**;; ==== bomb-class.ss**

```
;; ---------------------------------------------------------------------
;; klass bomb, timestamp och delay för att räkna ut när det ska explodera.
;; Radius för att räkna ut vad som skall tas bort
;;----------------------------------------------------------------------
(define bomb%
  (class object%
    (super-new)
    (init-field x-pos y-pos delay radius owner)
    (field
     (type 'bomb)
     (timestamp (*current-m-sec*))
     (bomb-font (make-object font% 10 'modern 'normal 'bold 'smoothed)))


    (define/public (set-x! x)
      (set! x-pos x))

    (define/public (set-y! y)
      (set! y-pos y))


    ;;returnera tidsstämpel från när bomben skapades.
    (define/public (get-timestamp)
      timestamp)

    ;;returnerar sant om bomben has sprängts.
    (define/public (gone-off?)
      (<= (+ timestamp delay) (*current-m-sec*)))

    ;; skickas in (x,y) och och returnerar vilken typ som
    ;; bomben kolliderar med, annars returneras falskt.
    (define/public (collition? xpos ypos)
      (if(and (= xpos x-pos)
           (= ypos y-pos)
           (< (+ timestamp 500) (*current-m-sec*)));dvs 1/2 sek att röra sig på
           type
           #f))

    ;;bitmap som används för att rita bomb m.m
    (define bitmap
      (new drawing%
           [width *blocksize*];;canvas-/bitmapsstorlek
           [height *blocksize*]))

    ;;uppdatera bitmapen för bomb med tidsskrift och olika bombbilder
    (define/public (update-bitmap)
      (send bitmap clear)
      (send bitmap background-transp)
      (cond
        ((< (- (+ timestamp delay) (*current-m-sec*)) 2000)
         (send bitmap draw-bitmap-on-bitmap
               (send *image-store* get-image 'bomb-1) 0 0))
        (else
         (send bitmap draw-bitmap-on-bitmap
               (send *image-store* get-image 'bomb-2) 0 0)))
      (send bitmap draw-text
            (number->string (/ (- (+ timestamp delay) (*current-m-sec*)) 1000))
            0 0 bomb-font))

    ;;Skickar bitmapen, anropas från spellogiken för att uppdatera skärmen
    (define/public (get-bitmap)
      (update-bitmap)
      (send bitmap get-bitmap))))
```

**;; ==== draw-class.ss**

```
;; ---------------------------------------------------------------------
;; Klass f�r att rita objekt i en bitmap
;; ---------------------------------------------------------------------
(define drawing%
  (class object%
    (super-new)
    (init-field width height)
    (define draw-buffer (make-object bitmap% width height #f #t))
    (define draw-dc (make-object bitmap-dc% draw-buffer))

    ;;f�r att rita upp igen
    (define/public (clear)
      (send draw-dc erase))

    ;;En metod som gör det möjlig att skicka in bitmapen från
    ;; objectet in i en dc på en canvas
    (define/public (get-image canvas dc)
      (send dc draw-bitmap draw-buffer 0 0))

    ;;skickar nuvarande bitmap
    (define/public (get-bitmap)
      draw-buffer)

    ;;returnerar bredd
    (define/public (get-width)
      width)

    ;;returnerar h�jd
    (define/public (get-height)
      height)

    ; En procedur som s�tter bakgrundsf�rgen p� GUI (p� slumpartat vis)
    (define/public (background)
      (send draw-dc set-background
            (make-object color% (random 255) (random 255) (random 255))))

    ; En procedur som s�tter bakgrundsf�rgen p� GUI
    (define/public (set-background-color! r g b a)
      (send draw-dc set-background  (make-object color% r g b a)))

    ;; En procedur som s�tter bakgrundsf�rgen p� GUI till genomskinlig
    (define/public (background-transp)
      (send draw-dc set-background  (make-object color% 0 0 0 0)))

    ;;Sätt alphakanalen på bitmappen
    (define/public (set-alpha! a)
      (send draw-dc set-alpha a))

    ;; En procedur som ritar en ellips
    (define/public (draw-circle x y size-x size-y pen brush)
      (send draw-dc set-pen pen)
      (send draw-dc set-brush brush)
      (send draw-dc draw-ellipse x y size-x size-y))

    ;; En procedur som ritar en rektangel
    (define/public (draw-rectangle x y size-x size-y pen brush)
      (send draw-dc set-pen pen)
      (send draw-dc set-brush brush)
      (send draw-dc draw-rectangle x y size-x size-y))

    ;; En procedur som ritar en linje
    (define/public (draw-line x y size-x size-y pen brush)
      (send draw-dc set-pen pen)
      (send draw-dc set-brush brush)
      (send draw-dc draw-line x y (+ x size-x) (+ y size-y)))

    ;; En procedur som ritar text
    (define/public (draw-text text x y font)
      (send draw-dc set-font font)
      (send draw-dc draw-text text x y))

    ;; En procedur som ritar en bild fr�n en bitmap
    (define/public (draw-bitmap-on-bitmap bitmap x y)
      (send draw-dc draw-bitmap bitmap x y))))
```

**;; ==== flame-class.ss**

```
;; ----------------------------------------------------------------------
;; klass flame
;; ----------------------------------------------------------------------
(define flame%
  (class object%
    (super-new)
    (init-field
     center-x-pos
     center-y-pos
     delay
     owner
     limits)
    (field
     (type 'flame)
     (timestamp (*current-m-sec*))
     (changed #f))

    ;;Yttre gränserna för var flammorna ska komma
    (define x-upper (cdr (assq 'l limits)))
    (define x-lower (cdr (assq 'r limits)))
    (define y-upper (cdr (assq 'u limits)))
    (define y-lower (cdr (assq 'd limits)))

    ;;göra om den relativa positionen till position i planen
    (define calc-x-pos (- center-x-pos x-upper))
    (define calc-y-pos (- center-y-pos y-upper))

    ;;Värden för höjd och bredd
    (define calc-height (+ 1 y-upper y-lower))
    (define calc-width (+ 1 x-upper x-lower))


    ;;funktioner som returnerar den absoluta positionen
    (define/public (get-x-pos)
      calc-x-pos)

    (define/public (get-y-pos)
      calc-y-pos)


    ;;returnerar tidsstämpel från när bomben skapades
    (define/public (get-timestamp)
      timestamp)

    ;;returnerar sant om bomben har sprängts
    (define/public (gone-off?)
      (<= (+ timestamp delay) (*current-m-sec*)))

    ;;tar en punkt (x,y) och kollar om en kollision sker,
    ;; och i sådana fall med vad. Annars returneras falskt.
    (define/public (collition? xpos ypos)
      (if(or
          (and (= xpos center-x-pos)
               (<= ypos (+ center-y-pos y-lower))
               (<= (- center-y-pos y-upper) ypos))
          (and (= ypos center-y-pos)
               (<=  xpos (+ center-x-pos x-lower))
               (<= (- center-x-pos x-upper) xpos)))
         type
         #f))

    (define bitmap
      (new drawing%
           [width (* *blocksize* calc-width)];;canvas-/bitmapsstorlek
           [height (* *blocksize* calc-height)]))

    ;;Funktion för att rita ut flammor, typen anger om det är i x-led eller y-led
    (define/private (draw-flames type)
      (define (draw-x from to)
        (if(<= from to)
           (begin
             (send bitmap draw-bitmap-on-bitmap
                   (send *image-store* get-image type 'x)
                   (* *blocksize* from)
                   (* *blocksize* y-upper))
             (draw-x (+ 1 from) to))))
    (define (draw-y from to)
      (if(<= from to)
         (begin
           (send bitmap draw-bitmap-on-bitmap
                 (send *image-store* get-image type 'y)
                 (* *blocksize* x-upper)
                 (* *blocksize* from))
           (draw-y (+ 1 from) to))))

    (draw-x 0 (+ 1 x-upper x-lower))
    (draw-y 0 (+ 1 y-upper y-lower)))

   ;;uppdateringsfunktion för att byta flamma efter en viss tid
   (define/public (update-bitmap)
     (cond
       ((< (- (+ timestamp delay) (*current-m-sec*)) 1000)
        (draw-flames 'flame-small))
       (else
        (draw-flames 'flame-big))))

   ;;Skickar bitmapen, anropad från spellogiken för att uppdatera skärmen
   (define/public (get-bitmap)
     (send bitmap clear)
     (update-bitmap)
     (send bitmap get-bitmap))))
```

**;; ==== game-board-class.ss**

```
;; --------------------------------------------------------------------
;; board% definera en spelplan med en viss längd och bredd
;; --------------------------------------------------------------------

(define board%
  (class object%
    (super-new)
    (init-field height width height-px width-px)
    (field
     (gamevector (make-vector  (* (+ 1 height) (+ 1 width))))
     (changed #f))

    ;;lägger till ett objekt på en given position
    ;; och sätter att ändrat till sant.
    (define/public (add-object-to-board! x y type)
      (vector-set! gamevector (get-pos x y) type)
      (set! changed #t))

    ;;Tar bort objekt från brädan och om det inte går, returneras falskt
    (define/public (delete-object-from-board! x y)
      (let((object (get-object-at-pos x y)))
        (if (not (eq? object 0))
            (begin
              (vector-set! gamevector (get-pos x y) 0)
              (set! changed #t))
            #f)))

    ;; Ger en punkt (x,y):s motsvarande position i vektorn
    (define/public (get-pos x y)
      (+ x (* y width)))

    ;;Räknar ut x och y-pos utifrån given pos i vektorn.
    ;; (x-pos . y-pos)
    (define/public  (get-pos-invers pos)
      (cons (remainder pos (+ 0 width)) (quotient pos (+ 0 width))))

    ;;Returnerar objekt som ligger i en viss (x,y)-position
    (define/public (get-object-at-pos x y)
      (vector-ref gamevector (get-pos x y)))

    ;;funktion för att ta bort block i spelplanen utifrån position
    ;; och sprängradie, kollar i de olika riktningar som finns.
    (define/public (delete-destruct-from-board-radius! x y radius)
      (let ((x1-run? #t)
            (y1-run? #t)
            (x2-run? #t)
            (y2-run? #t))
        (define limits '())
        (define emptyspaces '())
        (define delete-block '())
        (let loop ((x1-temp x) ;; den som ökar
                   (y1-temp y) ;; den som ökar
                   (x2-temp x) ;;den som minskar
                   (y2-temp y));; den som minskar

          (cond
            ((and (<= x1-temp (+ x radius)) x1-run?)
             (cond
               ((eq? 'destructeble-stone (collision? x1-temp y))
                (set! delete-block (cons (list x1-temp y 'r) delete-block))
                (set! x1-run? #f)
                (loop x1-temp y1-temp x2-temp y2-temp));;hoppa ur denna loop

               ((eq? 'indestructeble-stone (collision? x1-temp y))
                (set! x1-run? #f)
                (loop x1-temp y1-temp x2-temp y2-temp));;hoppa ur denna loop
               (else
                (set! emptyspaces (cons (list x1-temp y 'r) emptyspaces))
                (loop (+ x1-temp 1) y1-temp x2-temp y2-temp))))

            ((and (>= x2-temp (- x radius)) x2-run?)
             (cond
               ((eq? 'destructeble-stone (collision? x2-temp y))
                (set! delete-block (cons (list x2-temp y 'l) delete-block))
                (set! x2-run? #f)
                (loop x1-temp y1-temp x2-temp y2-temp))

               ((eq? 'indestructeble-stone (collision? x2-temp y))
                (set! x2-run? #f)
                (loop x1-temp y1-temp x2-temp y2-temp))

               (else
          (set! emptyspaces (cons (list x2-temp y 'l) emptyspaces))
          (loop x1-temp y1-temp (- x2-temp 1) y2-temp))))


      ((and (<= y1-temp (+ y radius)) y1-run?)
       (cond
         ((eq? 'destructeble-stone (collision? x y1-temp))
          (set! delete-block (cons (list x y1-temp 'd) delete-block))
          (set! y1-run? #f)
          (loop x1-temp y1-temp x2-temp y2-temp))

         ((eq? 'indestructeble-stone (collision? x y1-temp))
          (set! y1-run? #f)
          (loop x1-temp y1-temp x2-temp y2-temp))

         (else
          (set! emptyspaces (cons (list x y1-temp 'd) emptyspaces))
          (loop x1-temp (+ y1-temp 1) x2-temp y2-temp))))

      ((and (>= y2-temp (- y radius)) y2-run?)
       (cond
         ((eq? 'destructeble-stone (collision? x y2-temp))
          (set! delete-block (cons (list x y2-temp 'u) delete-block))
          (set! y2-run? #f)
          (loop x1-temp y1-temp x2-temp y2-temp))
         ((eq? 'indestructeble-stone (collision? x y2-temp))
          (set! y2-run? #f)
          (loop x1-temp y1-temp x2-temp y2-temp))

         (else
          (set! emptyspaces (cons (list x y2-temp 'u) emptyspaces))
          (loop x1-temp y1-temp x2-temp (- y2-temp 1)))))

      (else
       ;;gränserna relativt till bombens position
       (set! limits (list
                     (cons 'r (- x1-temp x 1))
                     (cons 'd (- y1-temp y 1))
                     (cons 'l (- x x2-temp 1))
                     (cons 'u (- y y2-temp 1)))))))

    ;;returnerar lista av objekt att ta bort, för att lägga till flammor
    (list emptyspaces delete-block limits)))


;; #f innebär tomt, annars returneras
;; vilken typ av objekt som ligger på positionen.
(define/public (collision? x y)
  (if(and (<= 0 x) (<= 0 y) (< x width) (<= y height))
     (let((object (get-object-at-pos x y)))
       (if (eq? object 0)
           #f
           object))
     #f))
```

```
;;hjälpfunktion för att kolla om man ska lägga
;;till en sten på en position utanför startplatserna för spelaren
(define/private (add-destruct-stone? x y)
  (and
   (not (or
         (and (< x 3) (< y 3));;första hörnet
         (and (<= (- width 3) x) (< y 3));;andra hörnet
         (and (< x 3) (<= (- height 3) y));;fjärde hörnet
         (and (<= (- width 3) x) (<= (- height 3) y));;tredje hörnet
         ))
   (= 0 (random 2))))

;;funktion för att placera ut stenarna på spelplanen,
;; både oförstörbara och förstörbara
(define/public (randomize-stones)
  (define (x-led x)
    (if (< x width)
        (begin
          (y-led 0 x)
          (x-led (+ x 1)))))

  (define (y-led y x)
    (if (< y height)
        (begin
          (cond
            ((= x 0)(add-object-to-board! x y 'indestructeble-stone))
            ((= y 0)(add-object-to-board! x y 'indestructeble-stone))
            ((= x (- width 1))
             (add-object-to-board! x y 'indestructeble-stone))
            ((= y (- height 1))
             (add-object-to-board! x y 'indestructeble-stone))
            ((and (even? y) (even? x))
             (add-object-to-board! x y 'indestructeble-stone))
            ((add-destruct-stone? x y)
             (add-object-to-board! x y 'destructeble-stone)))
          (y-led (+ y 1) x))))
  ;;starta
  (x-led 0))


;;huvudbitmap
(define bitmap
  (new drawing%
       [width width-px];;canvas-/bitmapsstorlek
       [height height-px]))

;;bitmap för att generera bakgrund i
(define background
  (new drawing%
       [width width-px];;canvas-/bitmapsstorlek
       [height height-px]))

;;Fixar rutmönstret på spelplanen
(define/public (set-bg!)
  (define (x-led x)
    (if (< x width)
```

```scheme
          (begin
            (y-led 0 x)
            (x-led (+ x 2)))))

      (define (y-led y x)
        (if (< y height)
            (begin
              (send background draw-bitmap-on-bitmap
                    (send *image-store* get-image 'bg)
                    (* *blocksize* y) (* *blocksize* x))
              (y-led (+ y 2) x))))

      (send background clear)
      (x-led 0));;starta


    ;;Metod för att uppdatera spelplanens bitmap om den har ändrats
    (define/public (update-bitmap)
      (define (loop index)
        (if (< index (vector-length gamevector))
            (begin
              (if (vector-ref gamevector index);;finns det något där eller inte?
                  (update-bitmap-help
                   (vector-ref gamevector index)
                   (get-pos-invers index)))
              (loop (+ 1 index)))))

      (if changed
          (begin
            (send bitmap clear)
            (send bitmap draw-bitmap-on-bitmap
                  (send background get-bitmap) 0 0)
            (loop 0))))

    (define/private (update-bitmap-help type pos)
      (cond
        ((eq? type 'indestructeble-stone)
         (send bitmap draw-bitmap-on-bitmap
               (send *image-store* get-image 'non-dest-block)
               (* *blocksize* (car pos))
               (* *blocksize* (cdr pos))))
        ((eq? type 'destructeble-stone)
         (send bitmap draw-bitmap-on-bitmap
               (send *image-store* get-image 'dest-block)
               (* *blocksize* (car pos))
               (* *blocksize* (cdr pos))))))

    ;;retunerar spelplanens bitmap
    (define/public (get-bitmap)
      (send bitmap get-bitmap))))
```

**;; ==== game-board-class.ss**

```scheme
;; ----------------------------------------------------------------------
```

```scheme
;; class game-logic%, huvudlogiken samt
;; hanterar utritning av bitmaps av alla object
;; ----------------------------------------------------------------------
(define game-logic%
  (class object%
    (super-new)
    (init-field height width height-px width-px)
    (field
     ;; lista med alla aktiva bomber sparade som list-object%
     (bombs (new list-object%))
     ;;  lista med alla aktiva players sparade som list-object%
     (players (new list-object%))
     ;;  lista med alla aktiva keyboard-players sparade som list-object%
     (keyboard-players (new list-object%))
     (powerups (new list-object%))
     (to-do-list (new list-object%))
     (bomb-flames (new list-object%)))

    ;;bitmap för statuspanelen
    (define game-status-bitmap
      (new drawing%
           [width 170];;canvas-/bitmapsstorlek
           [height height-px]))

    ;;bitmap för spelet
    (define game-board-bitmap
      (new drawing%
           [width width-px];;canvas-/bitmapsstorlek
           [height height-px]))

    ;;själva spelplanen
    (define game-board
      (new board%
           [height height]
           [width width]
           [height-px height-px]
           [width-px width-px]))

    ;;initiera spelplanen, sätt bg och randomisera stenar
    (define/public (init-gameboard)
      (send game-board randomize-stones)
      (send game-board set-bg!))



    ;;metod som tar emot key events från gui-delen
    ;; key - lista med knappar nedtryckta
    ;; Key events skickas hit från gui-klassen en gång per loop
    (define/public (handle-key-event key)
      (for-each
       (lambda (proc)
         (let((action (assq key (cdr proc))))
           (if action
               (move-dir (cdr action) (car proc)))))
       (get-field inner-list keyboard-players)))
```

```
;;Metod som lägger till keyboard-players
;;new-name - sträng
;;x y - start koordinater
;; number-of-lives - int
;;keybord-bindings - lista med tangenter och korisponderande händelse-
;;'((#\w . u)(#\a . l)(#\s . d)(#\d . r)(#\space . drop)
;; u=upp, l = vänster, d = ner, r = höger, drop = anropar drop-bomb-metoden.
(define/public (add-key-board-player new-name
                                     x y dxy
                                     number-of-lives
                                     player-color
                                     keybord-bindings)
  (let((temp-player
        (new player%
             [x-pos x]
             [y-pos y]
             [dxdy dxy]
             [name new-name]
             [lives number-of-lives]
             [color player-color])))

    (send players add-to-list! temp-player)
    (send keyboard-players add-to-list!
          (cons temp-player keybord-bindings))))

;;metod för att kolla om möjligt att förflytta sig
;; samt hanterar kollisioner med objekt.
;;Retunerar #t om möjligt att förflytta sig.
(define (move? player dir)
  (let((collition #f)
       (new-x (get-field x-pos player))
       (new-y (get-field y-pos player)))

    (cond
      ((and (eq? 'd dir)
            (not (= 0 (remainder (send player get-x-pos-px) *blocksize*))))
       (set! collition #t))
      ((and (eq? 'r dir)
            (not (= 0 (remainder (send player get-y-pos-px) *blocksize*))))
       (set! collition #t))
      ((and (eq? 'u dir)
            (not (= 0 (remainder (send player get-x-pos-px) *blocksize*))))
       (set! collition #t))
      ((and (eq? 'l dir)
            (not (= 0 (remainder (send player get-y-pos-px) *blocksize*))))
       (set! collition #t))
      ((eq? 'u dir)(set! new-y (-(get-field y-pos player) 1)))
      ((eq? 'd dir)(set! new-y (+(get-field y-pos player) 1)))
      ((eq? 'l dir)(set! new-x (-(get-field x-pos player) 1)))
      ((eq? 'r dir)(set! new-x (+(get-field x-pos player) 1))))

    (for-each
     (lambda (powerup)
       (if(and
           (send powerup collition? new-x new-y)
           (not collition);; F = ingen kollision
           )
          (begin
            (send powerup use-power-up player)
            (send powerups remove-from-list! powerup))))
     (get-field inner-list powerups))

    (for-each
     (lambda (bomb)
       (if(and
           (send bomb collition? new-x new-y)
           (not collition))
            (set! collition #t)))
     (get-field inner-list bombs))

    (if(and
        (send game-board collision? new-x new-y)
        (not collition))
       (set! collition #t))

    (not collition)))

;;Flytta spelaren
(define/private (move-dir dir player)
  (if (move? player dir)
      (cond
        ((eq? 'u dir)
         (send player set-y-pos-px!
               (-(send player get-y-pos-px) (get-field dxdy player))))
        ((eq? 'd dir)
         (send player set-y-pos-px!
               (+(send player get-y-pos-px) (get-field dxdy player))))
        ((eq? 'l dir)
         (send player set-x-pos-px!
               (-(send player get-x-pos-px) (get-field dxdy player))))
        ((eq? 'r dir)
         (send player set-x-pos-px!
               (+(send player get-x-pos-px) (get-field dxdy player))))
        ((eq? 'drop dir)
         (add-bomb
          (get-field x-pos player) (get-field y-pos player) player)))
      (cond;;flytta om möjligt i rutan
        ((and (eq? 'u dir)
              (<= (get-field dxdy player)
                  (remainder (send player get-y-pos-px) *blocksize*)))
         (send player set-y-pos-px!
               (-(send player get-y-pos-px) (get-field dxdy player))))
        ((and (eq? 'd dir)
              (<= (get-field dxdy player)
                  (remainder (send player get-y-pos-px) *blocksize*)))
         (send player set-y-pos-px!
               (+(send player get-y-pos-px) (get-field dxdy player))))
        ((and
          (eq? 'l dir)
          (<= (get-field dxdy player)
              (remainder (send player get-x-pos-px) *blocksize*)))
        (send player set-x-pos-px!
              (-(send player get-x-pos-px) (get-field dxdy player))))
       ((and
         (eq? 'r dir)
         (<= (get-field dxdy player)
             (remainder (send player get-x-pos-px) *blocksize*)))
        (send player set-x-pos-px!
              (+(send player get-x-pos-px) (get-field dxdy player))))))

  (if(not (eq? 'drop dir))
     (send player set-dir! dir)))

;;Metod  för att lägga till bomber till en position och ge bomben en ägare
(define/private (add-bomb x y owner)
  (if(send owner can-bomb?)
     (begin
       (add-bomb-help x y owner)
       (send owner add-bomb))))

(define/private (add-bomb-help x y own)
  (let((temp-bomb
        (new bomb%
             [x-pos x]
             [y-pos y]
             [delay (get-field delay own)]
             [radius (get-field radius own)]
             [owner own])))
    (send bombs add-to-list! temp-bomb)))


(define/private (on-bomb-explosion bomb)
  (define result
    (send game-board
          delete-destruct-from-board-radius!
          (get-field x-pos bomb)
          (get-field y-pos bomb)
          (get-field radius bomb)))

  (define flames (car result))
  (define to-blow-up (cadr result))
  (define flame-limits (caddr result))

  ;;sätt antal bomber ute på spelaren
  (send (get-field owner bomb) remv-bomb)

  ;; ta bort bomben från bomberna
  (send bombs remove-from-list! bomb)

  ;;kolla mot olika powerups och bomber för kedjesprängning
  (for-each  (lambda (flame)

              ;;spräng alla bomber
              (for-each  (lambda (bomb-to-check)
                           (if(send bomb-to-check collition?
                                    (car flame)
                                    (cadr flame))
                              (on-bomb-explosion bomb-to-check));;spräng
                           )
                         (get-field inner-list bombs))

              ;;spräng alla powerups
              (for-each  (lambda (powerup-to-check)
                           (if(send powerup-to-check collition?
                                    (car flame)
                                    (cadr flame))
                              (send powerups remove-from-list!
                                    powerup-to-check)))
                         (get-field inner-list powerups)))
             flames)

  ;;Gör en ny flammgrupp och lägg till den i flammlistan
  (send bomb-flames add-to-list!
        (new flame%
             [center-x-pos (get-field x-pos bomb)]
             [center-y-pos (get-field y-pos bomb)]
             [delay 1500]
             [owner (get-field owner bomb)]
             [limits flame-limits]))

  ;;lägg till att-göra-listan, för att ta bort nästa loop
  (send to-do-list add-to-list!
        (new make-timer%
             [delay 0];;spräng så fort som möjligt
             [proc (lambda (arg)(remove-blocks arg))]
             [args (list to-blow-up)])))

(define/private (remove-blocks block-list)
  (for-each  (lambda (block)
              (if (and
                   (send game-board;;kolla om borttagning lyckades och ta bort
                         delete-object-from-board!
                         (car block);x
                         (cadr block));y
                   (= 2 (random 5)));en på fem
                  (send powerups add-to-list!
                        (new powerup%
                             [x-pos (car block)]
                             [y-pos (cadr block)]))))
             block-list))

(define/private (on-die player flame)
  (if (send player possible-to-die?)
      (send player die))

  (if (= (get-field lives player) 0)
      (begin
        (send player set-x! 10000)
        (send player set-y! 10000))))

;; skickar in alla trackade objekt bitmaps i en viss positon.
(define/public (update-scene draw-class)
  (send game-board update-bitmap)
  (update-game-logic)
  (update-game-status-bitmap)
  (send draw-class draw-bitmap-on-bitmap
        (send game-board get-bitmap) 0 0)
  (send draw-class draw-bitmap-on-bitmap
        (send game-board-bitmap get-bitmap) 0 0)
  (send draw-class draw-bitmap-on-bitmap
        (send game-status-bitmap get-bitmap) width-px 0))

;;uppdatera statuspanelens bitmap
(define/private (update-game-status-bitmap)
  (send game-status-bitmap clear)
  (send game-status-bitmap draw-bitmap-on-bitmap
        (send *image-store* get-image 'bg-status) 0 0)

  (define row-px 140)

  (for-each  (lambda (player)
               (send game-status-bitmap draw-bitmap-on-bitmap
                     (send player get-status-bitmap)
                     0
                     row-px)
               (set! row-px (+ row-px 100)))
             (get-field inner-list players)))

;;updatera spelplanen och allas objektposition i olika bitmaps.
;; Samt kollar om spelaren kolliderar med flammorna
(define/private (update-game-logic)
  (send game-board-bitmap clear)

  ;;håll koll på alla bomberna i bomblistan
  (for-each  (lambda (bomb)
               (send game-board-bitmap draw-bitmap-on-bitmap
                     (send bomb get-bitmap)
                     (* *blocksize* (get-field x-pos bomb))
                     (* *blocksize* (get-field y-pos bomb)))
               (if(send bomb gone-off?)
                  (on-bomb-explosion bomb)))
             (get-field inner-list bombs))

  ;;håll koll på alla bomberna i flammlistan och
  ;;kolla kollisioner mellan spelare och flammor
  (for-each  (lambda (flame)
               (map  (lambda (player)
                       (if(eq? 'flame ;;
                               (send flame collition?
                                     (get-field x-pos player)
                                     (get-field y-pos player)))
                          (on-die player flame)))
                     (get-field inner-list players))
               (send game-board-bitmap draw-bitmap-on-bitmap
                     (send flame get-bitmap)
                     (* *blocksize* (send flame get-x-pos))
                     (* *blocksize* (send flame get-y-pos)))
               (if(send flame gone-off?)
                  (send bomb-flames remove-from-list! flame)))
             (get-field inner-list bomb-flames))

  ;;håll koll på timers
  (for-each  (lambda (to-do)
               (if(send to-do gone-off?)
                  (begin
                    (send to-do run-proc)
                    (send to-do-list remove-from-list! to-do))));;run proc
             (get-field inner-list to-do-list))


  ;;håll koll på powerups
  (for-each  (lambda (powerup)
               (send game-board-bitmap draw-bitmap-on-bitmap
                     (send powerup get-bitmap)
                     (* *blocksize* (get-field x-pos powerup))
                     (* *blocksize* (get-field y-pos powerup))))
             (get-field inner-list powerups))

  ;;alla spelare
  (for-each  (lambda (player)
               (send game-board-bitmap draw-bitmap-on-bitmap
                     (send player get-bitmap)
                     (- (send player get-x-pos-px) 5)
                     (- (send player get-y-pos-px) 35)))
             (get-field inner-list players)))))
```

**;; ==== gui-class.ss**

```
;; ---------------------------------------------------------------------
;; GUI, skapa gui för spelet.
;; ---------------------------------------------------------------------
(define game-gui%
  (class object%
    (super-new)
    (init-field window-name width height image-buffer logic-class)
    (define gui-frame (new frame%
                           [label window-name]
                           [min-width width]
                           [min-height height]))



    ;; visa gui och fokusera tangentbord på canvas
    (define/public (show-gui)
      (send gui-frame show #t)
      (send gui-canvas focus));; flytta fokus till canvas, för att ta key events

    ;; göm gui och stoppar *game-loop*
    (define/public (hide-gui)
      (send gui-frame show #f)
      (send *game-loop* stop-loop))

    ;; uppdatera guit för att ladda om nya bitmaps
    (define/public (redraw)
  (send gui-canvas on-paint))

;;retunera bredd
(define/public (get-width)
  (send gui-canvas get-width))

;;retunera höjd
(define/public (get-height)
  (send gui-canvas get-height))


;;Hämta en ny bitmap från den globala bitmappen
;; som sattes via image-buffer-argumentet när objektet skapades.
(define (draw-canvas canvas dc)
  (send image-buffer get-image canvas dc))



;;Anropas utifrån för att skicka vidare
;;key-events från canvasen i denna klass.
(define/public (update-keys-down)
  (send gui-canvas send-key-events))

;;panelen där canvas är placerad
(define top-panel (new vertical-panel%
                       [parent gui-frame]
                       [alignment '(center center)]
                       [min-height (get-field height image-buffer)]
                       [min-width (get-field width image-buffer)]))

;;en samling av knappar
(define bottom-panel (new vertical-panel%
                       [parent gui-frame]
                       [alignment '(right top)]))

(define gui-canvas
  (new user-interact-canvas%
       [parent top-panel]
       [paint-callback draw-canvas]
       [on-key-event-callback
        (lambda(key)(send logic-class handle-key-event key))]
       [min-height (get-field height image-buffer)]
       [min-width (get-field width image-buffer)]
       [stretchable-width #f]
       [stretchable-height #f]))

;;kontrollpanel
(define controllpanel (new horizontal-panel%
                       [parent bottom-panel]
                       [alignment '(right center)]))

;;start-/pausknapp
(define startbutton (new button% [parent controllpanel]
                      [label "Paus"]
                      [callback (lambda (button event)
                                  (if(send *game-loop* running?)
                                     (begin
                                       (send *game-loop* stop-loop)
                                       (send startbutton set-label "Start"))
                                     (begin
                                       (send *game-loop* start-loop)
                                       (send startbutton set-label "Paus"))))]))


  ;;Stängknapp, stoppar *game-loop* för att spara cpu
  (new button%
       [parent controllpanel]
       [label "Quit"]
       [callback (lambda (a b) (hide-gui))])

  (define gui-menu-bar
    (instantiate menu-bar%
      (gui-frame)))

  (define gui-menu
    (instantiate menu%
      ("Menu" gui-menu-bar)))

  (instantiate menu-item%
    ("Quit" gui-menu (lambda (a b) (hide-gui))))))
```

**;; ==== help-classes.ss**

```
;; ---------------------------------------------------------------------
;; class list-object% för att lägga till och ta bort enkelt från listor
;; ---------------------------------------------------------------------

(define list-object%
  (class object%
    (super-new)
    (field
     (inner-list '()))

    ;;lägg till i listan
    (define/public (add-to-list! wath-to-add)
      (set! inner-list
            (cons
             wath-to-add
             inner-list)))

    ;;ta bort från listan
    (define/public (remove-from-list! wath-to-rem)
      (set! inner-list (remv wath-to-rem inner-list)))))
```

**;; ==== image-store.ss**

```
;; ---------------------------------------------------------------------
;; klass för enkelt ladda in bilder, samt retunera utifrån sök kriterier
;; ---------------------------------------------------------------------
(define image-store%
  (class object%
    (super-new)
(define image-list '())
(define anim 1)

;;add-rot-image name(symbol), load-list list ex:
;;'(('r . "img/r.bmp")('l . "img/l.bmp")
;;('d . "img/d.bmp")('u . "img/u.bmp"))
(define/private (add-rot-image name load-list)
  (define temp-list '())

  (map  (lambda (image)
          (if(string? (cdr image))
             (set! temp-list
                   (cons
                    (cons (car image)
                          (make-object bitmap% (cdr image) 'png/alpha))
                    temp-list))
             (set! temp-list
                   (cons
                    (cons (car image) (add-anim-image (cdr image)))
                    temp-list))
             ))
        load-list)
  ;;add to image list as (NAME .  '(('r . IMAGEDATA) ... ('u . IMAGEDATA)))
  (set! image-list (cons
                    (cons name temp-list)
                    image-list)))

;load data: ("img/red-player/r-" ".png" 5)
;returnerar '((0 . IMAGEDATA) ... (5 . IMAGEDATA)))
(define/private (add-anim-image load-data)
  (define temp-list '())
  (define prefix (car load-data))
  (define file-ending (cadr load-data))

  (define (loop i)
    (if(<= i (caddr load-data))
       (begin
         (set! temp-list
               (cons
                (cons i (make-object bitmap%
                          (string-append
                           prefix
                           (number->string i)
                           file-ending) 'png/alpha))
                temp-list))
         (loop (+ 1 i)))))

  (loop 0);;to load from 0
  temp-list)

;;detektera om ladda flera eller en bild
;;samt lägger till dessa i listor för senare bruk.
(define/public (add-image name image)
  (cond
    ((list? image)
     (add-rot-image name image))
    (else
     (set! image-list (cons
                       (cons name (make-object bitmap% image 'png/alpha))
                       image-list)))))



;;retunera en bitmap av en bild från listorna i klassen.
;; Söks fram med hjälp av assq
(define/public (get-image name . args)
  (let ((temp-cons (assq name image-list)))
    (cond
      ((not temp-cons)(error "error, wrong name " name))
      ((and
        (list? (cdr temp-cons));;kollar om det är flera bilder.
        (not (null? args))); och args inte tom
       (get-image-rot (car args) (cdr temp-cons) (cdr args)))
           ;; anropar sig själv med flera bild listan.
      ((list? (cdr temp-cons))(error "You need a argument to select image"))
      (else
       (cdr temp-cons)))))

;;hjälp funktion
(define/private (get-image-rot name image-list-2 . args)
  (let ((temp-cons (assq name image-list-2)))
    (cond
      ((not temp-cons)(error "error, wrong name 2"))
      ((and
        (list? (cdr temp-cons));;kollar om det är flera bilder.
        (not (null? args))); och args inte tom
       (get-image-anim (car args) (cdr temp-cons)))
      (else
       (cdr temp-cons)))))

;;hjälp funktion
(define/private (get-image-anim name image-list-2)
  (let ((temp-cons (assq (car name) image-list-2)))
    (cond
      ((not temp-cons)(error "error, wrong name 3" name))
      (else
       (cdr temp-cons))))))

;; ---------------------------------------------------------------------
```

**;; ====loop-class.ss**

```
;; ---------------------------------------------------------------------
(define loop-this-proc%
  (class object%
    (super-new)
    (init-field function-to-loop fps)
    (define should-run #f)
    (define paustime-timestamp-stop 0)
    (define paustime-tot 0)
    ;;metod för att starta loopen
    (define/public (start-loop)
      (when (not should-run)
        (set! should-run #t)
        (if(not (= 0 paustime-timestamp-stop))
           (begin
             (set! paustime-tot
                   (+ paustime-tot (- (current-inexact-milliseconds)
                                      paustime-timestamp-stop)))
             (set! paustime-timestamp-stop 0)))
        (thread loop)))

    ;;för att stoppa loopen
    (define/public (stop-loop)
      (set! should-run #f)
      (set! paustime-timestamp-stop (current-inexact-milliseconds)))

    ;;returnerar sant eller falskt beroende på om loopen körs
    (define/public (running?)
      should-run)

    ;;sekunder per frame
    (define (fps->seconds fps)
      (/ 1 fps))

    ;;hämta nuvarande millisekund
    (define/public (get-current-m-sec)
      (- (current-inexact-milliseconds) paustime-tot))

    ;;låter loopen sova visst antal sekunder för att den inte ska gå för fort
    (define sleep-time (fps->seconds fps))

    ;;loopen som kör igenom funktionen som ska loopas
    (define (loop)
      (when should-run
        (function-to-loop)
        (sleep sleep-time)
        (loop)))))
```

**;; ==== main.ss**

```
;; ----------------------------------------------------------------------
;; huvudfilen
;; ----------------------------------------------------------------------
(require scheme/date);; tid f�r bomber och liknande.
(require racket/string);; f�r att ladda in bilder
(load "help-classes.ss");; sm� hj�lpklasser f�r listor mm.
(load "draw-class.ss")
(load "image-store.ss")
(load "player-class.ss")
(load "game-board-class.ss")
(load "user-interact.ss")
(load "game-logic.ss")
(load "powerup-class.ss")
(load "bomb-class.ss")
(load "flame-class.ss")
(load "gui-class.ss")
(load "loop-class.ss")
(load "timer-class.ss")


;; ----------------------------------------------------------------------
;; globala objekt
;; ----------------------------------------------------------------------

;; Storleken p� blocken i spelplanen
(define *blocksize* 30)

;;Bildinladdningsfunktion s� att det inte ska lagga n�r senare.
(define *image-store*
  (new image-store%))

(send *image-store* add-image 'red-player
      '((r . ("img/red-player/r-" ".png" 5))
        (l . ("img/red-player/l-" ".png" 5))
        (u . ("img/red-player/u-" ".png" 5))
        (d . ("img/red-player/d-" ".png" 5))))

(send *image-store* add-image 'blue-player
      '((r . ("img/blue-player/r-" ".png" 5))
        (l . ("img/blue-player/l-" ".png" 5))
        (u . ("img/blue-player/u-" ".png" 5))
        (d . ("img/blue-player/d-" ".png" 5))))

(send *image-store* add-image 'invincible "img/invincible.png")


(send *image-store* add-image 'bomb-1 "img/bomb1.png")
(send *image-store* add-image 'bomb-2 "img/bomb2.png")

(send *image-store* add-image 'max-panel "img/max-panel.png")
(send *image-store* add-image 'power-panel "img/power-panel.png")
(send *image-store* add-image 'heart-panel "img/heart-panel.png")

(send *image-store* add-image 'flame-big
      '((x . "img/flame-big-h.png")
        (y . "img/flame-big-v.png")))

(send *image-store* add-image 'flame-small
      '((x . "img/flame-small-h.png")
        (y . "img/flame-small-v.png")))

(send *image-store* add-image 'powerup-multi-bomb "img/max-image.png")
(send *image-store* add-image 'powerup-speed "img/speed-powerup.png")
(send *image-store* add-image 'powerup-stronger-bomb "img/power-image.png")
(send *image-store* add-image 'non-dest-block "img/non-dest-block.png")
(send *image-store* add-image 'dest-block "img/dest-block.png")
(send *image-store* add-image 'bg "img/bg.png")
(send *image-store* add-image 'bg-status "img/bg-status.png")

;; ----------------------------------------------------------------------
;; Spellogik
;; ----------------------------------------------------------------------

;;Global klocka som kan pausas
(define (*current-m-sec*)
  (send *game-loop* get-current-m-sec))


;;Skapar en logik m.h.a. spellogiks-klassen
(define bomberman-logic
  (new game-logic%
       [height 21]
       [width 21]
       [height-px 630]
       [width-px 630]))

;;Globala bitmapen som laddas in via gui varenda loop.
(define *draw*
  (new drawing%
       [width 800];;canvas-/bitmapsstorlek
       [height 630]))

;;spelets fönser
(define *gui*
  (new game-gui%
       [window-name "Bomberman"]
       [width 800];;f�nsterstorlek
       [height 650]
       [image-buffer *draw*];;bildbuffer, laddar bilden till canvas
       [logic-class bomberman-logic]))
;; logic-class -logisk klass att s�nda tangentbords-nedtryckningar till.


;; ----------------------------------------------------------------------
;; L�gga till spelare
;; ----------------------------------------------------------------------
;;(add-key-board-player new-name x y dxy number-of-lives color keybord-bindings)
;;spelare 1
(send bomberman-logic add-key-board-player "Jocke" 1 1 10 5 'blue-player
      '((#\w . u)
        (#\a . l)
        (#\s . d)
        (#\d . r)
        (#\q . drop)))
;;spelare 2
(send bomberman-logic add-key-board-player "Pocke" 19 19 10 5 'blue-player
      '((#\i . u)
        (#\j . l)
        (#\k . d)
        (#\l . r)
        (#\b . drop)))
;;spelare 3
(send bomberman-logic add-key-board-player "Tocke" 1 19 10 5 'red-player
      '((up . u)
        (left . l)
        (down . d)
        (right . r)
        (#\0 . drop)
        (numpad0 . drop)))
;;spelare 4
(send bomberman-logic add-key-board-player "Focke" 19 1 10 5 'red-player
      '((#\8 . u)
        (#\4 . l)
        (#\5 . d)
        (#\6 . r)
        (#\7 . drop)
        (numpad8 . u)
        (numpad4 . l)
        (numpad5 . d)
        (numpad6 . r)
        (numpad7 . drop)))

;; Procedurerna som ritar om br�dan fr�n huvudtr�den.
(define (draw)
  ;Skicka tangenter till spellogiken en g�ng per loop-varv
  (send *gui* update-keys-down)
  (send *draw* clear);; rensa bitmap
  ;;uppdatera bitmap och skicka till main bitmappen
  (send bomberman-logic update-scene *draw*)
  (send *gui* redraw));; Rita om gui f�r att se nya bitmapen




;; ----------------------------------------------------------------
;; G�rs alltid innan start
;; ----------------------------------------------------------------
(send *draw* clear);; Rensar buffern som ritar
(send *gui* show-gui);; startar gui
(send *gui* redraw);; uppdaterar canvas
(send bomberman-logic init-gameboard)

;; ----------------------------------------------------------------
;; Huvudloop
;; ----------------------------------------------------------------
(define *game-loop*
  (new loop-this-proc%
       [function-to-loop draw]
       [fps 24]));; anropar draw spec i update-graphic


(send *game-loop* start-loop);; startar loopen

;; ----------------------------------------------------------------------
```

**;;====player-class.ss**

```
;;klass player%, skapar spelarobjektet
;; ----------------------------------------------------------------------
(define player%
  (class object%
    (super-new)
    (init-field x-pos y-pos dxdy name lives color)
```

```
(field
 (x-pos-px (* x-pos *blocksize*))
 (y-pos-px (* y-pos *blocksize*))
 (spawn-x-pos x-pos)
 (spawn-y-pos y-pos)
 (type 'player)
 (points 0)
 (radius 1)
 (delay 5000);; bombfördröjning i ms
 (bomb-count 1)
 (number-of-bombs 0);; hur många bomber på spelplanen
 (last-bomb-timestamp 0)
 (invincible-in-m-sec 10000)
 (timestamp-invincible 0)
 (last-bomb-place '());; (x . y)
 (direction 'd);;spelarens riktning
 (moving #f);;om spelaren rör sig eller inte
 (animation 1);;nuvarande frame i animationen
 (animation-start 1);;var den startar
 (animation-stop 5);;var den stannar
 (animation-duration 4);frames med samma bild
 (animation-duration-count 0);;frameräknare
 (name-font (make-object font% 15 'default 'normal 'bold))
 (status-font (make-object font% 10 'default 'normal 'bold)))

;;returnerar sant om tiden för odödlighet har gått ut
(define/public (possible-to-die?)
  (<= (+ timestamp-invincible invincible-in-m-sec)
      (*current-m-sec*)))

;;funktion för att se om det går att lägga bomber just då,
;;kollar med hur många man har på spelplanen
;;och hur många man har möjlighet att lägga
(define/public (can-bomb?)
  (and
   (< number-of-bombs bomb-count)
   (or
    (< (+ last-bomb-timestamp 1000)
       (*current-m-sec*)); en sek fördröjning eller
    (not (and
          (eq? x-pos (car last-bomb-place))
          (eq? y-pos (cdr last-bomb-place)))))));; inte samma ställe


;;funktion för att återfå bomber att lägga ut när de har sprängts
(define/public (remv-bomb)
  (if(< 0 number-of-bombs)
     (set! number-of-bombs (- number-of-bombs 1))))


;;lägga ut bomb, sätter timer och sparar platsen.
(define/public (add-bomb)
  (set! number-of-bombs (+ number-of-bombs 1))
  (set! last-bomb-timestamp (*current-m-sec*))
  (set! last-bomb-place (cons x-pos y-pos)))

;;dö-funktion som återsätter bomber mm till startvärde och minskar liv med 1
(define/public (die)
  (set! lives (- lives 1))
  (set! number-of-bombs 0)
  (set! bomb-count 1)
  (set! dxdy 10)
  (set! radius 1)
  (set-x! spawn-x-pos)
  (set-y! spawn-y-pos)
  (set! timestamp-invincible (*current-m-sec*))
  (set! direction 'd))

;;sätt px pos och logisk pos
(define/public (set-x-pos-px! x)
  (set! x-pos-px x)
  (set! x-pos (quotient
             (+ x (/ *blocksize* 2))
             *blocksize*)))

;;sätt px pos och logisk pos
(define/public (set-y-pos-px! y)
  (set! y-pos-px y)
  (set! y-pos (quotient
             (+ y (/ *blocksize* 2))
             *blocksize*)))

;;sätt px pos och logisk pos
(define/public (set-x! x)
  (set! x-pos x)
  (set! x-pos-px (* x *blocksize*)))

;;sätt px pos och logisk pos
(define/public (set-y! y)
  (set! y-pos y)
  (set! y-pos-px (* y *blocksize*)))

;;hämta px pos
(define/public (get-y-pos-px)
  y-pos-px)

;;hämta px pos
(define/public (get-x-pos-px)
  x-pos-px)

;; när man förflyttar sig
(define/public (set-dir! dir)
  (set! moving #t)
  (set! direction dir))


(define status-bitmap
  (new drawing%
     [width 170];;canvas-/bitmapsstorlek
     [height 100]))
```

```
;;Metod för att olika värden i status-bitmapen,
;;som ligger till höger när spelet körs
(define/public (update-status-bitmap)
  (send status-bitmap clear)
  (send status-bitmap set-background-color! 255 255 255 1)
  (send status-bitmap draw-text name 10 0 name-font)
  (send status-bitmap draw-bitmap-on-bitmap
        (send *image-store* get-image 'max-panel) 60 40)
  (send status-bitmap draw-bitmap-on-bitmap
        (send *image-store* get-image 'heart-panel) 20 40)
  (send status-bitmap draw-bitmap-on-bitmap
        (send *image-store* get-image 'power-panel) 100 40)

  (send status-bitmap draw-text
        (number->string lives) 40 40 status-font)
  (send status-bitmap draw-text
        (number->string bomb-count) 80 40 status-font)
  (send status-bitmap draw-text
        (number->string radius) 120 40 status-font))

;;Metod för att uppdatera status-bitmapen samt returnera den
(define/public (get-status-bitmap)
  (update-status-bitmap)
  (send status-bitmap get-bitmap))

(define bitmap
  (new drawing%
       [width 40];;canvas-/bitmapsstorlek
       [height 62]))



(define/private (update-animation-help)
  (if moving
      (if(< animation-duration-count animation-duration)
         (set! animation-duration-count (+ animation-duration-count 1))
         (begin
           (set! animation-duration-count 0)
           (if(< animation animation-stop)
              (set! animation (+ animation 1))
              (set! animation animation-start))))
      (set! animation 0)))

;;uppdatera bitmap, lägger ev. till ödödlighetsbubbla om ej möjligt att dö
(define/public (update-bitmap)
  (send bitmap clear)

  (update-animation-help)
  (set! moving #f)
  (send bitmap draw-bitmap-on-bitmap
        (send *image-store* get-image color direction animation) 0 0)
  (if(not (possible-to-die?))
     (send bitmap draw-bitmap-on-bitmap
           (send *image-store* get-image 'invincible) 0 0)))

;;uppdatera och returnera bitmap
  (define/public (get-bitmap)
    (update-bitmap)
    (send bitmap get-bitmap))))
```

**;;====powerup-class.ss**

```
;; ---------------------------------------------------------------------
;;klass för att skapa powerup
;; ---------------------------------------------------------------------
(define powerup%
  (class object%
    (super-new)
    (init-field x-pos y-pos)
    ;;sätt en slumpmässig typ av powerup vid skapande av objekt
    (field (type (cdr (assq (random 3)
                         '((0 . powerup-speed)
                           (1 . powerup-multi-bomb)
                           (2 . powerup-stronger-bomb))))))

    (define/public (set-x! x)
      (set! x-pos x))

    (define/public (set-y! y)
      (set! y-pos y))

    ;; Tar xpos och ypos, om en kollision sker
    ;;returneras vilken typ kollisionen sker mot, annars returneras falskt
    (define/public (collition? xpos ypos)
      (if(and (= xpos x-pos)
              (= ypos y-pos))
         type;;
         #f))

    ;;Funktion för att avgöra på vilket sätt ens förmågor ska
    ;;ändras beroende på vilken powerup som plockats
    (define/public (use-power-up player)
      (cond
        ((eq? type 'powerup-speed)(add-speed player))
        ((eq? type 'powerup-multi-bomb)(add-multi-bomb player))
        ((eq? type 'powerup-stronger-bomb)(add-stronger-bomb player))))

    (define/private (add-speed player)
      (if (< (get-field dxdy player) 15)
          (begin;;snabbfix för osynk i kollisionshanteringen
            (set-field! dxdy player (+ (get-field dxdy player) 5))
            (send player set-x! (get-field x-pos player))
            (send player set-y! (get-field y-pos player)))))

    (define/private (add-multi-bomb player)
      (set-field! bomb-count player (+ (get-field bomb-count player) 2)))

    (define/private (add-stronger-bomb player)
      (set-field! radius player (+ (get-field radius player) 1)))
    ;;skickar bitmapen, anropad från spellogiken för att uppdatera skärmen.
    (define/public (get-bitmap)
      (send *image-store* get-image type))))

;; ---------------------------------------------------------------------
```

**;; ====timer-class.ss**

```
;; ---------------------------------------------------------------------
;;Klass för att få till timers till bomberna
(define make-timer%
  (class object%
    (super-new)
    (init-field delay proc args)
    ;;tidsstämpel i form av millisekunder
    (define timestamp (*current-m-sec*))


    ;;returnerar sant om bomben har sprängts
    (define/public (gone-off?)
      (<= (+ timestamp delay) (*current-m-sec*)))
    ;; applicerar argument på en procedur
    (define/public (run-proc)
      (apply proc args))))
```

**;;====user-interact.ss**

```
;; --------------------------------------------------------------------
;;Klass för att interagera med canvas via tangentbordet
;; --------------------------------------------------------------------
(define user-interact-canvas%
  (class canvas%
    (override on-char)
    (init-field on-key-event-callback)

    (define keysdown '());;Lista med alla nedtrycka knappar
    (define (on-char key-event)
      (let ((release (send key-event get-key-release-code))
            (key (send key-event get-key-code)))

;;Kollar att det inte är ett release event och att den inte redan är nedtryckt
        ;; Annars tas den bort från keysdown listan
        ;; press eller down beroende på version av drracket
        (if(and (not (member key keysdown))
                (or (eq? release 'press) (eq? release 'down)))
           (set! keysdown (cons key keysdown))
           (set! keysdown (remv release keysdown)))));; end on-char

;;Skickar vidare alla nedtryckta knappar till on-key-event-callback-funktionen.
;;Som defineras när klassen skapas. Som det är nu så skickas dem till game-logic
    ;;Denna metod anropas utifrån via gui-classen via
    ;;*game-loop* för att skicka vidare nedtrycka knappar.
    (define/public (send-key-events)
      (for-each
       (lambda (key)
         (on-key-event-callback key))
       keysdown))

    (super-instantiate ())))
```