Photo courtesy of Takahiro Omura

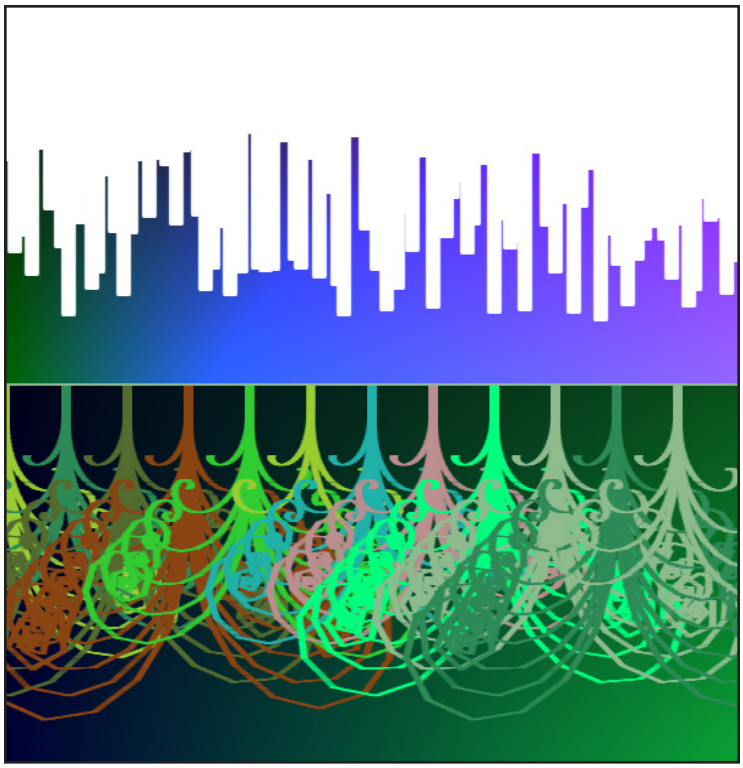
Ajuna Kyaruzi is a third-year Computer Science major at Grinnell College from Dar es Salaam, Tanzania. She has been working as the web designer of Rootstalk *since Spring 2015. Ajuna joined the journal to tie her interests in Computer Science with publishing, which is a new field for her. Having grown up in Tanzania, she has found living in the prairie for three years quite different and inspiring. While not working on* Rootstalk, *Ajuna directs Grinnell Appdev, tutors students in Computer Science, and is an active member of the African Caribbean Student Union.*

Underground Skies

Ajuna Kyaruzi

Underground Skies is digital artwork inspired by the small Iowa town of Grinnell's beautiful sunsets and the restored prairie in the Conard Environmental Research Area (CERA). The piece uses computer-generated fractals to highlight the intricate links between and beneath the prairie region's natural and urban landscapes. Kyaruzi draws on Professor Sam Rebelsky's work in Media Scheme and GIMP—open-source software that helps his Introduction to Computer Science students create images with code—to foreground the richness of the natural and social codes that lie just beneath the prairie world we inhabit.

Click on the image below to visit a YouTube video of the image being generated. The code used to create this image is to the right of this page.



Screen shot courtesy of Ajuna Kyaruzi

```
#lang racket
(require gigls/unsafe)
;;; File:
;;;   RTK-UndergroundSkies.rkt
;;; Authors:
;;;   Ajuna S. Kyaruzi
;;;   Samuel A. Rebelsky
;;; Summary:
;;;   Code for digital art submission to the Spring 2016
;;;   issue of Rootstalk
;;;   https://github.com/kyaruzia17/RTK-UndergroundSkies

; +----------------+---------------------------------
; | Main Procedure |
; +----------------+
;;; Procedure:
;;;   turtle-prairie
;;; Parameters:
;;;   side, a natural number
;;; Purpose:
;;;   Creates randomized digital art piece “Underground
;;;   Skies”
;;; Produces:
;;;   img, an image
;;; Pre-conditions:
;;;   [No additional]
;;; Post-conditions:
;;;   img is a square image of side length side
(define turtle-prairie
  (lambda (side)
    (turtle-wander (image-show (prairie-background side)))))


; Create the colorful square background of sidelength side
(define prairie-background
  (lambda (side)
    (image-compute (lambda (col row)
                     (cond
                       [(< row (/ side 6)) RGB-WHITE]
                       [(< (/ side 6) row (/ side 2))
                        (irgb (* col 0.30)
                              (* row 0.27)
                              (* col row 0.0052))]
                       [(= row (/ side 2)) RGB-WHITE]
                       [else  (irgb (* col 0.016)
                                    (* col row 0.00045)
                                    (* row 0.091))]))
                   side side)))


; Modifies the image with `cityscape` and `ground` procedures
(define turtle-wander
  (lambda (img)
    (let* ([colt (turtle-new img)]
           [side (image-width img)]
           [half (* 0.5 side)])
      (turtle-teleport! colt (* 0.1 side) half)
      (turtle-set-color! colt RGB-WHITE)
      (turtle-set-brush! colt “2. Block 01” 5)
      (turtle-face! colt 90)
      (cityscape colt side 0)
      (turtle-set-brush! colt “2. Block 01” 1)
      (ground colt side 0)
      (context-set-fgcolor! RGB-WHITE)
      (image-draw-line! img 0 half side half))))

; +--------------------------+---------------------------
---
; | Turtle Helper Procedures |
; +--------------------------+
; Directs the turtles to create the skycrapers above ground
(define cityscape
  (lambda (turtle side w)
    (let* ([half ( - (/ side 2) 1)]
           [max (/ side 6)]
           [increment (random 5)]
           [length (random (inexact->exact (/ side 4)))])
      (when (< w side)
        (turtle-teleport! turtle w max)
        (turtle-face! turtle 90)
        (turtle-forward! turtle length)
        (turtle-teleport! turtle (+ w 4) (/ length 2))
        (turtle-face! turtle 180)
        (turtle-forward! turtle 4)
        (cityscape turtle side (+ w 5 increment))))))


; Directs the turtles to create the intricate roots on the
; lower half of the image
(define ground
  (lambda (turtle side w)
    (turtle-set-color! turtle
                       (list-ref root-list
                                 (random (length root-
list))))
    (when (< w side)
      (turtle-roots turtle (+ 5 (random 15)) w (/ side 2))
      (ground turtle side (+ w 50)))))


; Directs the turtles to draw repetitive and layered “roots”
(define turtle-roots
  (lambda (turtle num x y)
    (let* ([xlist (lambda (n)
                    (map (lambda (x) (* x n 0.1))
                         (iota 30)))]
           [curr (if (odd? num)
                     (* -1 num)
                     num)])
      (when (> (abs num) 1)
        (turtle-teleport! turtle x y)
        (turtle-face! turtle 90)
        (for-each (lambda (len ang)
                    (turtle-forward! turtle len)
                    (turtle-turn! turtle ang))
                  (reverse (xlist (abs curr)))
                  (xlist (* 3 curr)))
        (turtle-roots turtle (- num 1) x y)))))

; +--------------------+---------------------------------
; | Color Definitions |
; +--------------------+
(define RGB-WHITE (irgb 255 255 255))
; List of brown and green colors to create the roots
(define root-list
  (append (map color-name->irgb
               (context-list-colors “brown”))
          (map color-name->irgb
               (context-list-colors “green”))))
```