Advanced Tech Report on Simulation Based Game

Alessandro Bufalino
19017120

University of the West of England

May 22, 2023

This report describes the development of an autonomous civilization simulation game featuring a procedurally generated map, resource gathering, AI agents, and a day cycle system. Algorithms such as Perlin noise, Voronoi, and cellular automata are employed to create a dynamic and engaging gameplay experience. Additionally, the game incorporates a graphing system that enables players to visualize data relationships.

1 Introduction

Simulation games provide an immersive experience for players to observe complex system growth, such as cities and civilizations. This report presents a novel simulation game built on Unity (Unity Technologies, 2023) that uses algorithms such as Perlin noise (Perlin, 1985) for the map generation and cellular automata for natural vegetation growth (Conway, 1970). The game also features a context-aware decision-making system and a graphing system for data visualization.

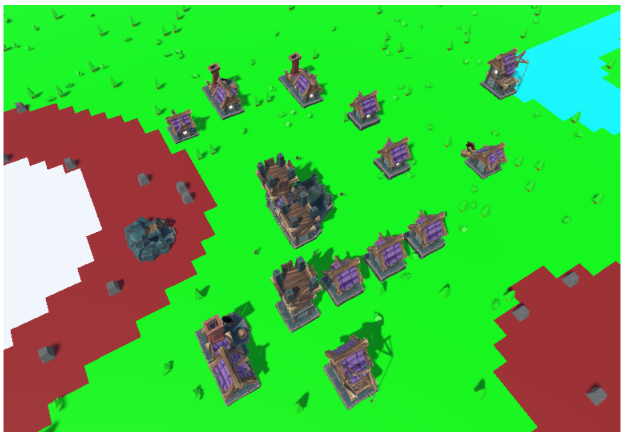


Figure 1: *Screenshot taken of the civilization from the project.*

2 Related Work

In this section, we review the existing literature and projects related to the simulation of the evolution and growth of civilizations, focusing on decision-making by artificial intelligence agents using the current state of the game as context.

AI decision-making has been a crucial component of strategy games, where AI systems often manage resources, build structures, and control units. Games like Civilization (Firaxis Games, 2016) and Age of Empires (Ensemble Studios, 1997) feature AI opponents that make decisions based on the current state of the game and, most importantly, the player's current state or actions. The AI needs to dynamically adjust its difficulty level, especially at lower levels, to avoid overwhelming the player.

There are different types of implementation to use to get the AI to make decisions based on the current context. Long (2007) explores a couple of these approaches:

- Hierarchical Task Network (HTN): An AI technique that breaks down complex tasks into smaller, manageable subtasks, creating a hierarchy that simplifies decision-making and planning for agents in a game environment. Notably used in the game F.E.A.R (Monolith Productions, 2005).
- Goal-Oriented Action Planning (GOAP): An AI technique that allows agents to make context-sensitive decisions by choosing a series of actions to achieve a specific goal. Middle-earth: Shadow of Mordor (Monolith Productions, 2014) uses this method to create dynamic and adaptive enemy behaviours.

Fairclough et al. (2001) highlights AI techniques in game genres like strategy games. They note two AI applications: strategic and individual. Strategic AI is harder due to player adaptability, while individual AI enables units to follow orders. The challenge lies in AI acting independently, such as when a unit is attacked.

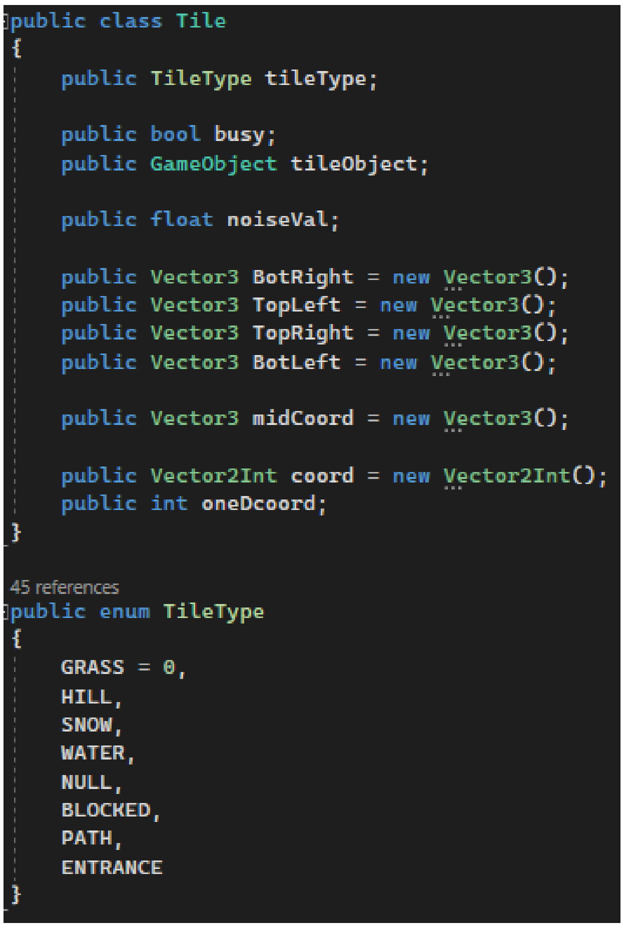
3 Method

3.1 map generation

3.1.1 Tile class

In this project the world where the player plays in is fully procedural generated and to achieve this firstly a Tile class is needed to store the data of the map. The class contains the following data (see Figure: 2):

- tileType: An enum type that describes the current state of the specific tile. The state dictates if things can grow on this tile and what can grow on it or if the tile has currently a building on top, meaning new buildings can't be placed there.
- busy and tileObject: Saves the resource that is available in that tile, so the agent can access it when looking for resources.
- The next four Vectors dictate the place of the tile in respect to their world location, this is used when the user clicks so the specific tile clicked on can be fetched.
- The 'coord' and 'oneDCoord' variables represent the 2D coordinate of a tile in a list of array and the 1D index of the same tile in a different array used for data gathering, respectively.



```
public class Tile
{
    public TileType tileType;

    public bool busy;
    public GameObject tileObject;

    public float noiseVal;

    public Vector3 BotRight = new Vector3();
    public Vector3 TopLeft = new Vector3();
    public Vector3 TopRight = new Vector3();
    public Vector3 BotLeft = new Vector3();

    public Vector3 midCoord = new Vector3();

    public Vector2Int coord = new Vector2Int();
    public int oneDcoord;
}

45 references
public enum TileType
{
    GRASS = 0,
    HILL,
    SNOW,
    WATER,
    NULL,
    BLOCKED,
    PATH,
    ENTRANCE
}
```

Figure 2: *Showing the variables held by each tile in the 2D grid that makes up the map.*

3.1.2 Perlin Noise

Perlin noise is used to generate believable, random maps by assigning weight values to each tile. Its natural appearance leads to terrain features resembling real landscapes. By setting weight thresholds, different terrain tile types are created, producing realistic terrains (see Figure 3).

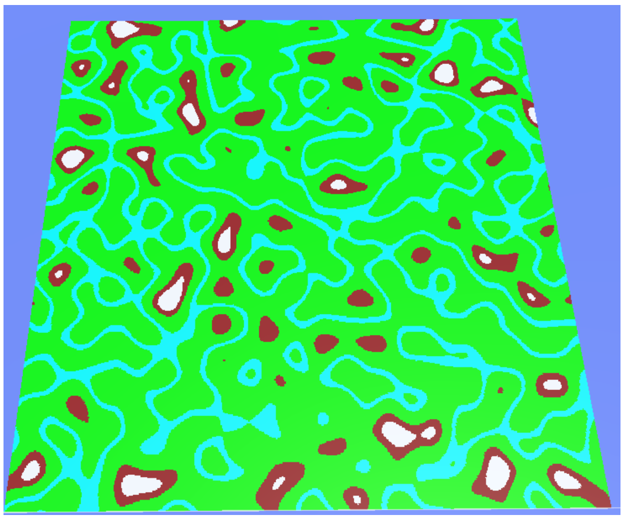


Figure 3: *Showing the Perlin noise generation with the tiles set for the map.*

3.2 Entities

3.2.1 Data

In the game, all entities inherit from an abstract class called ‘Entity‘. This class contains a string called ‘GUID‘, which stands for Globally Unique Identifier, a 128-bit value used to uniquely identify objects. Depending on the type of entity, either a building or an NPC, each entity is stored in a dictionary where the key is the GUID and the value is the class holding the entity’s data. This approach allows for a fast lookup table for each entity’s data.

```
public abstract class Entity
{
    public string guid;
    public Entity()
    {
        guid = System.Guid.NewGuid()
    .ToString();
    }
}
```

3.2.2 Buildings

To improve efficiency and expandability, the project uses Unity’s scriptable object data type to store building statistics (Figure 4). This allows for a lightweight, reusable, and easily configurable data assets. The stored data include:

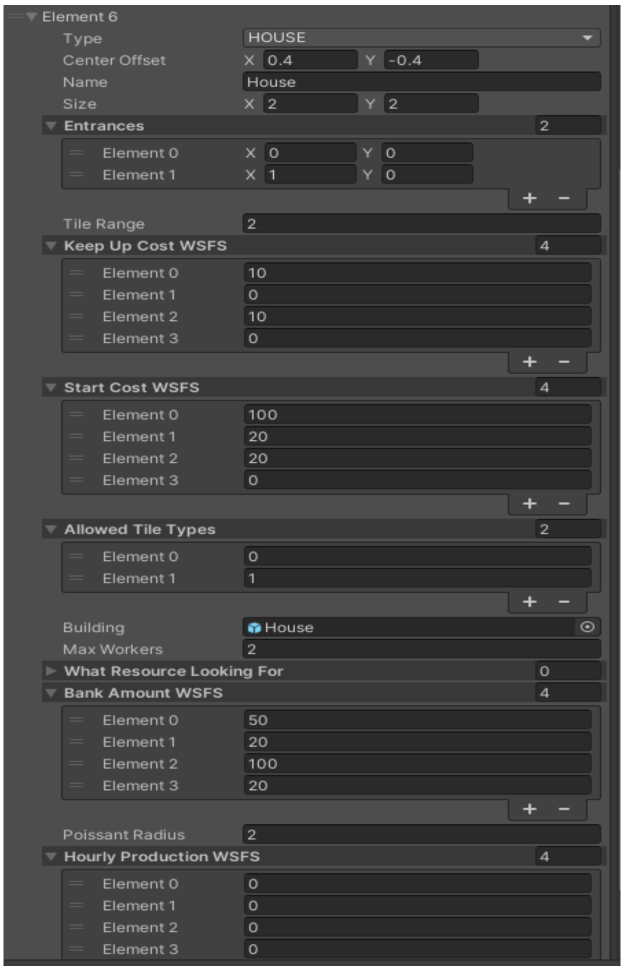
- Type: Enum denoting building type for UI.
- Center Offset: Offset for centering building on tile.
- Name: Building name.
- Size: Number of occupied tiles.
- Entrances: Tiles tagged as entrances for NPCs.
- Tile Range: Resource detection range.
- Keep Up costs WSFS: The number of resources needed per turn to keep the building healthy, the WSFS Denotes:

0	Wood
1	Stone
2	Food
3	Sand

- Start Cost WSFS: the needed cost to place such building.
- Allowed Types: The allowed types of tiles this building can be placed in, for example, 0 and 1 would equal to this building being only able to be placed in the grass or hill-type tile
- Building: this is the prefab object of the building.
- Max workers: The maximum amount of people that can be in that building
- What Resource Looking For: If this building were a buildings that needed to send the agent to forage for resources this array containing the integer that would be converted to the enum type of the necessary resource type to look for.

0	Stone
1	Food
2	Wood

- Bank Amount: The amount of each of the resources this building is going to add onto the communal maximum resource holding.
- Poissant Radius: Used for the AI to know what places are safe to generate another building.
- Hourly Production WSFS: amount of each resource generated per hour.

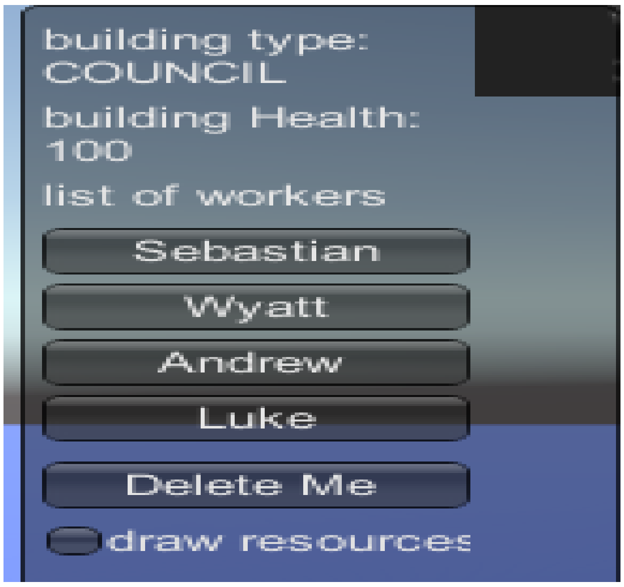


```
Element 6
  Type                     HOUSE
  Center Offset            X 0.4    Y -0.4
  Name                     House
  Size                     X 2      Y 2
  Entrances                                  2
    Element 0              X 0      Y 0
    Element 1              X 1      Y 0
                                               + -
  Tile Range               2
  Keep Up Cost WSFS                          4
    Element 0              10
    Element 1              0
    Element 2              10
    Element 3              0
                                               + -
  Start Cost WSFS                            4
    Element 0              100
    Element 1              20
    Element 2              20
    Element 3              0
                                               + -
  Allowed Tile Types                         2
    Element 0              0
    Element 1              1
                                               + -
  Building                 House
  Max Workers              2
  What Resource Looking For                  0
  Bank Amount WSFS                           4
    Element 0              50
    Element 1              20
    Element 2              100
    Element 3              20
                                               + -
  Poissant Radius          2
  Hourly Production WSFS                     4
    Element 0              0
    Element 1              0
    Element 2              0
    Element 3              0
```

Figure 4: *Showing the data inside the scriptable object for the house building.*

3.2.3 NPCs/Agents

Unlike the buildings, the NPCs have unique attributes apart from the GUID, such as their names, which help identify them on the map. For instance, when a player clicks on a building to see the workers inside, with their names displayed (see Figure 5).



building type: COUNCIL
building Health: 100
list of workers

Figure 5: *Showing the UI pop-up when clicking on a building and showing its workers.*

Another distinct feature of the NPC object is its ability to communicate with the agent class. The agent class serves as a visual representation of the NPC in the game world (see Figure 6), appearing when the NPC is given a destination to travel to. When the NPC is inside a building, the agent object is destroyed.

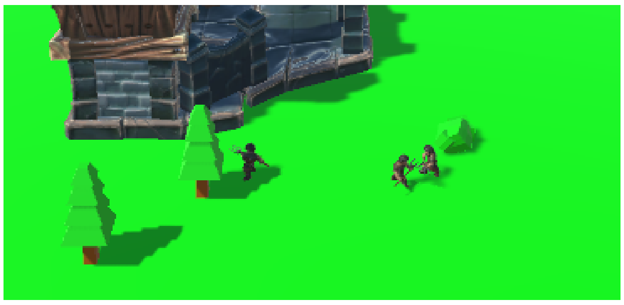


Figure 6: *Showing agents moving around the map.*

The destinations are typically provided by the buildings, which act as states. Each building object contains a prefab, and within that prefab, there is a unique class for each building type that defines how the agent should behave. In this sense, the agent exhibits finite state machine-like behaviour, with the states being the buildings themselves. This design approach allows for greater flexibility when adding new building types or modifying existing ones.

Additionally, the agents use A* pathfinding (Hart, Nilsson, and Raphael, 1968) to navigate the tile-based map efficiently. This algorithm ensures that they can effectively traverse the world, even as the map layout changes from new buildings being placed, or new obstacles are encountered.

3.3 Simulation

At the end of each day, the game manager decides which building to construct next on the map to facilitate the growth of the civilization. However, due to the maintenance cost of each building, the game will only proceed with construction when there is a genuine need for new buildings to avoid wasting resources.

The user has access to a menu that allows them to change the Game Manager's prioritization if they wish, as shown in Figure 7.

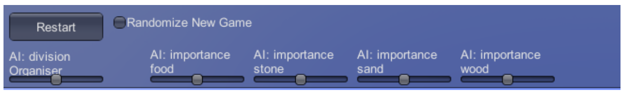


Figure 7: *Shows the menu available to the player where they can change values to have the AI focus on specific resources.*

If the game detects that there will be unemployed NPCs in the next turn, a two-step function is executed:

1 Resource Importance: A formula determines the importance of a resource every turn, which can be broken down into three parts. The lower the importance number, the higher the need for that resource.
The first part calculates the percentage difference from the maximum allowed amount for a resource. In theory, a resource with a lower number should be flagged as being in higher demand.

currentAmountOfResource / maximum-BankAmount

The second part considers the spending amount of each resource and estimates how long the resource will last if the player continues to spend at the same rate.

turnResourceSpending / currentAmountOfResource

Finally, the modifier stage adjusts resource importance based on the values set by the player in the menu (see Figure 7). Certain resources may be multiplied, increasing their importance number and making them less crucial. Alternatively, the user can instruct the Game Manager to focus primarily on one aspect of the equation by using the division organizer slider, also found in the menu (see Figure 8).

2 Once the appropriate resource is chosen, the correct building needs to be placed to continue producing the resource in demand. To ensure proper placement without overlapping other buildings and leaving enough space for agents to roam, a variation of the Poisson-disc sampling algorithm (Bridson, 2007) with varying radii is used. The algorithm first considers all currently placed buildings and then attempts to place new buildings without overlapping the radii of each building, which is stored in the data within the scriptable object section (see Figure 9).

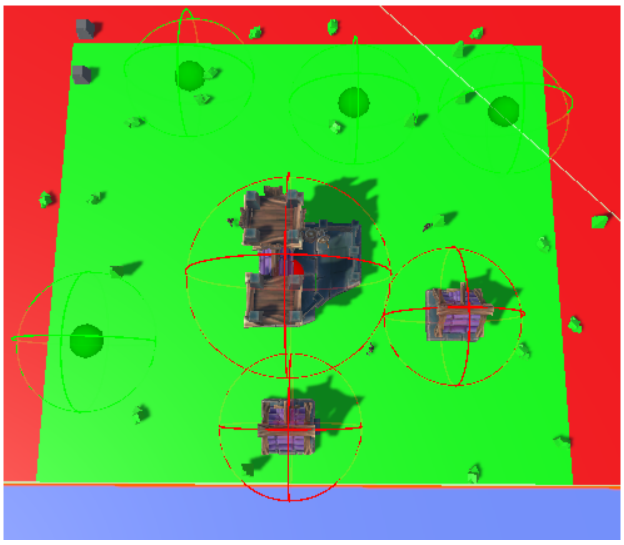


Figure 9: *Showing possible spots to build new buildings in green.*

3.4 Graphing

The game features a graph tool that shows correlations between two data points. This allows players to observe relationships between aspects of the city, such as NPC population growth and food expenses. The tool provides insights for informed decision-making to ensure sustainable civilization development (see Figure: 10).

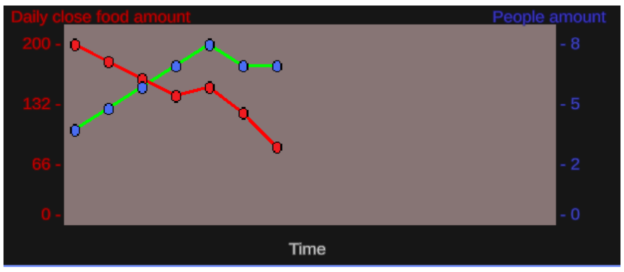
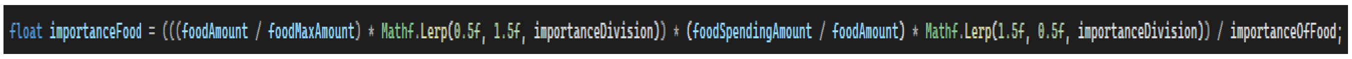


Figure 10: *Showing the graph available to the player.*



```
float importanceFood = (((foodAmount / foodMaxAmount) * Mathf.Lerp(0.5f, 1.5f, importanceDivision)) * (foodSpendingAmount / foodAmount) * Mathf.Lerp(1.5f, 0.5f, importanceDivision)) / importanceOfFood;
```

Figure 8: *Showing the whole formula in code.*

4 Evaluation

4.1 AI Agent Behaviour and Decision-making

The finite state machine-like behaviour allows for a wide range of agent actions, leading to diverse game-play experiences. However, further improvements could be made to enhance the game-manager AI decision-making and adaptability in response to changing game conditions and context.

4.2 Scalability and Modularity

The game's design is modular and scalable, making it easy to introduce new features and expand upon existing ones thanks to the use of scriptable objects and a finite state machine model, therefore future work could focus on adding new types of AI agents, resources, or building types to further diversify the implementation.

5 Conclusion

In conclusion, the autonomous civilization simulation game demonstrates the successful integration of advanced algorithms, AI agent behaviour, and data visualization. The modular and scalable design provides a solid foundation for future enhancements, such as refining agent decision-making, introducing new game elements, and improving the graphing system's capabilities. Overall, the project showcases the potential for engaging and immersive simulation experiences driven by innovative game design and technology.

Bibliography

Bridson, Robert (2007). "Fast Poisson disk sampling in arbitrary dimensions". In: *ACM Transactions on Graphics (TOG)* 26.3, pp. 1–9.

Conway, John H. (1970). "The Game of Life". In: *Scientific American* 223.4, pp. 120–123. URL: `https://www.scientificamerican.com/article/the-game-of-life/`.

Fairclough, Chris et al. (2001). "Research Directions for AI in Computer Games". In: *Proceedings of the 2001 Workshop on AI and Interactive Entertainment*. Trinity College Dublin. Dublin. URL: `http://www.tara.tcd.ie/handle/2262/13098`.

Games, Firaxis (2016). *Sid Meier's Civilization® VI*. URL: `https://store.steampowered.com/app/289070/Sid_Meiers_Civilization_VI/`.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Long, Edmund (2007). "Enhanced NPC Behaviour using Goal Oriented Action Planning". MSc thesis. Dundee, UK: University of Abertay Dundee. URL: `https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=472acfbf6b9a1a3faaec7bf61e7f018b471272c1`.

Perlin, Ken (1985). "An Image Synthesizer". In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, pp. 287–296.

Productions, Monolith (2005). *F.E.A.R.* URL: `https://store.steampowered.com/app/21090/FEAR/`.

– (2014). *Middle-earth: Shadow of Mordor*. URL: `http://www.tara.tcd.ie/handle/2262/13098`.

Studios, Ensemble (1997). *Age of Empires*. URL: `https://store.steampowered.com/developer/AgeOfEmpires`.

Unity Technologies (2023). *Unity Game Engine*. URL: `https://unity.com/`.