

Introduksjon til objektorientert programmering

Vi lager et rollespill

Innholdet i prosjektet

- 1) Hva er objektorientering?
- 2) Hva er rollespill?
- 3) Objektorientert modellering, med eksempler fra rollespill
- 4) Spillmotoren
- 5) Klasser og metoder i spillmotoren
- 6) Utviklingsmetodikk
- 7) Prosjektarbeid i grupper

1) Objektorientert programmering

- ❖ Fra Wikipedia: Konseptet stammer fra arbeidet nordmennene Kristen Nygaard og Ole-Johan Dahl gjorde ved Norsk Regnesentral med programmeringsspråket Simula på 1960-tallet, noe de ble belønnet med både Turing-prisen^{[1][2]} og John von Neumann-medaljen for. OOP-konseptet fikk stor utbredelse gjennom bruk i andre programeringsspråk, Smalltalk i 1970-årene, C++ i 1980-årene og Java i 1990-årene.
- ❖ Følgende prinsipper er sentrale i OOP:
 - ❖ *Objekter* – pakke data og funksjonalitet sammen i enheter i programmet. Dette er basis for *modularitet*, en av kvalitetene man prøver å oppnå.
 - ❖ *Abstraksjon* – gjøre at programmereren underveis kan ignorere noen av detaljene ved implementasjon av det som jobbes med.
 - ❖ *Innkapsling* – skjule den interne tilstanden til et objekt fra andre. Dette gjør at utenforstående kode ikke kan endre på tilstanden til objektet på uforutsette måter.
 - ❖ *Polymorfi* – gjøre at et objekt kan oppføre seg som et annet, bare den oppfyller den «kontrakten» grensesnittet spesifiserer.
 - ❖ *Arv* – lette arbeidet med innkapsling og polymorfi ved å tillate programmereren å lage objekter som er mer spesialiserte utgaver av andre objekter.
- ❖ De fleste mest brukte programeringsspråkene i dag benytter seg av en objektorientert programmeringsmodell.

Objektorientert programmering - objekter

- ❖ Objekter
 - ❖ Som navnet tilsier, handler objektorientert programmering, om å programmere *objekter*. Et objekt sier vi gjerne er en forekomst (instans) av en *klasse*.
 - ❖ En *klasse* kan sees på som en oppskrift for en objekt. Klassen definere hvilke egenskaper objektet skal få. En klasse kan også ha *metoder*. Dette kan vi se på som hvilke handlinger et objekt kan utføre.
 - ❖ I spillet vårt vil vi se at vi har en klasse for selve spilleren (Character). Denne klassen gir spilleren noen egenskaper som f.eks et navn, helsepoeng, styrke, smidighet osv. Spillerklassen har en rekke metoder. Den har f.eks metoder for bevegelse, det å plukke opp ting og lignende. Den har også mer abstrakte metoder, som f.eks å sjekke om den kan bevege seg til et gitt sted.

Objektorientert programmering - abstraksjon

- ❖ Abstraksjon
 - ❖ Abstraksjon kan bety at en skiller ut komplekse oppgaver til egne metoder. Dette betyr at en f.eks kan vente med å skrive mer avanserte oppgaver til en har fått på plass mer sentrale oppgaver i programmet.
 - ❖ I spillet vårt er f.eks kartklassen (TileMap) full av abstraksjoner. Denne klassen har metoder for å gjøre om et langt array av tall til et spillbrett vi kan se på skjermen. En kunne strengt tatt gjort ferdig resten av spillet før en begynte på dette, men litt av poenget med oppgaven jeg gir dere nå, er at de mest komplekse abstraksjonene allerede er ferdige, slik at vi kan konsentrere oss om det gøye.

Objektorientert programmering - innkapsling

- ❖ Innkapsling dreier seg om å skjule den interne tilstanden til et objekt fra resten av koden. Dette er viktig for at en eller annen funksjon plutselig endrer en viktig variabel et annet sted i koden.
- ❖ Eksempler på dette kan f.eks være encounters-objektet vårt. Hver encounter har sin egen egenskap «visited», som sier spillet om vi har vært et sted eller ikke. Dersom dette hadde vært definert i en global variabel, kunne det fort blitt kluss. I stedet må vi kjenne til objektet vi behandler, og presist bruke dotnotasjon for å endre visited for et gitt encounter til true. Dersom vi ikke er i et gitt instans av et encounter, kan vi heller ikke endre visited med metodene i encounter-klassen.

Objektorientert programmering – polymorfi

- ❖ Polymorfi har vi ikke gode eksempler på i spillet vårt.
- ❖ Et klassisk eksempel på dette er at dersom en har en klasse for *former*, vil denne være polymorf, dersom en har en *areal*-metode som beregner riktig areal uavhengig av hvilken form et initiert objekt har.
- ❖ Dette innebærer at form-klassen må ha egenskaper som ikke nødvendigvis er i bruk i hvert form-objekt, men som heller ikke er i veien når arealet skal regnes ut.

Objektorientert programmering

- arv

- ❖ Arv innebærer at en klasse kan «arve» egenskaper og metoder fra en annen klasse.
- ❖ Dette kan være nyttig for å gjenbruke kode som kan være relevant for mange ulike klasser.
- ❖ I vårt spill har vi f.eks levende-klassen, som gir alle levende ting egenskapene styrke, smidighet, utholdenthet, visdom, intelligens og karisme.
- ❖ Monster-klassen og Character-klassen arver disse egenskapene fra levende, slik at vi slipper å programmere det på nytt to ganger.

Objektorientering i JavaScript

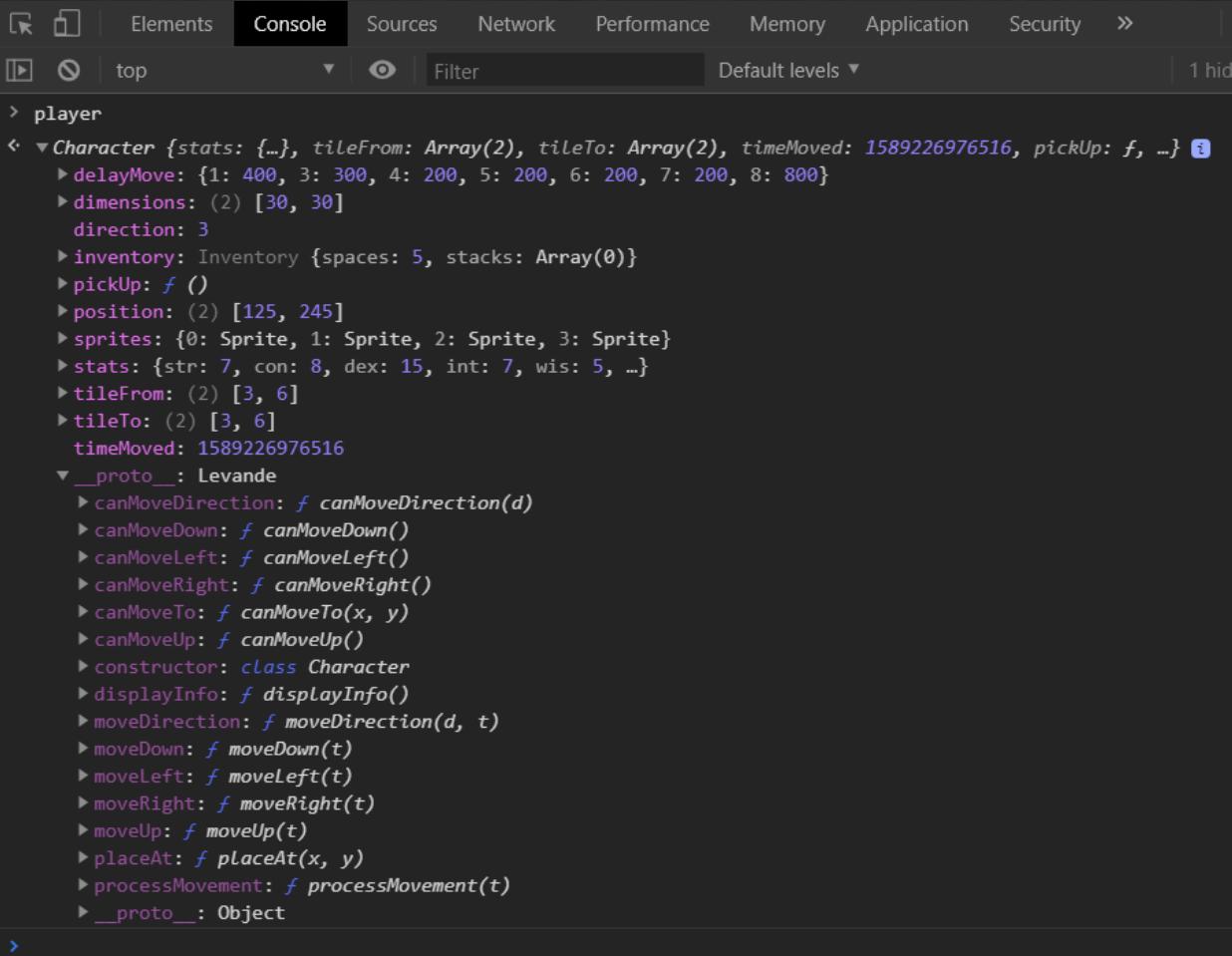
- ❖ JavaScript har ikke hatt ordentlig støtte for objektorientering før siste versjon (ES6 – EmmaScript 6)
- ❖ Med ES6 fikk vi støtte for class, og skikkelig arv. Det gikk an å emulere dette i tidligere versjoner, men det har gjerne blitt litt uglesett blant ordentlige OOP-entusiaster (og er det kanskje fremdeles)
- ❖ En klasse har denne syntaksen:

```
class MinKlasse {
    constructor(egenskap) {
        this.egenskap = egenskap;
    }
    minMetode(){
        alert(this.egenskap);
    }
}
```

Merk at klassenavn begynner med stor bokstav, i motsetning til variabelnavn som skrives med liten forbokstav og camelCase. Det fungerer å skrive på andre måter, men vi kan godt lære oss til å følge denne konvensjonen.

Objektorientering i JavaScript fortsetter

- ❖ Vi kan inspisere objekter ved å skrive navnet på en instans i konsollen.



The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. At the top, there's a toolbar with icons for back, forward, refresh, and search, followed by tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, and a 'More' button. Below the toolbar, the URL bar shows 'top' and a 'Filter' input field. A dropdown menu says 'Default levels' with '1 hidden' selected. The main area displays the object 'player' with its properties and methods. The object structure is as follows:

```
> player
  ↳ Character {stats: {...}, tileFrom: Array(2), tileTo: Array(2), timeMoved: 1589226976516, pickUp: f, ...}
    > delayMove: {1: 400, 3: 300, 4: 200, 5: 200, 6: 200, 7: 200, 8: 800}
    > dimensions: (2) [30, 30]
    > direction: 3
    > inventory: Inventory {spaces: 5, stacks: Array(0)}
    > pickUp: f ()
    > position: (2) [125, 245]
    > sprites: {0: Sprite, 1: Sprite, 2: Sprite, 3: Sprite}
    > stats: {str: 7, con: 8, dex: 15, int: 7, wis: 5, ...}
    > tileFrom: (2) [3, 6]
    > tileTo: (2) [3, 6]
    > timeMoved: 1589226976516
    ↳ __proto__: Levande
      > canMoveDirection: f canMoveDirection(d)
      > canMoveDown: f canMoveDown()
      > canMoveLeft: f canMoveLeft()
      > canMoveRight: f canMoveRight()
      > canMoveTo: f canMoveTo(x, y)
      > canMoveUp: f canMoveUp()
      > constructor: class Character
      > displayInfo: f displayInfo()
      > moveDirection: f moveDirection(d, t)
      > moveDown: f moveDown(t)
      > moveLeft: f moveLeft(t)
      > moveRight: f moveRight(t)
      > moveUp: f moveUp(t)
      > placeAt: f placeAt(x, y)
      > processMovement: f processMovement(t)
      > __proto__: Object
```

Merk listen over metoder tilgjengelig under `__proto__`

Objektorientering i JavaScript fortsetter

- ❖ Vi kan få verdien til en egenskap ved å bruke dot-notasjon

```
> player.stats.str  
< 7  
>
```

- ❖ En kaller også metoder gjennom dot-notasjon

```
> player.displayInfo()  
< undefined  
> |
```

- ❖ En ny instans blir initiert gjennom «new»-kodeordet

```
> let vakt = new Encounter("vakt")
```

- ❖ Variablen vakt vil her inneholde et objekt utledet av Encounter-klassen.

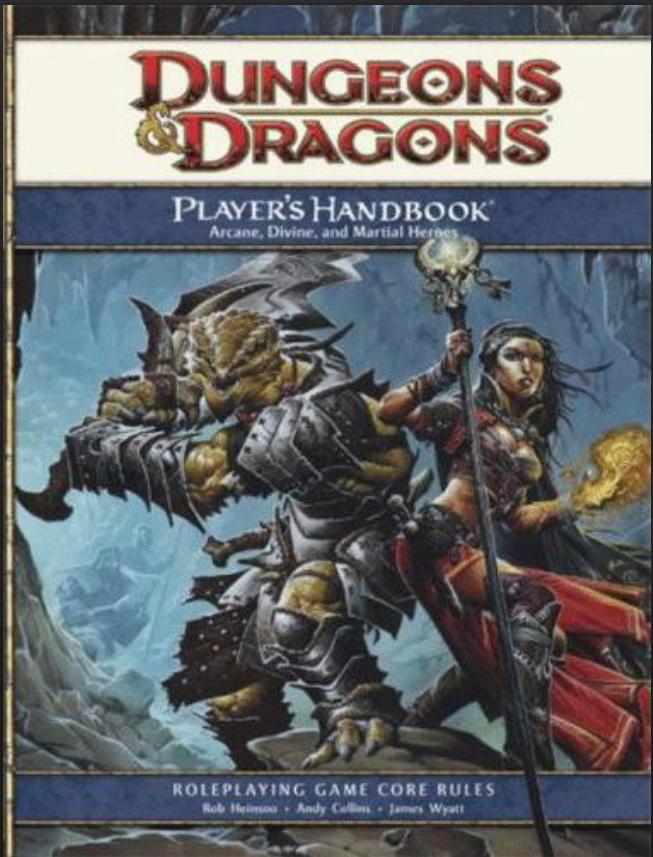
Objektorientert programmering i Javascript – arv

- ❖ For at en klasse skal arve en annen, bruker vi nøkkelordet *extends*, og deretter funksjonen *super()* i konstruktøren, for å hente inn egenskapene og metodene fra klassen en arver fra.

```
class Character extends Levande {  
    constructor() {  
        super();  
        this.tileFrom = [1, 1];  
        this.tileTo = [1, 1];  
        this.timeMoved = 0;  
        this.dimensions = [30, 30];  
        this.position = [45, 45];  
    }  
  
    class Levande {  
        constructor(str = tilfeldigTall(3, 18), con = tilfeldigTall(3, 18),  
            this.stats = {  
                str: str,  
                con: con,  
                dex: dex,  
                int: int,  
                wis: wis,  
                cha: cha  
            }  
        }  
    }  
}
```

```
▼Character {stats: {...}, tileFrom: Array(2), tileTo: Array(2), timeMoved:  
    ► delayMove: {1: 400, 3: 300, 4: 200, 5: 200, 6: 200, 7: 200, 8: 800}  
    ► dimensions: (2) [30, 30]  
    direction: 0  
    ► inventory: Inventory {spaces: 5, stacks: Array(0)}  
    ► pickUp: f ()  
    ► position: (2) [45, 45]  
    ► sprites: {0: Sprite, 1: Sprite, 2: Sprite, 3: Sprite}  
    ▼stats:  
        cha: 13  
        con: 6  
        dex: 7  
        int: 17  
        str: 7  
        wis: 3  
        ► __proto__: Object  
    ► tileFrom: (2) [1, 1]  
    ► tileTo: (2) [1, 1]  
    timeMoved: 0  
    ► __proto__: Levande
```

2) Hva er rollespill



Dette bildet av ukjent forfatter er lisensiert under [CC BY-SA](#).

Rollespill er en samlebetegnelse for spill hvor en spiller en karakter som er ute på ett eller annet oppdrag. Rollespillet har regler for hvordan en lager karakteren, samt regler for ulike handlinger karakteren kan utføre.

I Dungeons and Dragons har hver karakter 6 såkalte *stats*: Strength, Dexterity, Constitution, Wisdom, Intelligence og Charisma

For å få hver stat ruller en 3 vanlige sekserterninger (i rollespill har en flere enn den vanlige terninger med 6 kanter. En har også terninger med 4, 8, 10, 12 og 20 kanter. En terninger med 4 sider kalles d4, 8 terninger d8 osv. Tre sekserterninger skriver en 3d6), og summerer disse. Hver stat kan dermed være mellom 3 og 18.

Videre har en karakter en rase og en klasse. Alver har for eksempel en bonus på +1 på dexterity. Dexterity for en alv blir dermed mellom 4 og 19. Karakteren har til slutt en klasse, som f.eks kriger, trollmann, prest eller tyyv. Disse har igjen innvirkning på forskjellige egenskaper.^{11.05.2020}

Rollespill fortsetter

- ❖ En kriger er gjerne sterk og utholdende. De har derfor flere helsepoeng enn f.eks en trollmann. En kriger vil trille en d10 for helsepoeng, mens en trollmann bare får trille en d4. I tillegg får både krigeren og trollmannen en bonus eller ulempe basert på hvor høy utholdenhetsnivå den har. 16 i utholdenhetsnivå (Constitution) gir f.eks en bonus på +3. For en kriger vil dermed helsepoengene avgjøres ved å trille en d10 og plusse på 3 på resultatet.
- ❖ Et siste element ved karakteren er at etter hvert som den får erfaringspoeng vil den gå opp i nivå (level). For hvert nytt nivå skjer forskjellige ting, som f.eks at karakteren får trille en ny terning for å få flere helsepoeng. En bruker da samme terning som på første nivå.



Rollespill fortsetter

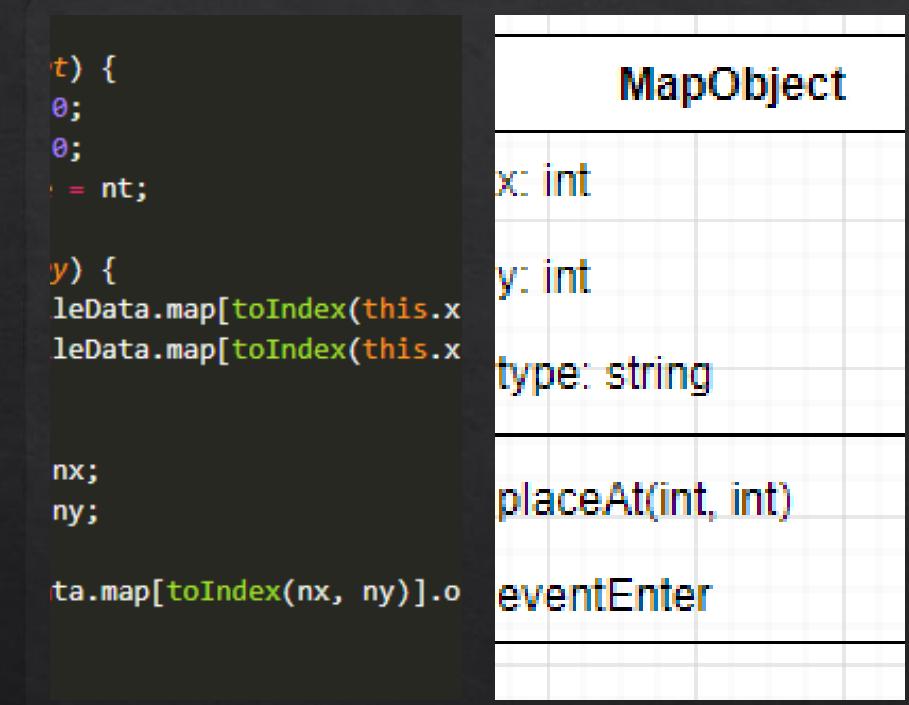
- ❖ Rollespillreglene er veldig gode å begynne med for å forstå objektorientert programering og arv.
- ❖ En karakterklasse f.eks arve fra både en rase-klasse, en klasse-klasse og en en klasse for alle levende ting (alle levende ting i Dungeons and Dragons har de samme seks statsene vi så på tidligere)
- ❖ Vi skal fremover forsøke å omgjøre noen av reglene for Dungeons and Dragons i rollespillet vårt. Dere vil få låne regelbøker av meg, og jeg skal hjelpe dere i gang. Før vi setter i gang med programmeringen er det derimot lurt å planlegge litt. Det skal vi se på i neste økt, når vi lærer oss klassemodellering i UML.

3) Objektorientert modellering (klassemodellering)

- ❖ I modelleringen forsøker vi å tenke gjennom programmet vi skal skrive før vi gjør det. Vi deler det inn i de ulike klassene, noterer oss hvilke egenskaper og metoder hver klasse skal ha, og viser hvordan de henger sammen med arv.
- ❖ Vi skal bruke et modelleringsspråk som heter UML. Dette skiller seg noe fra ER-diagrammene dere er vant med fra IT1. UML står for Unified Modelling Language, og er et modelleringsspråk som kan brukes i mange ulike sammenhenger. Vi skal konsentrere oss om klassediagrammer i UML.
- ❖ For å tegne diagrammene skal vi bruke webapplikasjonen <https://www.draw.io>

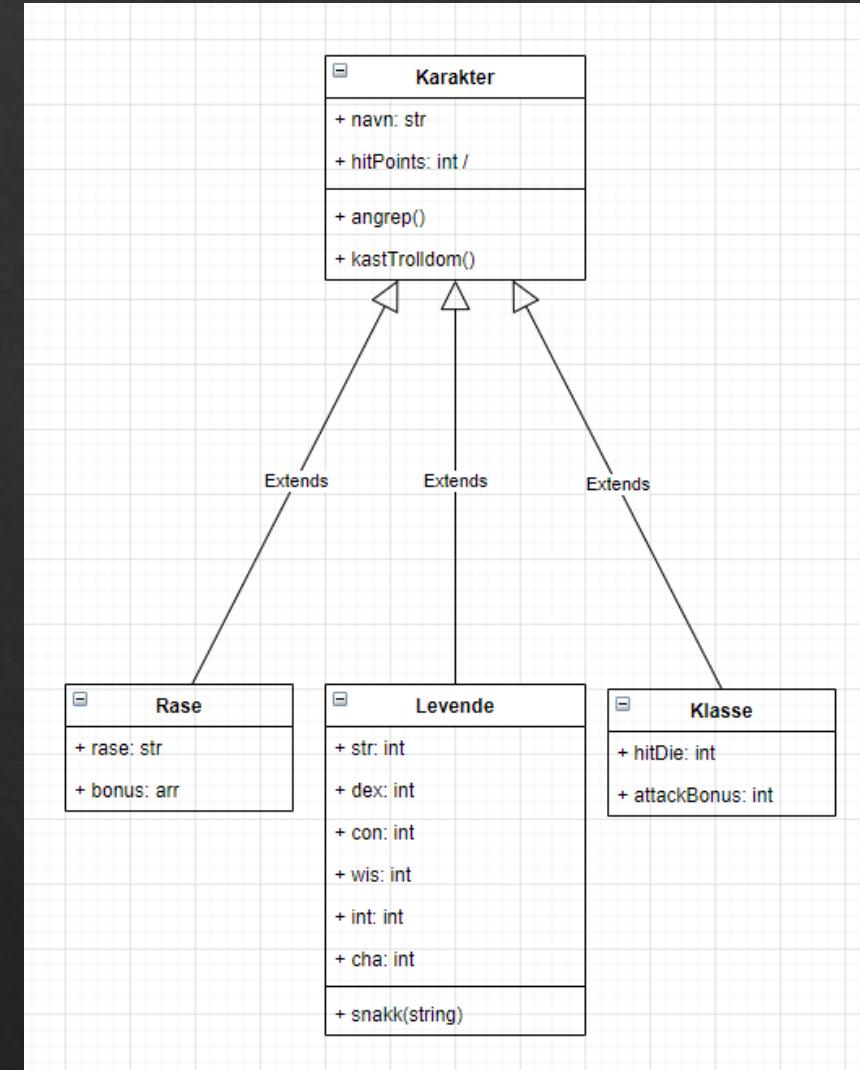
Klassediagrammer

- ❖ Et eksempel på et klassediagram i UML



Klassediagrammer

- ❖ Et klassediagram som viser arven som leder til karakterklassen kan f.eks se noe slik ut. Merk pilene som viser arv. Skråstreken etter hitPoints i karakter, viser at denne er utledet fra andre egenskaper.



Oppgave i modellering

- ❖ Med det dere har lært så langt, vil jeg nå at dere forsøker å lage et klassediagram for hvordan en monsterklasse kan se ut. Hvilke metoder skal den ha? Hva kan den arve fra?

4) Spillmotoren

Dere har fått utdelt en zip-fil med en spillmotor vi kan bruke for å lage et enkelt rollespill.

Mye i denne er for komplisert å forklare for vårt nivå, men jeg skal gå gjennom delene vi trenger for å komme i gang med å utvile vårt eget.

❖ Innholdet i zip-filen



index.html

HTML – Canvas og noen dver
JavaScript – Gameloopen, drawloopen
samt kall til filene under.



globals.js

En del variabler som styrer ting som fart og dimensjoner på spillbrett ol. Vi bruker denne i svært liten grad.



classes.js

Her foregår mye av magien. Her ligger klassene som styrer hvordan spiller foregår. Denne skal vi jobbe mye i, bla. med å lage nye metoder.



encounters.js Her ligger innholdet i handlingen, samt ting vi finner i spillet.



map.js

Her ligger kartet, samt informasjon om grafikk og forskjellige terrenghyper



tileset.png

I denne filen ligger grafikken. Det er denne vi må redigere viss vi vil legge til ny grafikk.

```
13     var mapTileData = new TileMap();
14     var player = new Character();
15     var viewport = new Viewport();
16
17     window.onload = function () {
18         ctx = document.getElementById('game').getContext("2d");
19         requestAnimationFrame(drawGame);
```

På disse linjene skjer det mye spennende.

index.html

Vi skal se litt nærmere på initieringen, funksjonene og løkkene som ligger i index.html

Merk at når jeg bruker begrepet gameloop, er det ikke snakk om en løkke i formell forstand. Den er egentlig en funksjon, men fungerer som en løkke fordi den kaller på seg selv som siste steg. Dvs at den begynner på nytt når den er ferdig med en gjennomkjøring.

Linje 13 initierer kartet vårt. Hva som skjer kan du se i classes.js linje 450->. Til å begynne med blir det bare laget et tomt objekt med egenskapene i linje 453-456 og metodene på linje 458-496

Linje 14 initierer spilleren vår. På samme måte som med kartet, blir det initiert et tomt objekt med noen standard egenskaper og metoder.

På linje 17 ser vi en funksjon som kjører med en gang spillet vårt er ferdig å laste inn. Linje 19 begynner selve gameloopen, som vi finner på linjene 97-223

```
97     function drawGame() {
98         if (ctx == null) {
99             return;
100        }
101        if (!tilesetLoaded) {
102            requestAnimationFrame(drawGame);
103            return;
104        }
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1
```

classes.js

Denne filen inneholder alle klassene spillet bruker. Mange av dem trenger ikke vi forholde oss til i det hele tatt, men enkelte skal vi se nærmere på.

❖ Character

- ❖ Denne klassen innholder mye som har med med plassering og bevegelse å gjøre. Dette er ting vi ikke trenger å endre på, men det kan være verdt å merke seg linjene som begynner med this.sprites.osv
- ❖ Dersom vi ønsker å forandre karakteranimasjonen må vi endre her. X og y referer her til x og y-koordinater i tileset.png. W og h betyr bredde og høyde. Om en vil endre grafikk, legger en derfor til bildene en vil bruke i tileset.png, finner x og y koordinatene ved å holde musen i øverste høyre hjørne, før en måler hvor bred og høy figuren er fra dette punktet (Jeg bruker GIMP til dette)
- ❖ Disse koordinatene blir brukt til å initiere en ny Sprite-instans. (se Sprite-klassen)

classes.js

Litt mer om Sprite-klassen

- ❖ Dersom vi ser på sprite-klassen, ser vi at constructoren tar imot en parameter data. Dette er koordinatene vi så på forrige slide.
- ❖ Hver gang vi ønsker ny grafikk, må vi lage en ny Sprite-instans.
- ❖ Data-parameteret tar i mot et array som holder på ett eller flere objekter med koordinanter.
- ❖ Formatet blir dermed slik: new Sprite([{X: 1, Y: 1, W: 1, H: 1}]) Om det er flere objekter i arrayet blir spriten animert. Formatet blir slik:
 - ❖ new Sprite([{X: 1, Y: 1, W: 1, H: 1}, {X: 1, Y: 1, W: 1, H: 1},])
 - ❖ I praksis lager vi sprites gjennom MapObject-klassen.

classes.js

Mer om MapObject-klassen, og en metode for å jukse litt

- ❖ Som sagt bruker vi MapObject for å lage ting som skal bli plassert på brettet. Denne klassen bruker et triks som er ganske vanlig i spill- og en del andre programmer.
- ❖ Som viser lager man en et nytt MapObject ved å skrive new MapObject(int)
- ❖ Tallet refererer til et objekt lagret i variabelen objectTypes, som ligger i filen map.js
- ❖ Trikset er altså at en lagrer data som skal brukes i spillet i objekter. Dette spesifikke eksemplet følger nesten JSON-standarden (men ikke helt.) JSON er et veldig vanlig format å bruke for denne type lagring.
- ❖ Ved å sende variablen som parameter får hvert nytt MapObject alle egenskapene som ligger innenfor et gitt tall.

```
class MapObject {  
    constructor(nt) {  
        this.x = 0;  
        this.y = 0;  
        this.type = nt;  
    }  
    3: {  
        name: "Tree top",  
        sprite: new Sprite([  
            x: 80,  
            y: 160,  
            w: 80,  
            h: 80  
        ]),  
        offset: [-20, -20],  
        collision: objectCollision.solid,  
        zIndex: 3  
    },  
    4: {  
        name: "Vakt",  
        sprite: new Sprite([  
            x: 355,  
            y: 49,  
            w: 30,  
            h: 30  
        ]),  
        offset: [0, 0],  
        collision: objectCollision.none,  
        zIndex: 1  
    }  
}
```

classes.js

MapObject fortsetter

- ❖ MapObject har også en metode for å plassere seg selv på kartet. Den heter placeAt()

```
placeAt(nx, ny) {  
    if (mapTileData.map[toIndex(this.x, this.y)].object == this) {  
        mapTileData.map[toIndex(this.x, this.y)].object = null;  
    }  
  
    this.x = nx;  
    this.y = ny;  
  
    mapTileData.map[toIndex(nx, ny)].object = this;  
}
```

- ❖ placeAt() tar i mot en x og y koordinat. Her snakker vi om koordinater på spillbrettet, ikke i tilesettet.
- ❖ MapObject-instansene blir plassert ut i onLoad() i index.html (men kan også plasseres ut via andre metoder senere om det ikke skal ses fra starten av. (Test dette i konsollen nå.)

```
63 | (local var) vakt: any | object(4);  
64 | vakt.placeAt(2, 2);  
65 | var mo2 = new MapObject(2);  
66 | mo2.placeAt(2, 3);
```

classes.js

TileMap

- ❖ TileMap fungerer mye som MapObject. Den henter data fra objektet tileTypes i map.js. Vil du legge til flere typer terreng endrer du i dette objektet.
 - ❖ TileMap har en metode som tegner opp kartet. Denne heter buildMapFromData(), som bygger et kart basert på arrayet gameMap.js i map.js. Hvert tall står for en type terreng, og en endrer kartet ved å endre i arrayet. Det er satt opp som et kvadrat for at det skal være lettere for oss å redigere i. Gjør noen endringer nå og se hva som skjer.

```
var tileTypes = {
  0: {
    colour: "#685b48",
    floor: floorTypes.solid,
    sprite: new Sprite([
      {x: 0,
       y: 0,
       w: 40,
       h: 40
     }])
  },
  1: {
    colour: "#5aa457",
    floor: floorTypes.grass,
    sprite: new Sprite([
      {x: 40,
       y: 0,
       w: 40,
       h: 40
     }])
  },
}
```

classes.js

Noen flere klasser og metoder dere kan snuse på

- ❖ Nå klarer dere kanskje å tenke dere til hva metoden addRoofs og klassen placedItemStack gjør. Se på disse i par, og finn ut hvor de heter data fra.
- ❖ Encounter-klassen er det jeg som har begynt på. Denne skal styre hendelser og karakterer vi møter på i spillet. Neste oppgave for dere blir å leke dere med hvordan dere kan utvide denne klassen. Prøv om dere kan lage noen encounter. Lag grafikk og plasser disse på kartet ved hjelp av metodene vi har sett på. Se linje 58-61 for hvordan dere kobler et encounter til et sted på kartet.

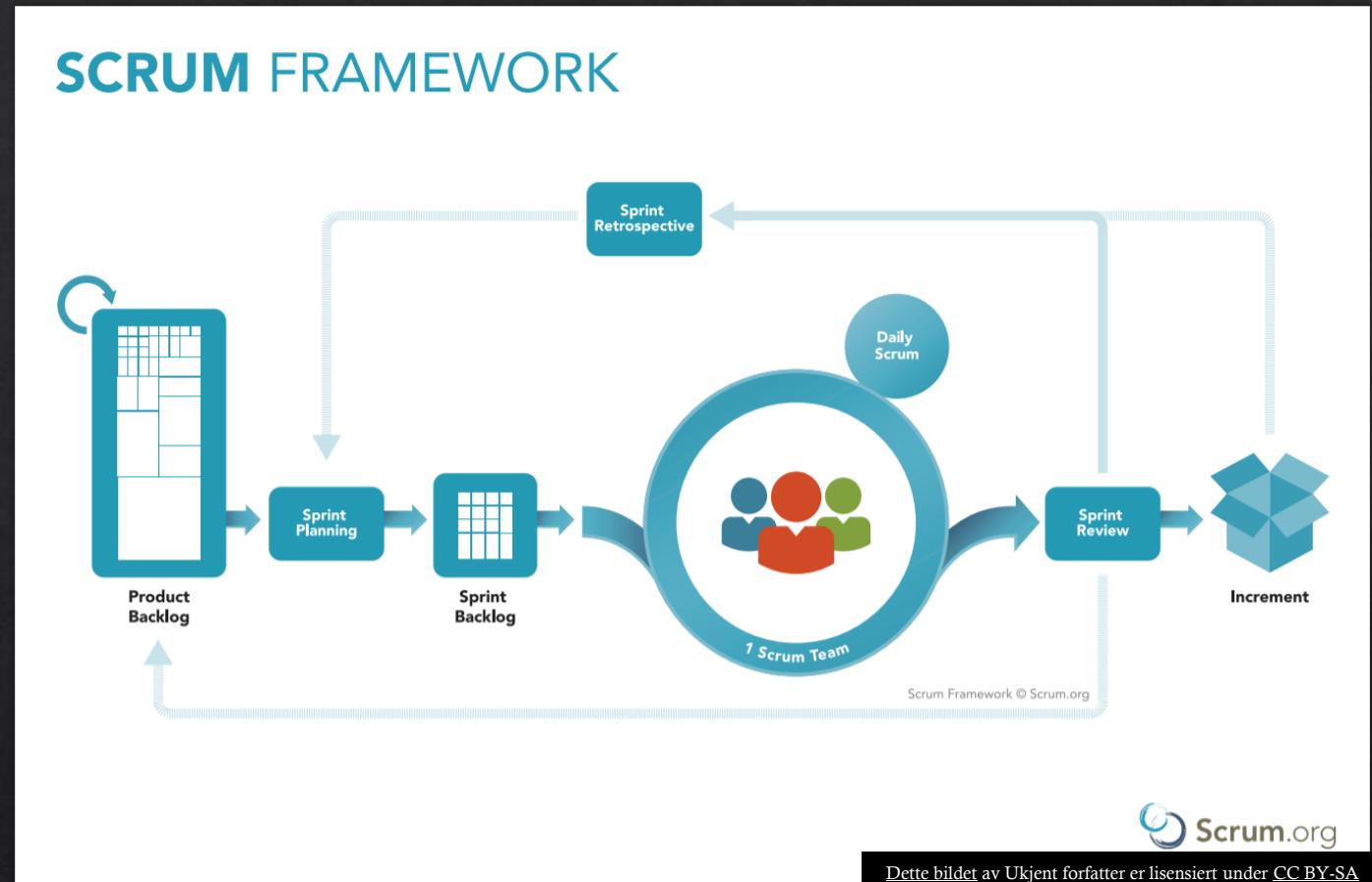
```
mapTileData.map[((2 * mapW) + 2)].eventEnter = function () {  
    var encounter = new Encounter("vakt");  
    encounter.startEncounter();  
};
```

6) Utviklingsmetodikk

SCRUM

SCRUM

- ◊ Når dere nå straks skal gå i gang med prosjektet deres, skal dere bruke utviklingsmetodikken SCRUM. Dette gjør vi for at dere skal få føle litt på hvordan virkelig utvikling foregår, og reflektere over hvorvidt metodikken hjelper oss i arbeidet vårt.
- ◊ Vi skal ikke forholde oss veldig strengt til metodikken, men i det minste forsøke å forholde oss til fasene som ligger i denne metodikken.



<i>Backlog</i>	<i>To Do</i>	<i>In Progress</i>	<i>Testing</i>	<i>Done</i>
<p><i>Feature</i> 10 hrs HIGH</p> <p><i>Bug Fix</i> 2 hrs <i>Medium</i></p> <p><i>Updated Research</i> 4 hrs <i>Low</i></p> <p><i>Content</i> 2 hrs HIGH</p>	<h1>SCRUM – Konsept/planleggingsfase</h1> <ul style="list-style-type: none"> Den første delen i metodikken kaller vi konseptfasen. I denne delen skal dere forsøke å definere konseptet for spillet deres. Hva ønsker dere å lage? Litt av poenget i denne fasen er å bryte prosjektet ned i mindre biter. Skriv ned hver del av spillet dere ønsker på en postit-lapp. Til slutt samler dere sammen de ulike lappene og henger disse opp på en tavle. Dette er den såkalte «backloggen» for prosjektet deres. I planleggingsfasen kan dere fargekode lappene etter hva som er viktigst å begynne med, eller gruppere de i ulike grupper av arbeid (f.eks grafikk, koding, innhold, testing og lignende). Flytt til slutt lappene dere vil begynne med til To-Do. 			
				13.05.2020

SCRUM - Gjennomføring

- ❖ Gjennomføringsfasen deler vi inn i sprinter. I vårt tilfelle blir det en sprint i hver time vi har på skolen.
- ❖ Hver time begynner med et kjapt møte rundt SCRUM-tavlen. Dere bestemmer dere for hvem som skal gjøre hva fra TO-DO, og flytter lappen til «In progress». Når dere er ferdige flytter dere lappen enten til «Done» eller «Testing», og velger en ny lapp. Om nødvendig kan dere flytte en lapp tilbake i «Backlog» eller «To-do.» Arbeid som ikke er ferdig i slutten av sprinten flyttes til backlog.
- ❖ Det kan være lurt å samle all koden som er ferdig på en maskin, slik at alle kan kopiere siste versjon ut fra denne i slutten av timen. Evt kan dere bruke et verktøy som Git, men det har vi ikke tid til å gå gjennom i denne omgang.
- ❖ Dere kan lese mer om [SCRUM her](#)

6) Prosjektarbeid

Da er det bare å sette i gang.
Lykke til!