Advanced Comp Sci Lab - Project 2

For Project 2 we are going to build a “Falling Sand Game” as described [here](https://en.wikipedia.org/wiki/Falling-sand_game). This app will include a graphical user interface for selecting the item to place as well as for controlling environmental components like gravity and Brownian (and other effects you will come up with on your own). You also will build the game environment engine to apply the world rules.

For an example of the type of experience you will build, you can download [The Powder Toy](https://powdertoy.co.uk/) and try it out.

# Requirements

For this project, you will need to deliver the following:

1. A GUI app that runs in a Window (not command line) for either a Mac or Windows device. You can choose to build this as a web app or for iOS or Android – but I think those will be significantly harder to debug.
2. A graphical user experience as described in the UX section.
3. A game engine that implements the Game Of Sand experience.
4. A set of 2-4 additional effects with at least one of them being a “growth” effect and another reacting to something other than gravity or Brownian (e.g. interacting with other elements, wind, etc.). You may copy effects from the Powder Toy but you must document that you took it from that game and how you adapted it.
5. You will provide unit tests for your game engine and instructions on how to run them.
6. You will provide test instructions for me in your readme.md so I can run the app and fully test all your features including running the unit tests.

You will be doing this project in teams of 2. I suggest one person own the UX and the other own the game engine. You will each need to implement an additional effect or 2 and add them to game canvas.

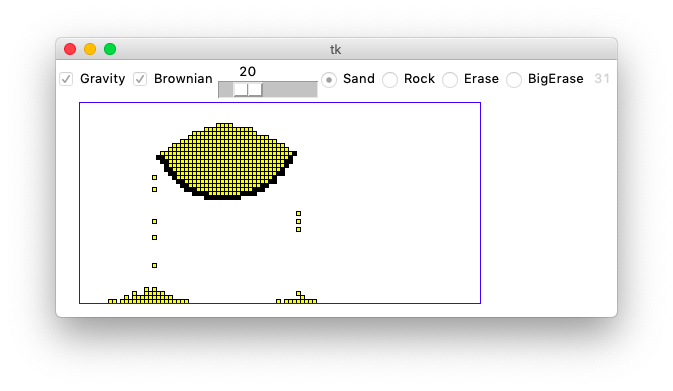
You may use whatever language and UX toolkit you choose. However, note that not every UX toolkit is available in every language and so your choice will need to be made as a team.

# Graphical User Experience

The app will have at a minimum, the following UX chrome:

* A single main Window.
* It will contain at least 1 toolbar and a 2D canvas for interacting with the game. You should draw blocks using color or other image effects in the canvas.
* The toolbars will support at minimum the following items:
  + Ability to select the tool you want to use in the 2D canvas.
    - Erase a single block.
    - Erase 4 block radius (called bigerase).
    - Sand
    - Rock
    - Additional effects you create.
  + Ability to turn gravity on and off.
  + Ability to turn Brownian on and off.
  + A slider or equivalent to set Brownian effect from 0-100.
  + Any controllers for your additional effects.

# Game of Sand

The first part of the project is to implement the Game of Sand.

## The Game Experience

You can select from either rock (black) or sand (yellow) and place them on the screen. The rocks cannot move but the sand is affected by the gravity and so moves down one position every second. So, you can create a rock pattern and then drop the sand to see how it flows over and around the rocks. You can also choose to erase sand or rock with a small eraser or a big eraser.

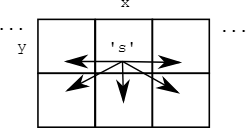
You can also control whether gravity is on or off from the checkbox. We are introducing a concept called Brownian which we will explain in more detail later, but this adds a bit of randomness to the game so sand can move left and right creating a more realistic experience.

## The 2D Sand Grid

The game engine contains a 2-dimensional grid of chars where every position holds one of three things:

1. Sand represented by 's'
2. Rock represented by 'r'
3. Empty represented by none or null.

In the sand world, each “turn” a sand 's' can move in one of five directions or stay where it is.



## Move Rules

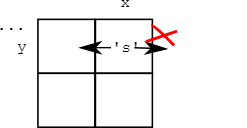
Assume we have a method called, is\_move\_ok, which takes a position containing sand and decides whether the rules of the world allow the sand to move from the initial position to the target position. For each sand on the board, we will test against the 5 possible options in the following order: left, right, down, down-left, and down-right. As soon as we find one that is allowed, we will move the sand to that position and remove it from the existing position. As you create new effect or object types, you can decide these rules for yourself and apply your effects.

For example, you may have an effect that grows and so it will not move to a new position, but may “grow” into that new position if it is empty or it may combine with another object to cause a reaction.

Here are the rules for deciding if sand can move:

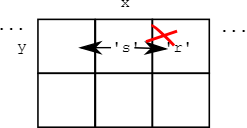
### Out of Bounds Rule

If the destination is out of bounds of the grid, the move is not ok.



### Empty Rule

If the destination square is not empty, the move is not ok.

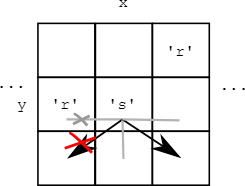


In the above picture, the left move is ok, but the right move is blocked by a rock and is not ok. Sand there would also block the move; the logic should detect a non-empty square.

### Corner Rule

Of the five possible moves, this rule only applies to the diagonal down-left and down-right moves. For a down-left or down-right move, if the "corner" square above the destination square is not empty, the move is not ok.

Consider the down-left and down-right moves of the 's' here, with the diagonal moves shown in black and the others in gray:



The down-left move of the sand is not ok because of the corner rule, but down-right is ok. A sand in the place of the rock would also block the down-left move. The straight-left move is blocked since the square is not empty. The straight right move is ok. The corner rule applies to diagonal down-left or down-right moves, but not the other three moves.

## Gravity

In order to keep your world alive and interactive, we will introduce gravity. You will want to plan how often to update the 2D canvas on screen as well as how often to update gravity. If you apply it too fast, the world will collapse on itself and if too slow, then it will not feel interactive. Part of your project is to make a world that is fun to play so play around with this by setting a timer callback (usually included in every UX framework you can set a timer to call you back so look for that API).

For a given cell in the grid, gravity behaves as follows:

1. If there is not a sand 's' at (x,y), do nothing, the move is over.
2. down: if the sand can move down, do it, this ends the move.
3. down-left: otherwise if the sand can move down left, do it, this ends the move.
4. down-right: otherwise if the sand can move down right, do it, this ends the move.

You will then iterate over the entire grid every “frame” and apply gravity to each cell. You should think hard about the order you apply the effect to make sure it behaviors the way you would expect it to. You also may need to apply different effects in different orders depending on how the effects interact from cell to cell.

## Brownian

Now for the last little bit of algorithm. [Brownian motion](https://en.wikipedia.org/wiki/Brownian_motion) is a real physical process, documented first by Robert Brown, who observed tiny pollen grains jiggling around on his microscope slide.

The "Brownian" value is a number in the range 0..100 inclusive (from your UX slider). When Brownian is 20, that means there is a 20% chance that each sand will randomly try to move one square left or right each turn.

### Here are the steps for applying Brownian to a single square.

1. Check if the square is sand. Proceed only if it is sand.
2. Create a random number in the range 0..99. Proceed only if number < Brownian. In this way, for example, if Brownian is 50, we'll do the Brownian move about 50% of the time.
3. Decide if the random move will be to the left or right. Perform a random "coin flip" by choosing a random number between 0..1.
4. If the coin is 0, try to move left. If the coin is 1, try to move right. The coin indicates the direction for a single attempted move, and that's it. In this way the Brownian motion is evenly balanced between left and right moves. If it cannot move, just ignore it and skip Brownian for that cell.

# Additional Effects

Each student in the group must create 1-2 additional effects. One effect from the team needs to be a growth effect and the other must include something other than gravity or Brownian as an environment impact. The reason is that this will force you to apply a different “update” pass for the new environmental impact and a new set of rules for the growth effect.

You need to document your effect in a similar fashion to the way I have documented the elements sand and rock – and how they interact. As well as how gravity works and Brownian functions in impacting the environment during each “iteration” of the world.

# Building a GUI with a game loop AND a UX event loop

You will need to implement both an event loop (usually built into UX frameworks) and a game loop (usually implemented by having a timer loop that updates the game engine and then makes an update to a 2D or 3D canvas – forcing a graphical refresh.

I do not suggest you use a Game Engine like Unity or Unreal to implement this as it is overkill for the simplicity of the experience. If you really want to take this approach, please talk with me ahead of time and get approval.