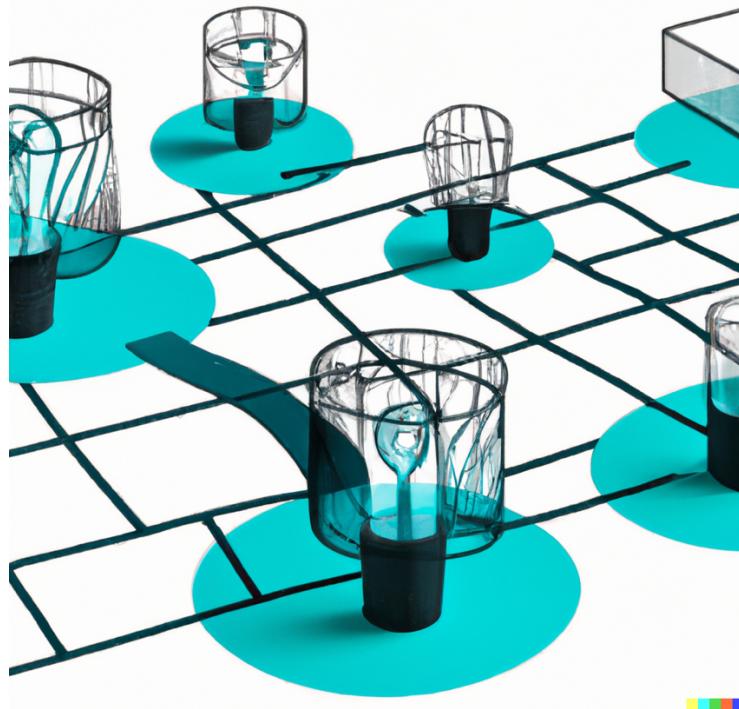


# Predictions V2 – By Nitzan

---

## Intro

Welcome to Predictions, an advanced simulation engine designed for researchers who want to model and simulate complex scenarios. This platform provides a powerful and customizable framework for creating diverse simulations by simply uploading an XML file that describes the world. Whether you're simulating ecological systems, social dynamics, or other phenomena, Predictions is the perfect tool to turn your research concepts into dynamic simulations.



## About The Project:

Student Name: Nitzan Ainemer (ניצן אינמאר) 313289472

Email: [Nitzan63@Gmail.com](mailto:Nitzan63@Gmail.com)

Bonuses Implemented: Simulate Single Tick (5), Animations (2) and Themes (1).

# Design Overview:

There are three modules implemented in this project:

1. Engine Module – the “heart” of the project. The Engine runs all the logic of the program.  
From handling file and validating data to running the simulation.
2. GUI – the graphic user interface of the program. It manages the interaction with the user and displaying the simulation results.
3. DTO – data transfer objects module, a module designed to be the “buffer” between the engine and the UI. Any data that the engine needs to provide for the user, it does that using the DTO, and the other way around.

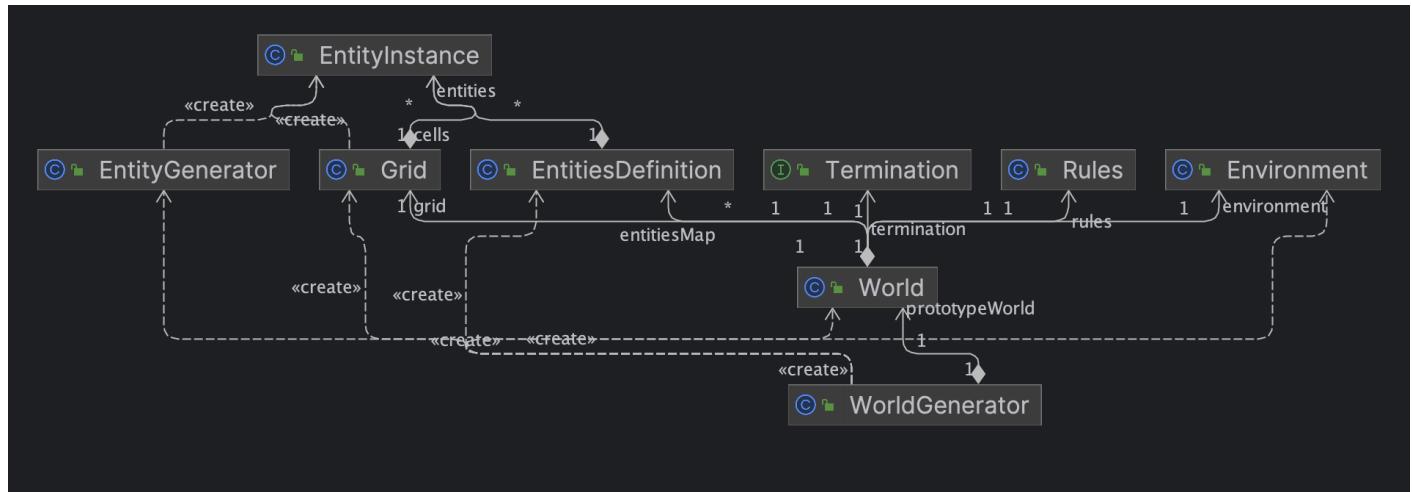
## Engine Module:

The Engine Has many roles and functions. In this section I will present the main components of the Engine module.

1. **File Handling:**
  - a. The engine gets a path to a file and uses JAXB to unmarshall it to JAXB classes.
  - b. After the unmarshalling, the engine runs a “static” validation system that validates the data loaded to the JAXB classes. If it fails – it throws Exceptions from the engine to the UI, using a class in the DTO called ErrorDTO.  
If the “static” validation succeeded, we move on.
  - c. Mapping – the Engine maps the JAXB classes to the domain classes (That will be presented in this file).
  - d. After mapping validation – the engine runs another validation, that requires the domain objects to be “alive”. For example – this is where the “byExpression” validation happens.
  - e. The world created by the XML processor is a prototype world, which will be the base for the world instances that the simulation will actually simulate.

## 2. WORLD:

- The engine holds and manages all the domain classes of the “world”. Here Is a class diagram to represent the classes and relations to “World”:

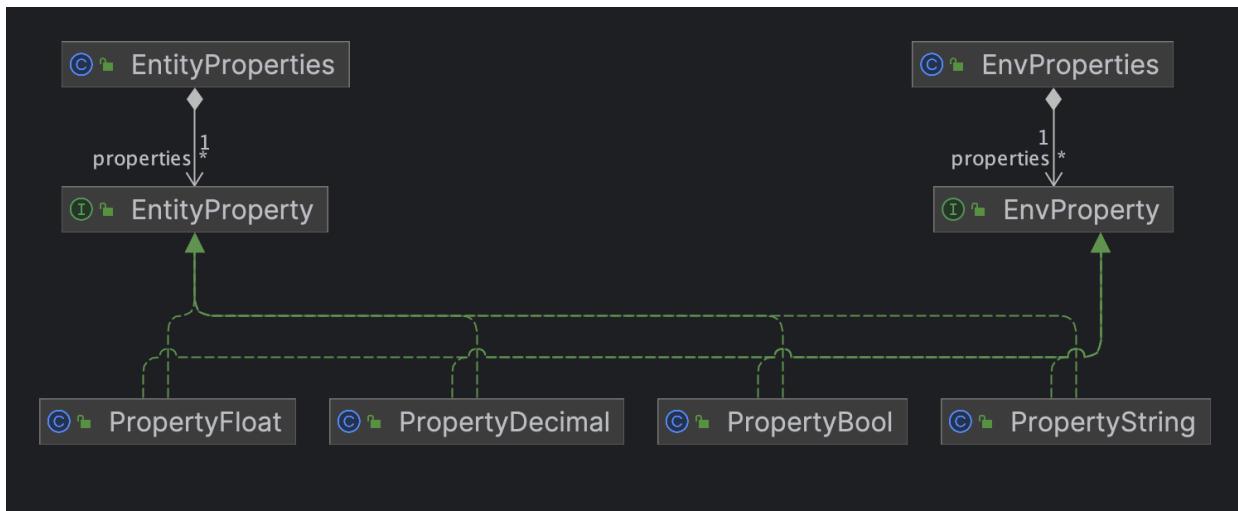


## 3. World Generator

- The world generator uses the prototype world to generate a world instance, for each simulation run.
- There is a “helper” Entity Generator that is dedicated to generating the entity instances.

#### 4. Entities, Environment and properties:

- a. In my implementation, I have created a class called EntitiesDefinition, which holds the name of the entity, the population and a Map that holds all the EntityInstances.
- b. The Environment class holds a list of EnvProperties.
- c. For the properties – I've created different interfaces for entity properties and environment properties. For the implementation I've used the same classes for them:

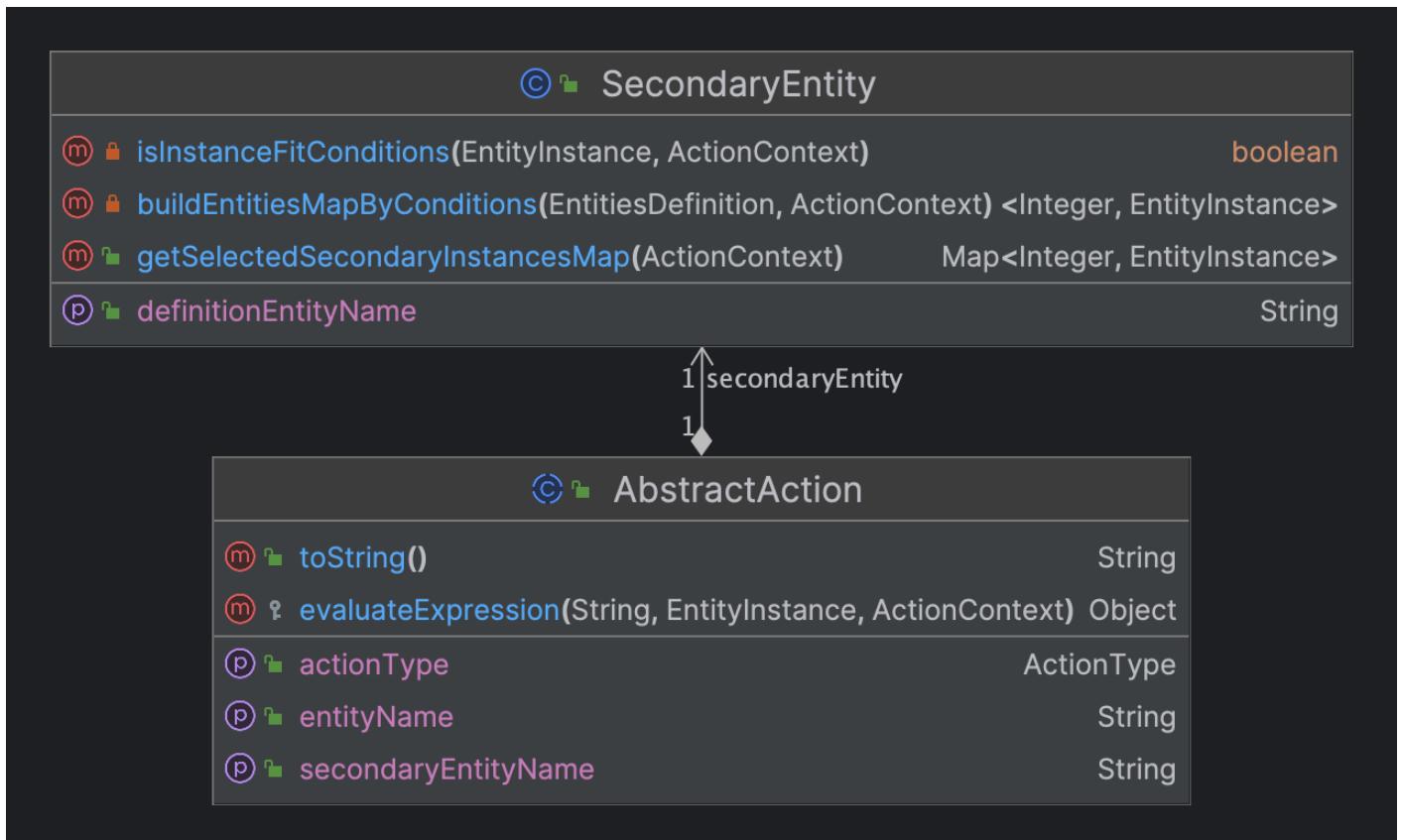


## 5. Rules and actions:

- Rules and Actions where one of the most challenging implementation in the domain class and logics.
- I managed to use interfaces and abstract classes to simplify it and make it work.
- Here is a diagram that represents the relations between them:



- Also added secondary entity class, which holds the logic of how to select the number of secondary entity instances, and then delivers them to the actual actions to perform:

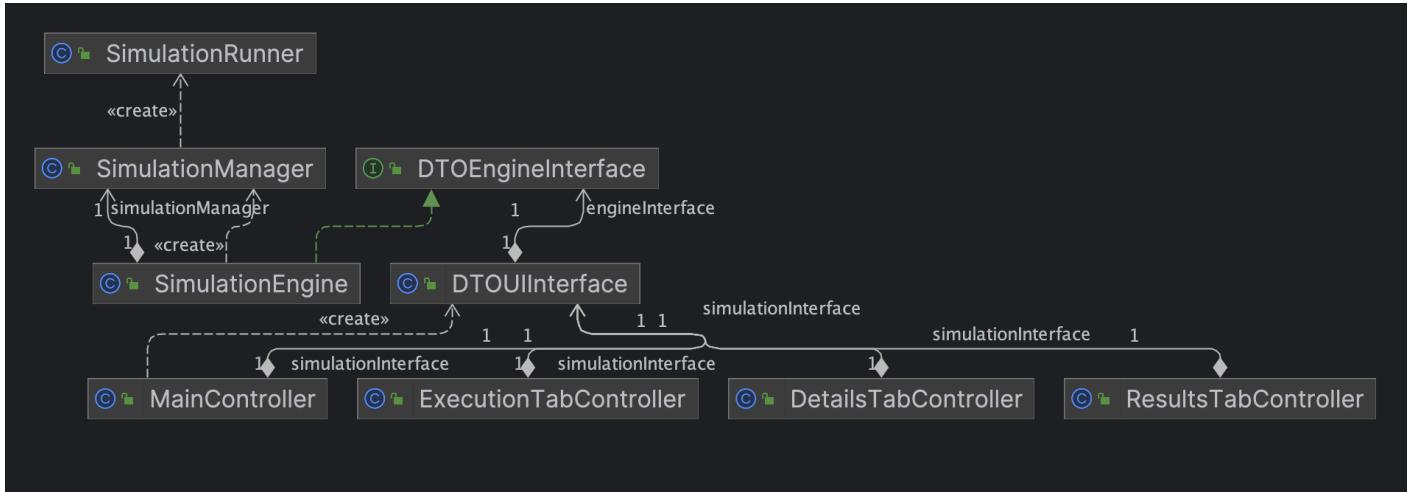


## **6. Threads:**

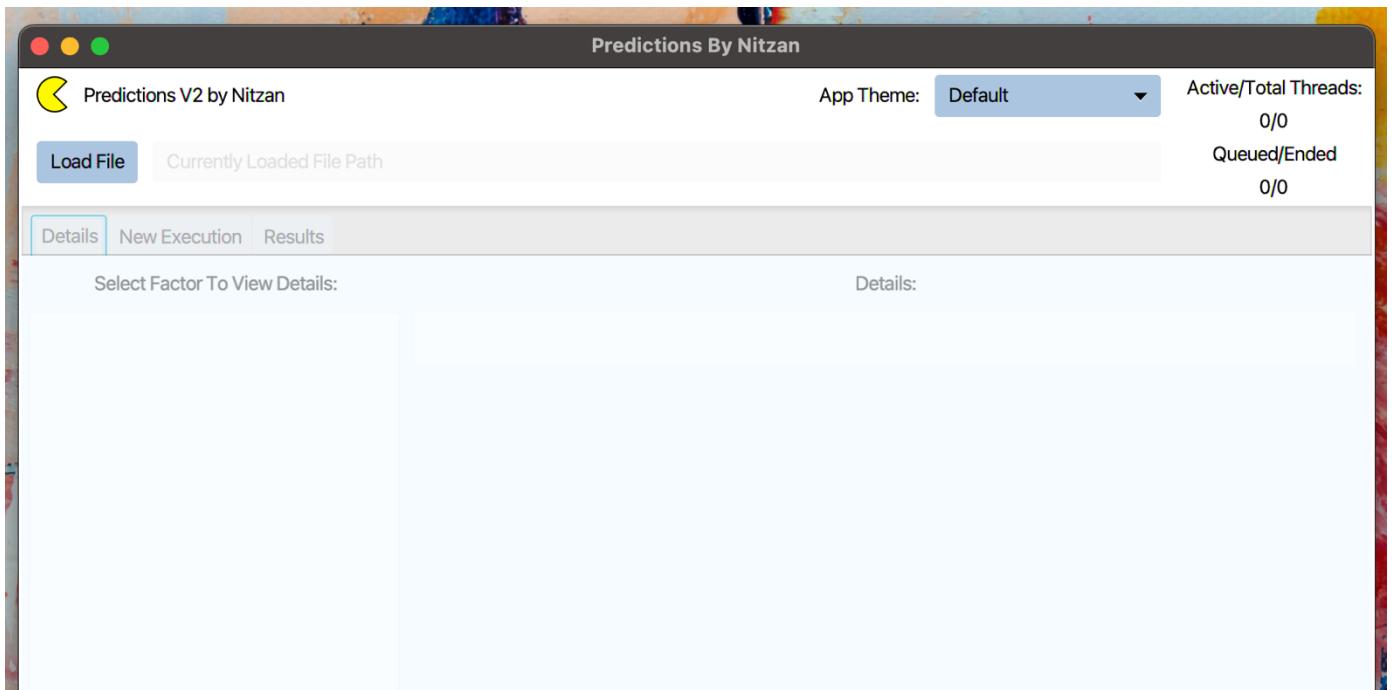
- a. The program let's the user to run many simulation simultaneously.
- b. It's done by using multiple threads – managed in a thread pool.
- c. The number of threads is set by the xml file.
- d. When a user starts a new simulation, the simulation can be in a 4 different states:
  - i. QUEUED
  - ii. LIVE
  - iii. PAUSED
  - iv. COMPLETED
- e. At first, the simulation is queued. When there is an available thread in the thread pool, it takes the simulation and runs it.
- f. The user can pause and resume the simulation as he wants, using wait() and notifyAll() logic.
- g. The simulation can be completed in 2 different ways – stopped by the user or reached termination condition (if exists).
- h. When a simulation is done, the thread is released to take a new simulation that is queued.

# Simulation Run

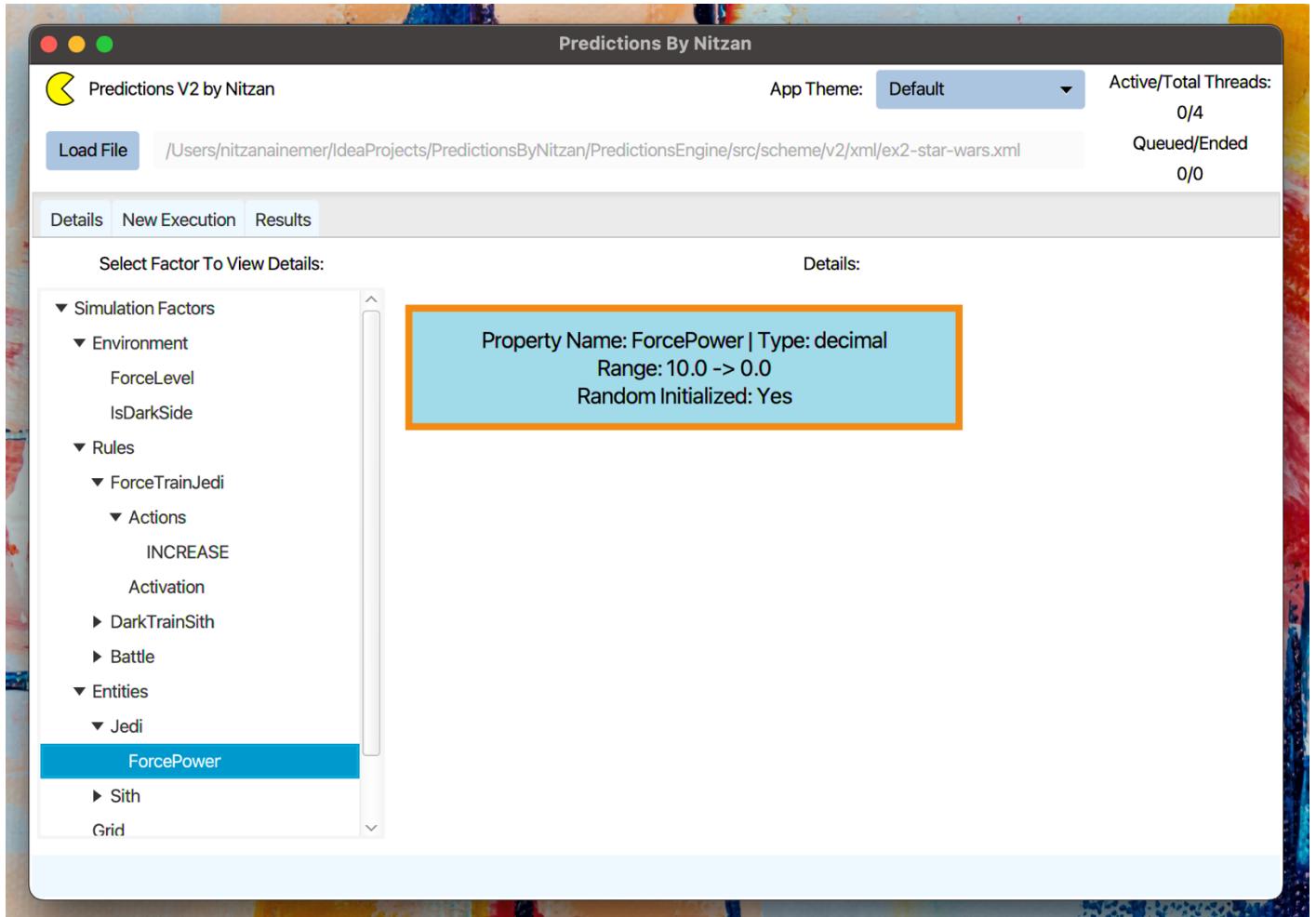
In the design of the simulation run, I designed a separated classes that interfaces between each other in a way that keeps the engine separated from the UI, and the opposite. Here is a diagram that shows the layout of the classes from the GUI to the Simulation Runner.



1. The **SimulationGUI** class (Not in this scheme) in the GUI module, has the **Main** function that starts the program running. It sets the stage and the scene:



- Once the user loaded a valid xml file, the simulation tabs are enabled and the user can see the simulation details:



3. When the user wants to run a simulation, he goes to the new execution tab. There he can set the population of each entities (default is 0) and environment properties (default is random):

The screenshot shows the 'Predictions By Nitzan' application window. At the top, there's a title bar with the application name and a file icon. Below it, a toolbar includes a 'Load File' button and a file path: '/Users/nitzanainemer/IdeaProjects/PredictionsByNitzan/PredictionsEngine/src/scheme/v2/xml/ex2-harry-potter.xml'. On the right of the toolbar are status indicators for 'App Theme: Default', 'Active/Total Threads: 0/1', and 'Queued/Ended 0/0'. A navigation bar at the bottom has tabs for 'Details', 'New Execution' (which is selected), and 'Results'. The main content area is divided into two sections: 'Set Entities Population' and 'Set Environment Properties Values'. The 'Set Entities Population' section contains a table with two rows: 'DarkCreature' with a value of '10' and 'Wizard' with a value of '100'. The 'Set Environment Properties Values' section contains a table with two rows: 'MagicalFieldLevel' with a type of 'decimal', range from '1.0' to '5.0', and a 'Set Value' of '2', and 'DarkPresence' with a type of 'decimal', range from '1.0' to '5.0'. At the bottom left is a 'Clear' button, and at the bottom right is a 'Start!' button.

Entity Name	Set Population
DarkCreature	10
Wizard	100

Name	Type	Range From	Range To	Set Value
MagicalFieldLevel	decimal	1.0	5.0	2
DarkPresence	decimal	1.0	5.0	

4. After pressing start, the simulation starts, and the user is automatically transferred to the results tab, there he can watch the final environment properties selected/generated and the live progress of the simulation and past runs:

The screenshot shows the 'Predictions By Nitzan' application window. At the top, it displays the title 'Predictions By Nitzan' and the file path '/Users/nitzanainemer/IdeaProjects/PredictionsByNitzan/PredictionsEngine/src/scheme/v2/xml/ex2-harry-potter.xml'. It also shows the app theme as 'Default' and the status 'Active/Total Threads: 1/1' and 'Queued/Ended 0/0'. Below the header, there are tabs for 'Details', 'New Execution', and 'Results'. The 'Results' tab is active, showing a simulation titled 'Simulation 12092023-1 Ongoing...'. The progress bar indicates 'Ticks Progress: 669531' and 'Time Elapsed: 4 Seconds'. The progress bar itself is at 40.0%. On the left, a sidebar lists 'Simulations' with one entry: '12092023-1'. The main results area has three tabs: 'Simulation Details', 'Entities Population Statistics', and 'Property Histogram'. The 'Entities Population Statistics' tab is selected, displaying a table with two rows:

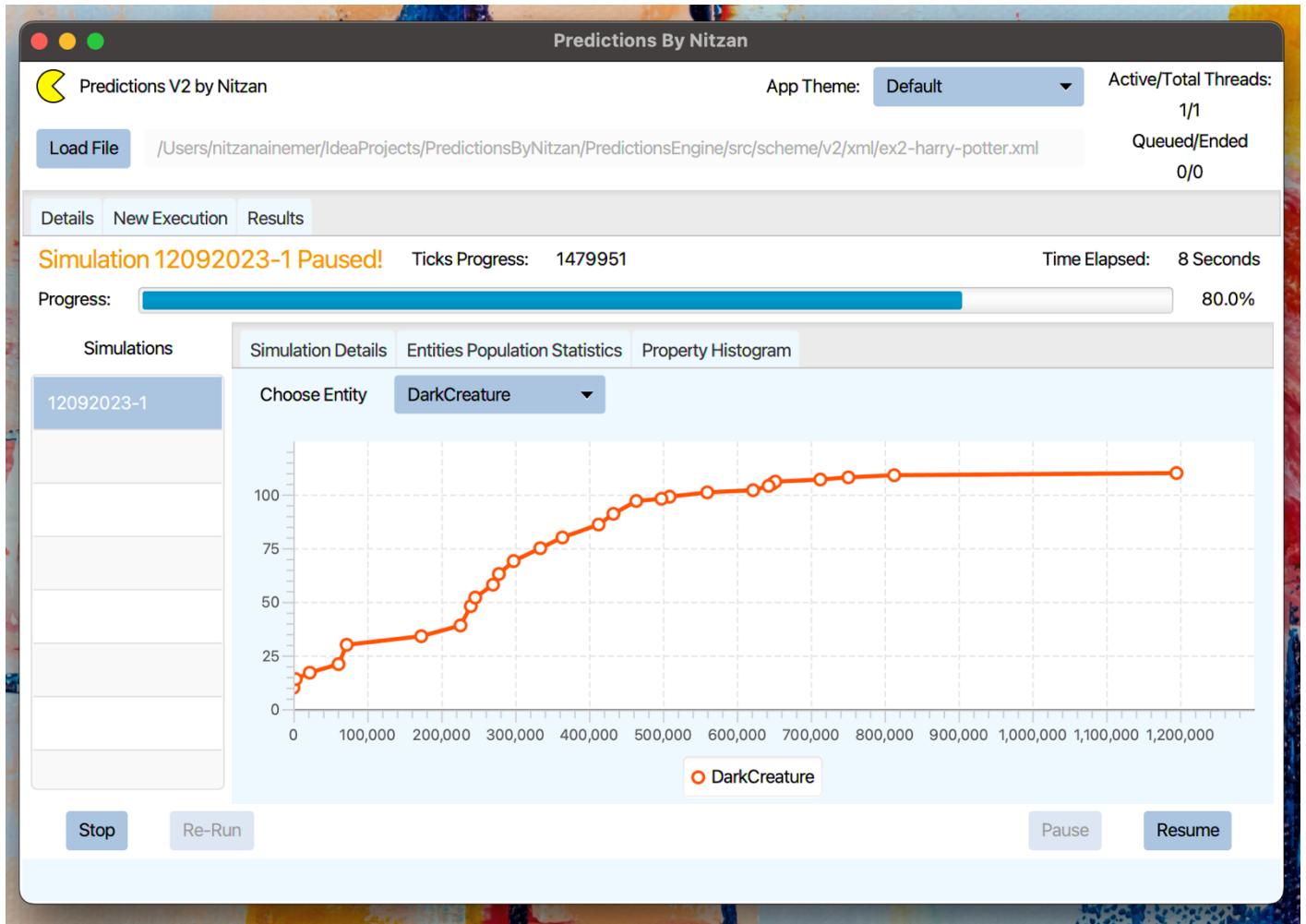
Entity	Population
DarkCreat...	106
Wizard	4

Next to this table is another table for 'Environment Properties':

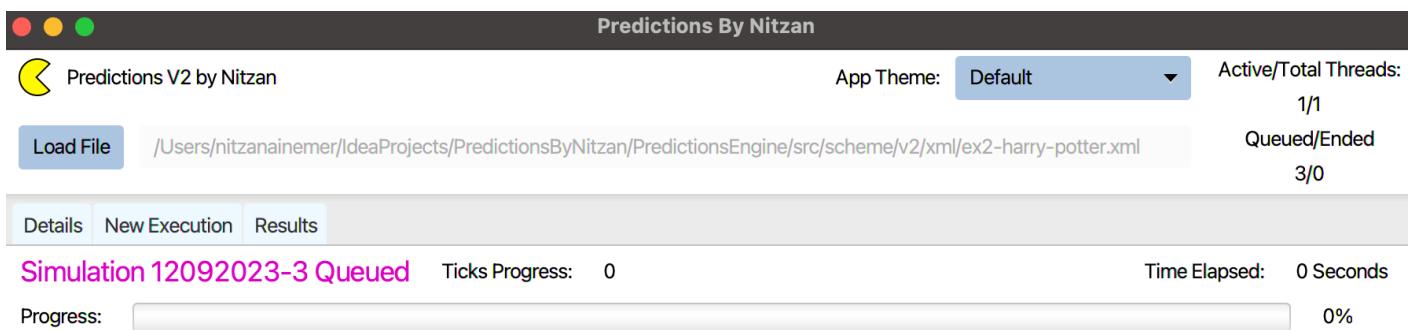
Property ...	Property ...
MagicalFie...	2
DarkPrese...	2

At the bottom of the results panel are buttons for 'Stop', 'Re-Run', 'Pause', and 'Resume'.

5. The user can pause the simulation at any time while it's "live", and watch the current status of the entities population and properties:



6. When trying to run more simulations than there are available threads, a queue is formed:



7. Eventually, when the simulation ends, the user can see its full details and results, while other simulations are live:

