**Adding a New Game to Balance or Disrupt**

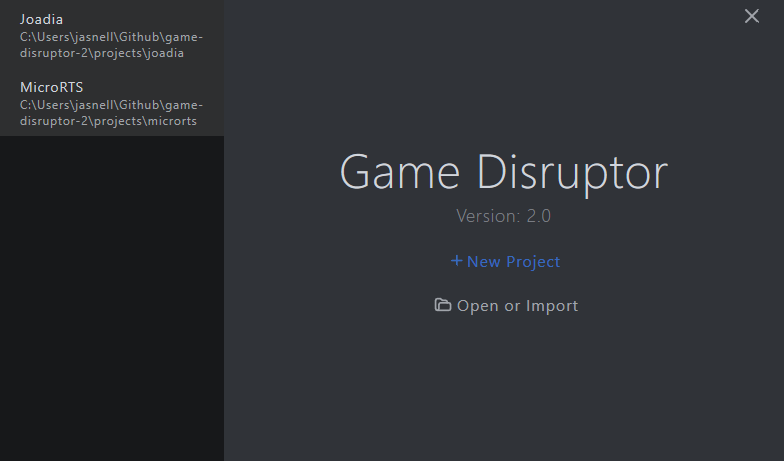
**Game Disruptor 2.0 Documentation**

# Overview

This document outlines the necessary steps to add a new game to balance for the game disruptor tool and the requirements that a game must met in order to be added. The game disruptor 2.0 tool allows a game or wargame designer to balance their game without requiring any code changes to the tool itself. Instead, minor code changes will be needed to made to the game or simulator that will be used, this is to allow communication between the tool and the game. This limits the number of games a designer can use with the game disruptor tool, to the number of games that they have access to the source code or can request changes to be made on their behalf.

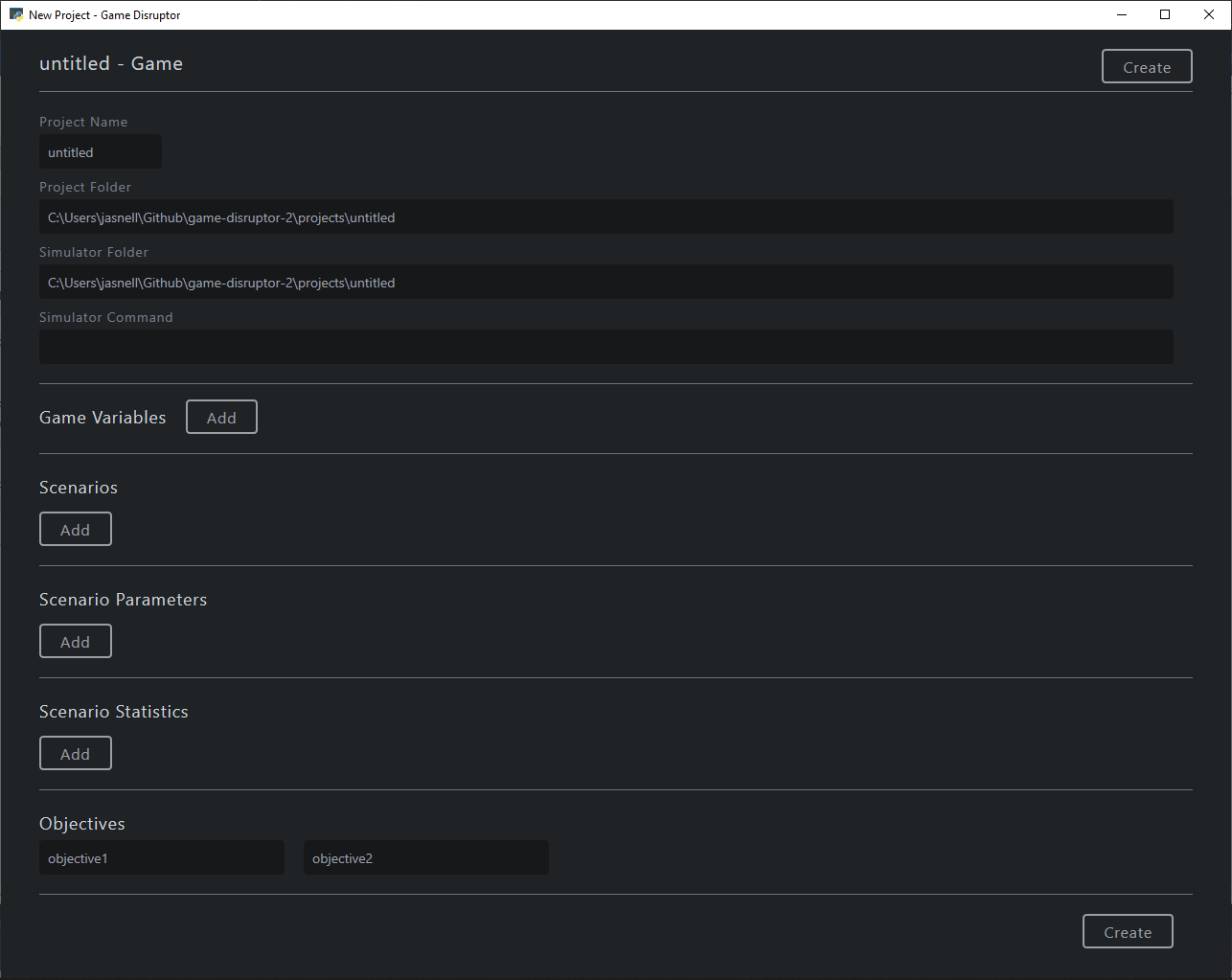
# New Project

The first step in being able to use a game or simulator with the game disruptor tool is to create a new project. This project holds all the game variable and evolution parameters as well as the command to run to evaluate a solution. The project file is stored in the json format which can be easily viewed in a text editor if required however all the values can be changed using the UI in the game disruptor tool. A new project can be created by clicking the “New Project” button on the welcome screen when the tool is first opened see Figure 1. The welcome screen can also be used to load previously created projects, shown in the left panel.



**Figure 1: The game disruptor welcome screen shown after loading.**

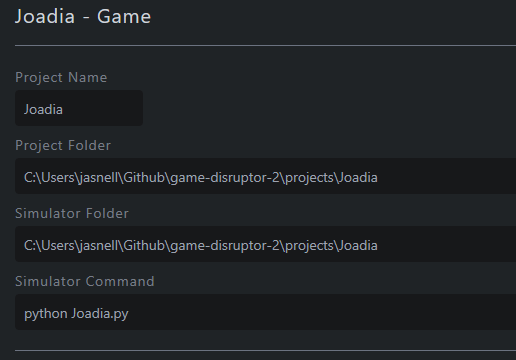
The new project screen is split into six sections, simulator information, game variables, scenarios, scenario parameters, scenario statistics and objectives. See Figure 2.



**Figure 2: A blank new project screen.**

## Simulator Information

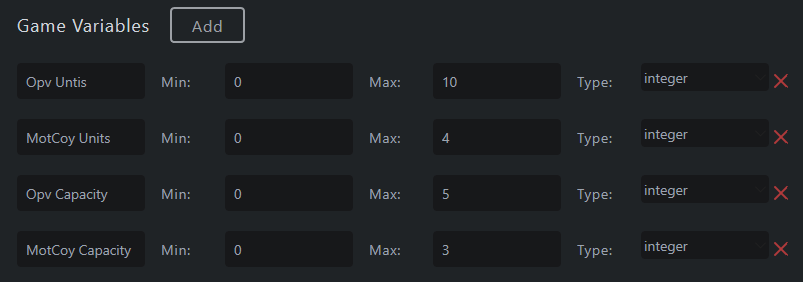
This section is where the path to the simulator or game is configured as well as the command to use to execute it. If the simulator is an executable, then the simulator command could be as simple as calling the .exe file such as “.\Joadia.exe”. The simulator command field can take any shell command, for a python script it could be “python Joadia.py” or for a java package “java -cp Joadia.jar Main”. This flexibility is what allows the game disruptor tool the ability to interface with any game, provided that the necessary software to run the game is installed. Along with the simulator information a project requires a name, and a directory to hold the project file. (See Figure 3)



**Figure 3: Example values for the simulator fields.**

## Game Variables

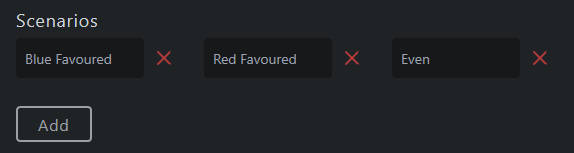
The new project screen allows a user to easily specify the game variables that the game disruptor tool should balance. A game variable can be added by clicking the add button located next the game variables section heading. Each game variable requires four fields to be filled out, the name of the variable, the minimum value, the maximum value and whether the variable is an integer or a real number, See Figure 4. The minimum and maximum value are used to specify the search space of each game variable for the game. The game variables can be removed by clicking the red X button and like any of the project settings, editable later after creating the project. The names assigned to the game variables can be any value however they must be unique, the names are used when the values are later sent to the simulator for evaluation.



**Figure 4: Example values for game variables settings for a game.**

## Scenarios

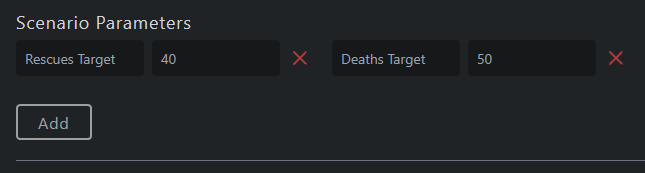
The scenarios sections can be used to specify the names of different levels or maps a game or simulator may include. This section is optional and only applicable to games with multiple scenarios that share the same game variables. For an example see Figure 5, a scenario could be designed to favour a particular team or be completely balanced.



**Figure 5: Example values for the scenario fields.**

## Scenario Parameters

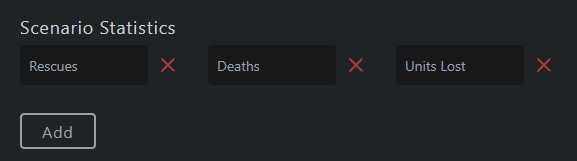
The scenario parameters can be used to control aspects of the scenario such as the amount of uncertainty or be used to specify targets that the game disruptor tools optimized solutions aim to achieve. (See Figure 6)



**Figure 6: Example scenario parameters that are sent to the simulator.**

## Scenario Statistics

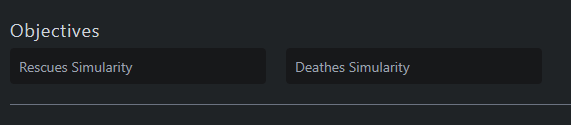
This section allows a user to outline the statistics the simulator or game should record and send back to the tool after evaluation. Example statistics could include the number of wins, losses, the average hit points remaining of the units after the battle, see Figure 7. Some statistics it might make sense to represent as a distribution instead of a single value, therefore any statistic may also have a corresponding distribution calculated by the simulator or game during evaluation.



**Figure 7: Example statistics that a game may have.**

## Objectives

The objectives are used by the multi-objective evolution process to access the performance of different solutions for a games balance. The new project settings screen only requires specifying the names to assign the objectives, the calculation for each objective is defined by the code changes to the game or simulator outlined in the next section. For example, the objective for the evolution might be to match the targets defined in the scenario parameters as closely as possible, see



**Figure 8: Example names for the objectives.**

Once all the necessary project information has been filled in, the new project can be created and saved by clicking the create button located at the top and bottom of the new project screen. Once created the project is ready to be used with the game, however the game must first be modified to communicate with the tool which is detailed in the next section.

# Simulator or Game Code Changes

The game disrupter tool can be made to interface with any game provided that the game or simulator can be slightly modified to communicate with the tool. The new project user interface has a simulator command field that is run every time a new solution of game variables is to be evaluated. For the game disruptor tool to communicate the solution that needs evaluation, it sends the game variables and other scenario parameters to the standard input of the process in the JSON format. This then requires that the game be modified to read in the JSON from the standard input and update the values for the game variables to match those provided in the JSON input. The simulator can the use the game variables and evaluate it’s win-rate if applicable by running many simulations. Finally, the evaluation results for the specific game variables and communicated back to the game disruptor tool by writing the results to the standard output in the JSON format. Communicating using inter-process communication in the JSON format was chosen to make it simple for as many games to be easily modified as possible regardless of the programming language used for development.

## The JSON Input Structure

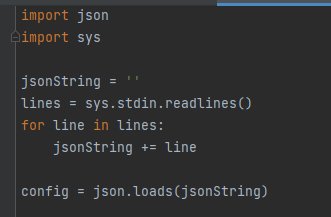
Every time the simulator command is called by the game disruptor tool, it sends a JSON string containing all the necessary information to evaluate the solution. The core structure of the JSON consists of the game variables, scenario parameters, evolutionary parameters, the names of the objectives and the names of the statistics, see Figure 9. The names for each of the parameters match those defined when filling out the new project screen user interface. The structure of the JSON closely matches that of the projects JSON file located in the game disruptor project folder created when started a new project.

{  
 "evolution\_parameters": {  
 "Seed": "",  
 "Generations": 10,  
 "Population": 8,  
 "Mutation-Rate": 0.15,  
 "Crossover-Rate": 0.3,  
 "Simulation Count": 100,  
 "Injection Count": 1,  
 "Injection Interval": 5  
 },   
 "game\_variables": [  
 {  
 "Blue Health": 20,  
 "Blue Damage": 10,  
 "Blue Attack Range": 2,  
 "Blue Movement Time": 8,  
 "Blue Attack Time": 6,  
 "Blue Health 2": 20,  
 "Blue Damage 2": 10,  
 "Blue Attack Range 2": 2,  
 "Blue Movement Time 2": 8,  
 "Blue Attack Time 2": 6  
 }  
 ],  
 "scenarios": [  
 "even",  
 "blue\_advantage",  
 "red\_advantage",  
 "default"  
 ],  
 "scenario\_parameters": {  
 "Target Win-rate": 0.0,  
 "Damage Uncertainty": 100.0,  
 "Movement Time Uncertainty": 100.0,  
 "Attack Time Uncertainty": 100.0  
 },  
 "scenario\_statistics": [  
 "Blue Wins",  
 "Red Wins",  
 "Draws",  
 "Blue Win-rate",  
 "Average Hit Points Remaining"  
 ],  
 "objectives": [  
 "Win-Rate Similarity",  
 "Changed Similarity"  
 ]  
}

**Figure 9: Example structure of the JSON string sent to the simulator.**

## Receiving Game Variables

For a game to receive the game variables that need to be evaluated, it must be modified to read the JSON string from the standard input stream. Some languages such as python can achieve this with built-in methods without needing to install any extra libraries or packages, as shown in Figure 9. Other languages such C#, Java or C++ may require a JSON parser library to make reading the game variables from the JSON input simple.



**Figure 10: Example of reading the json config from the standard input in python.**

## Loading Game Variables

Once the JSON input has been read from the input stream, the game variables can be loaded ready for evaluation. For languages that have a JSON parser as a library or built-in method this can be very simple see Figure 10 and Figure 11. If the simulator or game is using a language that doesn’t have a JSON parser the games designer may be required to manually parse the JSON which could be achieved using basic string operators such as split and substring.



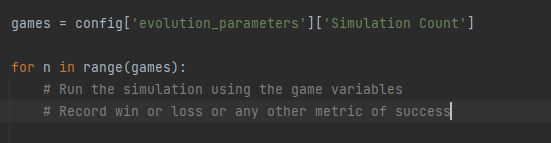
**Figure 11: Example of loading a game variable from json config in python.**



**Figure 12: Example of loading a game variable from the json config in java.**

## Running The Simulation

After the game variables have been updated to the correct values, the next step is to create a core loop to perform many simulations. When games or wargames have uncertainty or non-deterministic behaviour many simulations may be required to calculate an accurate win-rate or another reliable measure of success. The JSON configuration provided by the game disruptor tool includes the “Simulation Count” which is the number of times the game or simulator must be evaluated per a combination of game variables. Calculating the win-rate for example could be achieved by using a simple loop and tracking whether a win or a loss occurred after each simulation. (Figure 12)

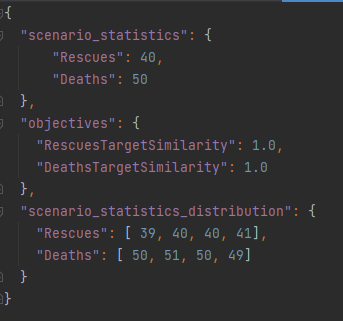


**Figure 13: Example of an evaluation loop in python that can be used to evaluate a set of game variables.**

If possible, a game should be evaluated without rendering to the screen in a headless mode which can drastically increase the number of simulations that can be achieved per second. Additionally, when the user clicks the “Show Simulation” button in the game disruptor tool, the simulator command is called with “display” passed as a command parameter. If the simulator command is “python Joadia.py” then clicking the “Show Simulation” button will cause the simulator command to temporarily become “python Joadia.py display”. A games designer has the option to check for this parameter run a single simulation using the game variables while rendering the output to the screen. This allows a user to quickly analyse the effectiveness of a given set of game variables while also only showing the result visually when required.

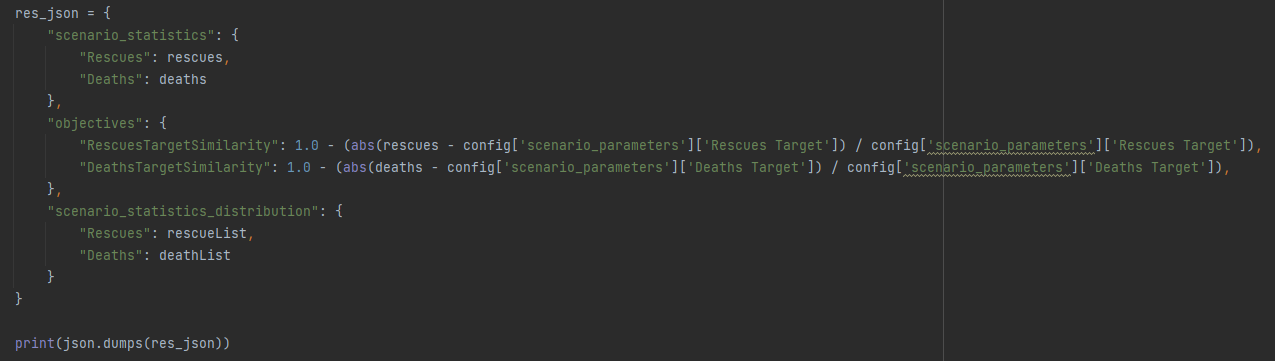
## Reporting The Results

The final step after evaluating a solution, is to send the results back to the game disruptor tool. The results must be formatted using JSON with the names for the scenario statistics, objectives and distributions matching those defined in the game disruptor project configuration, see Figure 13.



**Figure 14: Expected structure of json response sent to the game disruptor tool.**

Creating the JSON response, like the parsing, can be performed using an external library a built-in package or by hand. The scenario statistics field contains the values for the statistics defined when creating the new project, such as wins, losses, win-rate for example. The objectives contain the values for the two objectives that the game disruptor tool optimizes for. The values for the objectives should be designed so the higher the value the better the solution as the evolutionary process will try to maximum both values. Figure 15 shows an example of how the objectives calculation could be defined and how to send the final JSON string back to the game disruptor tool by writing to the standard output stream.



**Figure 15: Example of defining the json response after evaluating a solution in python.**

# Summary

This document outlines the necessary steps and requirements to use a new game, wargame or simulator with the game disruptor tool. Each of the decisions made were chosen to make it a simple as possible for a game designer to add support for their game without having to modify code for the tool. Even though changes are required to the game itself, designers would often be much more family with the code base of the game they are balancing than the tool the use to help balance it and so this was decided as the necessary trade off. Although a number of screenshots included examples in either Python or Java, any programming language that support inter-process communication is supported by this approach. The scope of this document is limited to the steps necessary to add a new game to use with the game disruptor tool (future document title here) outlines how to balance a game once the steps in this document have been completed.