National College of Ireland

BSc (Honours) in Computing - Evening

2023/2024

Michal Eichler

x20139462

x20139462@student.ncirl.ie

Computing Project

Technical Report

Executive Summary

1.0 Introduction

1.1. Background

I chose to undertake this project as an opportunity to learn web graphics, game programming and developing AI-powered applications. The development methodology is to emphasize learning and creativity.

1.2. Aims

There are two end goals for the project itself. One is to create a game, e.g. something playful, light-hearted and if possible, humorous. The other is to introduce the audience (and me) to some basics of economics, by means of play.

The project's source code should open and be available free of charge for anyone (post grading). If possible, I'd like to make the game playable for free as well, but as currently developed, this could incur non-trivial cost, due to the use of a paid AI model.

1.3. Technology

The front-end is developed using Three.js (threejs.org, n.d.) and standard web technologies. Three.js enables 3D graphics.

The back-end is a FastAPI (fastapi.tiangolo.com, n.d.) app. A PostgreSQL (www.postgresql.org, n.d.) database stores the game messages (the user selections and the AI responses), without identifying the user.

The AI used for the project is Mistral Small (docs.mistral.ai, n.d.) large language model, accessed over a REST API.

2.0 System

2.1. Requirements

The requirements of the application are few and straightforward, whilst challenging in implementation. This is on purpose, to maximize learning opportunities in development.

2.1.1. Functional Requirements

1. The user should be able to select a game.

2. The user should be able to update economic policy.

3. The user should be able to receive economic metrics updated over time in response to the policy.

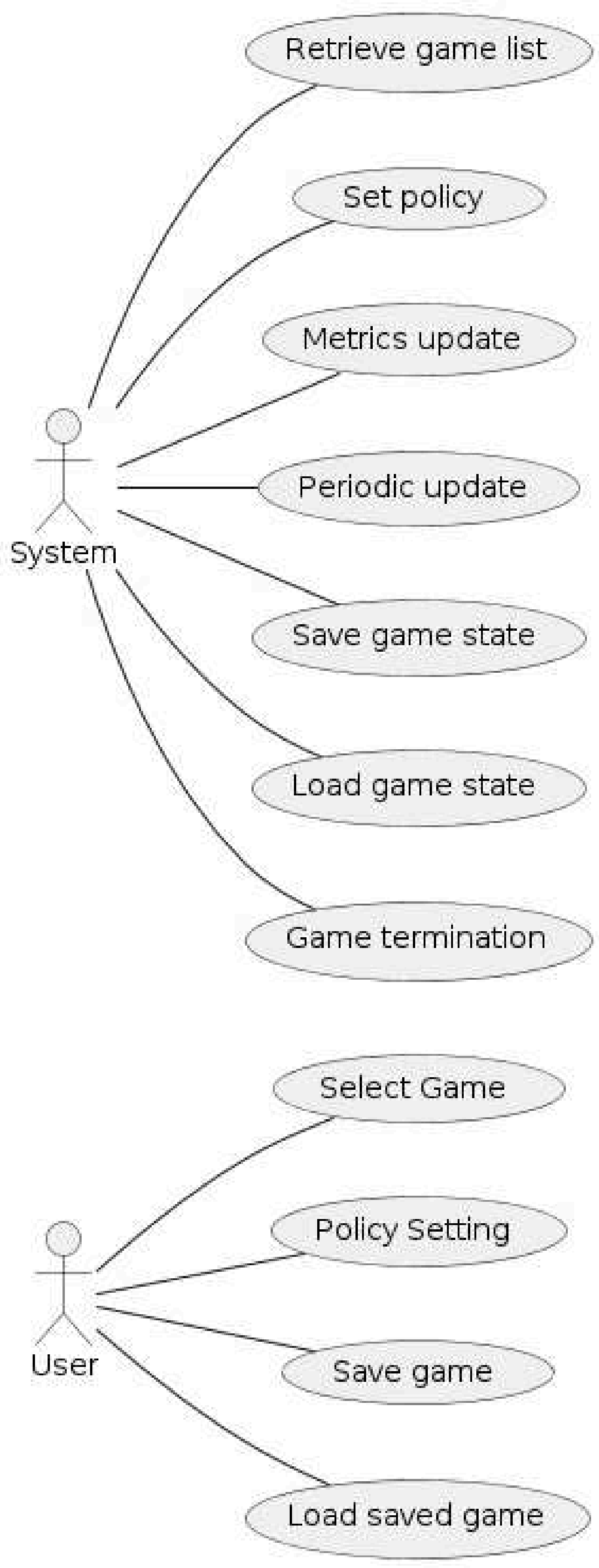
4. The game should periodically generate updated metrics, if the user does not set the policy.

5. The user should be able to save the state of the game to his local computer.

6. The user should be able to load a saved state.

7. The user should be able to terminate the game by winning or losing.

2.1.1.1.1. Use Case Diagram



2.1.1.2. Requirement 1: Game selection

Priority: essential.

The user should be able to select one of three games. All are essentially the same, however each is governed by a different set of rules. The rules of each game are based on the economic principles of one of three famous economists: Adam Smith, Karl Marx and John Maynard Keynes. The rules are given by the system prompt. The LLM decides what values to return in response to users input.

2.1.1.3. Requirement 2: Policy setting

Priority: essential.

The user selects values for the relevant economic settings of each game (e.g. money supply, interest rate, government spending, tax rate). This can be done often or not at all, the game starts with a hard-coded pre-set for each value.

2.1.1.4. Requirement 3: Metrics update

Priority: essential.

The game responds to the user's policy settings by updating a set of metrics (population increase, unemployment, government debt, etc). These metrics are to be updated over a set time (a month in game time for example), simulating the effects of real-world economic policies.

2.1.1.5. Requirement 4: Periodic update

If the user does not set the policy over a set period of time, the game will generate updates from existing state.

2.1.1.6. Requirement 5: Saving

Priority: essential.

The user should be able to save the entire state of the game to the database.

2.1.1.6.1. Requirement 6: Loading

Priority: essential.

The user should be able to load a previously saved game.

2.1.1.6.2. Requirement 7: Termination

Priority: essential.

The user should be able to either lose or win the game, based on reaching or failing to reach certain metrics values over a given time or based on reaching values that are considered a “lost game”.

2.1.1.6.3. Use Cases

UC1 Select a Game

- Description: User selects a game to play
- Actor: User
- Triggers: The user selects one of the game titles
- Pre-condition: The user reached the site
- Post-condition: Game starts to play
- Normal flow:
  - The user clicks the game title of the game they want to play.
  - The screen changes, the selected game starts loading
- Exceptional flow:
  - E1. Connection error (frontend can't access the server)
  - E2. DB error (backend can't access the database)

UC2 Policy Setting

- Description: The user selects the policy update UI element, sets the policy and clicks “Send”.
- Actor: User
- Triggers: the user clicks the "Policy Settings" button
- Pre-condition: The game is being played.
- Post-condition: A new policy is set.
- Normal flow:
  - The user clicks on the “Policy” button.
  - The game presents the user with a policy selection UI.
  - The user sets the policy and clicks send.
- Alternate flow: The user cancels the action..
- Exceptional flow:
  - E1. Connection error (frontend can't access the server)
  - E2. DB error (backend can't access the database)

UC3 Metrics Update

- Description: The game updates metrics.
- Actor: System
- Triggers: UC2
- Pre-condition: The game is being played
- Post-condition: Metrics are updated
- Normal flow: The game receives a response from the LLM with the new data and displays it to the user.
- Exceptional flow:
    - E1. Connection error (frontend can't access the server)
    - E2. DB error (backend can't access the database)
    - E3. The LLM replies with an unparseable response.
    - E4. The LLM can't be reached.

UC4 Periodic Update

- Description: The game automatically updates metrics.
- Actor: System
- Trigger: Internal game clock.
- Pre-condition: The user has not updated policy over a certain time period
- Post-condition: Metrics are updated.
- Normal flow: The system determines that the list of previously obtained metrics is almost exhausted.. The game sends the current policy to the LLM.
- Exceptional flow:
    - E1. Connection error (frontend can't access the server)
    - E2. DB error (backend can't access the database)
    - E3. The LLM replies with an unparseable response.
    - E4. The LLM can't be reached.

UC5 Save Game

- Description: The user saves the game.
- Actor: User
- Trigger: The user clicks on the “Save Game” button
- Pre-condition: The game is being played

- Post-condition: The game is saved.
- Normal flow:
  - The system saves the game state to the database.
  - The system saves the game ID in the user's clipboard.
- Exceptional flow:
  - E1. Connection error (the frontend can't access the the server)
  - E2. DB error (the backend can't access the database)

UC6 Load Game

- Description: The user loads a previously saved game.
- Actor: User
- Trigger: The user enters the game ID and clicks on the "Load Game" button.
- Pre-condition: The user has reached the site.
- Post-condition: The game starts playing.
- Normal flow: The system loads the previously saved game state.
- Exceptional flow:
  - E1. Connection error (frontend can't access the server)
  - E2. DB error (backend can't access the database)
  - E3. The game ID is not found in the database.

UC7 Game Ends

- Description: The game play comes to a conclusion.
- Actor: System
- Trigger: Internal game clock
- Pre-condition: The game has been played for a set amount of game time.
- Post-condition: The game has ended.
- Normal flow: The user is presented with a screen determining whether the game was won or lost.
- Exceptional flow:
  - E1. Connection error (frontend can't access the server)
  - E2. DB error (backend can't access the database
  - E3. The LLM replies with an unparseable response.

2.1.2. Data Requirements

The game stores the following data points

- Scenarios: 3 different scenarios that included system prompts and relevant images for display.
- Games played, with times and results (if played to the end)
- Metrics: sets of metrics that were used by all of the games. Some were generated by the LLM, some were interpolated.
- Policy settings, entered by the user (or the default "all zeros", if the user never entered any)
- Raw messages: the stored text of the exchanges between the API and LLM

These are all stored in the above-mentioned Postgresql database. No personal identifiable information (IP addresses, names, etc.) is kept.

2.1.3. User Requirements

User should be a regular person, capable of playing web games and reading and understanding economic articles in the Irish Times.

2.1.4. Environmental Requirements

The final deployment will require a Linux virtual private server with Python and Caddy installed.

2.1.5. Usability Requirements

The game should be intuitive and not require a manual. Internationalization and accessibility features are not planned.

2.2. Design & Architecture

The back-end is a FastAPI application that serves the front-end and provides communication between the front-end and the LLM. Standard routing is used.

The front-end is a web interface, which consists of a main page with the game scene, the game selection page and an information page. The game gives the impression that several other pages are displayed (loading and end of game) however those are just overlays on the main scene page.

The main page contains a main canvas element . ThreeJS is used to draw the UI on the page and ensure user interaction.

The LLM is hosted by Mistral AI and configured by the "system" part of each message.

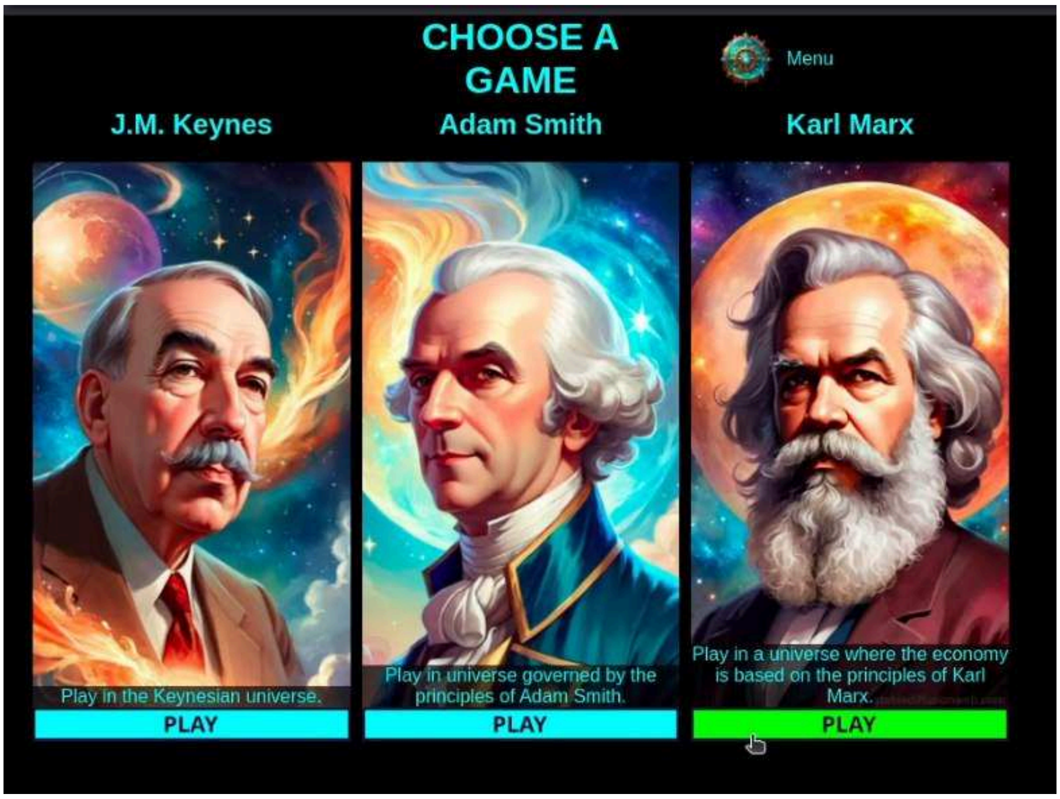
2.3. Graphical User Interface (GUI)

The GUI is created from ThreeJS objects, a model made in Blender (Blender Foundation, 2019), HTML elements and some assets generated using Stable Diffusion (stablediffusionweb.com, n.d.).

The AI-generated images have Stable Diffusion watermarks on them. In order to reduce the cost of this project, I have not upgraded to the tier that does not include them. The Stable Diffusion FAQs (stablediffusionweb.com, n.d., 2) do not mention any restricted use for their images.

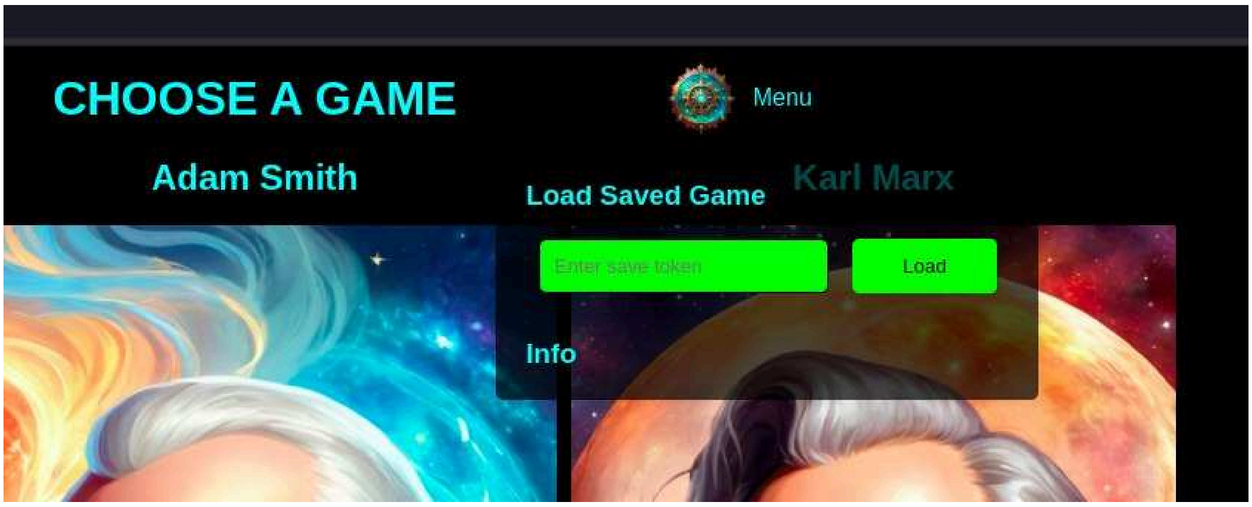
There is a background image of a space nebula on the main screen, which is sourced from Space Spheremaps (TonyS, 2024). I have processed the original image to reduce its size.

The first page that the user encounters offers the game selection.


CHOOSE A GAME
Menu
J.M. Keynes
Adam Smith
Karl Marx
Play in the Keynesian universe.
PLAY
Play in universe governed by the principles of Adam Smith.
PLAY
Play in a universe where the economy is based on the principles of Karl Marx.
PLAY

Each game is represented by a generated image of the economist in question. The user can click on the "PLAY" button under any of the images, to start the given game.

This page also has a menu that allows for loading of a previously saved game and provides a link to the info page.



When the user selects a game, it takes a while to load (in the background the system is querying the LLM). When everything loads, the user is presented with the main game screen.

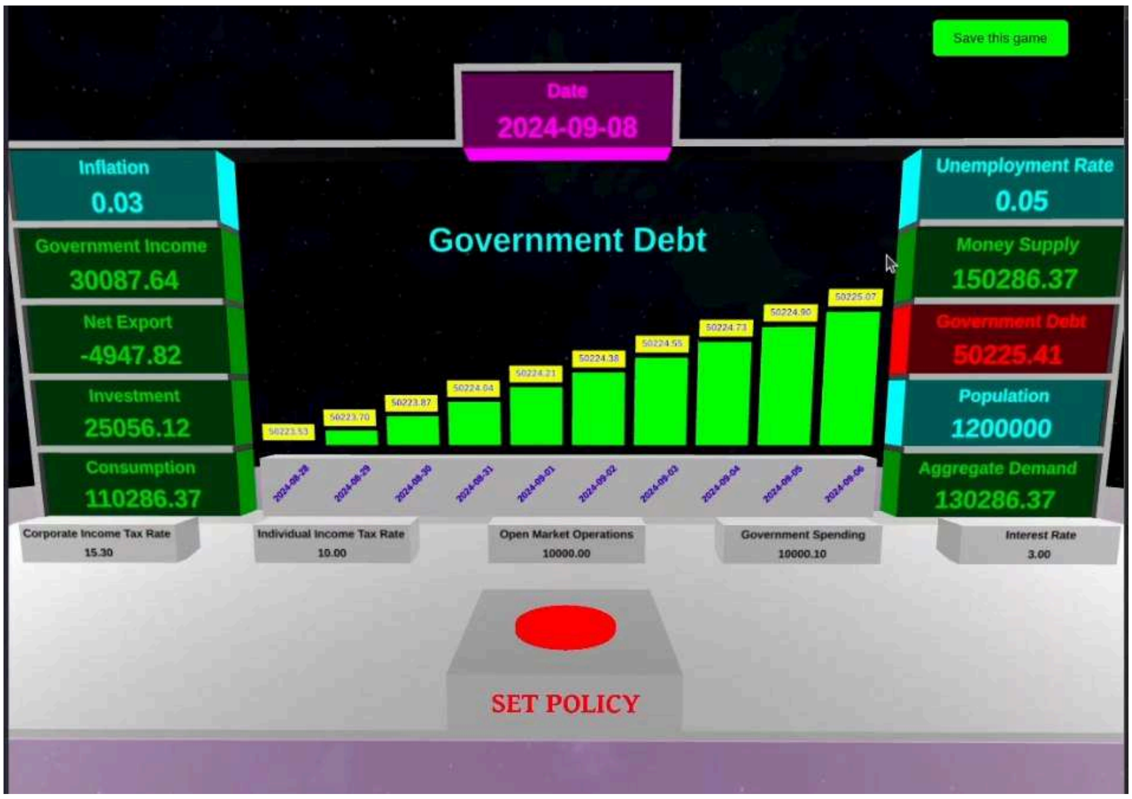
This screen shows a console with display panels and a single button.

On the top of the console is a "calendar" panel that shows the in-game date. The game starts at the current date and lasts 6 in-game months. Each in-game day is 5 real-time seconds, however pressing the "Set Policy" button stops the in-game clock.

The panels show the 10 different economic metrics. Each panel starts coloured in cyan. As long as the metric value is the same as the previous day, the display's color is cyan. When the value improves, the display turns green, when it worsens, the display turns red.

In the middle, between the panels, is a chart showing the last 10 days worth of values of a single metric. The UI periodically switches between the metrics. The user can set the chart to a specific metric by clicking on the metric's display panel. Clicking on the same panel again, restarts the rotation.

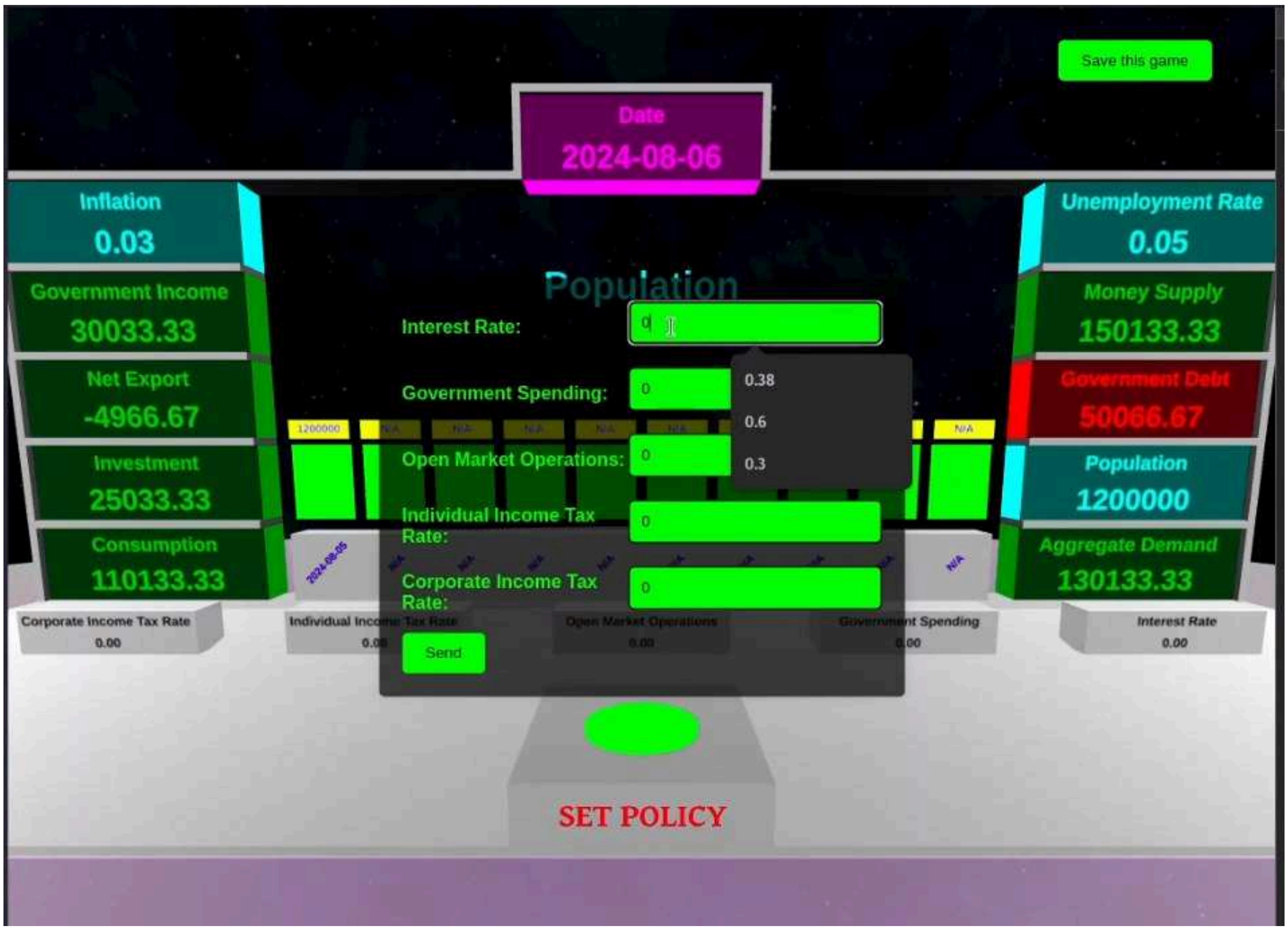
There is a limited zoom/pan/dolly movement available, using the built in ThreeJS controls.



Behind the console, the user can see a slowly rotating nebula. When the set policy button is clicked and policy setters show, the nebula stops turning, reinforcing the impression of stopped time.

The user can set economic policy by clicking on the red “SET POLICY” button. This will display a form. The user can click on the button again to dismiss the form, or send the policy to the LLM. The policy does not have to be changed when sent. Sending the same policy extends the period of time before the system sends one automatically.

The current policy is displayed on five silver translucent panels between the policy button and the metrics displays.



Interest Rate: q

Government Spending: 0

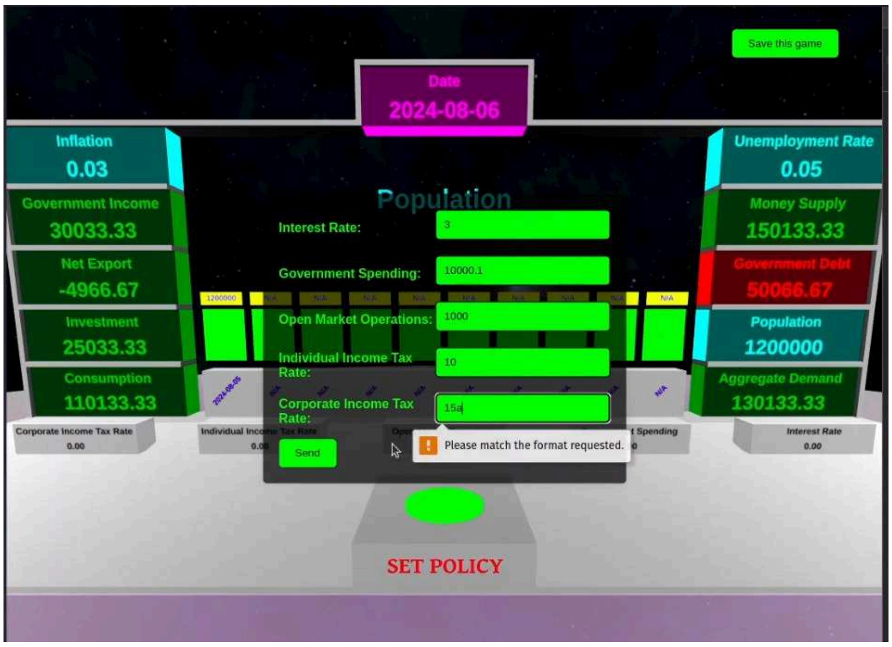
Open Market Operations: 0

Individual Income Tax Rate: 0

Corporate Income Tax Rate: 0

0.38
0.6
0.3

The form validates input, only integers and floats are allowed.



Interest Rate: 3

Government Spending: 10000.1

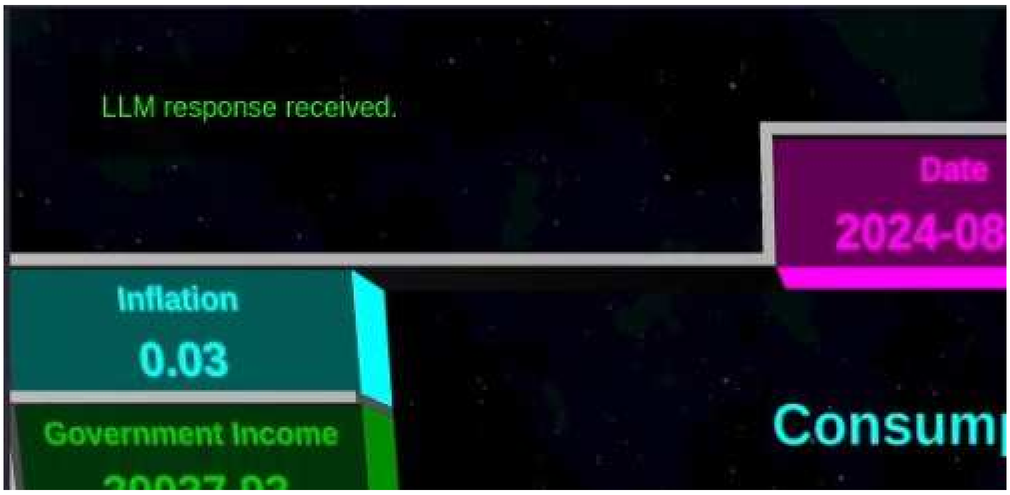
Open Market Operations: 1000

Individual Income Tax Rate: 10

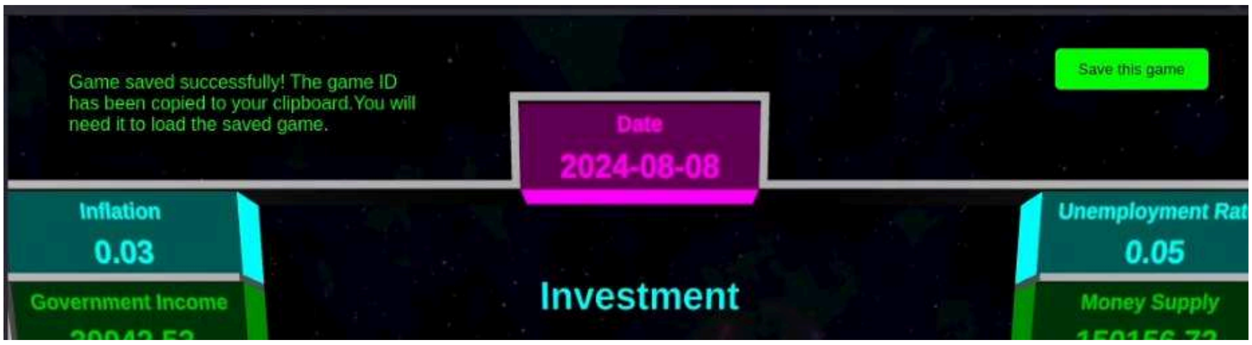
Corporate Income Tax Rate: 15a

Please match the format requested.

When the user sends the policy, or when the game sends one automatically, the LLM is query and responds with a new array of 30 days worth of metrics. This user is notified of this by a notification in the top left corner of the screen.

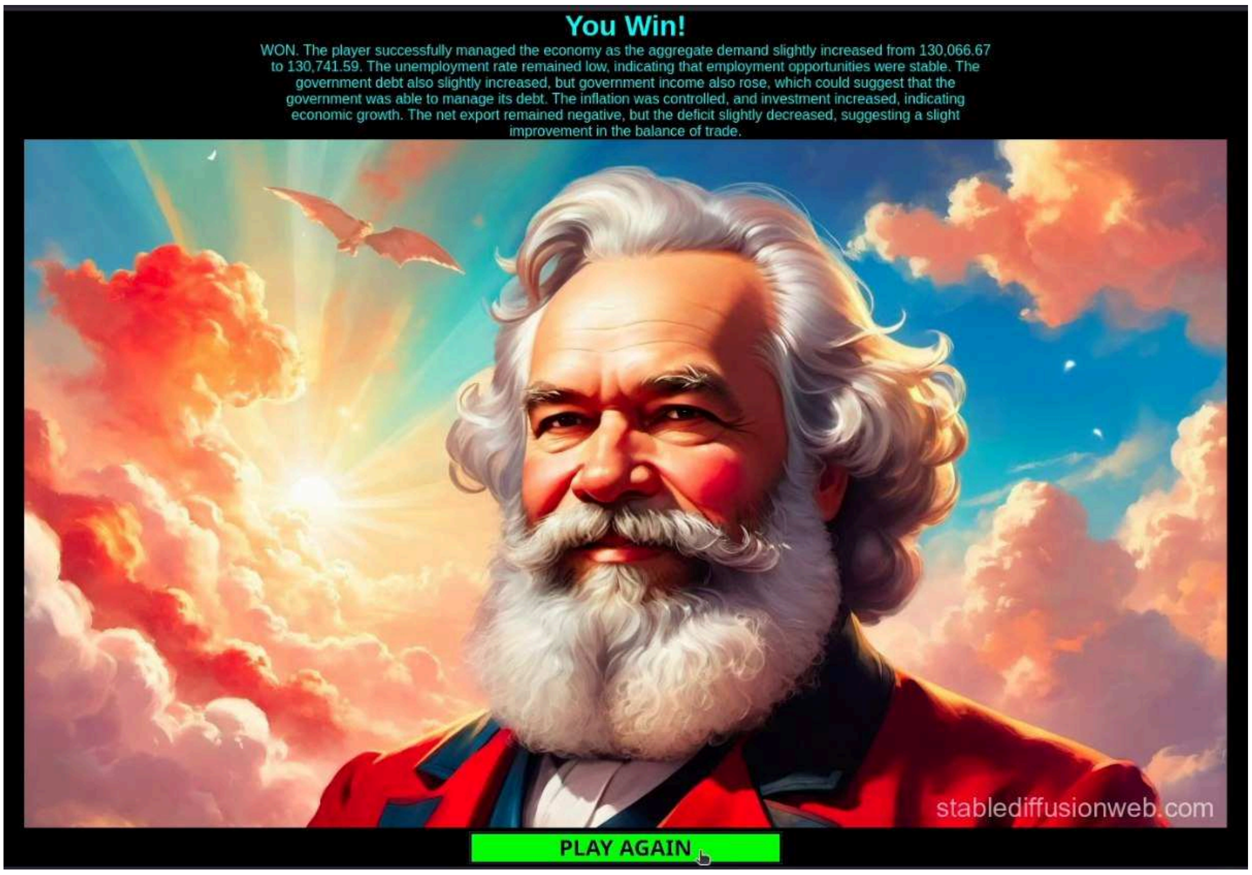

LLM response received.

The screen also provides the possibility to save the game state into the database. When the state is saved, the user is notified.


Game saved successfully! The game ID has been copied to your clipboard. You will need it to load the saved game.

The game runs for a set amount of 6 in-game months. When the system determines that this period is finished, it asks the LLM to generate a result and a verbal evaluation of the user's game. The LLM is provided with two sets of metrics to achieve this, one set from the start of the game and one from the end. When the LLM sends back a response, the user is presented with a screen that shows whether the game was won or lost. The screen also provides the LLM evaluation.

The screen is dominated by a generated image of the given economist. The image reflects the result, i.e. the image for a won game differs from the image of a lost one. The “PLAY AGAIN” button leads to the game selection screen.


You Win!

WON. The player successfully managed the economy as the aggregate demand slightly increased from 130,066.67 to 130,741.59. The unemployment rate remained low, indicating that employment opportunities were stable. The government debt also slightly increased, but government income also rose, which could suggest that the government was able to manage its debt. The inflation was controlled, and investment increased, indicating economic growth. The net export remained negative, but the deficit slightly decreased, suggesting a slight improvement in the balance of trade.

2.4. Backend API and Client Side Processing

The FastAPI application accepts incoming connections from the web frontend, facilitates communication with the LLM and stores and retrieves data from the database. It provides formatting and calculations on the retrieved, user-entered and generated data. It also generates some data itself.

The core functionality of the backend is a HTTP server. It provides several routes that can be accessed using either GET or POST methods.

The main route (“/”) leads to the game selection screen. When a user selects a game, first a GET request navigates the browser to the main page. When the page loads, the loaded JavaScript sends a POST request that starts the initial metrics generation.

At the first time, the API sends a request for two sets of metrics to the LLM - one for now and one for 30 days hence. When the LLM responds, the API parses the response and interpolates metrics for each day between the two endpoints (using simple linear interpolation). This is done in order to limit the cost (both monetary and processing time). The array of 30 metrics is then returned to the frontend process.

The client side code saves the array in an object that keeps the game state. This object is a singleton, and is generated once for each game session.

The main game loop is provided by a function named `tick()` which runs from the game start until the game end.

The game loop calculates the days (a new one every 5 real-time seconds). On each new day the set of metrics from the state object is selected to populate the displays. Independently, the middle chart is rotated, unless frozen by the user.

The game loop and the game state object are used to keep track of how long (in game days) the average communication with the LLM takes, from the request being sent to the receipt of the response. This average time is multiplied by 2 to calculate the size of a buffer of the metric array. Once the buffer is reached (i.e. once there are only so many days worth of previously generated metrics left to be displayed), the game loop triggers an automated request for new metrics with the currently set policy settings. To ensure stability, the minimum days in the buffer is hardcoded to 3.

After the initial requests, all subsequent requests (user triggered or automatic) set the current metrics and the policy settings to the LLM. The LLM is only asked to provide one set of new metrics, an outlook of what it predicts the economy is going to look like in 30 days. The API interpolates the data between the current metrics and the ones generated by the LLM.

The API stores all interaction in the database, in multiple formats - separated into tables (metrics, policy settings), as JSON objects using the Postgresql JSONB format (game state). The communication between the API and the LLM is also captured in a table.

I have exported all the tables after a single whole game, the data (in CSV and SQL formats) is available here: TheGame DB Exports.

The user has the option to save the game. When the button is clicked, several pieces of data from the game state object is exported as JSON and the API stores it in the database. Unlike all the other tables, which are indexed using regular integer primary keys, this table is indexed on UUIDs generated by the API. The UUID is then sent back to the client side, which stores in the user's system clipboard. The user can use the form on the game selection page to load the game and play it from the save point.

When the game loop determines that the game has taken 6 in-game months, it sends two sets of metrics to the API, which passes them to the LLM. The LLM responds with a judgment, as described above. This concludes the game.

2.5. The LLM

The large language model is configured by system prompts (one for each scenario). It is used as the only decision-making engine in the game.

The non-deterministic nature of large language models carries significant risks of miscomunication. In my testing, an unparseable reply from the LLM was the only source of non-coding runtime errors. Utmost care has been taken to remedy this - the API tries various ways to parse the response before throwing an error, and the system prompts have been engineered to ensure maximum message compatibility.

However in some cases the LLM's response is unusable. If this happens at the first load, the user is notified and redirected back to the game selection, where they can start a new game. If this happens on subsequent loads, the user is notified, and they can try re-sending the policy settings.

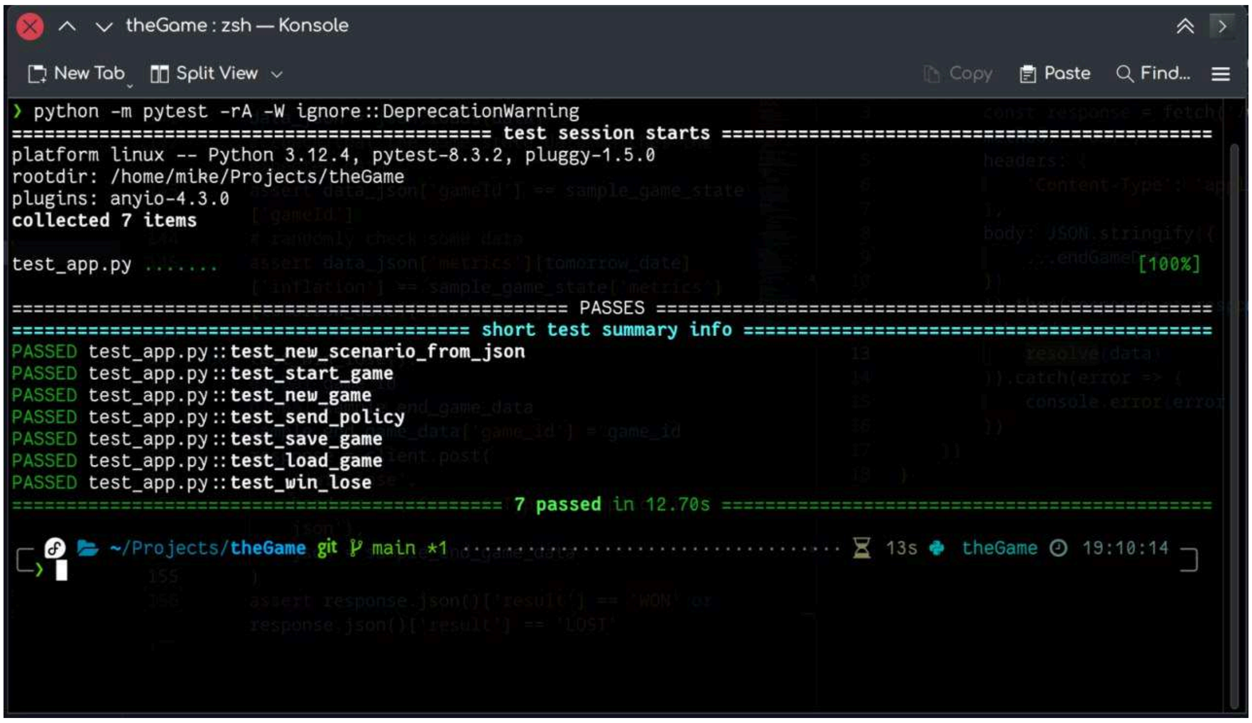
The final response (the "judgment") is not so sensitive to formatting, because it does not contain numbers that need to be precisely processed.

2.6. Testing

To test the GUI and the user facing functionality, I let the game play many times. A video of one such session, is provided here 🎬 2024-08-05 14-11-48.mkv. As shown, the game largely plays itself, if the user does not provide any input. This is useful for the tester, who can check that specific events trigger database writes or notification. The tester can of course also participate in the game whilst testing.

A separate Pytest tests have been run on all the API routes (excluding "/" and "/info", which are just HTML pages and can sufficiently be tested visually).

The tests passed. The code repository has the relevant data and test code for replication purposes.



```
❯ python -m pytest -rA -W ignore::DeprecationWarning
=========================== test session starts ============================
platform linux -- Python 3.12.4, pytest-8.3.2, pluggy-1.5.0
rootdir: /home/mike/Projects/theGame
plugins: anyio-4.3.0
collected 7 items

test_app.py .......                                                  [100%]

================================== PASSES ==================================
========================= short test summary info ==========================
PASSED test_app.py::test_new_scenario_from_json
PASSED test_app.py::test_start_game
PASSED test_app.py::test_new_game
PASSED test_app.py::test_send_policy
PASSED test_app.py::test_save_game
PASSED test_app.py::test_load_game
PASSED test_app.py::test_win_lose
============================ 7 passed in 12.70s ============================

~/Projects/theGame git main *1 ........................ 13s theGame 19:10:14
```

Note, the `ignore::DeprecationWarning` flag in Pytest ensures that the output is not littered by warnings about code that I have not written.

3.0 Conclusions

The project certainly has a potential beyond mere curiosity, in multiple aspects:

- In my opinion, LLMs are well suited for game engines. They provide unpredictability that can be leveraged for engaging gameplay. Whilst this project can not claim to be the most engaging game, it shows the possibilities in this direction. There are 2 major hurdles that any developer must work around: latency and parsing the nondeterministic responses.
- ThreeJS (and the WebGL and WebGPU technologies that it is built on) provide a pleasant and efficient framework for 3D interface

development. What is more, because it is available in the browser, the user does not need to install an application.
- Whilst the previous point is good for the user, it would affect monetization. However, making money off this has never been the goal.
- This project generates interesting data sets that can be used to compare multiple LLMs (the code would have to be expanded for this). I can see something like this project being used for generating all manner of data, to study both human and machine behaviour.

There are several points where I feel the project could be better, even given the time it was developed in:

- Engaging game play
- Design

Those would require me collaborating with people who can think up games and with artists (such collaboration is of course not possible on a graded project).

4.0 Further Development or Research

I will keep on further exploring both 3D web development and interactions with LLM, as the subjects fascinate me.

5.0 Bibliography

- threejs.org. (n.d.). three.js – JavaScript 3D library. [online] Available at: https://threejs.org.
- fastapi.tiangolo.com. (n.d.). FastAPI. [online] Available at: https://fastapi.tiangolo.com.
- www.postgresql.org. (n.d.). PostgreSQL: The world's most advanced open source database. [online] Available at: https://www.postgresql.org.
- docs.mistral.ai. (n.d.). Models | Mistral AI Large Language Models. [online] Available at: https://docs.mistral.ai/getting-started/models/.
- Blender Foundation (2019). blender.org - Home of the Blender project - Free and Open 3D Creation Software. [online] blender.org. Available at: https://www.blender.org.
- stablediffusionweb.com. (n.d.). Stable Diffusion Online. [online] Available at:

https://stablediffusionweb.com.

- stablediffusionweb.com. (n.d., 2). Stable Diffusion Online. [online] Available at: https://stablediffusionweb.com/#faq.
- TonyS (2024). Blue Nebulae Spheremaps - Space Spheremaps. [online] Space Spheremaps. Available at: https://www.spacespheremaps.com/blue-nebulae-spheremaps/ [Accessed 5 Aug. 2024].

6.0 Appendices

National College of Ireland

Computing Project (BSCCYBE4)

Project Proposal

BSc (Honours) in Computing - Evening
2023/2024
Michal Eichler
x20139462
x20139462@student.ncirl.ie

Table of Contents

1.0 Objectives

This projects has two objectives:

1. Educational, learning how to develop a web game, using state of the art technologies and learning the basics of economics.
2. Effective, to produce a game that aims to educate its players about mainstream economic theories in a highly engaging and graphically submersive environment. The game is to be free of charge, available over the internet and playable in a browser. The source code is to be released under a permissive open source license (after the project is graded).

2.0 Background

I chose to undertake this project because I want to both learn how to make a web game and gain basic understanding of economic theories.
My motivation to make a game is to make something which is simultaneously demanding and not serious. I have made plenty of data processing Python scripts, but a game is new and it should be fun to make (as well as to play).
My motivation to make an educational game in the branch of economics is that I want to understand economics. I want to gain some insight into how economists think. Economics is the one big part of our society that I understand only very little.
My motivation to make an online game that's free of charge, is so that everyone can play it. I have a preference for unusual hardware and software combinations, ones that games vendors usually don't support. Everyone with a computer, or even a smartphone has a web browser. I will prioritize efficiency, so as to minimize hardware requirements on behalf of the user.
My motivation behind the open source licensing is dual: for one, I am an open source enthusiast and I believe that all source code should be open so that we all can learn from it and built upon it. For two, given that there should be no fee to pay, I won't be able to afford to scale up the infrastructure in case the game will achieve any degree of popularity. Making it open source will enable everyone to host their own instance.
And finally, my motivation to make the game graphically submersive is so that it's fun to play. The danger of educational games is that they prioritize the education over the gaming (see any gamified workplace security quiz or tutorial.) I would like to avoid that danger by making a game that won't feel like a chore.
The above points are inline with the objectives from section 1.0, and I will fulfil those objectives by following my motivation.

3.0 State of the Art

There are educational games on the web, like Vecon Labs [1] or Economics-games.com [2]. Those are usually not what I would call graphically engaging, some resemble actual spreadsheets more than anything else.
Then there are games that have an economy as part of their gameplay, usually strategy games like Anno 1800 [3]. In those, the economy is just one of the parts of the game (others being warfare, diplomacy, etc). Most of these games, definitely the most known examples, are expensive to buy, and require a programmed installed on a specific platform and often come with high hardware requirements.
The unique point of my project is that it will feel like a game, and it will be fun and engaging.

4.0 Technical Approach

4.1 Project Management

The principle place for project management will be GitHub Projects. This provides a Kanban board, an issue tracker, Gantt timeline view and a space for documentation. I will use elements of the agile methodology, namely incremental development in short cycles, flexibility and kanban. Some parts of agile don't apply here, since I am not working on this with anyone else.
The flexibility and short iterations are important to me, because I am not at all familiar with the technology that I'm going to use and I might discover that I need to adjust the requirements or move parts of the timeline around.
Kanban is just what I'm used to for task management.

4.2 Identifying requirements

I am the only real stake holder in this project, or more to the point, I assume that I can represent the intended audience (by virtue of intending the audience to be a lot like me). Therefore I can base the requirements on the motivation and on the capabilities of the technology that I'm going to use.
Provisionally, many functional requirements are related to the gaming paradigm, eg. keeping score, saving state, randomized game bahaviour, difficulty levels, scenario selection, options setting etc. Another set of functional requirements will be derived from the economics part of the game, choosing the theory for the game environment (eg. play as a gold trader in a fully Keynesian world).
Non functional requirements will be driven by my desire to make the game run as well as possible with the lowest possible system requirements.
In order to finalize the requirements, I need the domain knowledge, both in terms of technology and economics. To that effect, I'm planning on educating myself first. I have found several materials to utilize:

- Master Java Web Services and REST API with Spring Boot [4], which will help me with the backend, which is to be built using Spring Boot
- Mathematics for Game Programming and Computer Graphics [5], as an introduction the the underlying mathematics
- Build Your Own 2D Game Engine and Create Great Web Games: Using HTML5, JavaScript, and WebGL2 [6], to for the frontend
- A Little History of Economics [7], an overview of economic theories
  Once I have the theoretical background, I will be able to establish a clear picture of what the technologies allow me to do. Then I will be able to think about how to express the theories I've learned about as a game.
  After this, I will be able to fully establish the functional and non-functional requirements, use cases

and testing procedures. I will first plan everything in a Gantt chart, and then translate it into issues, milestones tickets, etc.

5.0 Technical Details

There will be 3 components to the deployed application: frontend, server and database. The application is going to be containerized using Podman or Incus containers and running behind a reverse proxy (possibly Caddy).

5.1 Frontend

The principal technology here will be WebGL [8]. I would like to be on the cutting edge and use WebGPU [9], sadly there is very little adoption [10] yet and using such an unavailable technology would go against my principle of availability.
WebGL is a low level graphics API, good for making games. I'm not sure yet which specific algorithms or libraries I will use, as I have to do the above mentioned research first.
For the rest of the frontend (settings, account, leaderboards, about page, etc) I'm considering Thymeleaf [11], because it works well with Spring.

5.2 Backend

I'll be using Spring Boot [12], a Java-based framework for the backend. This is a mature and well tested framework that I hope will provide me with reliability and security. I'm going to use the Model-Viewer-Controller pattern using the Spring MVC module.

5.3 Database

The database used will be Postgresql [13], chosen because it's mature with a reputation for stability and a wealth of documentation that I can draw on.

5.4 Container and Reverse proxy

I will use Podman [14] or Incus [15] containers for the application to run in. These technologies are quite different, Podman is close to Docker whilst Incus is an application container (eg, well suited for running both the Spring Boot app and the Postgresql database in a single container). I have some experience with both and will decide after further research, which one of these technologies is better suited for the project.

6.0 Special Resources Required

Except for the books mentioned above and other documentation, and my GitHub account, I will be using my home lab for development and early testing.
For further testing and actual deployment, I will be using a Hetzner VPS [16] running either a Debian or a Ubuntu server.
A generative AI service will be used to create assets, both graphics and audio.

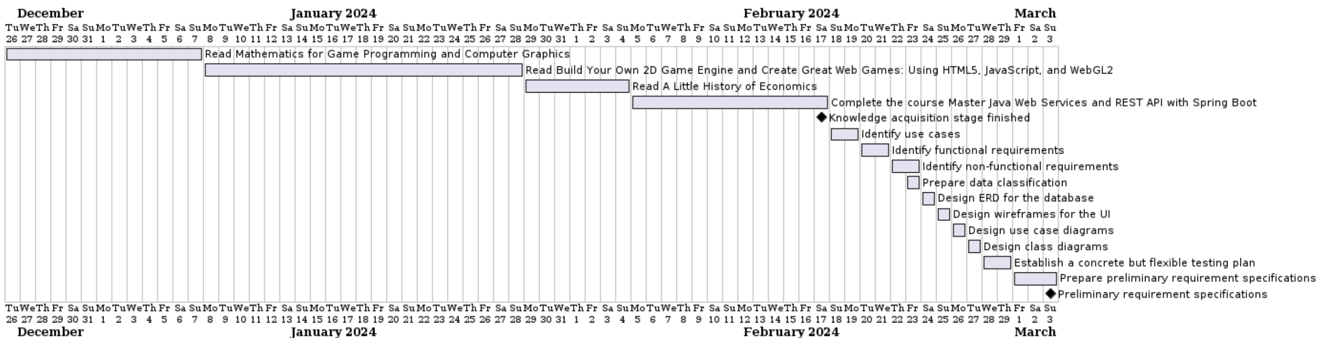
7.0 Project Plan

The project has several distinct stages, marked by the milestones on the GitHub project page:

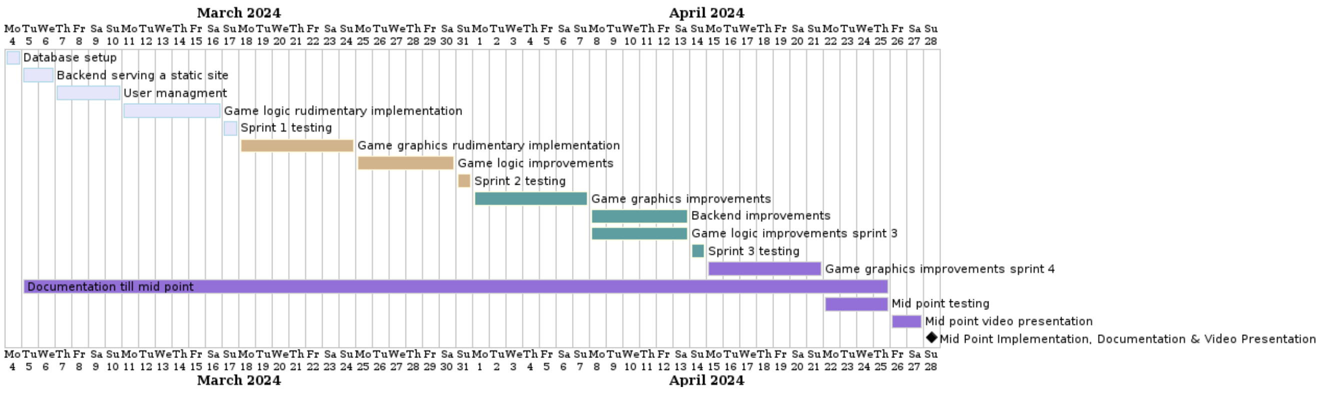
1. Knowledge acquisition, runs from 2023-12-26 to 2024-02-17
2. Preliminary requirement specification, runs from 2024-02-18 to 2024-03-03
3. Mid point implementation, runs from 2024-03-04 to 2024-04-28
4. Final implementation, runs from 2024-04-29 to 2024-08-05
   The first stage will consist mostly of reading the above mentioned books and completing the above mentioned course. I might do some exploratory coding, but won't attempt to seriously start putting the project together yet. The information materials are sorted from the most demanding to the least, if I get slowed down by vector maths, I can speedup through the less challenging subject of Spring Boot development.
   During the second stage will create the necessary planning materials for the development, eg ERD, use cases, etc.
   The third stage will start the actual development. From now on, each 2 week period marks a sprint, during which a distinct part of the work will be done. At the end of stage 3, I expect to have all parts of the application in a running if rudimentary state, with features missing, but a usable prototype.
   The fourth stage will continue with iterative development. I expect to finish all development by the end of sprint 10 (2024-06-21). The last week before the project submission is a buffer week, reserved for emergencies.
   Documentation is to be continuously updated during stages 3 and 4.
   The plan for those stages is still rather vague, I need to first acquire the knowledge (stage 1) so that I can design and plan the app (stage 2), before I can formalize the needed tickets.

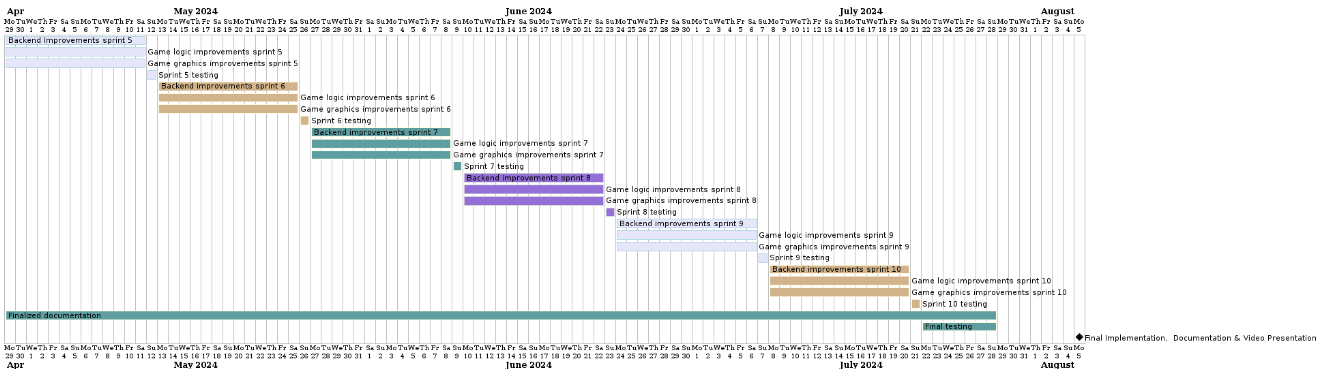
7.1 Gantt Charts

The project plan is captured as a Gantt chart, kanban and issue list in the GitHub repository, which is private. Please let me know if you require access.



Sprints are identified by colours





7.2 Task List

#	Title	Start Date	Due Date	Milestone	Sprint
1	Read Mathematics for Game Programming and Computer Graphics	2024-01-01	2024-01-07	Knowledge acquisition stage finished	
2	Read Build Your Own 2D Game Engine and Create Great Web Games: Using HTML5, JavaScript, and WebGL2	2024-01-08	2024-01-28	Knowledge acquisition stage finished	
3	Read A Little History of Economics	2024-01-29	2024-02-04	Knowledge acquisition stage finished	
4	Complete the course Master Java Web Services and REST API with	2024-02-05	2024-02-17	Knowledge acquisition stage finished	

#	Title	Start Date	Due Date	Milestone	Sprint
	Spring Boot				
5	Identifiy use cases	2024-02-18	2024-02-19	Preliminary requirement specifications	
6	Identify functional requirements	2024-02-20	2024-02-21	Preliminary requirement specifications	
7	Identify non-functional requirements	2024-02-22	2024-02-23	Preliminary requirement specifications	
8	Prepare data classification	2024-02-23	2024-02-23	Preliminary requirement specifications	
9	Design ERD for the database	2024-02-24	2024-02-24	Preliminary requirement specifications	
10	Design wireframes for the UI	2024-02-25	2024-02-25	Preliminary requirement specifications	
11	Design use case diagrams	2024-02-26	2024-02-26	Preliminary requirement specifications	
12	Design class diagrams	2024-02-27	2024-02-27	Preliminary requirement specifications	
13	Establish a concrete but flexible testing plan	2024-02-28	2024-02-29	Preliminary requirement specifications	
14	Prepare preliminary requirement specifications	2024-03-01	2024-03-03	Preliminary requirement specifications	
15	Database setup	2024-03-04	2024-03-04	Mid Point Implementation, Documentation & Video Presentation	1
16	Backend serving a static site	2024-03-05	2024-03-06	Mid Point Implementation, Documentation & Video Presentation	1
17	User managment	2024-03-07	2024-03-10	Mid Point Implementation, Documentation & Video Presentation	1
18	Game logic rudimentary implementation	2024-03-11	2024-03-16	Mid Point Implementation, Documentation & Video Presentation	1
19	Sprint 1 testing	2024-03-17	2024-03-17	Mid Point Implementation, Documentation & Video Presentation	1
20	Game graphics rudimentary implementation	2024-03-18	2024-03-24	Mid Point Implementation, Documentation & Video Presentation	2
21	Game logic improvements	2024-03-25	2024-03-30	Mid Point Implementation, Documentation & Video Presentation	2
22	Sprint 2 testing	2024-03-31	2024-03-31	Mid Point Implementation, Documentation & Video Presentation	2

#	Title	Start Date	Due Date	Milestone	Sprint
23	Game graphics improvements	2024-04-01	2024-04-07	Mid Point Implementation, Documentation & Video Presentation	3
24	Backend improvements	2024-04-08	2024-04-13	Mid Point Implementation, Documentation & Video Presentation	3
25	Game logic improvements sprint 3	2024-04-08	2024-04-13	Mid Point Implementation, Documentation & Video Presentation	3
26	Sprint 3 testing	2024-04-14	2024-04-14	Mid Point Implementation, Documentation & Video Presentation	3
27	Game graphics improvements sprint 4	2024-04-15	2024-04-21	Mid Point Implementation, Documentation & Video Presentation	4
28	Documentation till mid point	2024-03-05	2024-04-25	Mid Point Implementation, Documentation & Video Presentation	4
29	Mid point testing	2024-04-22	2024-04-25	Mid Point Implementation, Documentation & Video Presentation	4
30	Mid point video presentation	2024-04-26	2024-04-27	Mid Point Implementation, Documentation & Video Presentation	4
31	Backend Improvements sprint 5	2024-04-29	2024-05-11	Final Implementation, Documentation & Video Presentation	5
32	Game logic improvements sprint 5	2024-04-29	2024-05-11	Final Implementation, Documentation & Video Presentation	5
33	Game graphics improvements sprint 5	2024-04-29	2024-05-11	Final Implementation, Documentation & Video Presentation	5
34	Sprint 5 testing	2024-05-12	2024-05-12	Final Implementation, Documentation & Video Presentation	5
35	Backend improvements sprint 6	2024-05-13	2024-05-25	Final Implementation, Documentation & Video Presentation	6
36	Game logic improvements sprint 6	2024-05-13	2024-05-25	Final Implementation, Documentation & Video Presentation	6
37	Game graphics improvements sprint 6	2024-05-13	2024-05-25	Final Implementation, Documentation & Video Presentation	6
38	Sprint 6 testing	2024-05-26	2024-05-26	Final Implementation, Documentation & Video	6

#	Title	Start Date	Due Date	Milestone	Sprint
				Presentation	
39	Backend improvements sprint 7	2024-05-27	2024-06-08	Final Implementation, Documentation & Video Presentation	7
40	Game logic improvements sprint 7	2024-05-27	2024-06-08	Final Implementation, Documentation & Video Presentation	7
41	Game graphics improvements sprint 7	2024-05-27	2024-06-08	Final Implementation, Documentation & Video Presentation	7
42	Sprint 7 testing	2024-06-09	2024-06-09	Final Implementation, Documentation & Video Presentation	7
43	Backend improvements sprint 8	2024-06-10	2024-06-22	Final Implementation, Documentation & Video Presentation	8
44	Game logic improvements sprint 8	2024-06-10	2024-06-22	Final Implementation, Documentation & Video Presentation	8
45	Game graphics improvements sprint 8	2024-06-10	2024-06-22	Final Implementation, Documentation & Video Presentation	8
46	Sprint 8 testing	2024-06-23	2024-06-23	Final Implementation, Documentation & Video Presentation	8
47	Backend improvements sprint 9	2024-06-24	2024-07-06	Final Implementation, Documentation & Video Presentation	9
48	Game logic improvements sprint 9	2024-06-24	2024-07-06	Final Implementation, Documentation & Video Presentation	9
49	Game graphics improvements sprint 9	2024-06-24	2024-07-06	Final Implementation, Documentation & Video Presentation	9
50	Sprint 9 testing	2024-07-07	2024-07-07	Final Implementation, Documentation & Video Presentation	9
51	Backend improvements sprint 10	2024-07-08	2024-07-20	Final Implementation, Documentation & Video Presentation	10
52	Game logic improvements sprint 10	2024-07-08	2024-07-20	Final Implementation, Documentation & Video Presentation	10
53	Game graphics improvements sprint 10	2024-07-08	2024-07-20	Final Implementation, Documentation & Video Presentation	10

#	Title	Start Date	Due Date	Milestone	Sprint
54	Sprint 10 testing	2024-07-21	2024-07-21	Final Implementation, Documentation & Video Presentation	10
55	Finalized documentation	2024-04-29	2024-07-28	Final Implementation, Documentation & Video Presentation	11
56	Final testing	2024-07-22	2024-07-28	Final Implementation, Documentation & Video Presentation	11

8.0 Testing

At the start of each sprint, I will come up with relevant test cases, which are meant to be passing at the end of the iteration. Most sprints have the last day reserved for testing (this is always a Sunday, so a day is enough).
This includes unit testing for backend and non-WebGL parts of the fronted using the testing framework built into Sprint Boot [17], and for the WebGL part I plan on using GLCheck [18]. I hope that this emulation of Test-Driven Development [19] will enable me catch and solve bugs early.
Each sprint will also conclude with integration tests of what's been done, using the Spring built in test framework, and Selenium [20]. I'm choosing Selenium over say Puppeteer, because it can use Firefox as well as Chrome, again in accord with my principle of access.
Non-functional tests are going to be performed using JMeter [21] for the Java part, and browser developer tools for the frontend part. Security testing will be done using ZAP proxy [22], also at the end of each sprint.
The last sprint before the midpoint and the final half sprint before the buffer week will be dedicated to testing. All tests will be repeated multiple times to ensure the quality of the finalized project.

References

[1] “Attacker Defender Game Features,” veconlab.econ.virginia.edu. https://veconlab.econ.virginia.edu/ad2/ad2.php (accessed Dec. 18, 2023).
[2] “Classroom Games for Teaching Economics,” economics-games.com. https://economics-games.com (accessed Dec. 18, 2023).
[3] “Anno 1800 on Steam,” store.steampowered.com. https://store.steampowered.com/app/916440/Anno_1800/ (accessed Dec. 18, 2023).
[4] “Master Java Web Services and REST API with Spring Boot,” learning.oreilly.com. https://learning.oreilly.com/course/master-java-web/9781789130133/ (accessed Dec. 18, 2023).
[5] P. de Byl, Mathematics for Game Programming and Computer Graphics. Birmingham: Packt Publishing, 2022. Accessed: Dec. 18, 2023. [Online]. Available: https://learning.oreilly.com/library/view/mathematics-for-game/9781801077330/
[6] K. Sung, J. Pavleas, M. Munson, and J. Pace, Build Your Own 2D Game Engine and Create Great Web Games: Using HTML5, JavaScript, and WebGL2. New York: Apress, 2021. Accessed: Dec. 18, 2023. [Online]. Available: https://learning.oreilly.com/library/view/build-your-own/9781484273777/
[7] N. Kishtainy, Tran., A Little History of Economics. New Haven: Yale University Press, 2018. Available: https://www.amazon.co.uk/Little-History-Economics-Histories-ebook/dp/B06XDHTFTL
[8] “WebGL - OpenGL ES for the Web,” The Khronos Group, Jul. 19, 2011. https://www.khronos.org/webgl/
[9] “WebGPU,” www.w3.org. https://www.w3.org/TR/webgpu/
[10] “‘webgpu’ | Can I use... Support tables for HTML5, CSS3, etc,” caniuse.com. https://caniuse.com/?search=webgpu (accessed Dec. 18, 2023).
[11] “Thymeleaf,” www.thymeleaf.org. https://www.thymeleaf.org/
[12] Spring, “Spring Projects,” Spring.io, 2019. https://spring.io/projects/spring-boot
[13] The PostgreSQL Global Development Group, “PostgreSQL: The world’s most advanced open source database,” Postgresql.org, 2019. https://www.postgresql.org/
[14] “Podman,” podman.io. https://podman.io/
[15] “Linux Containers - Incus - Introduction,” linuxcontainers.org. https://linuxcontainers.org/incus/introduction/ (accessed Dec. 18, 2023).
[16] “Dedicated Server, Cloud, Storage & Hosting,” www.hetzner.com. https://www.hetzner.com/?country=ie (accessed Dec. 18, 2023).
[17] “41. Testing,” docs.spring.io. https://docs.spring.io/spring-boot/docs/1.5.7.RELEASE/reference/html/boot-features-testing.html
[18] “glcheck,” npm, Dec. 02, 2019. https://www.npmjs.com/package/glcheck (accessed Dec. 19, 2023).
[19] J. Acosta and K. Gajda, “https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/,” www.ibm.com. https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/
[20] Selenium, “SeleniumHQ Browser Automation,” www.selenium.dev, 2023. https://www.selenium.dev/
[21] Apache Software Foundation, “Apache JMeter - Apache JMeterTM,” Apache.org, 2019. https://jmeter.apache.org/
[22] OWASP, “OWASP ZAP,” Zaproxy.org, 2020. https://www.zaproxy.org/