Rapport INF-1400

Obligatorisk oppgave 1

Martin Soria Røvang
Universitetet i Tromsø *

20. februar 2019

1 Introduksjon

I denne oppgaven ble en klone av spillet "Breakout" laget med mål om å bruke objektorientert design så godt som mulig. I denne rapporten blir det diskutert hvordan dette ble designet og implementert.

1.1 Krav

1. Implementer spillet etter objektorientert design, bruk objekter og klasser.
2. Platformen skal kunne kontrolleres med mus eller tastatur.
3. Ballen skal sprette i forskjellige retning basert på hvor på platformen den treffer.
4. En brikke skal forsvinne når ballen treffer den.
5. Spillet er vunnet når alle brikkene er fjernet, spillet er tapt hvis ballen går under skjermen ved platformens side.
6. Strukturert og kommentert kode.

2 Teknisk bakgrunn

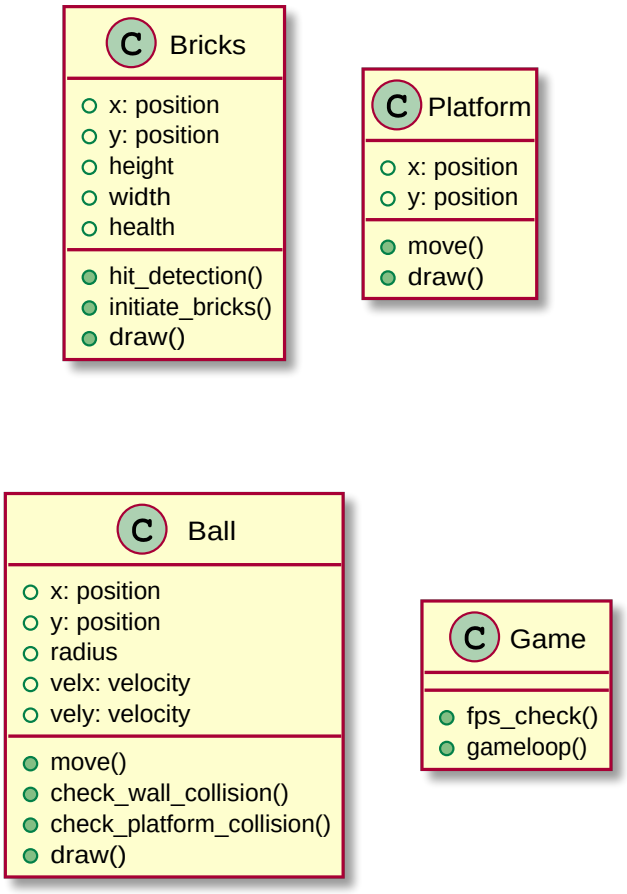
Pygame modul: Pygame modul[1] blir brukt for å tegne til skjermen. I dokumentasjonen til Pygame står det også om hvordan man initialiserer pygame vinduet som er nyttig for å komme igang.
Attributter: Attributter er egenskaper/verdier som ligger i klassen og/eller instansene som er laget fra klassene. Disse verdiene kan endres innad i hver enkelt instans.[Dusty [Oktober 2018] p.37]
Klassemetoder: Klasse-metodene er funksjoner som ligger inni instansene, slik at de kan bli kalt på direkte fra instansene. Disse funksjonene må ha med ett argument som normalt kalles *self*. Denne gir deg mulighet til å hente ut metodene og attributtene fra klassen inn i denne spesifikke funksjonen. Hvis man legger med en @staticmethods over funksjonene gir man muligheten til å kalle på metoden fra klassen også(her vil man ikke få self argument).[Dusty [Oktober 2018] p.38]

*.
[1] https://www.pygame.org/wiki/GettingStarted

3 Design

3.1 Klassestruktur

I denne oppgaven har vi klassene som vist i figur(1) under,



Figur 1: UML diagram for alle klassene i prosjektet

Her har vi brukt staticmethod på tre av metodene i "Bricks" fordi *hit_detection*, *update_bricks* og *initiate_bricks* er bricks relevante metoder, men kjøres ikke direkte på objektene selv.

3.2 Klassene

Klasser fungerer som en blåkopi av et objekt man har lyst på. I denne blåkopien kan man fastsette *attributtene* man vil at objektet skal starte med, dette gjøres ved spesial metoden init. Her setter man hvilke argumenter man har lyst til

at objektet skal få inn når den blir laget og hva slags attributter som objektet starter med. Det kan også være generelle ting man vil at skal skje når dette objektet blir laget.

I dette prosjektet fikk f.eks ballen, attributtene posisjon, hastighet og radius. I koden ser man også hvordan de globale variablene har *selvforklarende*[2] navn slik at det vil være enkelt for dem som programmerer og andre personer å vite hva som gjør hva i koden.

Klassene kan også inneholde metoder som objektet vil få, i dette tilfellet er en av metodene til ballen *move*, Denne metoden flytter ballen ved å ta forrige posisjon og summe på en en distanse gitt ved hastigheten ganget med en konstant(BALLSPEED) som angir farten til ballen. BALLSPEEDkonstanten er gitt i konfigurasjonsfilen som blir forklart senere.

I brikke klassen er det implementert en *@staticmethod*, dette gjør metoden om til en klassemetode slik at jeg kan kalle på den via selve klassen *Bricks*. Dette ble gjort slik at jeg kan holde brikke relaterte metoder i klassen selvom de ikke a angår selve objektet. Bricks klassen har en liste(klasseattributt) som inneholder alle brikkene i spillet. Hvis en brikke blir ødelagt av ballen vil *update_bricks* metoden fjerne den brikken fra listen.

3.3 Gameloop/(Spill-løkke)

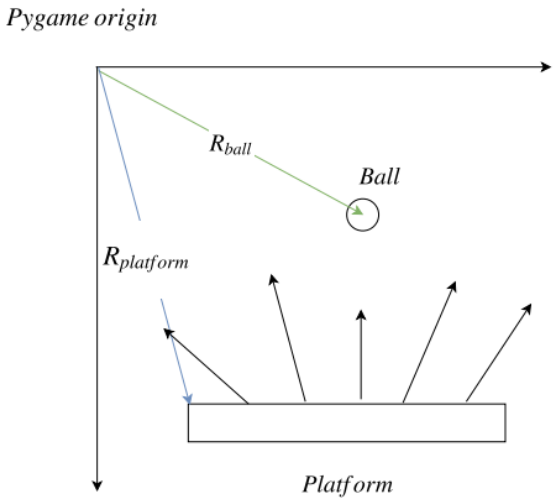
Spillløkken er det som holder spillet i gang. Denne kjører kontinuerlig ved bruk av en "imensløkke"(whileloop). Her starter programmet i *spillende* tilstand der man kan styre platformen og at en ball aktivt spretter på vegger, platform og på brikkene. Dette gjøres ved at objektene sine bevegelse- og tegnefunksjoner blir brukt i samsvar med at hele spilleskjermen blir fylt med svart slik at tidligere bilder av posisjonene blir visket vekk. I denne oppaven har spill-løkken blitt lagt inn som en metode i *Game* klassen slik at denne må startes fra en instans av Game klassen.

3.4 Platform(spilleren)

Platformen styres av spilleren til å sprette vekk ballen for å knuse blokkene. Ballen skal sprette i en vinkel gitt hvor på platformen den treffer, slik som vist i figur(2) under;

I pygame vil koordinatsystemet ha origo i venstre hjørnet og med invertert y-akse, slik som i figur(2). Herfra trekkes det vektor ned til venstre side av platformen for å translatere origo dit. Deretter lages det en liste med verdier fra en cosinus funksjon i intervallet π til 0. Da vil verdier på platformen være -1 til 1, fordeler man dette ut på platformen slik at midten av platformen vil ha verdi 0($\cos(\pi/2) = 0$) kan man sette hastigheten i x-retning til ballen til disse verdiene. Da vil ballen ha negativ hastighet i x-retning hvis den treffer på venstre siden av sentrum på platformen og en positiv hastighet hvis den treffer på høyre siden av sentrum. Verdien vil bli større i x retning etter hvor lengre ut

[2]Dette gjør det enkelt å vite hva man skal endre på i konfigurasjonsfilen for å gjøre tilpassede endringer, både for personen(e) som jobber med prosjektet eller andre som vil bruke/teste det.



Figur 2: Hvordan ballen skal sprette i forhold til hvor på platformen den treffer.

mot sidene ball treffer slik at ballen får en brattere vinkel mot sidene. Platformen skal også kunne styres av spilleren ved hjelp av piltastene. Hastigheten på platformen må følge FPS(frames per second) som er låst til 60(styres av pygame clock funksjon) slik at den ikke går kjappere hvis man spiller på en kraftigere pc.

move: Flytter posisjonen til ballen basert på x og y hastighetskomponentene, i dette tilfelle vil det kun være bevegelse i x-retning. Det spesielle her er at platformen kontrolleres ved hjelp av tastaturet så her brukes *pygame.key.get_pressed()* metoden for å hente ut tastetrykk. Her blir time_passed_seconds sendt inn i metoden for å skalere ned farten slik at den er konstant uavhengig av FPS.

def draw(self): Tegnefunksjonen som vil tegne platformen ut på spillvinduet.

3.5 Ball

Ballen skal sprette av både platformen og blokkene. Ved å gange hastigheten med -1 vil ballen reflektere ut med samme vinkel uten demping. Ballen skal ha en maks fart uansett hvor stor komponent det er i x eller y, så her er det brukt enhetsvektor med en skalering.

$$\boldsymbol{V}_{ball} = \frac{\boldsymbol{v}_{ball}}{||\boldsymbol{v}_{ball}||}\alpha \tag{1}$$

,der α er en skalar for hva farten skal være slik at,

$$||\boldsymbol{V}_{ball}|| = \alpha, \forall\, t \tag{2}$$

Dette vil gi oss posisjonen gitt ved,

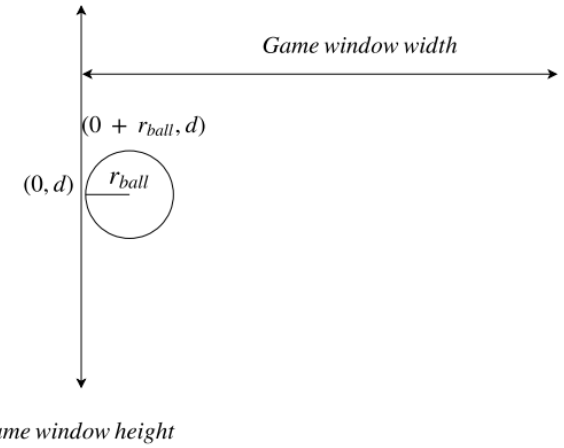
$$(x, y) = \boldsymbol{V}_{ball} t^3 \tag{3}$$

Ballen skal også knuse blokker når den treffer dem og sprette vekk igjen, men dette vil bli kontrollert av bricksobjektene i denne oppgaven.

Disse metodene vil bli brukt i spill-løkken som forklart i seksjon(3.3).

move: Flytter posisjonen til ballen basert på x og y hastighetskomponentene. Her blir time_passed_seconds sendt inn i metoden for å skalere ned farten slik at den er konstant uavhengig av FPS.

check_wall_collisions: Sjekker om ballen treffer veggene, her brukes størrelsen på spillvinduet og radiusen til ballen for å sjekke for kollisjon,i figur(3) under vises dette.



Figur 3: Siden origo til ballen ligger i sentrum må det testes for når ballens posisjon og radius kommer utafor skjermen.

check_platform_collision(self, platform): Her sjekkes det om ballen treffer platformen, her skal noe av det samme skje som med veggene, men at ballen også skal bli gitt en x-hastighet som er større hvor lengre ut på sidene ballen treffer på platformen. Dette vil forårsake at ballen blir reflektert med en vinkel. Her blir platform objektet tatt inn som argument og deretter brukes posisjonen

[3]I programmet vil ikke dette være tid direkte, men ved FPS. Hvis man har 60 FPS vil denne operasjonen skje 60 ganger per sekund. Her vil vi derfor legge inn en skalarverdi fra pygame sin clock funksjon som skalerer farten slik at den holder seg konstant for all FPS.

til platformen får å finne ut om den kolliderer.

def draw(self): Til slutt er det tegnefunksjonen som vil tegne ballen ut på spillvinduet, dette vil være felles for alle objektene.

3.6 Bricks

Brikkene skal bli borte når de blir truffet av ballen. Ballen skal også sprette vekk igjen når den treffer brikkene. Brikkene plasseres ved siden av hverandre i en gitt distanse og med N rader, dette blir styrt ved en global variabel i konfigurasjonsfilen. Brikkene har en *health*(helse) attributt som bestemmer hvor mange ganger ballen må treffe før den blir ødelagt(I denne oppgaven blir brikkene gitt 1 helse). Ved å lage en sammenheng mellom farge og helse kan man sette forskjellige farger etter hvor mye helse en brikke har.
hit_detection: Denne metoden tar inn Ballobjektet for å finne posisjonen til ballen og en brick for å få posisjonen til brikken. Dermed sjekkes det om ballen er innenfor brikkens høyde og bredde. Hvis dette er sant returneres verdien True. Legg også merke til at denne metoden er en @staticmethod, dette betyr at den kan bli kalt på via klassen.

update_bricks(Ball): Sjekker om ballen har truffet en brikke(ved hjelp av *hit_detection* metoden) og fjerner så et liv fra brikken. Hvis helsen til brikken er under 0 vil brikken bli slettet fra brick_list som er en klasseliste og holder på alle brikkene ute i spillet. Hvis en brikke blir fjernet fra denne listen vil den ikke bli tegnet i spill vinduet.

initiate_bricks(rows): Denne metoden blir initialisert når programmet starter. Denne laster inn brikker gitt ved variabelen NUMBER_OF_BLOCK_HORIZONTAL og rows som angir hvor mange rader med brikker det skal være.

I denne koden kan man se hvordan en dobbel for-løkke blir brukt til å lage rad og kolonner med brikker. Her blir et nytt objekt av typen *Bricks* lagt til i klasse listen *bricks_list* og der globale variablene fra konfigurasjonsfilen hjelper til med å lage distanse imellom brikkene.

def draw(self): Tegnefunksjonen som vil tegne brikkene ut på spillvinduet.

3.7 Konfigurasjonsfilen

Konfigurasjonsfilen(config.py) inneholder alle globale variabler. Denne blir brukt slik at man unngår *magic numbers*(tall i koden det er vanskelig å tolke hva gjør) og at man enkelt kan endre på innstillingene i programmet.

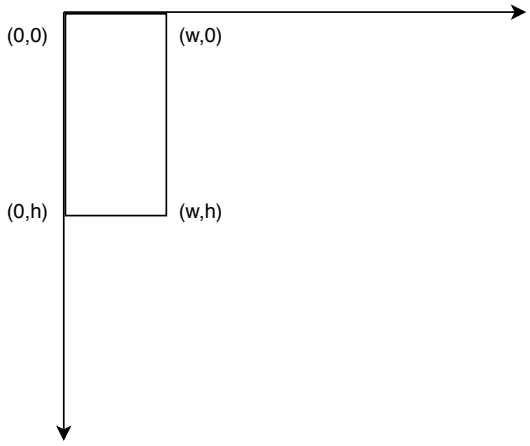
4 Implementasjon

Koden er skrevet i Python versjon 3.[4]
OS: Windows 10
Systemtype: 64-bit OS, x64-basert prosessor
Skjermkort: NVIDIA Geforce 920MX
CPU: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
RAM: 4GB
Pygame [5] Version: 1.9.4
Nympy [6] Version: 1.14.5

5 Diskusjon

Første gang denne oppgaven ble gjennomført ble oppgaven løst ved å kun oppfylle kravene. Da spillet var ferdig ble menyer, toppliste, forskjellige spill nivåer og musikk implementert. Dette førte til at koden ble veldig ustrukturert og vanskelig å navigere fordi det ikke var planlagt når oppgaven ble påbegynt. Derfor ble oppgaven løst på nytt med bedre kodestruktur(med kun kravene). Det å planlegge alt først før man gjennomfører viser å være viktig for at kodestrukturen skal være så fin som mulig. I oppgaven ble det også brukt *@staticmethod* et "problem" i dette tilfelle er at disse gjorde en god del krevende arbeid som dobbel for-løkke osv, og er ment for å kun kjøre fra et kall til klassen. @staticmethod tillater også instanser fra klassen å kalle på disse metodene, slik at hvis man er "uheldig" og kaller på disse inne i løkkene som jobber med metodene til objektene vil det sakte ned pc'en betraktelig. Det er også viktig å merke seg er at koordinatsystemet i pygame er i tredje kvadrant med snudd y-akse som vist i figur(4) under.

[4] https://www.python.org/
[5] https://www.pygame.org/wiki/GettingStarted
[6] http://www.numpy.org/



Figur 4: En rektangel tegnet i posisjon (0,0) med bredd w og høyde h.

Hvordan ballen samhandler med platformen kunne hvert gjort på mange måter. I denne oppgaven gjorde vi det veldig enkelt ved å "strekke" en sirkel ut i en en-dimensjonell array som inneholder trigonometriske verdiene slik at $\cos(\pi)$ er på første indeks og $\cos(0)$ på siste.

```
anglelist = [\cost(\pi),\cost(..), .....,\cost(0)]
```

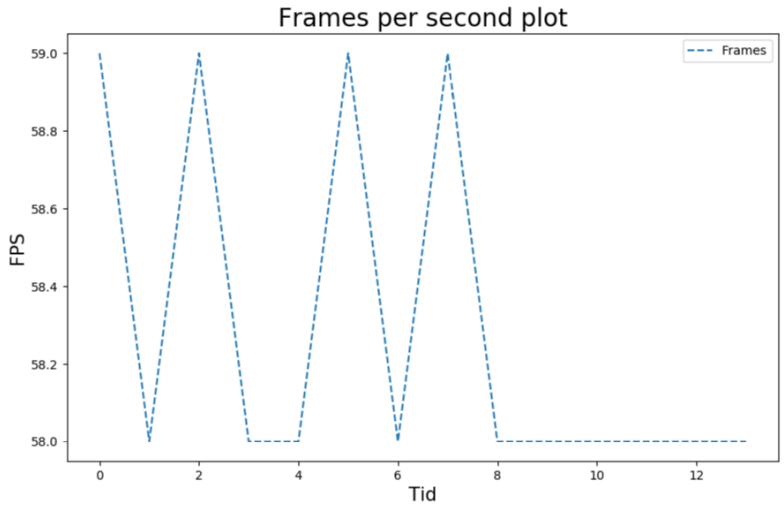
Disse ble fordelt utover platform bredden slik at $\frac{\pi}{2}$ ligger ca i midten. Disse verdiene blir gitt til x-hastigheten til ballen etter hvor på platformen den treffer. Plasseringen til ballen på platformen angir hvilken indeks verdi som blir gitt til ballens x-hastighet og derfor vil dette da bli en tilnærming da indeksene er diskré verdier. Et eksempel er hvis ballen treffer på plass 0.12 vil denne bli rundet ned til indeks 0 som er $cos(\pi) \approx -1$.

5.1 Evaluasjon

Implementerte en FPS(Frames per second) funksjon for å se hvor optimalisert koden er. I figur(5) ser man resultatet av FPS'en. Den ser ut til å flukturere rundt 60 FPS som har blitt satt ved å pygames clock funksjon. FPS'en vil variere etter hvor kraftig pc man har, men den skal være låst til rundt 60.

6 Konklusjon

I denne oppgaven ble en klone av spillet *Breakout* laget ved bruk av python, Pygame og objektorientert design. Her vises det at det kan enkelt bli gjennomført strukturert. Et problem som oppsto er det at koden blir ustrukturert hvis man implementerer flere funksjoner til koden som man ikke hadde planer om å ta med fra starten av uten å måtte veldig mye tid på å restrukturere koden.



Figur 5: Frames per second(FPS) under testing av koden, her er x-aksen i sekunder.

A Appendix

Referanser

Phillips Dusty. *Python 3 Object-Oriented Programming*. Packt Publishing Ltd., third edition, Oktober 2018.