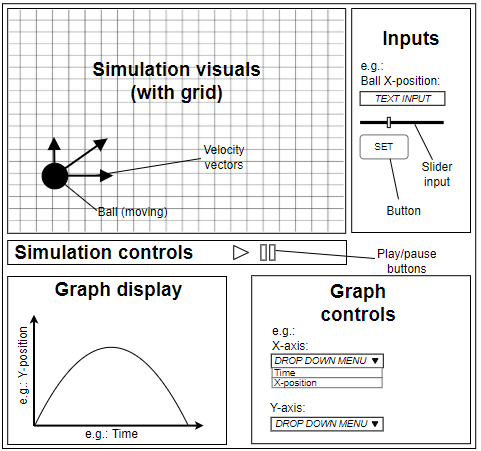
## Criterion B: Design

### Page layout



The inputs section will have the following inputs (which are also outputs, as the the value of the variable will be displayed in the text input area):

1. X-position
2. Y-position
3. X-Velocity
4. Y-Velocity
5. Net velocity
6. Angle of velocity
7. Canvas dimensions

### System variables/constants

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier** | **Type** | **Description** | **Initial value** | **Used for input? (y/n)** |
| canvasWidthReal | float | Stores the width of the canvas in meters, used for calculating physics. | 50 | y |
| canvasHeightReal | float | Stores the height of the canvas in meters, used for calculating physics. | 25 | y |
| canvasWidth | float | Stores the width of the canvas in pixels, used for drawing animations. | 1000 | n |
| canvasHeight | float | Stores the height of the canvas in pixels, used for drawing animations. | 500 | n |
| startTime | float | The time at which the animation starts, when user presses play. | null | n |
| isRunning | boolean | True if animation is actively running, false if it is paused. | false | n |
| startVelX | float | The X-velocity at the start of each phase of the animation, after play is pressed. | 0 | y |
| startVelY | float | The Y-velocity at the start of each phase of the animation, after play is pressed. Updated after every pause by v=u+at. | 0 | y |
| starPosX | float | The X-position at the start of each phase of the animation, after play is pressed. Updated to current position after every pause. | 0 | y |
| startPosY | float | The Y-position at the start of each phase of the animation, after play is pressed. Updated to current position after every pause. | 0 | y |
| timerTime | float | Time that is displayed on the timer. | 0 | n |
| startTimerTime | float | The timer time at the start of each phase of the animation, after play is pressed. | 0 | n |
| canvasArea | constant float | The area of the canvas that is kept constant as the width and height change. | 1000\*500 | n |
| ball | object | Stores the properties and methods of the ball. | N/A | n |
| g | float | Acceleration due to gravity. | -9.81 | n |

### Ball object properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier** | **Type** | **Description** | **Initial value** |
| posRealX | float | Stores the x-position of the ball in meters. | 0 |
| posRealY | float | Stores the y-position of the ball in meters. | 0 |
| posX | float | Stores the x-position of the ball in pixels. | 0 |
| posY | float | Stores the y-position of the ball in pixels. | 0 |
| velX | float | Stores the x-velocity of the ball in ms⁻¹. | 0 |
| velY | float | Stores the y-velocity of the ball in ms⁻¹. | 0 |
| speed | float | Stores the net velocity of the ball in ms⁻¹. | 0 |
| angle | float | Stores the angle of the ball’s velocity off the horizontal in degrees. | 0 |
| radius | float | Radius of the ball. Used for ensuring the ball does not clip through the “floor” of the simulation area. | 0 |

### System methods

##### onLoad( ):

BALL.draw()

drawGrid()

printData(0)

##### animate( ):

input STARTTIME

input TIME

input ISUSINGTIMER

IMPACT = false

IMPACTTIME = (-STARTVELY - sqrt(STARTVELY^2 - 2\*G\*STARTPOSY)) / G

if TIME > IMPACTTIME

BALL.POSREALY = 0

ISRUNNING = false

IMPACT = true

else

BALL.POSREALY = STARTPOSY + (STARTVELY\*TIME) + (0.5\*G\*TIME^2)

BALL.POSREALX = STARTPOSX + (STARTVELX\*TIME)

BALL.VELX = STARTVELX

BALL.VELY = STARTVELY + G\*TIME

if IMPACT = true

if TIMER = true

TIMERTIME = STARTTIMERTIME + IMPACTTIME

output "ball has reached the ground at" + TIME

else

if ISUSINGTIMER = true

TIMERTIME = STARTTIMERTIME + TIME

BALL.draw()

printData(TIMERTIME)

if ISRUNNING = true

animate()

else

STARTTIME = null

STARTTIMERTIME = TIMERTIME

STARTVELX = BALL.VELX

STARTVELY = BALL.VELY

STARTPOSX = BALL.POSREALX

STARTPOSY = BALL.POSREAL

##### startClick( ):

isRunning = true

animate()

##### ball.draw( )

input VECTORS

input TRAIL

clearScreen()

drawCircle(THIS.POSX, THIS.POSY, THIS.RADIUS, CANVAS1)

if VECTORS = true

SCALE = 5

drawArrow(THIS.POSX, THIS.POSY, THIS.POSX + (THIS.VELX \* SCALE), THIS.POSY, CANVAS1)

drawArrow(THIS.POSX, THIS.POSY, THIS.POSX, THIS.POSY - (THIS.VELY \* SCALE), CANVAS1)

drawArrow(THIS.POSX, THIS.POSY, THIS.POSX + cos(THIS.ANGLE)\*THIS.SPEED\*SCALE, THIS.POSY - sin(THIS.angle)\*THIS.SPEED\*SCALE, CANVAS1)

if TRAIL = true

if ISRUNNING = true

drawDot(THIS.POSX, THIS.POSY, CANVAS1)

##### printData(TIME):

output truncate(BALL.POSREALX, 4)

output truncate(BALL.POSREALY, 4)

output truncate(BALL.VELX, 4)

output truncate(BALL.VELY, 4)

output truncate(BALL.SPEED, 4)

output truncate(BALL.ANGLE, 4)

if NOT time = null

output truncate(TIME, 4) + "s"

##### resizeCanvas( ):

if ISRUNNING = false {

input WIDTHIN

input HEIGHTIN

if validateFloat([widthIn, heightIn]) = true

CANVASWIDTHREAL = WIDTHIN

CANVASHEIGHTREAL = HEIGHTIN

CANVASWIDTH = sqrt(canvasArea \* CANVASWIDTHREAL / CANVASHEIGHTREAL)

CANVASHEIGHT = sqrt(canvasArea \* CANVASHEIGHTREAL / CANVASWIDTHREAL)

BALL.draw()

drawGrid()

printData(null)

##### drawArrow(X1, Y1, X2, Y2):

HEADLENGTH = 10

HEADANGLE = 30

ANGLE = arctan((Y2 - Y1)/(X2 - X1))

drawLine(X1, Y1, X2, Y2)

drawLine(X2, Y2, X2 - HEADLENGTH \* cos(ANGLE - HEADANGLE), Y2 - HEADLENGTH \* sin(ANGLE - HEADANGLE))

drawLine(X2, Y2, X2 - HEADLENGTH \* cos(ANGLE + HEADANGLE), Y2 - HEADLENGTH \* sin(ANGLE + HEADANGLE))

##### drawGrid():

clearCanvas(GRIDCANVAS)

loop for COUNT = 0 to CANVASHEIGHTREAL

POS = CANVASHEIGHT - (BALL.RADIUS + (COUNT \* CANVASHEIGHT / CANVASHEIGHTREAL))

drawLine(0, POS, CANVASWIDTH, POS, GRIDCANVAS)

loop for COUNT = 0 to CANVASWIDTHREAL

POS = BALL.RADIUS + (COUNT \* CANVASWIDTH / CANVASWIDTHREAL)

drawLine(POS, 0, POS, CANVASHEIGHT, GRIDCANVAS)

##### validateFloat(inputs[]):

OUT = true

loop for COUNT = 0 to INPUTS.length

if isNaN(inputs[COUNT]) = true

OUT = false

if NOT OUT = true

output "Inputs needs to be numbers!"

return OUT

### Expected methods

These methods are technical to the framework I’ll be working with and are abstracted as the following in the code above.

drawLine(x1, y1, x2, y2, canvas):  
Draws a line on the canvas from (x1, y1) to (x2, y2) on selected canvas.

drawDot(x, y, canvas):  
Draws a dot on the canvas at (x, y), at selected canvas.

clearScreen(canvas):  
Clears the canvas.

truncate(string, number):  
Truncates given string to number of significant figures.

isNaN(string):  
Ouptputs true if string is not a valid number, otherwise outputs false.