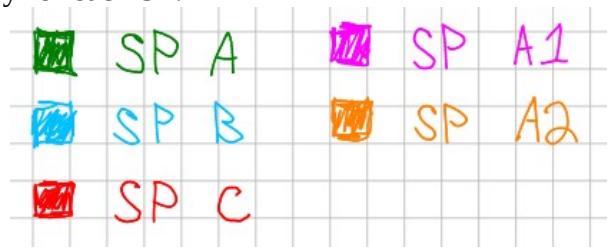
Christopher Jamieson
Project Proposal (Refined/Completed)
Due ASAP

Simulated Pixels Mechanic

- **The Concept/Theme**
  - Sometimes it's entertaining to watch a YouTube video of a swarm of ants create a colony in the ground at 10X speed. It makes you think, how can these things with their relatively simple brains create such complicated structures? Unfortunately, for those who are curious, this mechanic won't answer that. Instead, the simulated pixels mechanic takes this observation and gamifies it where instead of watching ants, pixels will be looked at, and instead of them creating colonies, they will be changing color. Now, that sounds less exciting. Pixels change color all of the time, what is different about these ones? Well, just like the ants, the pixels can have behaviors that can cause them to create their own structures. Now, that's much more interesting but how is this mechanic going to be implemented?
- **Implementing the Mechanic**
  - **Convenient Definitions:**
    - The ***dish*** is a grid where some of the cells in the grid have their own properties and rules which determine their behavior.
    - Every cell in the dish that has behavior is called a ***simulated pixel*** or ***SP***.
    - ***Properties*** are the attributes of each SP and are used by rules to determine whether their requirements are being met.
    - ***Rules*** are statements that say: "when all of these requirements are fulfilled, perform these actions." In other words, when certain properties in the SP meet a rule's requirements, then the SP should perform the actions defined in that rule.
    - To ***check*** a rule is to examine that rule's requirements and perform its defined actions if the requirements are being met.
    - A ***collision*** occurs when two or more SP's try to occupy the same cell.
    - The ***collided property*** is a special property every SP has that determines whether an SP collided with another SP and what SP it collided with.
    - A ***tick*** is an event that occurs when every rule from every SP in the dish is checked
  - **Fundamental Properties of the Mechanic:**
    - The game runs itself, therefore making it a **0 player** game
    - The rules and properties of each SP are defined before the game starts
    - Every SP must have a rule that deletes it.
      - This rule's requirements must be possible so that the SP can be deleted.
    - Every cell can have an unlimited number of SP's in it, but, there *must* be rules that ensure that there is only 1 SP per cell
    - The order that rules are checked is dependent on their creation *except* for rules that have the collision property as a requirement or an action that deletes the SP.
      - Rules with the collision property are checked before rules that have actions that delete the cell and they are then checked in the order of their creation
      - Rules that have an action that results in the SP being deleted are performed last and the ordering of these rules does not matter
    - The simulation updates every tick
      - The 0^{th} tick is the start of the simulation (No rules have been checked at this point)

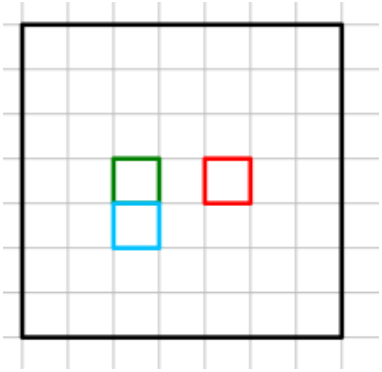
- The frequency of each tick is at a rate that is visually pleasing for the current simulation

- **A Simulated Example**
  - Let's start with an example game world: A 7 X 7 dish with 3 SP's labeled A, B, and C.
    - The frequency of each tick is 1 tick per 10 seconds so that we can see the changes (Obviously, this is a PDF so the changes can't be seen in a video format but imagine that each image below appears 10 seconds after the last)
    - Every SP has these 3 properties: health, energy, and damage where each property is represented as a number. (Of course they all also have the collided property but that is assumed to exist in every SP)
    - Every SP has these 3 rules: if the SP's energy is greater than 0, then lower their energy by 1 and move the SP to a random adjacent cell; if the SP has collided with another SP, lower the other SP's health by the amount of damage that the SP has; and lastly, if the SP's health is less than or equal to 0, delete the SP
      - The first and second rule work together to prevent SPs from occupying the same cell
      - The third rule complements the first and second by adding extra measures to prevent SPs from taking up the same cell while ensuring that every SP can be deleted
    - SP A has a rule where if the SP has 4 or more energy, then remove 4 energy from the SP and create another SP adjacent to it with: 5 health, 1 energy, and 1 damage with the same rules it has
    - SP C has a rule where if a SP collides with it, lower the other SP's health by double the amount of damage SP C has, and increase health and energy by 10 if the other SP's health is equal to or below 0
    - **The Simulation**
      - SP A is placed at point 3 X 4 and has health = 3, energy = 10, and damage = 1
      - SP B is placed at point 3 X 5 and has health = 20, energy = 5, and damage = 2,
      - SP C is placed at point 5 X 4 and has health = 5, energy = 5, and damage = 4,
        - A separate chart has been created showing each SP's properties during each tick (It's placed at the bottom of the PDF)
        - Here's the visual key for each SP:



When the game starts, the player will be shown a dish with SPs placed at their initial positions

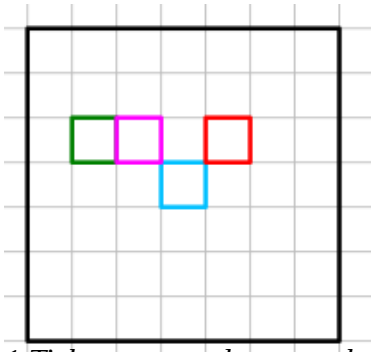
Tick 0



SP A, SP B, & SP C are placed on the dish

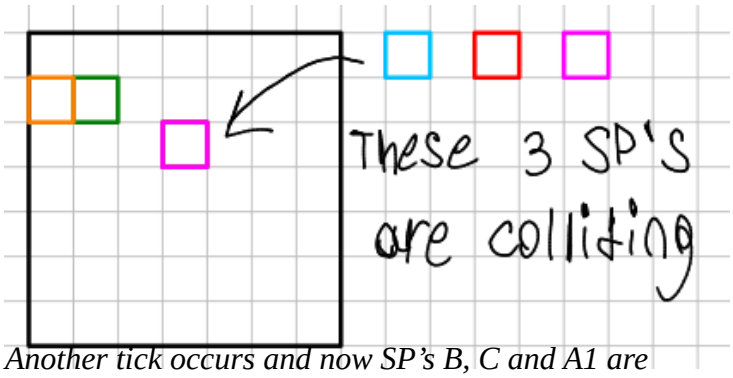
Then, as the game progresses, more ticks occur

Tick 1



1 Tick occurs and every rule in each SP is checked. SP A created a new SP (SP A1).

Tick 2



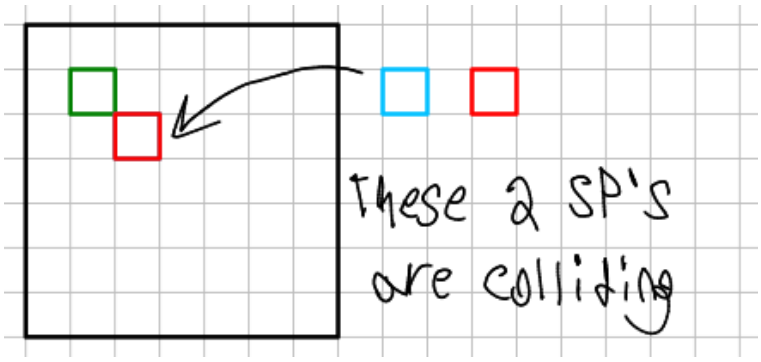
Another tick occurs and now SP's B, C and A1 are colliding with one another

Now that these 3 SPs are colliding, their collision property is active and therefore any rules checking for that property to be active can run

Tick 3

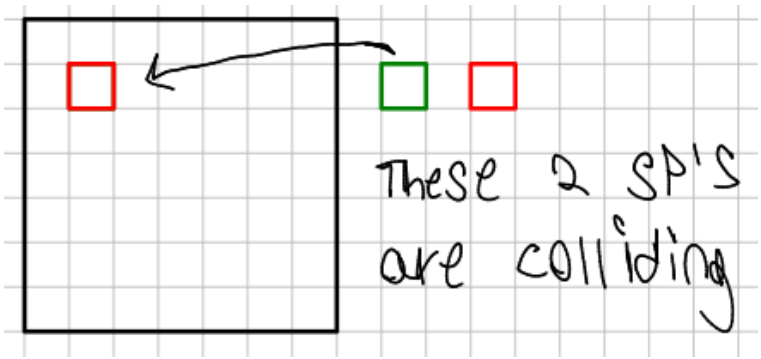
After the rules that handle the collision property are checked, 1 SP gets deleted and the rest move around

Tick 4



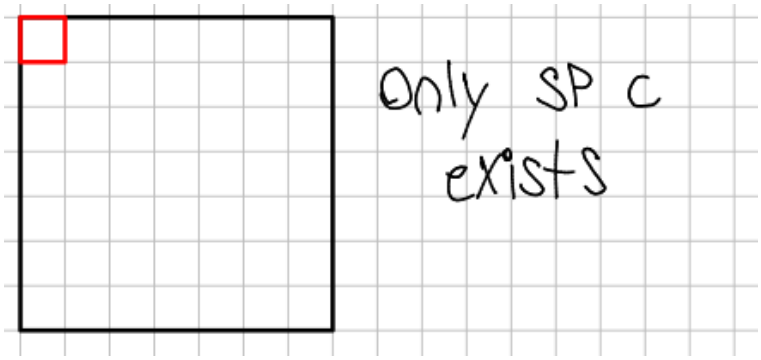
Now, SP's B & C are colliding

Tick 5



SP B is deleted after colliding with SP C and now SP C and A are colliding

Tick 6



After SP C & A collide, SP A gets deleted and only SP C remains

- **Target Audience**
  - My target audience would be the type of people who enjoy playing games that play themselves. However, they would be technical enough to define their own rules instead of relying on the ones predefined in the game. This would limit them to people familiar with programming as that might be how I allow user generated rules.
- **Visual Design**
  - The visual design of the game is going to be simple. SPs are going to be differentiated based on color so there would be an emphasis on finding colors that are easily differentiable from each

other. Also, for those who are colorblind, filters *can* be created so that they can still differentiate between colors. Furthermore, if someone is fully colorblind, then SPs can use grayscale or symbols can be drawn in them to help differentiate them.

- **Scope of the Demo**
  - **What to Create for the Mechanic**
    - The demo will include the base features of the mechanic like: the dish, rules and properties to create SPs, and something to handle the performing of rules in their proper order. This alone requires:
      - A scheduler to handle the ordering of rules
      - Code to construct SPs
        - Code to create rules
        - Code to create properties
      - Code that ensures that rules are checked at a predefined frequency
      - Code that notifies SPs of collisions
      - Code to check that there is a rule in each SP that has an action that deletes it
      - Code to warn the developer when more than one SP occupies a cell and by how long
      - APIs to provide rules the ability to move SPs to different cells
      - Code to check rules when their requirements are met
      - Code to render each tick in the game
      - Code allowing rules to access the properties of SPs
      - SPs need to be able to store a wide array of properties
    - (This idea came from AI but was expanded upon by me) Provide a visual way of showing the properties of a SP while playing the game
      - A GUI needs to created to show the properties of each SP
    - (Potential future idea) As the simulation gets more computationally intense, optimize the processing of rules
      - This can be done by moving some of the workloads to other threads so that multiple rules can be checked at once
    - (Potential future idea) Provide a way for users to upload their rules and properties to a database so that other users can use those rules/properties
      - Entire dishes can probably be encoded and sent to the database as well
  - **How They Connect**
    - The dish will likely contain the APIs necessary to allow rules to move SPs
    - Properties will be used by rules to determine if they can perform actions
    - The scheduler will work directly with the game engine to determine which rules to perform and in what order
    - The dish can activate the special collision property in SPs when they collide
    - The dish will provide APIs to rules allowing them to move SPs
    - APIs would be created allowing rules to read the properties of other SPs

- **The chart**

	Tick 0			Tick 1			Tick 2			Tick 3			Tick 4			Tick 5			Tick 6		
	Health	Energy	Damage	Health	Energy	Damage	Health	Energy	Damage	Health	Energy	Damage	Health	Energy	Damage	Health	Energy	Damage	Health	Energy	Damage
SP A	3	10	1	3	5	1	3	0	1	3	0	1	3	0	1	-7	0	1			
SP B	20	5	2	20	4	2	7	3	2	7	2	2	-3	1	2						
SP C	5	5	4	5	4	4	12	13	4	12	12	4	20	21	4	29	30	4	29	29	4
SP A1				5	1	1	-12	0	1												
SP A2							5	1	1	5	0	1									