Tracker One - Software Requirements Specification

# Introduction

This is a testing code project. Its purpose is to show coding and documentation capabilities. It should produce a running application to visualize entities movement on a board including their properties – shape, color, size etc’.

# Assumptions

* N input defines saving and trailing min 1, max 5.
* Entity location is the center of the shape image.
* Board is 500,500 pixels but 100,100 spots.
* A movement of 5 is 25 pixels

# Specs

* Display the entities at initial coordinates - x,y (as stored in a json file)
* Identify each entity by color, type, size and also name
* On startup, all entities will be displayed and the user will have the option to hide / show entities from the display
* A Start button will cause the entities to move until Stop button is pressed by the user.
* The user can input n last steps to be saved when Stop button is pressed.
* The entities will move in 5 seconds intervals. That is, all entities will perform a movement every round by 5 sec intervals.

## Movement specs:

* Movement direction - 4 options Up, Down, Left, Right
* Selected direction for each entity is random
* Each movement measures 5 units (pixels)
* The entity cannot go back to previous location on next step
* When Stop button is pressed the app will save the n last steps that the user set as input.

# Specs Notes

* The display grid is 100x100 (i.e. 0<=X<=100; 0<=Y<=100)
* There are three supported colors (red, green, blue)
* There are three supported entity types (circle, square and triangle)
* There are three supported entity sizes (small, medium, large)
* Max number of supported entities is 10
* JSON input file will be located at executable folder (testing input may vary).

# Specs Technical Notes

* The display grid is 100x100 is a gray square, located in a somewhat larger one to enable display of a shape that its center is at 0, 0 for example.
* We use a 500X500 pixel board. E.G. a single spot measures 5X5 pixels. A movement of 5, as required means a movement of 25 pixels.
* Display 0, 0 location, is bottom-left corner. Up/Down change at Y axis. Right/Left at X axis.
* Entity is a user control that combines an image of a shape and a label (entity id).
* The entity coordination represents the center of the shape.
* The note above can cause a shape to exceed the borders.
* Entity must remember, at minimum, its last location to prevent moving backwards, or at maximum n steps.
* We assume that as a default the app should save, at minimum, its current location at stop.

# Use Cases

## Startup

* Load data from json file
* Fill up data model – Note!! First 10 only. Consider log warming if json contains more than 10 entities.
* Show all existing entities (first 10 – see above).
* Draw check boxes (the number is dynamic!!!) and mark them checked. This should cause the entities to be shown.

## User presses Start button

* Timer starts to run at 5 seconds intervals.
* Every time elapsed all entities should move 5 spots in a random direction (Up, Down, Left, Right) but NOT backwards.
* The movement is tracked by a line drawn back in a length of **N** spots, as defined by the user (we enforce minimum 1 as default).

## User presses Stop button

* Timer stops
* Entities stop
* The N locations are saved to CSV file.

## User Check/Uncheck a Checkbox

* Show/Hide the related entity

## User selects N trailing spots

* On startup the default is 1
* On stop mode validate up to 5 and save N locations per entity.
* Os run mode (after start is pressed) use N as backwards steps counter for step trail. (Future feature that will not be implemented at this stage)

# Project Structure

The solution contains 2 projects:

* **Tracker\_One\_View** (UI) - This is the UI part and as minimum logic as possible.
  + Win form – We decided to kip it simple for now – KIS[S]. Although there may be issues with 3D later.
  + Holds the timer that instructs the logic layer and then refreshes the display.
* **Tracker\_One\_Core** – Holds the data access, data model and most of the logic
  + Data Access:
    - LoadJson() - Read json and parse to model
    - SaveCsv(List<Entities> data) - Save the data to csv
  + Business Logic
    - List of entities
    - Logging
    - RepositionEntities()
* Tracker\_One\_Test – Depends on time!!

# Tracker\_One\_Core - Business Logic & Model

## XEntity class

### Properties

* Id: string
* Name: string (100)
* Color: string (accept only red, green, blue)
* Shape: string (accept only circle, square, triangle)
* Size: String (accept only small, medium, large)
* X: int
* Y: int

## XentitiesMgr class

### Properties

* Data: List<XEntity>
* HistoryNunberOfSteps: int

### Methods

bool FillModel()

* Call access layer to retrieve json
* Fills the list (optionally by json deserializer)
* Returns true for success and false on error.

bool RepositionEntities()

* Iterates through the list of entities and for each of them (using sub methods)
  + Gets a random new direction (never back to previous step)
  + Calculates new position
  + Validates that new position is on the board (if needs return on these steps)
  + Sets new position
* Returns true for success and false on error.

Bool SaveDataToCsv()

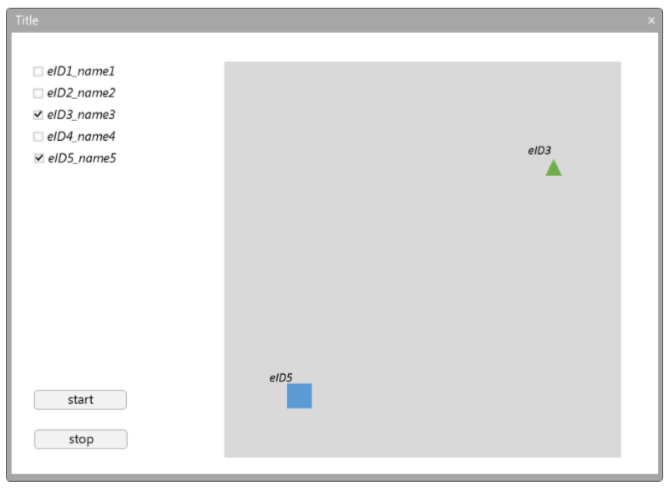
* Saves the data to csv file
* Save history steps according to HistoryNunberOfSteps
* Returns true for success and false on error.

Void SetHistoryNunberOfSteps(int numOfHisSteps = 1)

* Expects an int or default of 1
* Validate max 5
* Returns void

# Tracker\_One\_View - UI

## FrmMain

* Main UI Form   
  
* Entity\_UC – A single entity User Control  
  
* See Details below – Entity\_UC

## Entity\_UC

### Properties

* Id: string
* Name: string (100)
* Color: enum
* Shape: enum
* Size: enum
* X: int
* Y: int
* String: ImagePath
* RealX: int
* RealY: int

### Methods

Void SetPosision(int x, int y)

* Set RealX according to deta between X and RealX
* Set RealY according to deta between X and RealX

## HelperMethods

String GetImagePath(enum entityShape, enum EntitySize)

* Calculates the right image and return its path