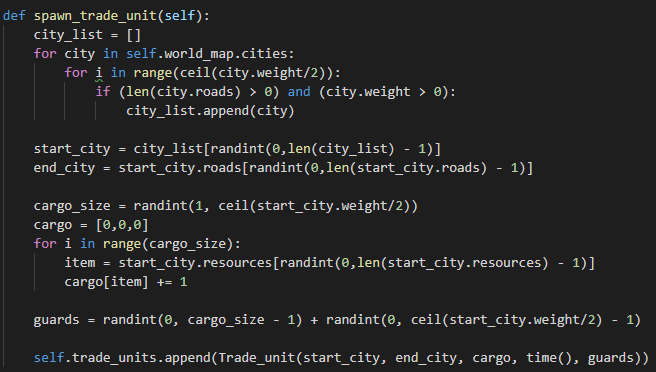


Fig. 19 Del 1 af spawn\_trade\_unit

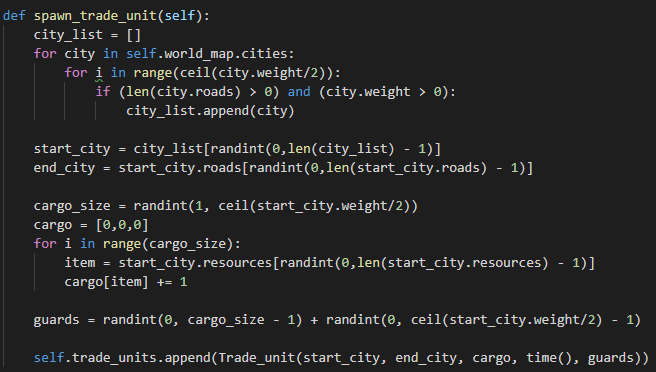


Fig. 20 Del 2 af spawn\_trade\_unit

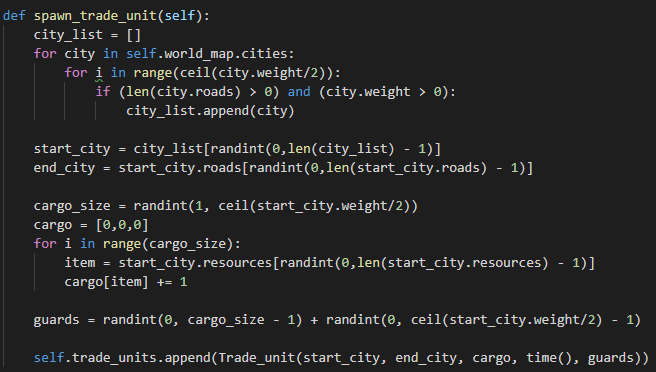


Fig. 21 Del 3 af spawn\_trade\_unit

## 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

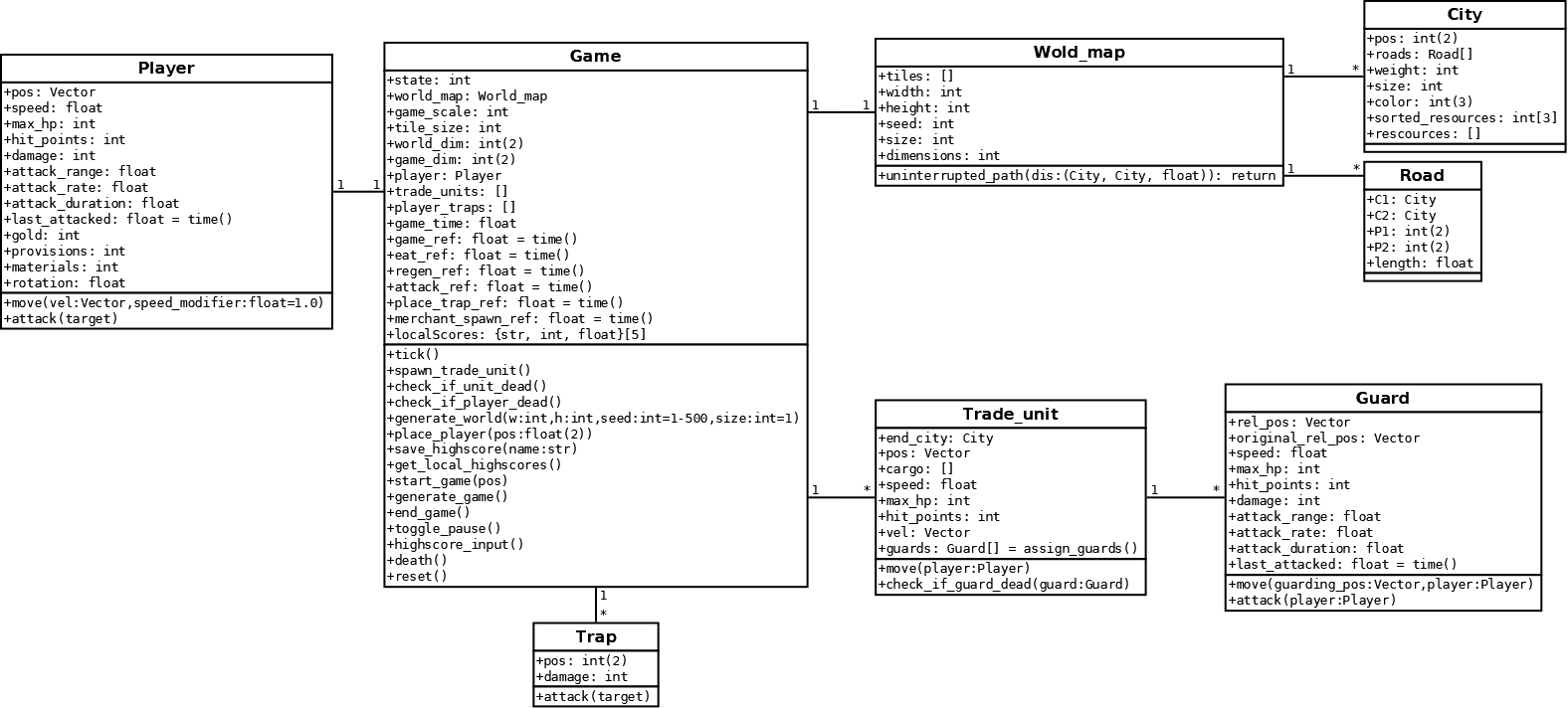


Fig. 23 Samlet klassediagrammet

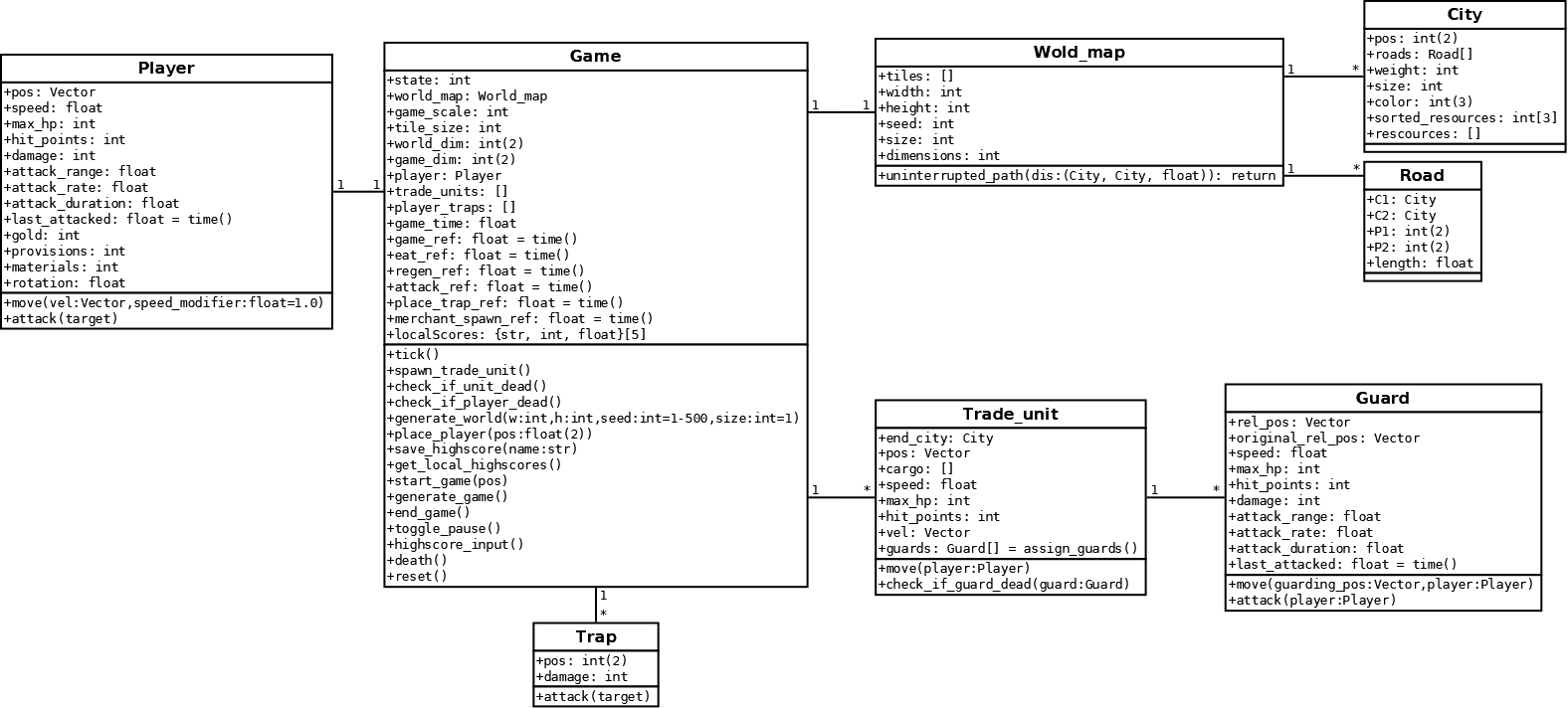


Fig. 24 Venstre side af klassediagrammet

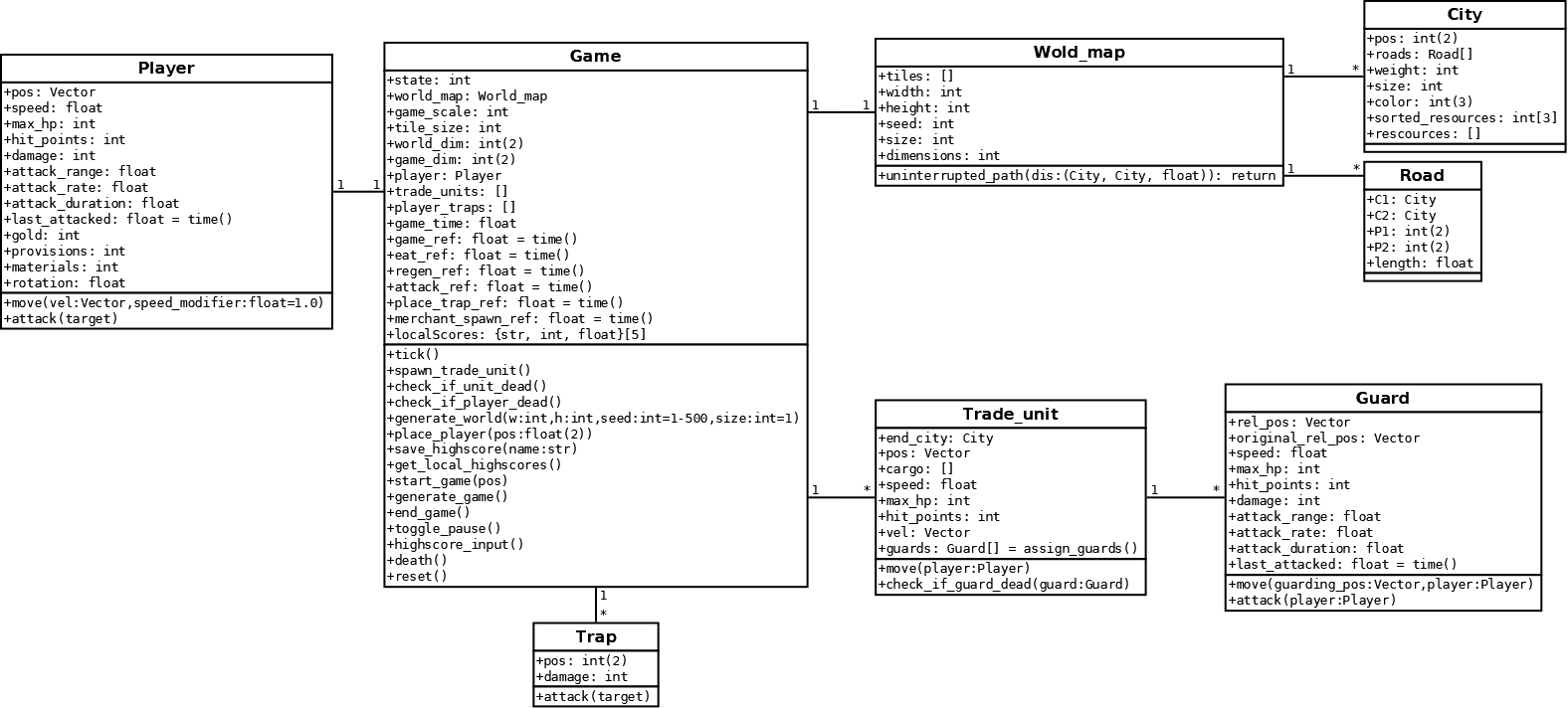


Fig. 25 Højre side af klassediagrammet

## Video

Kort test af spillet: [http://youtu.be/M4AzjC7Kj6k](http://youtu.be/M4AzjC7Kj6k?hd=1)

## 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


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)
22. running = True
24. game = Game(pygame.display.Info())
26. textinput = pygame\_textinput.TextInput()
28. clock = pygame.time.Clock()

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(' ')
37. **if** game.state == 0:

40. pygame.draw.rect(screen, (0, 0, 0), pygame.Rect(0, 0, w, h))
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))
46. **elif** game.state == 0.5:
48. map = game.world\_map # World\_map object
49. ts = game.tile\_size # size of an individual tile in pixels
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), (road.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), (road.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)


64. **elif** game.state == 1:
65. screen.fill((100, 100, 100))
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))
78. lB = 0 # LeftBoundary
79. rB = 0 # RightBoundary
80. tB = 0 # TopBoundary
81. bB = 0 # BottomBoundary
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

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))
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))
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))

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))
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.max\_hp)
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))
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)

149. # Hud
150. # Info
152. screen.blit(small\_font.render("FPS:{}".format(str(int(clock.get\_fps()))), 1, (0, 0, 0)), (w - Small\_size[0] \* 6 - 1, int(S \* ts \* 1/2 - Small\_size[1]//2)))
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)), (int(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)), (int(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)), (int(S \* ts \* 3/2 - Small\_size[0]), int(S \* ts \* 7/2 - Small\_size[1]//2)))
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)
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))
169. **if** game.state == 2 **or** game.state == 0:
171. controls = ["Movement: WASD", "Attack: Spacebar","Place trap: t" , "Pause: p", "Exit Game/New Game: ESC", "Sumbmit Score: Enter"]
172. pygame.draw.rect(screen, (30, 30, 30), pygame.Rect(w//2 - 150, 400, 300, (len(controls) + 1) \* (Small\_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)))

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']) + ' Time: ' + str(int(score['Time']/60)) + ':' + str(int(score['Time'] % 60)), 1, (255, 255, 255)), (w//2 - 175 + 5, 120 + (i + 1) \* (Small\_size[1] + 3)))

185. **elif** game.state == 3:
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()



197. **def** screen\_to\_world(pos):
198. **return** (int(pos[0]/game.tile\_size), int(pos[1]/game.tile\_size))
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]))

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()
224. pressed = pygame.key.get\_pressed()
226. game.tick(pygame, pressed)
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


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
21. # Worldmap variabels
22. self.world\_map = None
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)
29. # Game dependant variabels
30. self.player = None
32. self.trade\_units = []
34. self.player\_traps = []
36. self.game\_time = 0
38. # Highscores
39. self.localScores = self.get\_local\_highscores()[:5]

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)
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()
55. **if** self.state == 1:
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

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]))))
86. # Movement
87. # Player
88. p\_vel = Normalize(p\_vel)

91. p\_pos = vect(self.player.pos.x, self.player.pos.y)
93. speed\_modifier = self.world\_map.tiles[int(p\_pos.x)][int(p\_pos.y)][2]
95. next\_pos = p\_pos + p\_vel \* speed\_modifier \* self.player.speed
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)
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)
112. **for** unit **in** to\_pop[::-1]:
113. self.trade\_units.remove(unit)
115. # Attacks
116. # Player
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.player.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, angle\_to\_vector(self.player.rotation))) < pi/3:
129. self.player.attack(guard)
130. unit.check\_if\_guard\_dead(guard)

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()
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**

162. # Time Stuff
164. self.game\_time = time() - self.game\_ref
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()
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()
180. # Cities
181. **if** time() - self.merchant\_spawn\_ref > 3: # Merchant spawning
182. self.spawn\_trade\_unit()
183. self.merchant\_spawn\_ref = time()

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)
193. start\_city = city\_list[randint(0,len(city\_list) - 1)]
194. end\_city = start\_city.roads[randint(0,len(start\_city.roads) - 1)]
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
202. guards = randint(0, cargo\_size - 1) + randint(0, ceil(start\_city.weight/2) - 1)
204. self.trade\_units.append(Trade\_unit(start\_city, end\_city, cargo, time(), guards))
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)
213. **def** check\_if\_player\_dead(self):
214. **if** self.player.hit\_points <= 0:
215. self.death()

218. **def** generate\_world(self, w, h, seed=randint(1, 500), size=1):
219. self.world\_map = World\_map(w, h, seed, size)

222. **def** place\_player(self, pos):
223. self.player = Player(pos[0], pos[1], time())

226. **def** save\_highscore(self, name):
227. #Pickle database
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)
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()
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
292. **def** start\_game(self, pos):
293. **if** self.state == 0.5:
294. self.place\_player(pos)
295. self.state = 1
297. **def** generate\_game(self):
298. **if** self.state == 0:
299. self.state = 0.5
301. self.generate\_world(self.world\_dim[0],self.world\_dim[1],randint(1,500),3)

304. **def** end\_game(self):
305. **if** self.state > 0:
306. self.state = 0
308. **def** toggle\_pause(self):
309. **if** self.state == 1:
310. self.state = 2
311. **elif** self.state == 2:
312. self.state = 1
314. **def** highscore\_input(self):
315. **if** self.state == 1:
316. self.state = 3
318. **def** death(self):
319. **if** self.state == 1:
320. self.highscore\_input()
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


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
20. self.cities = []
21. self.roads = []
23. #Confine variabels
24. **if** size < 1:
25. size = 1
26. **elif** size > 10:
27. size = 10
29. self.dimensions = w
30. **if** w < h:
31. self.dimensions = h
33. #Generate map
35. p = pGrid(size, self.dimensions, self.seed)
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]
43. size\_val = max\_val
44. **if** size\_val < abs(min\_val):
45. size\_val = abs(min\_val)
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
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
65. self.tiles[x].append(tile\_info)
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))
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 cities and distance: (c1, c2, float: 'distance') and adds them to 'dists' list
86. connections = [[]] # Connected cites
87. unfound = self.cities.copy() # Unconnected cities
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**
97. **else**:
98. unfound.pop(unfound.index(un))


102. **while** len(unfound) != 0: #Keeps going until there are no more unfound citites (all cities are found)
104. # Finds the shortest connection which has both a connection to 'connected' and to 'unfound' insuring a correct connection
105. cons = []
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)
113. s\_dist = shortets\_dist(cons)
114. **if** s\_dist **is** **not** None:
115. connections[-1].append(s\_dist)  # Add new connection
117. i = 0                           # Delete city from unfound
118. **if** s\_dist[1] **in** unfound:        #
119. i = 1                       #
120. unfound.remove(s\_dist[i])       #

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)
130. s\_dist = shortets\_dist(cons)
131. **if** s\_dist **is** **not** None:
133. connections[-1].append(s\_dist)  # Add to new connection to connections
134. unfound.remove(s\_dist[0])       # and
135. unfound.remove(s\_dist[1])       # Remove now found cities
137. **else**:
138. unfound[:] = []
140. **else**:
141. **break**

144. **for** con **in** connections:                             # Create road objects and add to self.roads
145. **for** c **in** con:                                   #
146. self.roads.append(Road(c[0], c[1], c[2]))   #
147. self.cities[self.cities.index(c[0])].roads.append(c[1]) # Give cities connected cities as "roads"
148. self.cities[self.cities.index(c[1])].roads.append(c[0]) #
150. **def** uninterrupted\_path(self, dis):
151. c1pos, c2pos = dis[0].pos, dis[1].pos
152. V = vect(c2pos[0] - c1pos[0], c2pos[1] - c1pos[1])
153. length = Length(V)
154. V = Normalize(V)
155. pos\_V = vect(c1pos[0], c1pos[1])
156. uninterrupted = True
157. **for** i **in** range(int(length)):
158. Pos = pos\_V + V \* i
159. tile = self.tiles[int(Pos[0])][int(Pos[1])]
160. **if** **not** (2 <= tile[0] <= 3):
161. uninterrupted = False
162. **return** uninterrupted

165. **def** shortets\_dist(dists):
166. # Finds the 'dist' touple with the smallest distance value
167. shortest = None
168. **for** d **in** dists:
169. **if** shortest == None:
170. shortest = d
171. **elif** d[2] < shortest[2]:
172. shortest = d
173. **return** shortest

Perlin (Bibliotek) - perlingrid (egen funktion)

1. **import** numpy as np
3. **def** perlin(x,y,seed=0):
4. # permutation table
5. np.random.seed(seed)
6. p = np.arange(256,dtype=int)
7. np.random.shuffle(p)
8. p = np.stack([p,p]).flatten()
9. # coordinates of the top-left
10. xi = x.astype(int)
11. yi = y.astype(int)
12. # internal coordinates
13. xf = x - xi
14. yf = y - yi
15. # fade factors
16. u = fade(xf)
17. v = fade(yf)
18. # noise components
19. n00 = gradient(p[p[xi]+yi],xf,yf)
20. n01 = gradient(p[p[xi]+yi+1],xf,yf-1)
21. n11 = gradient(p[p[xi+1]+yi+1],xf-1,yf-1)
22. n10 = gradient(p[p[xi+1]+yi],xf-1,yf)
23. # combine noises
24. x1 = lerp(n00,n10,u)
25. x2 = lerp(n01,n11,u)
26. **return** lerp(x1,x2,v)
28. **def** lerp(a,b,x):
29. "linear interpolation"
30. **return** a + x \* (b-a)
32. **def** fade(t):
33. "6t^5 - 15t^4 + 10t^3"
34. **return** 6 \* t\*\*5 - 15 \* t\*\*4 + 10 \* t\*\*3
36. **def** gradient(h,x,y):
37. "grad converts h to the right gradient vector and return the dot product with (x,y)"
38. vectors = np.array([[0,1],[0,-1],[1,0],[-1,0]])
39. g = vectors[h%4]
40. **return** g[:,:,0] \* x + g[:,:,1] \* y
42. **def** perlingrid(b, res, seed):
44. lin = np.linspace(0,b,res,endpoint=False)
45. x,y = np.meshgrid(lin,lin)
47. **return** perlin(x,y,seed)

City

1. **from** random **import** randint
2. **from** math **import** ceil
4. **class** City:
5. **def** \_\_init\_\_(self, pos):
6. self.pos = pos
7. self.roads = []
8. self.weight = randint(0,9)
9. self.size = self.weight + 8
10. start\_color = randint(100,215)
11. self.color = (start\_color + randint(0,80) - 40, start\_color + randint(0,80) - 40, start\_color + randint(0,80) - 40)
12. self.sorted\_resources = [0,0,0]
13. **for** i **in** range(ceil(self.weight/2)):
14. self.sorted\_resources[randint(0,2)] += 1
15. self.resources = []
16. **for** i **in** range(3):
17. **for** j **in** range(self.sorted\_resources[i]):
18. self.resources.append(i)


22. **def** \_\_repr\_\_(self): #For debugging purposes
23. **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
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

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)
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
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
17. self.vel = Normalize(e\_pos - s\_pos)
18. self.guards = []
19. **if** guards > 0:
20. self.guards = assign\_guards(guards, self.vel, time)

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)
41. **def** check\_if\_guard\_dead(self, guard):
42. **if** guard.hit\_points <= 0:
43. self.guards.remove(guard)


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
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
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
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
6. **def** attack(self, target):
7. target.hit\_points -= self.damage

Dist

1. **from** math **import** sqrt
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)
8. **def** \_\_add\_\_(self, val):
9. **return** Point( self[0] + val[0], self[1] + val[1] )
11. **def** \_\_sub\_\_(self,val):
12. **return** Point( self[0] - val[0], self[1] - val[1] )
14. **def** \_\_iadd\_\_(self, val):
15. self.x = val[0] + self.x
16. self.y = val[1] + self.y
17. **return** self
19. **def** \_\_isub\_\_(self, val):
20. self.x = self.x - val[0]
21. self.y = self.y - val[1]
22. **return** self
24. **def** \_\_div\_\_(self, val):
25. **return** Point( self[0] / val, self[1] / val )
27. **def** \_\_mul\_\_(self, val):
28. **return** Point( self[0] \* val, self[1] \* val )
30. **def** \_\_idiv\_\_(self, val):
31. self[0] = self[0] / val
32. self[1] = self[1] / val
33. **return** self
35. **def** \_\_imul\_\_(self, val):
36. self[0] = self[0] \* val
37. self[1] = self[1] \* val
38. **return** self
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")
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")
56. **def** \_\_str\_\_(self):
57. **return** "(" + str(self.x) + "," + str(self.y) + ")"
58. Point = Vector
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)

1. **from** math **import** atan2, pi, acos, sqrt, cos, sin
2. **from** Vector **import** Vector, Dot, LengthSqrd
4. **def** vect\_to\_angle(v):
5. x,y = v[0],v[1]
7. **if** x == 0 **and** y == 0:
8. **return** 0
9. **else**:
10. **return** atan2(y,x)
12. **def** vectors\_to\_angle(v1, v2):
13. **return** acos(Dot(v1,v2)/(sqrt(LengthSqrd(v1)\*LengthSqrd(v2))))
15. **def** angle\_to\_vector(a):
16. **return** Vector(cos(a), sin(a))

pygame\_textinput (bibliotek)

1. """
2. Copyright 2017, Silas Gyger, silasgyger@gmail.com, All rights reserved.
4. Borrowed from https://github.com/Nearoo/pygame-text-input under the MIT license.
5. """
7. **import** os.path
9. **import** pygame
10. **import** pygame.locals as pl
12. pygame.font.init()

15. **class** TextInput:
16. """
17. This class lets the user input a piece of text, e.g. a name or a message.
18. This class let's the user input a short, one-lines piece of text at a blinking cursor
19. that can be moved using the arrow-keys. Delete, home and end work as well.
20. """
21. **def** \_\_init\_\_(
22. self,
23. initial\_string="",
24. font\_family="monospace",
25. font\_size=35,
26. antialias=True,
27. text\_color=(0, 0, 0),
28. cursor\_color=(0, 0, 1),
29. repeat\_keys\_initial\_ms=400,
30. repeat\_keys\_interval\_ms=35):
31. """
32. :param initial\_string: Initial text to be displayed
33. :param font\_family: name or list of names for font (see pygame.font.match\_font for precise format)
34. :param font\_size:  Size of font in pixels
35. :param antialias: Determines if antialias is applied to font (uses more processing power)
36. :param text\_color: Color of text (duh)
37. :param cursor\_color: Color of cursor
38. :param repeat\_keys\_initial\_ms: Time in ms before keys are repeated when held
39. :param repeat\_keys\_interval\_ms: Interval between key press repetition when helpd
40. """
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
48. **if** **not** os.path.isfile(font\_family):
49. font\_family = pygame.font.match\_font(font\_family)
51. self.font\_object = pygame.font.Font(font\_family, font\_size)
53. # Text-surface will be created during the first update call:
54. self.surface = pygame.Surface((1, 1))
55. self.surface.set\_alpha(0)
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
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
70. self.clock = pygame.time.Clock()
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
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]
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. )
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. )
95. **elif** event.key == pl.K\_RETURN:
96. **return** True
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))
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)
106. **elif** event.key == pl.K\_END:
107. self.cursor\_position = len(self.input\_string)
109. **elif** event.key == pl.K\_HOME:
110. self.cursor\_position = 0
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
121. **elif** event.type == pl.KEYUP:
122. # \*\*\* Because KEYUP doesn't include event.unicode, this dict is stored in such a weird way
123. **if** event.key **in** self.keyrepeat\_counters:
124. **del** self.keyrepeat\_counters[event.key]
126. # Update key counters:
127. **for** key **in** self.keyrepeat\_counters:
128. self.keyrepeat\_counters[key][0] += self.clock.get\_time()  # Update clock
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. )
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))
140. # Re-render text surface:
141. self.surface = self.font\_object.render(self.input\_string, self.antialias, self.text\_color)
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
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))
156. self.clock.tick()
157. **return** False
159. **def** get\_surface(self):
160. **return** self.surface
162. **def** get\_text(self):
163. **return** self.input\_string
165. **def** get\_cursor\_position(self):
166. **return** self.cursor\_position
168. **def** set\_text\_color(self, color):
169. self.text\_color = color
171. **def** set\_cursor\_color(self, color):
172. self.cursor\_surface.fill(color)
174. **def** clear\_text(self):
175. self.input\_string = ""
176. self.cursor\_position = 0