COMPUTER SCIENCE

Capstone Report - Spring 2020

Parallelization of Monopoly®Game Simulations

Yijie Zou,
Yixiang Xiao,
Xiyan Cai

supervised by
Almaz Zelleke & Olivier Gilles Marin

Abstract

The Monopoly game can be a tool for economists to study real-world economic equality. In this project, researchers need a program to simulate numerous Monopoly games with changeable parameters; and the interface should be aimed at people with no computer science background. This project builds a Monopoly game simulation system which can create parallel simulation jobs on HPC(High-Performance Computing) and a user interface to connect to the system remotely. Users can acquire statistics about a large number of Monopoly game simulations on HPC by filling a simple form.

1 Introduction

Players in the Monopoly game try every means "to become the wealthiest player through buying, renting, and selling property" [1]. It seems that monopoly is encouraged; however, the game was originally designed to be an anti-monopolistic "Landlord's Game" by an acolyte of progressive economist Henry George [2].

In the past, the Monopoly game was utilized to "create a simulation of business and economic realities" [3]. There is a substantial connection between the rules of the game and economic practices in the real world. This project uses a simulation of the board game Monopoly to explore economic security in reality. In the game, only one at the end winner who possesses all the resources. However, an ideal society is a place where everyone has a balanced proportion of wealth to survive. This project will modify different parameters of the game (toll, land value, .etc) and add new features (subsidy, tax, .etc) to explore the impact of economic institutions on social equality. By simulation, the project acts as a sandbox for economists and policymakers to examine factors that affect financial security, and finally, explore a rule to avoid economic inequality. Therefore, the ultimate goal is to make the game last as long as possible, so that it simulates a balanced social wealth system.

To explore game duration under different sets of parameters, our project adopts parameter sweep on clusters. Researchers used Computational Grid to perform parameter sweeps in other fields, such as geoscience. They show the feasibility of performing parameter sweeps on clusters. The similarity between previous researches and this project is that in both scenarios, multiple parameters are examined, and the goal is finding the best matches. Methodology and algorithm play an essential part in the simulation process [4] [5].

The project is divided into two stages.

During the first stage, the main goal is to create a game and enable AI players. Users can change different parameters of the game and refer to the simulation log for details about player activities.

The second stage consists of user interface designing and the assessment of the game output. We created a user-friendly interface for people not familiar with computer science to conveniently create simulation jobs. After submitting the job via the interface, the simulation runs in parallel on HPC clusters at NYU Shanghai and generates statistical assessment and graphs from the simulation output.

2 Related Work

There are three aspects to explore in our project. First, we need to understand the current usage and the feasibility of the monopoly application in the real world. Second, we need to explore player strategies to create an intelligent application. Last but not the least, we need to find a user-friendly way to do parameter sweep on computational clusters.

2.1 Current Usage Of Monopoly Game Simulation

The simulation of the game has been widely used for academic purpose, especially in class. In one Introductory Financial Accounting Class taught by Stephen B. Shanklin, University of Southern Indiana, the simulation of Monopoly game has been used because "it is an effective way to illustrate the accounting cycle and to present an engaging approach to the financial accounting articulation method of income calculation."[6] In another class taught by An Ansoms at UCLouvain related to poverty and inequalities, the rules of the original Monopoly game are even modified to better reflect social stratification and inequalities in the context of developing countries[7].

In all these cases above, the players in the game focuses more on how to win the game under different rules so that the students can better understand how reality makes a difference in people's life. However, our project gives attention to the other direction: how to keep the game running as long as possible. According to one previous research from Cornell University, the probability for a Monopoly game to go on forever is around 12 percent. However, this result is limited to two-player Monopoly game with original rules and no trades[8]. Based on this research, our project will go deeper and look at how different rules and different numbers of players will affect the duration of the game.

2.2 Winning strategy exploration

There are relatively older but effective strategies drawn from thousands of simulations. Hentzel suggests calculating the probability of landing on each property and expected per dollar return on the invested amount by computer programs. Although there is no way to guarantee to win the game, players can use the calculation results as a reference to make decisions [9].

Some researchers have tried reinforcement learning methods to address the Monopoly game simulation problem. Ballis et al. represent the Monopoly game as a Markov Decision Process. They treat it as a single-agent game since the results of previous single-agent experiments are positive. The agent learns winning strategies in a dynamic environment and demonstrates intelligent playing methods [6].

Besides,Ash et al. have also explored the Monopoly game as a Markov Chain problem and provided rigorous mathematical explanation. The paper introduces a parameter to indicate the distortion of the model from the original game and a higher-eigenvalue-calculation method. Based on the innovative simulation, we can estimate the frequency of pawns' occupation in each position. Thus, the expected income and the estimation for the stability of the property distribution can also be illustrated [10].

However, Ellingsen explores the problem from buyers' perspective. Instead of passively accept the set pricing of the properties, buyers also actively engage in modifying pricing, which maintains an active market. Two formal models are introduced: the lottery model of Gorden Tullock and the perfect discriminating model. The result presents the varied buyer activities according to the bidding rules and strategies [11].

These papers present different winning strategies. Hentzel's gives us a hint that we can explore the best strategy and variable sets by running for thousands of times. The conclusions from Reinforcement Learning offer us game settings and variable sets to refer to. We can also use the naive and greedy method as a base model and improve on it according to different exploring strategies. Ellingsen, on the other hand, provides us with an aspect that we can further work on after finishing the base requirements of the project. We can try adding as many buyers' activities as possible to best simulate the real-world case.

2.3 Parameter Sweep and Computational Clusters

Numerous examples can be found in other fields that use computational clusters to run simulations. In the architecture developed by Hollman and Marti, a PC cluster is used to run a real-time power system simulator, while a single computer can only deal with a limited number of nodes in the system. PC-cluster architecture is necessary to simulate large scale real-world power systems [12].

Another example is the integration of computational grid to air pollution simulation. This integration is based on two conditions. First, the air pollution model is complicated and time-consuming. The system requires efficiency to predict future air pollution problems in the real world [13].

Compared to the previous examples, computational clusters in our project are not as necessary. This project runs simulations on the game Monopoly, whose model is not as complicated as the power system or air pollution. Also, there is no real-world application related to the project that has efficiency demand. With the help of computational clusters, simulations of the game can be run parallel to generate samples efficiently, but the absence of computational cluster should not become the bottleneck of the project.

Computational clusters may not be the pre-requisite of the project. However, running parallel simulation can improve efficiency in parameter sweep applications.

Researchers have already integrated automation to game design. Edward et al.[14] used an auto-playtesting system to design casual game stages. This system can find flaws in the game to improve user experience. For a single parameter, setting a range, step-size and number of trials on each variation and finally, the relationship between parameter and result can be clearly illustrated by running simulations. However, this auto-playtesting system focus on games that require human actions such as touching and sliding. Also, these games are single-player. The Monopoly game is multi-player and does not require human actions as it is turn-based. The system is not generic enough to be expanded to all types of games.

In the research conducted by Casanova et al.[15], heuristics could lead to good performance when running simulations of parameter sweep applications in grid environments. Compared with running simulations on a single computer, using grid environments and heuristic algorithms effectively improves the efficiency. However, the research is general and cannot be directly applied to simulations of the Monopoly game. Based on this research, our project needs to tweak the algorithm and make it fit the context of the Monopoly game.

3 Solution

There are three main parts to our solution: the Monopoly game itself, player strategies, and the user interface.

We started with writing out a runnable Monopoly game because the game itself is the core of our project.

Before we start writing the game, we had to decide which programming language to use, we chose Python after careful consideration. The main reason for the choice is that we found an existing but incomplete implementation of Monopoly game written in Python. This saved us a lot of work by modifying the existing implementation to meet our requirements instead of writing the game from scratch. There are also libraries in Python to optimize our program based on the HPC environment.

After creating a runnable game, since strategy for buyingtrading is an important fact in the game, we started with writing three different naive player strategies towards trading. The first one is purely random. Among the remaining two strategies, one is more aggressive, and one is more conservative. With these three naive strategies, we can have a start point to explore the influence of strategy on the game duration and explore more strategies to make the game duration longer.

To make our solution accessible to researchers. We developed a desktop application. With the help of our application, researchers can create Monopoly game simulations on HPC from their personal computers by simply filling a form. Our application can also display detailed statistics of the simulation and draw graphs to illustrate the relationship between changeable parameters and game duration.

3.1 Monopoly

This section presents how we create a runnable Monopoly game.

3.1.1 Game Rule Simplification

Discussions with our client and non-CS faculty supervisor lead us to simplify some original game rules that are not important to economic and social research purposes. We detail the reasons for the change in Section 5.1

1. In the original rules, apart from normal property, there are two groups of special property: utilities and railroads. These two groups of property cannot be upgraded after a player buys them and the rent collected when other players land on them is related to the total number of utilities/railroads that the owner has. We deleted this rule and implemented all properties as the same normal property.

2. In the original rules, normal properties are of different colors and a player can only upgrade a property when he/she owns all the property of the same color. We delete this rule and players can choose to upgrade any property whenever he/she lands on his/her own property.

3. In the original rules, the bidding may happen when a player tries to buy a house or sell a house. In our game, we did not allow bidding. Instead, each player can only buy a property at its original price. When a player wants to sell a property, he/she can first try to sell it to other players at the original price and if no one would like to buy the property, he/she can then sell it to the bank at 90% of the original price.

4. In the original rules, the mortgage is allowed when a player is in need of cash. We did not allow mortgage. In cases when a player needs cash, he/she can only sell properties for cash.

5. In the original rules, a player goes bankrupt when he/she owes more than he/she can pay. Since we did not allow mortgage in our implementation, we also changed the rule for bankruptcy: a player will go bankrupt if he/she has no property and his/her cash is strictly smaller than 0.

3.1.2 Game Logic and Implementation

After we simplified the rules, we wrote out the game logic to better help us write the implementation.

Algorithm 1 Monopoly Game

```
function TURN($Player$)
    Roll two dices and get $dice1$ and $dice2$
    $Player$ move to next place
    if $Player$ lands on cell "$GotoJail$" then
        $Player$ go to jail
    else if $Player$ lands on $Chance$ block then
        $Player$ picks a $Chance$ card.
    else if $Player$ lands on $Community$ block then
        $Player$ picks a $Community$ card.
    else if $Player$ lands on a property block $P$ then
        if $P$ has no owner then
            $Player$ buys $P$ if satisfying strategy condition.
        else
            if $P$'s owner is $Player$ then
                $Player$ upgrades $P$ if satisfying strategy condition.
            else
                Pay tax to the owner of $P$, sell a house if necessary.
            end if
        end if
    end if
    if $dice1 == dice2$ then
        turn($Player$)
    end if
end function
function GAME($Players$)
    while More than one player has not bankrupted do
        for $Player$ in $Players$ do
            if $Player$ does not bankrupt then
                turn($Player$)
            end if
        end for
        if thenPlayer gets negative cash
            $Player$ bankrupts
        end if
    end while
end function
```

When we implemented the game, we used the Object-Oriented Programming(OOP) methodology. We created separated classes for players and buildings. In these classes, natural activities are represented by class methods. For example, player class has methods like move, buy/upgrade buildings, and pay taxes. This architecture makes our code easy to refactor because different parts of the game are loosely coupled.

3.1.3 Parallel Programming

After finishing the game implementation, our goal became running our program on HPC to see the performance. Thus, we enable our program to run with multiple processes so that it can distribute work over different nodes. We achieve this using a package called "Multiprocessing" in Python.

3.2 Player Strategies

Beside throwing the dice, there are three conditions in which players need to make decisions: buying properties, upgrading buildings and trading. We created three strategies for players to choose each time:

1. The player randomly choose to give up or do the task if there is enough cash (see Algorithm 2).
2. The player make the decision based on a fixed proportion of the current cash/properties (see Algorithm 3). This can be interpreted as an conservative strategy since the player will keep more cash when his/her total property increase.
3. The player do the task if there is enough cash and the cash is more than a fixed constant (see Algorithm 4). This can be interpreted as an aggressive strategy since regardless of the total property a player has, he/she will always a fixed value of cash, and this fixed value will account for smaller proportion when total property increases.

Algorithm 2 Strategy 1

```
procedure STRATEGY 1($price$)
    $do \leftarrow False$
    $dice \leftarrow int(random(0,1))$
    if $dice == 1$ & $cash \geq price$ then
        $do \leftarrow True$
    end if
    return $do$
end procedure
```

Algorithm 3 Strategy 2

```
procedure STRATEGY 2($cash, property, percentage, price$)
    $do \leftarrow False$
    if $\frac{cash}{cash+property} \geq percentage$ & $cash \geq price$ then
        $do \leftarrow True$
    end if
    return do
end procedure
```

Algorithm 4 Strategy 3

```
procedure STRATEGY 3($cash, price, constant$)
    $do \leftarrow False$
    if $cash \geq price$ & $cash \geq constant$ then
        $do \leftarrow True$
    end if
    return do
end procedure
```

A player can use different strategies and different values for the boundary to better simulate real cases.

3.3 User Interface

Game simulation jobs are on HPC clusters. If a researcher wants to start game simulations on HPC, he/she needs to use terminal to log into HPC and submit jobs via command line. Also, users are required to have account accessible to HPC. We want researchers with no computer science background to start game simulation jobs conveniently, therefore, we developed a desktop application that can connect to HPC from personal computers and start game simulation jobs.

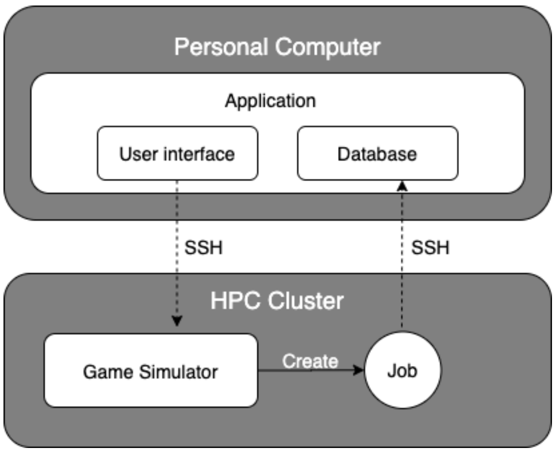


Figure 1: Architecture of our Monopoly game simulation system

3.3.1 Framework

Due to security constraint of browsers, JavaScript running in browser can only create HTTP requests. However, SSH is used to create the connection between client and HPC. The goal is not achievable by web applications. Therefore, we built a desktop application.

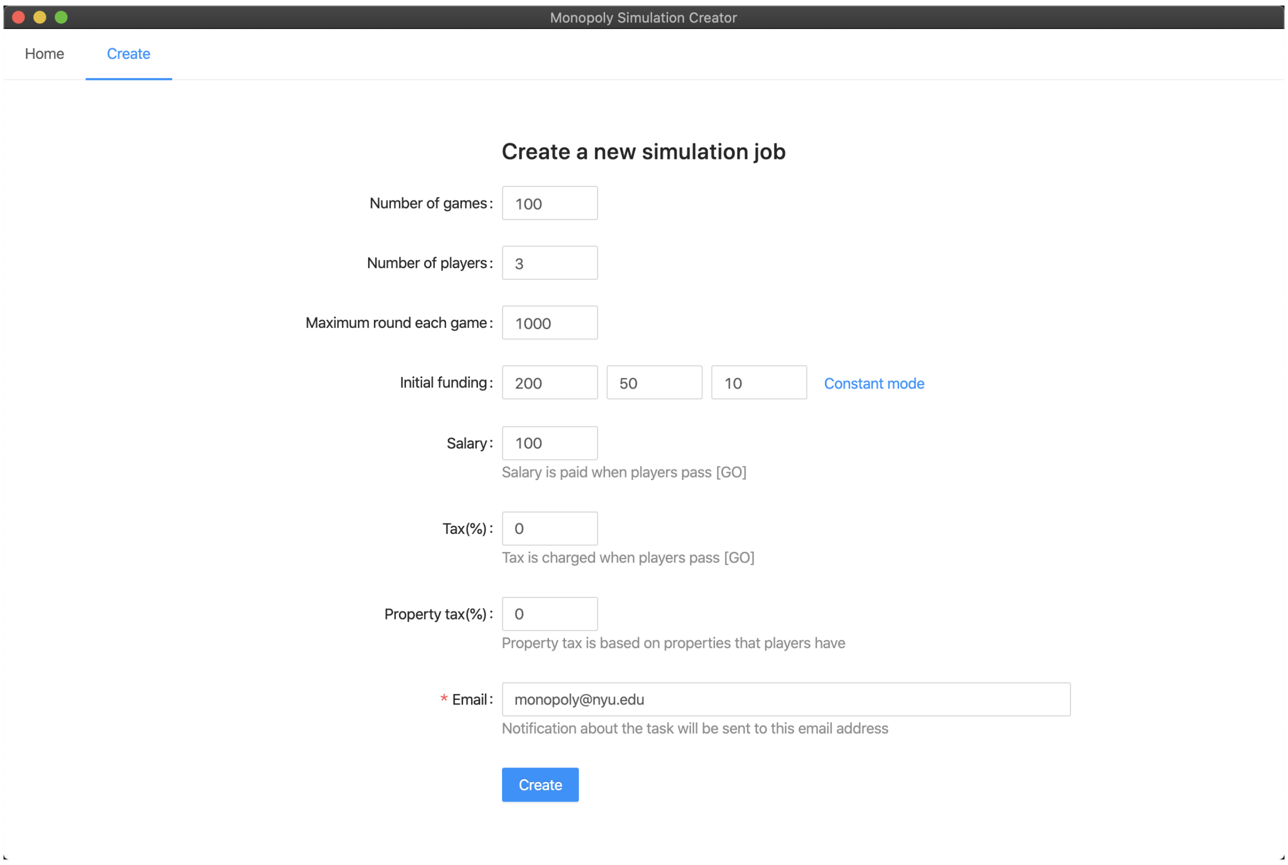
We selected Electron as the framework to build our desktop application. The framework puts Chromium and Node.js together in the same runtime. Therefore, development of user interface is conducted in the same as developing a web application. The application also has the ability in Node.js environment, so the application can have access to the operating system layer, including creating SSH connection.

Moreover, Electron is a cross-platform framework. With the same source code, we can have deliverable packages supporting Windows OS, Mac OS and Linux OS. Researchers are not required to have specific devices to use our solution, and they can conveniently transfer data among different devices on different platforms.

3.3.2 Creating Jobs

In the user interface, users only need to fill a form with configurable parameters. After user submit the form, the application assign an UUID(Universally Unique IDentifier) to the job, start an SSH connection to HPC, create a bash file and use SLURM workload manager to create a job on HPC[16].

The reason of developing such a user interface is that jobs are running on HPC so the output is stored on HPC. Users can actively connect to the HPC to start jobs but when a job is completed, the HPC cannot actively send data to clients. Therefore, UUID is needed in the future when fetching simulation results to ensure the job on the HPC matches the job information stored locally.



Create a new simulation job

Number of games: 100

Number of players: 3

Maximum round each game: 1000

Initial funding: 200 50 10 Constant mode

Salary: 100
Salary is paid when players pass [GO]

Tax(%): 0
Tax is charged when players pass [GO]

Property tax(%): 0
Property tax is based on properties that players have

* Email: monopoly@nyu.edu
Notification about the task will be sent to this email address

Figure 2: Screenshot of the job creating interface

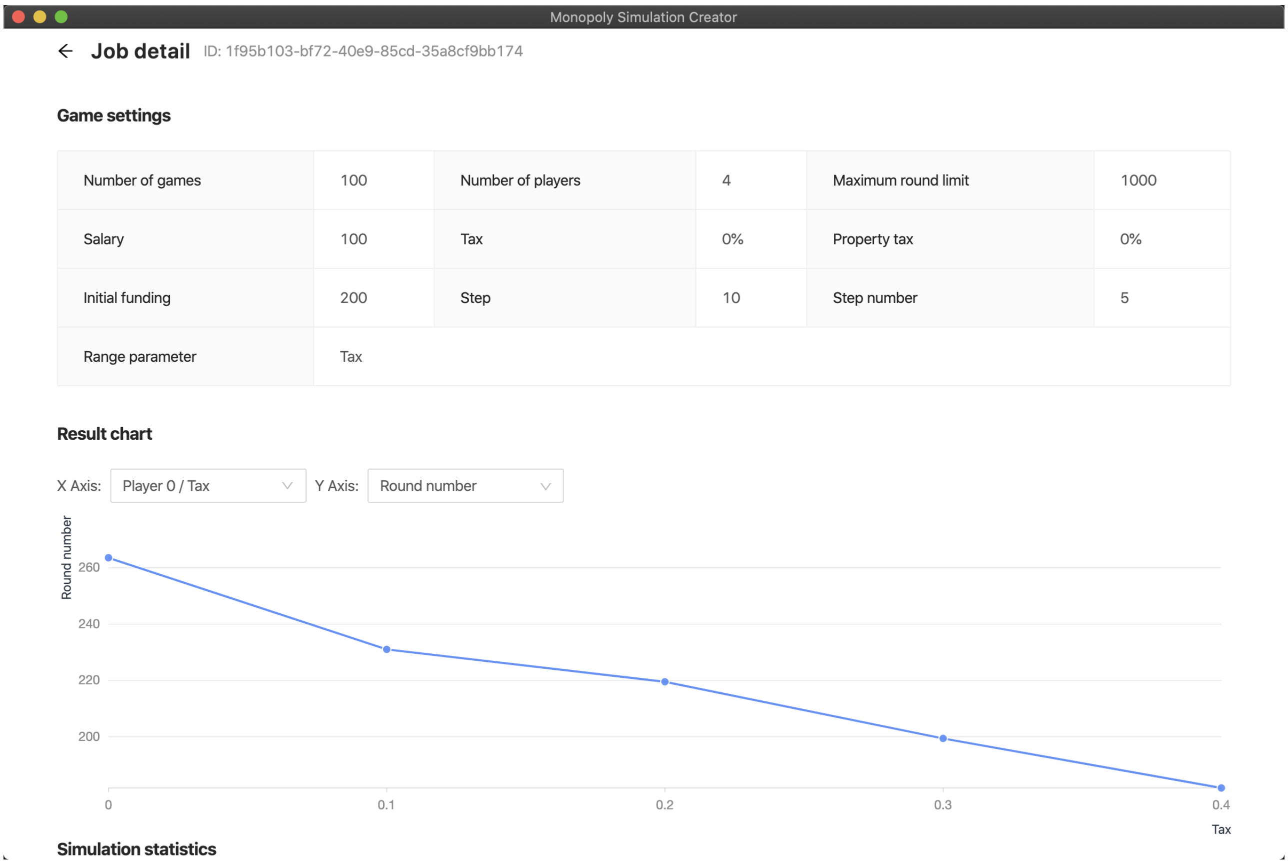
3.3.3 Retrieve Results

When the job is completed, results will be stored in a text file of JSON format on the HPC. Since the game and the user interface are not written in the same programming language, JSON can be a universal data format to share data between different platforms. Since the HPC cannot actively send data to clients, users need to manually retrieve simulation results when the job is finished. When users try downloading results from the HPC, the application will start a new SSH connection, read the output file, parse it and store the result in a local database. We chose to store data locally on users' machines because depending on the importing and exporting functions of our application, the data is portable. Also, network is not required to use the application.

Users can export the database as a file. The file can be stored in cloud storage or hardware storage devices. On a new device, users can import the database file to synchronize jobs and their results.

3.3.4 Display Data

Since results of jobs are stored locally, the application has the ability to read the results and display it in a user-friendly way. In the page containing job statistics, users can find general settings of the job and detailed result of a single simulation. There is also a built-in drawing function that allows users to visualize the relationship between parameters and game duration via a line chart.


Monopoly Simulation Creator

Job detail ID: 1f95b103-bf72-40e9-85cd-35a8cf9bb174

Game settings

Number of games	100	Number of players	4	Maximum round limit	1000
Salary	100	Tax	0%	Property tax	0%
Initial funding	200	Step	10	Step number	5
Range parameter	Tax				

Result chart

X Axis: Player 0 / Tax Y Axis: Round number

Simulation statistics

Figure 3: Screenshot of the job detail interface

4 Results

We ran game simulations with different sets of parameter combinations to explore the relationship between different parameters and game duration. Since we integrate our game simulation system to HPC, we also tested the optimization of our program in the HPC environment. We fixed the number of simulations and run these simulations with different computing resources. Moreover, we have economic researchers as users to test the usability of our job creator application.

4.1 Parameters' Influence on Game Duration

We used Control Variable Method to better analyze the influence of different variables on the game duration, that is, we only change one variable at a time. There are four main parameters in our program, and they are:

- initial funding, the money each player has at the beginning of the game, default value is 1500.
- income, the money a player can get when he/she passes [Go], default value is 200.
- tax, the money charged on players based on cash when he/she passes [Go], default value is 0.1.
- building tax, the money charged on players based on properties he/she has when he/she passes [Go], default value is 0.

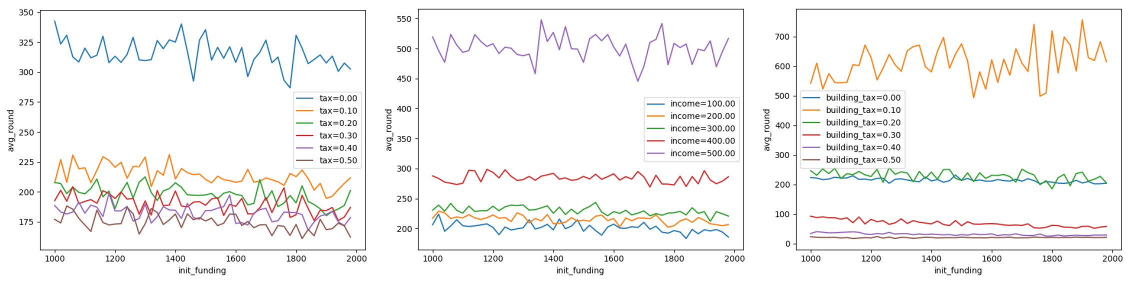
In the following analysis, parameters will use its default value if not mentioned specifically.

Besides these main parameters, we also explored the influence of number of players and player strategies on the game duration. For number of players, by default we run each game with 3 players, but we would also like to see if the game duration will change for games with different number of players. For strategies, by default each player chooses to buy/trade randomly, and we would like to see if the game duration will change when players use naive strategies to make the decisions.

Since we need to evaluate how long the game will last so how many rounds a game will go is a good measurement. Since the game is randomized and abnormal games may happen, we decided to simulate the same set of parameters for 100 runs, exclude the game with the minimum and maximum rounds and calculate the average rounds for the remaining games to lower the influence of abnormal games. Also, in case the game goes for too long without ending, we limit the game to a maximum of 2000 rounds. We chose the limit to be 2000 because after experiment we found that in most situations the game would end in 2000 rounds.

4.1.1 Initial Funding

We simulate the game with initial funding from 1000 to 2000 under different income, tax, building tax conditions to see how it will affect the game duration under different conditions. The following are the graphs.



(a) initial funding vs average round under different tax

(b) initial funding vs average round under different income

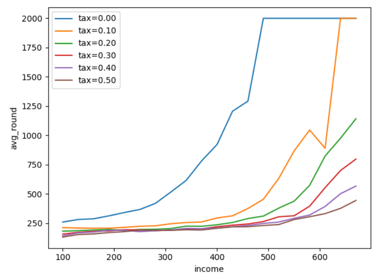
(c) initial funding vs average round under different building tax

Figure 4: Influence of Initial Funding Under Different Conditions

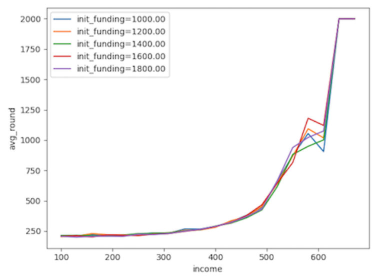
From Figure 4, we can see that no matter how other parameters change, the plot is always a horizontal line with small float up and down. This means that initial funding does not affect the game duration a lot.

4.1.2 Income

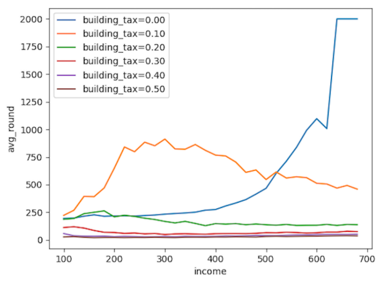
We simulate the game with income from 100 to 700 under different initial funding, tax, building tax to see how it will affect the game duration under different conditions. The following are the graphs.



(a) income vs average round under different tax



(b) income vs average round under different initial funding



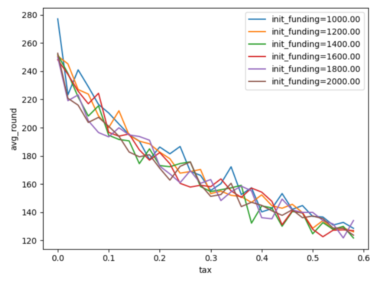
(c) income vs average round under different building tax

Figure 5: Influence of Income Under Different Conditions

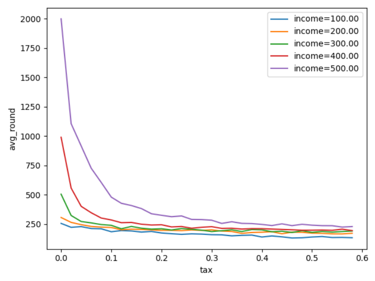
We can see from Figure 5 that overall the increase in income will lead to a longer game duration. But the effect is different under different conditions. As mentioned above, initial funding does not affect the game duration. Normal Tax and building tax both can lower the effect income has on the game duration. This is because a player with higher income will always have more money and buy more properties, thus paying higher normal tax and building, so higher normal tax and building tax will neutralize the benefit brought by higher income. Also, the line of tax=0 becomes flat as income goes beyond 500. This means games with income beyond 500 when tax=0 do not end within the limit of 2000 rounds. That is because when income goes beyond 500, players can earn more than they would lose in each round so it is almost impossible for them to bankrupt without a tax to limit them.

4.1.3 Tax

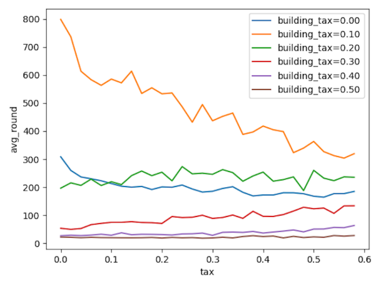
We simulate the game with tax from 0 to 0.6 under different initial funding, income, building tax to see how it will affect the game duration under different conditions. The following are the graphs.



(a) tax vs average round under different initial funding



(b) tax vs average round under different income



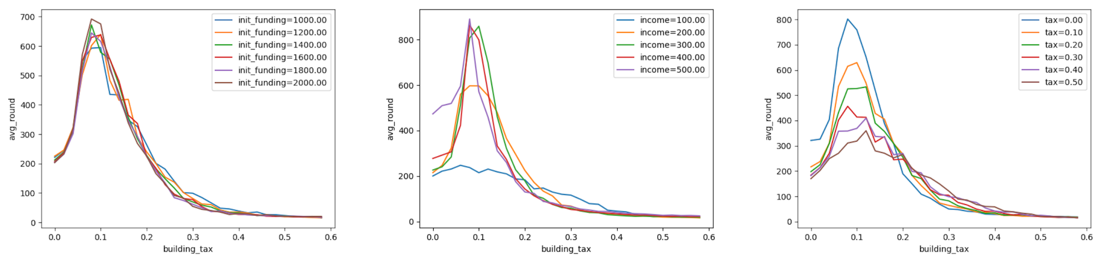
(c) tax vs average round under different building tax

Figure 6: Influence of Tax Under Different Conditions

We can see from Figure 6 that tax always has a negative effect on game duration as we expected because tax always make the player lose more money. The effect of tax is more obvious when players have a higher income because tax is based on the cash players have so more income means more cash and more tax.

4.1.4 Building Tax

We simulate the game with building tax from 0 to 0.6 under different initial funding, income, tax to see how it will affect the game duration under different conditions. The following are the graphs.



(a) building tax vs average round under different initial funding (b) building tax vs average round under different income (c) building tax vs average round under different tax

Figure 7: Influence of Building Tax Under Different Conditions

We can see from Figure 7 that no matter how initial funding, income, and tax change, average rounds always first increase and then decrease with the increase of building tax, and the average round of game reaches its maximum near building_tax=0.1. This trend makes sense because players in the game need land properties to get rent from other players and to make other players bankrupt. Therefore, when the building tax is low, it is always better for a player to own more properties because more properties mean more rent. Building tax will only function as a tool to lower the gap between the rich and the poor, so an increase in building tax will make the game last longer. However, as building tax increases, more properties may not always benefit players. The cost of building tax may exceed the rent the player can get from buying a property. More properties may even mean a larger chance for a player to bankrupt, and that's why the game duration decreases as building tax becomes too large. And in our game, when building tax is larger than 0.1, buying properties will no longer benefit players.

4.1.5 Number of Players

We simulate the games with 3 to 10 players with all other parameters set to default values. The following is the figure.

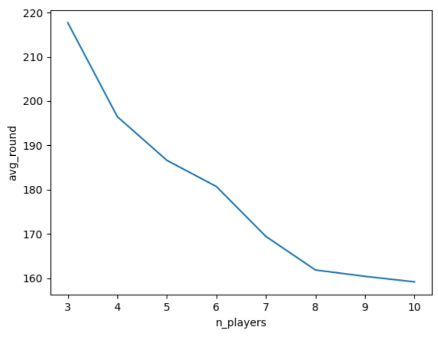


Figure 8: number of players vs average round

We can see from Figure 8 the average round of the game decrease with the increase of number players. We think this is because with more players the properties in the game are more dispersed, so players are more likely to bankrupt.

4.1.6 Player Strategies

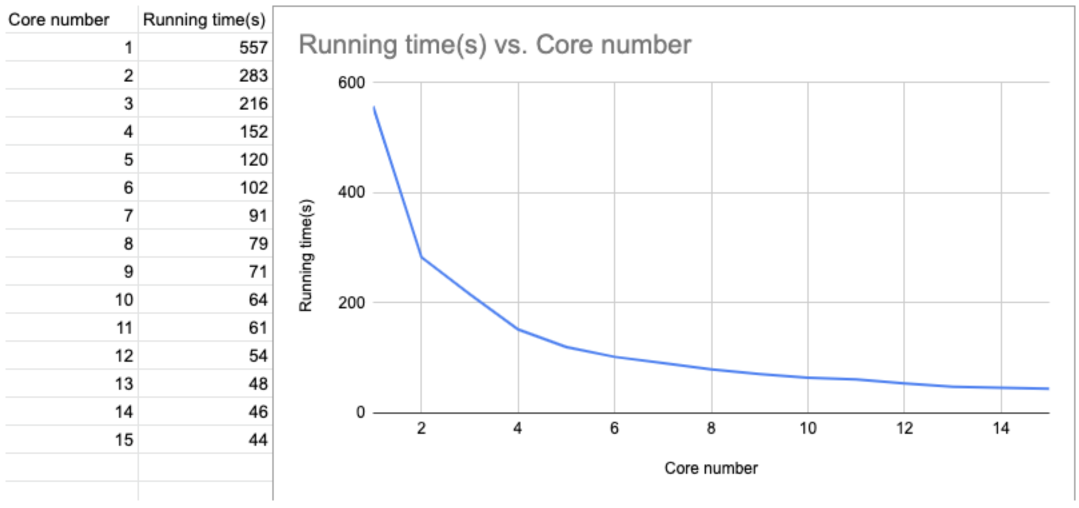
We simulate the game with three different strategies with all other parameters set to default values. With Algorithm 2, the average round is 217.62, with Algorithm 3, the average round is 560.78, and with Algorithm 4, the average round is 882.82. These results show that games where players use naive strategies will last longer than games where players only make decisions randomly. This means that strategy does make difference to game duration and further exploration in game strategies is meaningful.

4.2 Utilization of HPC Resources

When we programmed our game simulation system, we allowed simulations with different parameter combinations to be run in parallel on different processes. Before simulation starts, we need to configure the number of process used in the simulation. To examine the effect of the parallelization, we did some experiments where the number of simulations was set to constant and computing resources were variant. In these experiments, core number and node number are configurable fields of computing jobs while process number is set programmatically in our game implementation.

4.2.1 Utilization of CPU Cores

To examine the efficiency of multi-process computing, we fixed the simulation to have 100 combinations of parameters and 100 games in each combination. We also set the random seed to 0 to make workload consistent. We had all the simulations run on a single node to eliminate the influence of network.

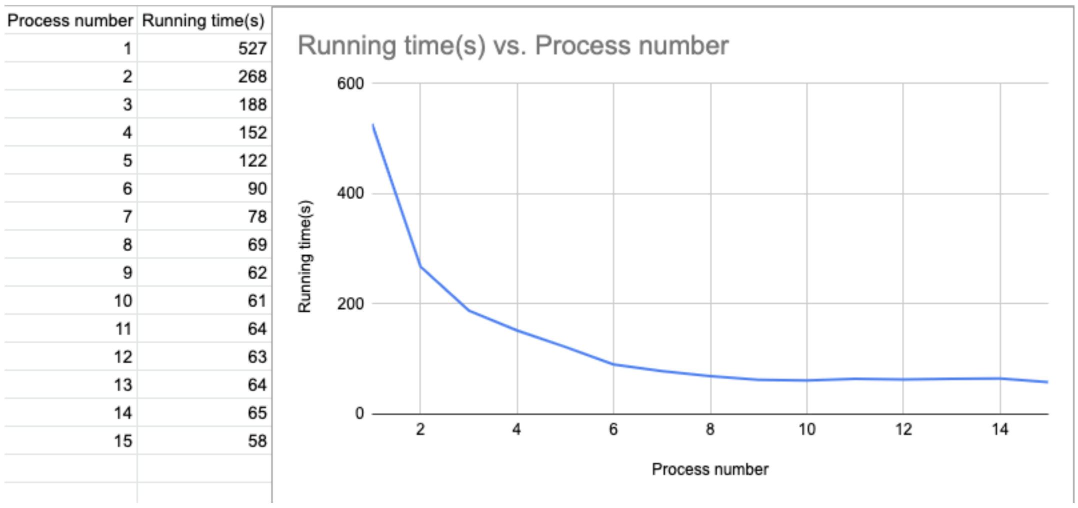


Core number	Running time(s)
1	557
2	283
3	216
4	152
5	120
6	102
7	91
8	79
9	71
10	64
11	61
12	54
13	48
14	46
15	44

Figure 9: Running time when core number ranges from 1 to 15 and process number = core number

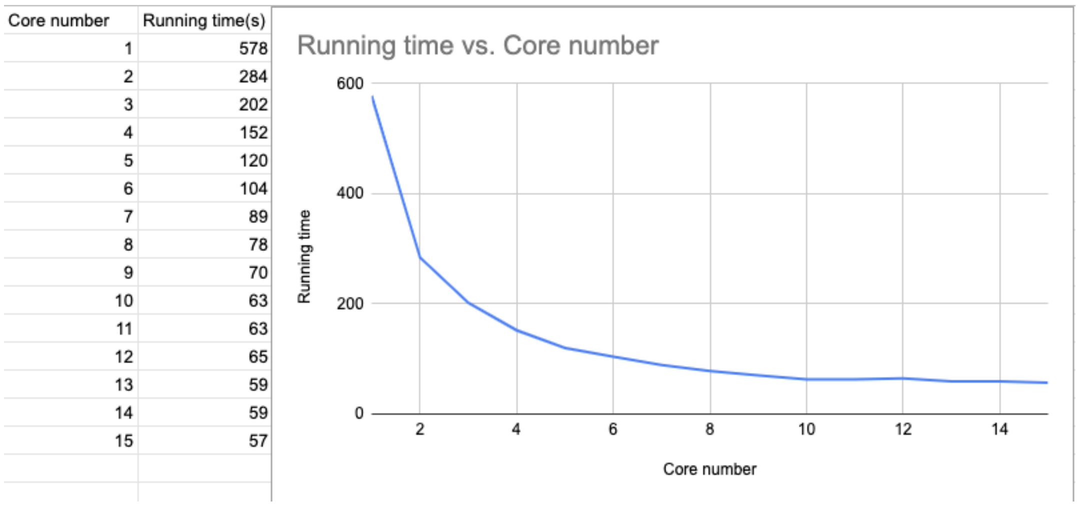
As what is shown in Figure 9, when we synchronously increased core number and process number, the running time would steadily decrease. The running time and core number have an inversely proportional relationship. The results illustrate that on HPC, we can distribute our computation load evenly to multiple cores to improve efficiency of our game simulation.

To explore the relationship between physical CPU cores and process required programmatically, we did another two experiments.



Process number	Running time(s)
1	527
2	268
3	188
4	152
5	122
6	90
7	78
8	69
9	62
10	61
11	64
12	63
13	64
14	65
15	58

Figure 10: Running time when process number ranges from 1 to 15 and core number = 10



Core number	Running time(s)
1	578
2	284
3	202
4	152
5	120
6	104
7	89
8	78
9	70
10	63
11	63
12	65
13	59
14	59
15	57

Figure 11: Running time when core number ranges from 1 to 15 and process number = 10

Figure 10 and 11 show that when either value exceeds the other value, the running time will not decrease. In a word, running time relates to the lower value out of the two. We believe the reason is that on HPC, one CPU core has one physical process. If the program asks for more process the device has, it needs to wait until processes to be released from previous running simulations. When the core number is larger than process number asked, some cores are idle in the simulation. This finding helps us configure core number and process number in an efficient way.

4.2.2 Utilization of Computational Nodes

Nodes refer to physical computers. When jobs are executed on multiple nodes, workload is distributed to different computers and results are collected via a network connecting these computers. When the load is low, efficiency improvement of parallel computing cannot counteract the cost of network transmission. In this experiment, we ran simulations with 1000 and 2000 combinations to verify the effect. To ensure nodes have equal computing power, we allocated 5 cores to each node.

Due to network transmission latency, improvement of computing efficiency is not proportional to the number of nodes. Effect of multiple nodes is more obvious when simulation number is

Node number	Simulation number	Running time (s)
1	1000	1274 s
3	1000	1247 s
5	1000	1209 s
1	2000	2540 s
3	2000	2359 s
5	2000	2231 s

Table 1: Comparison of running time between simulations running with different number of nodes

larger. The comparison between running 1000 combination and 2000 combination shows that huge load is the precondition of benefiting from multiple computational nodes.

4.3 User test for the desktop application usability

We have Professor Almaz Zelleke to test the usability of our desktop application as asks her to give mark from 1 to 5 for the simplicity/clarity and achievement. User test covers two core features of our application:

1. Submit simulation jobs.
2. Display simulation results.

Professor Zelleke gave 5 for simplicity because it was easy to change parameters and submit simulation jobs. As for the result displaying feature, Professor Zelleke could understand the user interaction after we explained to her in detail, so she gave 3 for clarity on this feature. Overall the application meets her requirements, so she gave 5 to achievement.

The job creation interface is convenient to use. However, from the feedback given by users, there is still work need to be done in the future to make the data displaying interface clearer.

5 Discussion

5.1 Game Rules

As mentioned above, to make it easier for the implementation of the game, we simplified several rules in the original game.

5.1.1 Property Types and Colors

We eliminated the features of property type and property color because they are not important for the game to reflect real life situations and deleting them can make it easier for us to implement the game within limited time. But adding more features to the game is always better. If we had enough time, we would add these features to the game.

5.1.2 Bidding and Mortgage

We did not allow bidding and mortgage in our program. Bidding is an interesting feature where players can compete for a property when more than one player wants it, while mortgage is an important feature with which the game can better reflect real life since mortgage is relatively common in the society. But implementing both of them is difficult. For bidding, it is hard to decide at which price the bid should begin and end, and how the price should increase. A possible naive strategy is that the bid ends when only one player can afford the property. However, in this way, the player with the most cash will always win the bid, which is not realistic. For mortgage, mortgage is used when a player is running out of cash. But selling properties is another choice in case a player needs cash. It is hard for a player to decide when he/she should choose mortgage, or he/she should sell a property.

5.2 Player Strategy

Player Strategy is an important part of the Monopoly game. Due to the limitation of time, we did not fully explore this area. We only enabled three relatively naive strategies for buying and trading in our program so that it can better reflect real life. These are not enough because human behavior is more complicated than these naive strategies. Also, some features cannot be implemented with only naive strategies such as the bidding and mortgage mentioned above. Therefore, if we had enough time, we would try several advanced algorithms involving machine learning such as the genetic algorithm and Markov Process to explore more about player strategies.

However, the current strategies are not too naive to simulate the reality. As mentioned above, the player can freely change strategies within a round of the game. Therefore, since the strategies are not robust, it may still simulate some real-case strategies.

5.3 User Interface

Simulation jobs are executed on HPC, but services such as web application cannot be deployed on HPC. Therefore, users need to submit simulation jobs by themselves. Here comes another problem, the process of submitting a simulation job. As developers, we can log into HPC and create jobs via command line, but users should not be required to have the skill of using command line. The core problem in the development of the application is how to let users access HPC in a friendly way. The solution we finally brought up was a desktop application.

Our solution has some drawbacks. For example, users need to connect to NYU Network before using our application. When a simulation job is completed, users have to manually download the results and the results are stored locally.

If we could have a server that not only has access to the HPC cluster, but is also accessible to the public internet, we would build a web application as the centralization of simulation jobs. Users can submit jobs in web browsers and when jobs are completed, results will be stored on the cloud automatically. This solution would be more convenient than our current implementation.

6 Conclusion

6.1 Project Summary

In this project, we developed a Monopoly game simulation system to run game simulations with changeable parameters. Players in the game have different strategies to simulate real society. The program is designed for research purpose so researchers can start numerous simulations to see the effect of game parameters.

We deployed the game simulation system on HPC to utilize multiple cores and nodes. Higher efficiency of running simulations is acquired on HPC compared to running them on personal computers.

We also developed a desktop application that requires no computer science experience for researchers to create simulation jobs and view results on their personal computers.

6.2 Limitations

Our project only supports the simulation of Monopoly game. Therefore, the research direction based on our project is limited.

In the implementation of our game simulation system, human activities such as trading cannot be perfectly simulated like real human players. The human decision-making process is complicated and cannot be perfectly reflected by code.

From the perspective of researchers, our application can be improved to have a clearer interaction. More features need to be added to the application to fulfill the requirements of academic research.

6.3 Future Extension

In the future, we plan to use machine learning techniques to improve the reliability of player activities. According to our previous research in the related work section, reinforcement learning can be applied to our project where the game is treated as a Markov Process.

We also want to make our project more generic. Our project can be extended as a general simulation platform for academic research. In the future, not only the Monopoly game, simulations in other forms should be supported to make the project a general tool for researchers.

Acknowledgement

We would like to express our great appreciation to Professor Olivier Marin and Professor Almaz Zelleke for their constructive suggestions and continuous support.

References

[1] "Monopoly parker brothers real estate trading game," https://www.hasbro.com/common/instruct/monins.pdf.

[2] M. Pilon, in *The Monopolists: Obsession, Fury, and the Scandal Behind the World's Favorite Board Game.* Bloomsbury, 2015.

[3] S. B. Shanklin and C. R. Ehlen, "Using The Monopoly Board Game As An In-Class Economic Simulation In The Introductory Financial Accounting Course," *Journal of College Teaching & Learning (TLC)*, vol. 4, no. 11, Nov. 2007. [Online]. Available: https://clutejournals.com/index.php/TLC/article/view/1525

[4] H. Casanova, F. Berman, G. Obertelli, and R. Wolski, "The apples parameter sweep template: User-level middleware for the grid," in *SC '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, Nov 2000, pp. 60–60.

[5] C. Youn and T. Kaiser, "Management of a parameter sweep for scientific applications on cluster environments," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 18, pp. 2381–2400, 2010. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1563

[6] C. R. E. Stephen B. Shanklin, "Using the monopoly® board game as an in-class economic simulation in the introductory financial accounting course," vol. 4, pp. 65–72, Nov 2007. [Online]. Available: https://clutejournals.com/index.php/TLC/article/view/1525

[7] S. G. An Ansoms, "Development monopoly: A simulation game on poverty and inequality," *Simulation & Gaming*, pp. 854–862, Aug. 2013. [Online]. Available: https://journals.sagepub.com/doi/pdf/10.1177/1046878112451877

[8] T. B. G. G. G. Eric J. Friedman, Shane G. Henderson, "Estimating the probability that the game of monopoly never ends," pp. 380–391, 2009. [Online]. Available: https://www.informs-sim.org/wsc09papers/036.pdf

[9] I. R. Hentzel, "Strategic buyers and the social cost of monopoly," *The Saturday Review*, pp. 44–50, 1973. [Online]. Available: https://www.unz.com/print/SaturdayRev-1973mar24-00044/Contents/

[10] R. B. Ash and R. L. Bishop, "Monopoly as a markov process," *Mathematics Magazine*, vol. 45, no. 1, pp. 26–29, 1972. [Online]. Available: http://www.jstor.org/stable/2688377

[11] T. Ellingsen, "Strategic buyers and the social cost of monopoly," *The American Economic Review*, vol. 81, no. 3, pp. 648–657, 1991. [Online]. Available: http://www.jstor.org/stable/2006526

[12] J. Hollman and J. Marti, "Real time network simulation with PC-cluster," *IEEE Transactions on Power Systems*, vol. 18, no. 2, pp. 563–569, May 2003, conference Name: IEEE Transactions on Power Systems.

[13] B. Xu, H. Lin, J. Gong, S. Tang, Y. Hu, I. A. Nasser, and T. Jing, "Integration of a computational grid and virtual geographic environment to facilitate air pollution simulation," *Computers & Geosciences*, vol. 54, pp. 184–195, Apr. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0098300412003937

[14] E. J. Powley, S. Colton, S. Gaudl, R. Saunders, and M. J. Nelson, "Semi-automated level design via auto-playtesting for handheld casual game creation," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sep. 2016, pp. 1–8.

[15] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*, May 2000, pp. 349–363.

[16] P. J. Leach, M. Mealling, and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace," library Catalog: tools.ietf.org. [Online]. Available: https://tools.ietf.org/html/rfc4122