L11.1 Herbst

Class Diagram

CanvasRenderingContext

Moveable
+ position: Vector
# velocity: vector
---
constructor(_position: Vector)
+ move(_timeslice: number): void
+ *draw(): void*

Vector
+ x: number
+ y: number

+ constructor(_x: number, _y: number)
+ set(_x: number, _y: number): void
+ scale(_factor: number): void
+ add(_addend: Vector): void
+ copy(): Vector
+ length(): number
+ random(_minLength, _maxLength): void

Leaf
- type
- scale
- rotation
- rotationSpeed
- color

+ constructor()
+ move(_timeslice: number): void
+ draw(): void

Squirrel
- facing: string
- size: number
- target: Nut

+ constructor()
+ move(_timeslice: number): void
+ draw(): void
+ search(_array: Moveable[ ]): void
- eat(): void
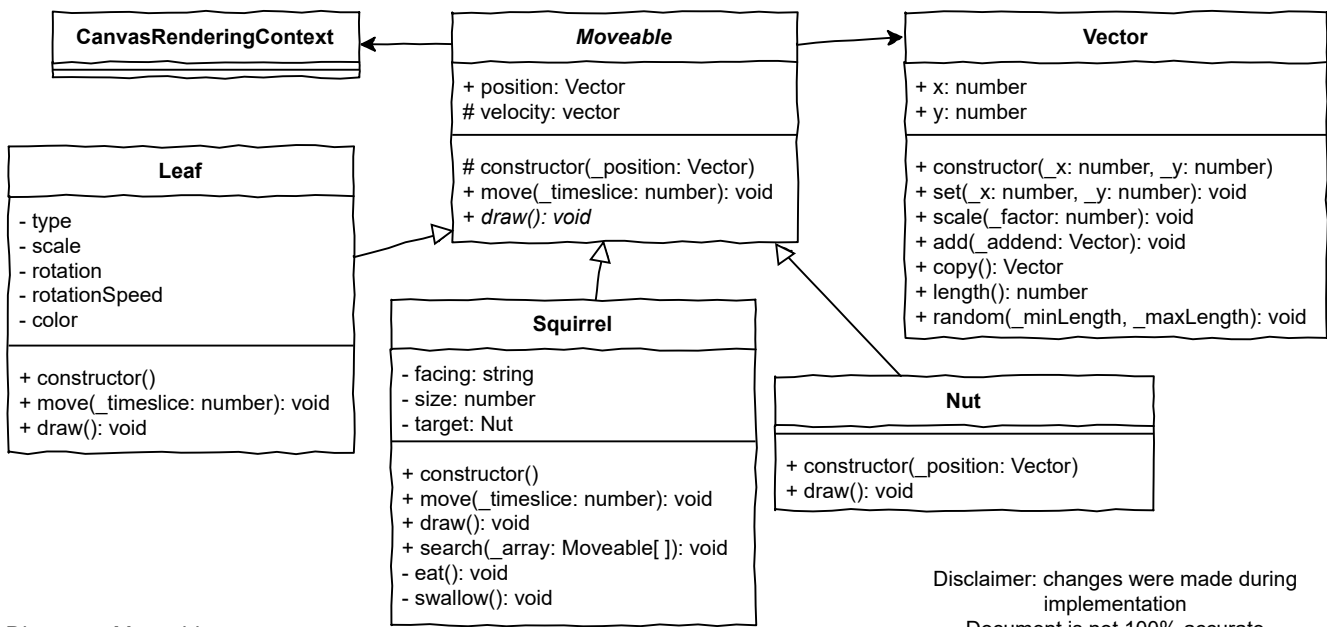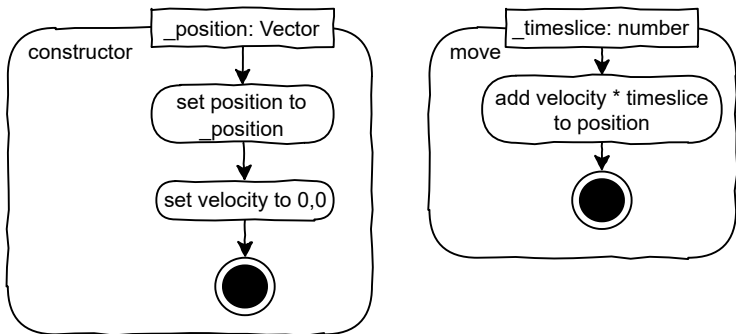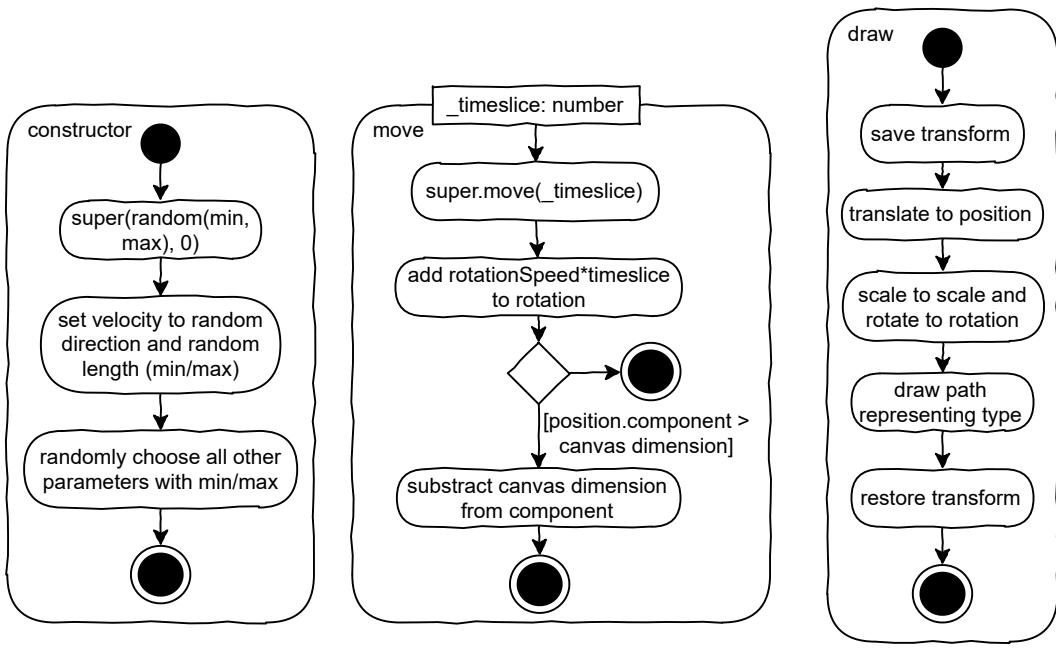- swallow(): void

Nut
+ constructor(_position: Vector)
+ draw(): void

Disclaimer: changes were made during implementation
Document is not 100% accurate anymore

Activity Diagram - Moveable

constructor
_position: Vector
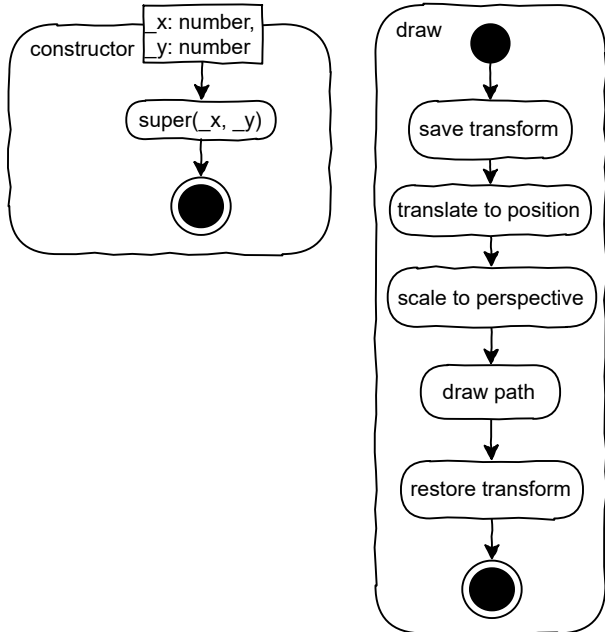- set position to position
- set velocity to 0,0

move
_timeslice: number
- add velocity * timeslice to position

Activity Diagram - Leaf

constructor
- super(random(min, max), 0)
- set velocity to random direction and random length (min/max)
- randomly choose all other parameters with min/max

move
_timeslice: number
- super.move(_timeslice)
- add rotationSpeed*timeslice to rotation
- [position.component > canvas dimension]
- substract canvas dimension from component

draw
- save transform
- translate to position
- scale to scale and rotate to rotation
- draw path representing type
- restore transform

Activity Diagram - Squirrel

constructor
- super(random(min, max), random(min, max))
- set velocity to random length and direction (min/max)
- set size to random(min, max)

move
_timeslice: number
- super.move(_timeslice)
- [position.component > canvas dimension]
- recalculate velocity away from border
- [velocity.x > 0]
- facing.left / facing = right
- [this.velocity.length < (distance to this target).length]
- set velocity to 0,0
- eat
- give information about this.target

search
_array: Moveable[ ]
- find nearest nut to this position in _array
- set velocity to random length (min, max) with direction to Nut position
- set target to Nut

draw
- save transform
- translate to position
- scale to perspective scale, considering size
- [facing=right]
- mirror along y
- draw path
- restore transform

Activity Diagram - Nut

constructor
_x: number, _y: number
- super(_x, _y)

draw
- save transform
- translate to position
- scale to perspective
- draw path
- restore transform

Activity Diagram - Main

- install LoadListener on window

load
- hndLoad

20fps
- update

pointerdown
- hndClick

eat
- hndEat

crc2: CanvasRenderinContext
golden: number = 0,62
horizon: number
background: ImageData
actives: Moveable[ ] = [ ]
passives: Leaf[ ] = [ ]

hndLoad
- get crc2
- set horizon to canvas height * golden
- push nSquirrels of new Sqiuirrels into actives
- push random(min, max) amont of new Leaves into passives
- draw background and save as background
- setInterval for update
- install PointerdownListener on canvas
- install eatListener on canvas

update
- restore background
- sort actives by y
  if nut nearby, set isHunting = true
  if Squirrel has reached nut, fire custom event
- for (active of actives)
  active.move(timeslice)
  active.draw()
- for (passive of passives)
  passive.move(timeslice)
  passive.draw()

hndClick
_event: PointerEvent
- [if pointer position within min/max of squirrel area]
- push new Nut(pointer position) in actives
- for (active of actives)
  if active = Squirrel
  active.search(actives)

hndEat
_event: CustomEvent
- remove Nut that was eaten
- for (active of actives)
  if active = Squirrel
  active.search(actives)