**Documentation**

* Domain

The purpose of this mini-project is to establish a design studio that utilizes Petri-Nets. A Petri net is defined as a triple (P, T, F), where P denotes a finite set of locations, T denotes a finite set of transitions, and F denotes a finite set of arcs. The function that represents the state of the network by assigning a non-negative integer to each location. A transition's Inplace is a collection of locations where each element of the set is connected to the transition. The outplaces of a transition are a collection of locations connected to the transition by arcs, with the locations serving as destinations and the transition serving as the source. The number of tokens at each inplace of the transition is nonzero. When an enabled transition is fired, it reduces the number of tokens in all inplace locations by one and increases the number of tokens in all outplace locations by one.

* Typical Use-Cases

The Petri Net Design Studio is applicable to a wide variety of situations. One of them may be incorporated into the game's design. The modeling languages typically used in this context are inefficient at validating the underlying game systems. Petri Nets have a number of advantages over alternative modeling languages. Their graphical notation is straightforward, but it is capable of modeling complex game systems. Their mathematically defined structure enables formal analysis of the modeled system, and their behavior simulation enables early detection of undesirable behaviors, loopholes, or balancing issues during the game design stage..

* Install the design studio

The easiest way to start using this project is to fork it in git. Alternatively, you can create your empty repository, copy the content and just rename all instances of 'WDeStuP' to your liking. Assuming you fork, you can start-up following this few simple step:

1. install [Docker-Desktop](https://www.docker.com/products/docker-desktop)
2. clone the repository
3. edit the '.env' file so that the BASE\_DIR variable points to the main repository directory
4. docker-compose up -d
5. connect to your server at [http://localhost:8888](http://localhost:8888/)

Main docker commands

All of the following commands should be used from your main project directory (where this file also should be):

* To **rebuild** the complete solution docker-compose build (and follow with the docker-compose up -d to restart the server)
* To **debug** using the logs of the WebGME service docker-compose logs webgme
* To **stop** the server just use docker-compose stop
* To **enter** the WebGME container and use WebGME commands docker-compose exec webgme bash (you can exit by typing command 'exit')

Start modeling & feature work

The Petri Net Design Studio is applicable to a wide variety of application scenarios. One of them could be incorporated into the design of the game. When it comes to validating the underlying game systems, the modeling languages typically used in this are inefficient. Petri Nets can be used in a way that other modeling languages cannot. Their graphical notation is straightforward, but it allows for the modeling of complex game systems. Their mathematically defined structure enables formal analysis of the modeled system, and their behavior simulation enables early detection of undesirable behaviors, loopholes, or balancing issues.

There is the "check" guest example. I've included four examples in the right corner. I adhere to the instructions and provide two examples. The large graphic example is labeled "petri nets," while the small graph example is labeled "test." The term "deadlock" is used to indicate that the network has reached a stalemate. The "stateMachine" example is the final one in the simulator that is used to test the code. In the simulator, the network is visualized similarly to the composition. Additionally, it should differentiate the enabled transitions. Once the user clicks on an enabled transition, firing should occur. Markings should progress in accordance with the number of shots fired. The visualizer's toolbar should include a'reset' button that returns the network to its initial marking. The model should not reflect the state of the simulation. When the network reaches a deadlock with no enabled transition, the user should be notified via a visual effect or an actual notification. The interpreter that should be associated with a toolbar button can check the individual classifications using any implementation. While using formulas is not required, it probably simplifies the checking process.