Final Project

Group 7

Agent-Based Modeling of Small Combat Situations

GitHub Repository

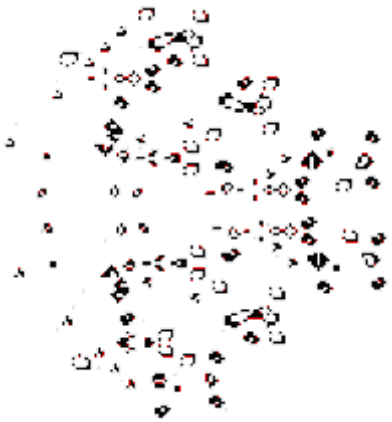
Bhalerao, A.M., Sathe, C.K., Ott, L., Orlandi, M., & Shenoy, K.S. (2023). Artificial Life Simulator. GitHub https://github.gatech.edu/kshenoy8/artificial-life-simulator/

Abstract

In this project we attempt to recreate a simulator for hypothetical small combat situations. Through an Agent-based probabilistic model we depict the development of a potential recognizance military mission. A series of agents, players, are tasked to discover all the enemies in the environment starting from a random position in the field. We implement our simulation through a combination of Python and Julia, to address our mathematical and visualization needs. A nuanced probabilistic approach to discovery, combined with detailed topological features add realism to our simulations. From our experimental results it is possible to observe not only the speed but the efficiency as well of our search models when combined with a complex environment. We believe our results could be of use not only in military applications, but in search-and-rescue missions, where the need to efficiently find unknown targets is paramount.

Project Description & Literature Review

The field of artificial life (ALife) officially began in 1987 when the term "artificial life" was coined by computer scientist Christopher Langton during the Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems at Los Alamos National Laboratory. Artificial life is a multidisciplinary field of study that explores the nature of life and life-like phenomena, with a primary focus on the inherent organizational aspects rather than the physical structures of living systems. Its core objective is to gain a deeper understanding of living systems by designing and simulating simplified versions of them. This provided a unifying label for interdisciplinary studies focused on lifelike behavior. ALife ideas, including the concept of cellular automata, can be traced back to mathematicians John von Neumann and Stanislaw Ulam in 1948. The Game of Life (see animation below), designed by mathematician John Conway in the 1970s, further strengthened the connection between cellular automata and artificial life, captivating players with its exploration of intricate patterns and complex behaviors. ALife has since evolved into three branches: "soft" ALife, simulating life in software; "hard" ALife, focusing on autonomous robots; and "wet" ALife, involving the creation of synthetic organisms through biochemistry. While soft ALife has made progress with software-based organisms like Tierra, hard ALife lags in creating self-assembling robots. Wet ALife, closely mirroring real biology, has garnered scientific respect with experiments involving synthetic genomes in bacteria and the creation of self-replicating xenobots from frog cells (Shi En, 2022).



ALife involves the study of artificial systems that display some specific life-like properties such as compartmentalization, homeostasis, the ability to reproduce, growth and development, adaptation, and evolution, among many others. Compartmentalization is considered of fundamental importance for life. Most of the specific life-like properties can be grouped into two generic processes of self-organization and emergence. Self-organization focuses on interactions on a local scale which follow an order and a set of rules and regulations. Emergence focuses on a global scale collective behavior which the parts of the system would not do alone. Soft ALife is linked to both these categories in many subdomains. Cellular Automata illustrates examples of self-organizing systems where a global controller is not involved. Using PDEs and dynamical systems the emergent and self-organizing properties are expressed by spontaneous pattern formation and spatio-temporal coherence. (Scirè and Annovazzi-Lodi, 2023)

While the field of ALife has emerged into several subcategories, our aim in this project is to mainly focus on 'soft' ALife. We plan to begin with Game of Life and augment it with a multiagent exploration scenario. This simulation can be well reflected in combat situations, where two opposing groups of agents interact with each other. At this checkpoint, we are exploring on ways to model decision making of these agents depending on the environment around them. We are considering options like where the two groups use weapons and attempt to harm each other, and in scenarios where the military is conducting a patrol and encounters individuals who may pose a threat. These scenarios are akin to border control interacts with non-favorable groups of enemies (be it violent or non-violent). This system could also very well represent a search-and-rescue (SaR) operation in disaster-struck lands. Even though this may seem like an abrupt change from the previous literature review, many core elements of agent-based systems and cellular automata remain identical in interactions between the military and unknown individuals. In fact, the first layer of this will be each of the cell in the CA system having its own data flow to achieve predefined goals. We call this the environment rules (ER). A second layer will be used to simulate the environment and how the agents interact with each other. The system will also incorporate line-of-sight visibility rules obscuring the agents' vision when visual barriers, such as trees, hills, or buildings, are present. (Cil & Mala, 2010).

Conceptual Model

For this simulation our conceptual model is abstract, but attempting to recreate human decision making:

Agents:

- Unit size, large count
- Move across the environment to scout the enemy camps (location and size)
- Will move in a stochastic or random fashion when there is no enemy in the view sight
- Will move in a strategic manner and hid in bushes when there are enemies nearby
- Can move a unit step in any direction for every simulation iteration

Enemy Camps:

- Varying sizes (in space), in forms of camps
- Move in a manner to avoid collisions with other camps and seek to achieve higher ground
- Have a restriction on maximum slope they can climb in a step due to a tendency to trip
- Can move a unit step in any direction for every 10 simulation iterations

Cellular Automata (ca-system.jl)

Topography

The CA system spans over multiple items. Firstly, it handles the environment, topography and bushes. The topography is generated by randomly choosing control points in the design space. Let's denote this set by P Then, for each pixel of land in the environment, the altitude of this pixel is calculated like so:

$$h_p = \frac{\sum_{p_i \in P} z[p_i]/d^3}{\sum_{p_i \in P} 1/d^3},$$

Here, $z[p_i]$ is the altitude of the control point p_i and d is the distance of this pixel from the control point. This equation is adapted from Shenoy's work on 2D color gradients [1]. The output of this system gives us a topography (for random seed 584) as shown in Fig 2.

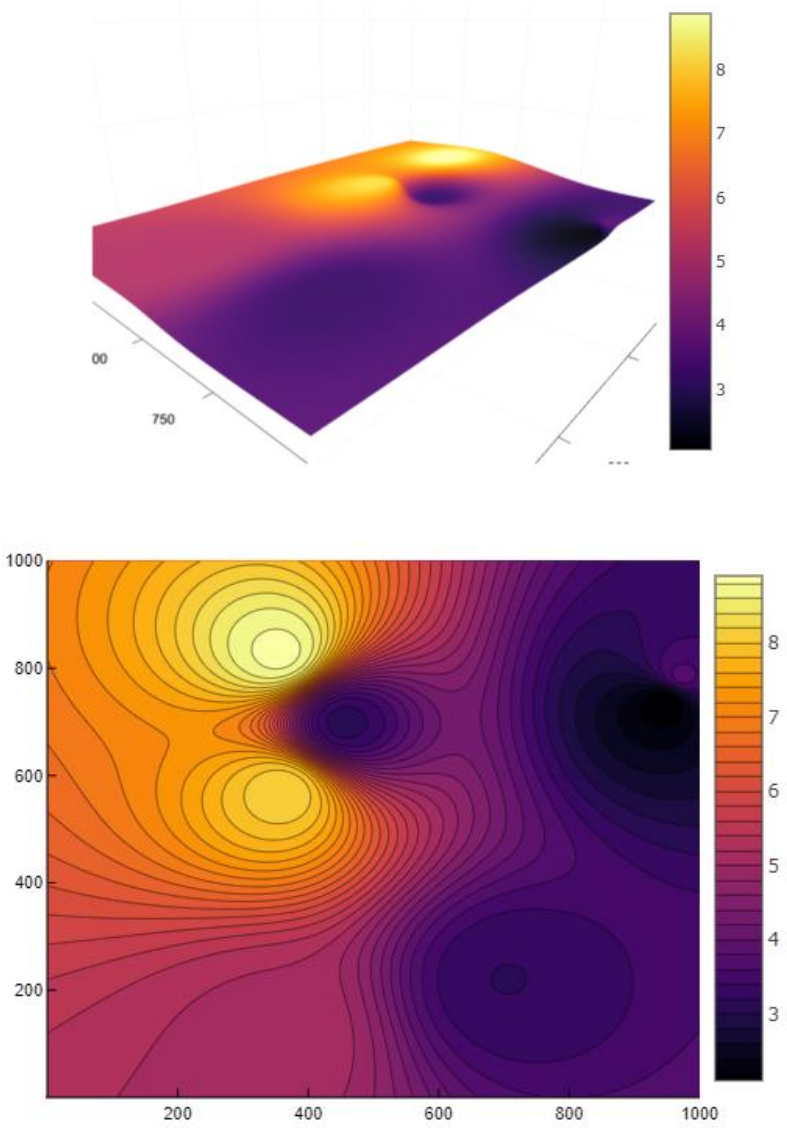


Fig 1: The topography of the system generated through random seed and non-interpolation based methods.

Vegetation

Now let's move on to the vegetation. You might wonder why there is a need for vegetation modeling in a combat modeling system. The idea is that the agents can hide inside the bushes of vegetation and gather information on the enemy camps. This helps us model a conflict-less system of agent-enemy combat. This conflict-less system can easily be transformed into a model for a Search and Rescue operation (SaR). The bushes are generated through an array of size $n \times n$ filled with random float values in the range $[0, \frac{L}{10}]$ where L is the size of the design space.

The topography with vegetation is shown in Fig 3. We might use Conway's life rules for this vegetation. This idea is debated because it inherently introduces "randomness" into the system which might perturb the dynamics of the agent-enemy interaction leading us to misrepresented conclusions.

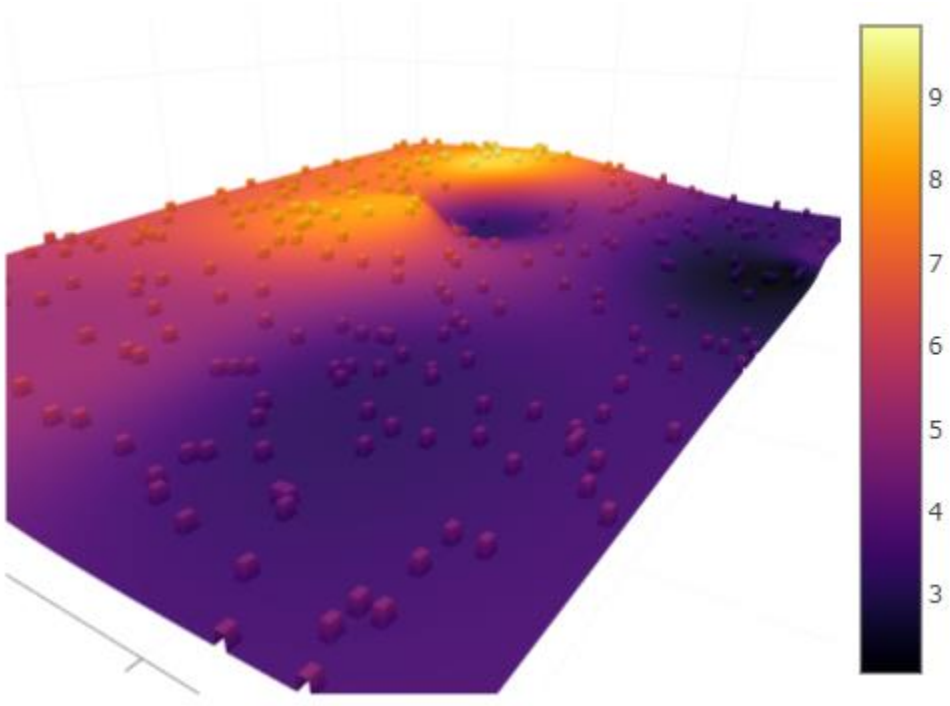


Fig 2: The topography of the system along with the generated vegetation.

Design Space

The design space in consideration is of the size $L \times L \times \frac{L}{10}$, with the x and y directions being subdivided into n parts. This subdivision makes the step value, $h_x = h_y = h$ in both these directions equal to,

$$h = \frac{L}{n}.$$

With this subdivision setup, we can move to the enemy cells of the system.

Enemies

All groups of enemy cells are circular in shape, and they occupy a von Neumann neighborhood around its center. The size of this neighborhood is variable, and this size also determines the

level of accuracy with which these cells calculate environment parameters to make decisions for movement.

Slopes

Given that we know the function that produces the topography, it might seem trivial to find the topographical slopes at each grid point. However, in order to introduce limitations in how the enemy cells gain information of the environment, we introduce a finite difference approach to numerically calculate the slopes at each point. At this moment, all enemy cells gain this information with the same level of accuracy regardless of their size.

We know that a function $f(x, y)$ can be expanded using Taylor series in the following manner,

$$f(x+h, y) = f(x+y) + h\,\frac{\partial f}{\partial x} + \frac{h^2}{2!}\,\frac{\partial^2 f}{\partial x^2} + \frac{h^3}{3!}\,\frac{\partial^3 f}{\partial x^3} + \mathcal{O}(h^4).$$

A central difference method gives us the slope with an error term $\propto h^2$ like so,

$$\Delta(h) = \frac{f(x+h, y) - f(x-h, y)}{2h} = \frac{\partial f}{\partial x} + \frac{h^2}{3!}\,\frac{\partial^3 f}{\partial x^3} + \frac{h^4}{5!}\,\frac{\partial^5 f}{\partial x^5} + \mathcal{O}(h^6).$$

If we look at the central difference with double this step,

$$\Delta(2h) = \frac{f(x+2h, y) - f(x-2h, y)}{4h} = \frac{\partial f}{\partial x} + \frac{2^2 h^2}{3!}\,\frac{\partial^3 f}{\partial x^3} + \frac{2^4 h^4}{5!}\,\frac{\partial^5 f}{\partial x^5} + \mathcal{O}(h^6),$$

we see that the h^2 term can be eliminated giving us slope with accuracy up to the fourth order,

$$\frac{4\Delta(h) - \Delta(2h)}{3} = \frac{\partial f}{\partial x} - \frac{h^4}{30}\,\frac{\partial^5 f}{\partial x^5} + \mathcal{O}(h^6).$$

We employ this equation in the code to get the slopes at each point.

```
dfbydx = 1/3*(4*(xph-xmh)/2 - (xp2h-xm2h)/4)
dfbydy = 1/3*(4*(yph-ymh)/2 - (yp2h-ym2h)/4)

dfbydx = (xph-xmh)/2
dfbydy = (yph-ymh)/2
```

Simulated Annealing

To gain strategic advantage, the enemies attempt to climb up and gain altitude. But, how would they do it? A simple Newton Raphson method would make them "stuck" in local maximas. This means that the enemies cannot directly use the slopes.

We introduce randomness to implement a simulated annealing process. At each timestep, the enemies decide to climb or climb down with a certain probability. They climb down if a "metropolis trip" occurs when,

$$\text{rand}() < e^{-\frac{2}{T}}$$

Now, initially they all have the same constant temperature, T, and after each movement, T is changed. If a metropolis trip occurs, the temperature increases. This is done to imitate a "velocity" to the cluster. With this velocity, an oscillation of cluster is prevented. We also have similar random movements when a cluster is stuck and boundaries and the slope tells it to go outside the environment.

Agent Based Models (search.py/agent.py)

We have demonstrated a simulation of the agents' interaction with their surroundings in a python code. We have used an area sweep search algorithm to move the agents around a 2D grid to detect enemy camps. The agents can sweep through the environment and estimate the number of enemies by corelating them with the area of enemy camps. For the base version of the search algorithm, the enemy is assumed to be fixed at their locations, so the agents can just move around and detect all the coordinates that are occupied by the enemy camp. Thus, the location and size of the camps can be determined. A more coordinated search has to be conducted for moving targets.

There are two types of searches an agent can perform, *Random or Strategic*.

- **Random Search:** the simplest out of the two searches happens when there are no visible enemies in the agent's field of view. In this scenario, the agent can move across each of the cardinal directions, and diagonally, in a random fashion. Given the starting position and the two Von Neumann neighborhoods centered around it, we simply randomly select one of the possible landing spots, given the agent's step size.

- **Strategic Search:** this is our more nuanced implementation. When an enemy appears in the agent field of view, the player is tasked with finding the size of the enemy camp. This can happen one of two ways:
    - The agent is able to detect corners of the Von Neumann neighborhood representing the enemy camp, and thus can store the enemy's UID as already detected and start moving away from it, with defined probabilities
    - In case the corners of the enemy camp do not mathematically represent an edge of a Von Neumann neighborhood, it means the enemy has moved during our scan. The agent will then enforce some directionality to its movement. With fine-tuned movement probabilities, aimed at reducing the risk of the agent being spotted by the enemy camp, it will try to approach the furthest of the detected points, in order to find the camp's new corner position and determine its shape.

Simulation

Our simulation approach is fairly straightforward. We first deployed a 2D testing environment in Python to verify our implementation relative to our conceptual model. We defined search rules for each Agent, that combine a mix of a random walk and a more strategic approach. When looking for an enemy, the agents have access to only a limited visibility field from which to gather information, corresponding to a double Von Neumann neighborhood. If their field of vision is free, then they would move randomly according to their step size, otherwise their movement is still probabilistic, but aimed at either discovering the size of the enemy camp, or hide from enemy sight, depending on whether the enemy has already been discovered.

In our small-scale simulations we tested various scenarios to ensure the completion of the search strategy (i.e. discovery of all enemy camps), we can summarize them into three categories of increasing complexity:

1. One agent, and one stationary enemy
2. Multiple agents, multiple stationary enemies
3. Multiple agents, multiple moving enemies

The usefulness of these tests was especially relevant in confirming the different movement strategy depending on the proximity of the agent to the enemy.

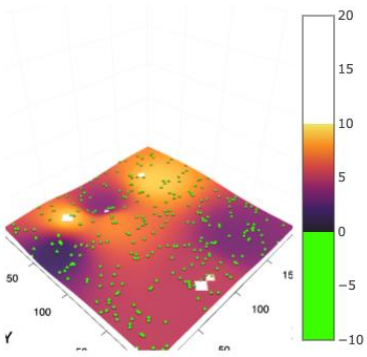
Experimental Results and Validation

We implemented our code with a mixture of Python and Julia, relying on an Intel GPU for a lot of the matrix algebra and storage needs of this project.

Here are some results of our simulation. Given that our simulator has more of a visual application, better described through our video, we will limit ourselves here in showing snapshots of particular frames of our simulator.

Our experimental procedure, just like in our conceptual validation followed a series of steps. We have two types of inputs: agents and enemies. Both are defined by their Von Neumann neighborhoods and their initial state corresponds to a randomly sampled position *(x,y,z)* in the environment. Additionally, both types of agents are driven by similar objectives, discovering all the agents of different kind. Nonetheless, we make some assumptions regarding the intrinsic capabilities of each agent in order to both simplify our constraints for ease of implementation, but also to set a more realistic stage. Agents are intrinsically faster than enemies, i.e. at each time step they can move further, however they have a smaller range of sight than their counterparties.

In order to find the active agents, the enemies will try to climb the slopes in the topography according to a *gradient ascend function*. Since it's unlikely that a troop can climb any slope, we will try to make them move in the direction with the max feasible slope. Let us observe the starting and final positions of a simulation over 800 time-steps *t*. The white elements in the field represent enemy camps, while green cubes represent bushes.



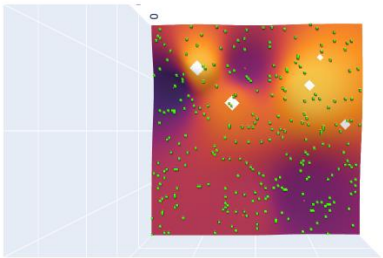
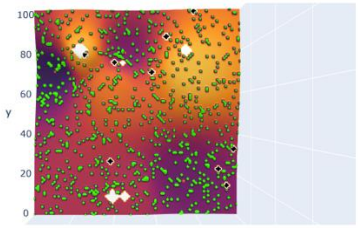


Fig. 3 Starting Position of Enemies

Fig 4. Ending Position of Enemies

We can clearly see the movement of the enemies across the environment. The intensity of the color red in the map indicates the steepness of the slope, so it isn't surprising to see that a lot of the clustered boxes haven't been able to make much progress compared to the camp that started in at (100, 50, 0). Having demonstrated the movement of the enemies across the simulated battlefield, it is time to check out what the interaction between agents and enemies will produce.



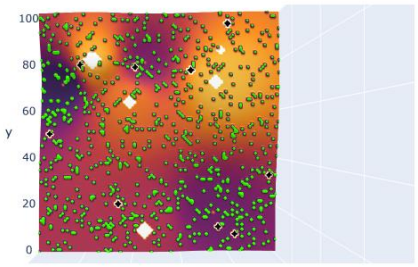


Fig 5. Initial configuration with Agents

Fig 6. Final Configuration with Agents

We added the density of bushes to increase complexity, and added 5 agents initiated at random locations around the environment. As we can see from our ending iteration (t=900), the results are not as promising as imagined. We were initially worried about the possibility of the agent getting stuck, or too close to the enemy for longer time steps than optimal, thanks to our testing, and to changing the rules in our *strategic search* we were able to solve this problem at a local level, nonetheless the agents often took a long time to reach their targets, or in some cases never reach them at all. There is a variety of hypothesized reasons for this, starting with a poor definition of movement rules, as a matter of fact we noticed that our players would jump further than determined, or get stuck in a loop. These findings are not necessarily surprising, as in many CA models there is a possibility for infinite loop.

Tuning our initial parameters: like agent's view sight, gather radius and step size helped in achieving more concrete results, however agents were often not able to discover all of the enemies within the simulation time.

Discussion and Conclusion

This project has given us the chance of working with simulated environments and decision making. It was interesting to try to recreate a real-life scenario through cellular automata and agent-based modeling, particularly because the nature of CA is much more deterministic than real life. We learned about the difficulty of deploying a nuanced movement of the agents through the grid when subject to topological and survival constraints, as well as how to combine Python and Julia for visualization.

Our implementation certainly didn't come without issues. Our agents encountered several roadblocks in their movement and often got stuck in areas of restricted movement (i.e. high density of bushes, or many enemies clustered together). Possible corrections to such issues may lie in the probabilistic approach to movement we have implemented. A more comprehensive comparison between movement approaches (deterministic vs. random search, parallel sweep, local search) might yield more accurate results. Additionally, the use of graph algorithms for MSTs or shortest paths have been discussed among the group. We believe this integration would also greatly benefit the results.

Overall, it has been a challenging but rewarding experience. The Julia implementation created more than a few implementation issues, however it was a wonderful opportunity to learn a useful language while gaining valuable practical experience.

In conclusion, through this project we wanted to tackle an interesting problem, not only from a simulation point of view but also from an efficiency perspective. Despite our struggles in achieving a satisfying solution (albeit with high quality standards set by group members), we are pleased with the efficiency of our code that was able to propose.

References

1. Kim, Shi En. "Life Evolves. Can Attempts to Create 'Artificial Life' Evolve, Too?" Scientific American, SCIENTIFIC AMERICAN, a Division of Springer Nature America, Inc., April 6, 2023, https://www.scientificamerican.com/article/life-evolves-can-attempts-to-create-artificial-life-evolve-too/.

2. Courgeau, D., Franck, R., Bijak, J., Hilton, J., Noble, J., Bryden, J. (2018). Methodological Investigations in Agent-Based Modelling With Applications for the Social Sciences. Open Access.
3. Walker, R. (1999). "Niche Selection" and the Evolution of Complex Behavior in a Changing Environment—A Simulation. Artificial Life, 5(3), 271–289. Massachusetts Institute of Technology.
4. Ling, M. H. T. (2012). An Artificial Life Simulation Library Based on Genetic Algorithm, 3-Character Genetic Code and Biological Hierarchy.
5. Scirè, A., Annovazzi-Lodi, V. The emergence of dynamic networks from many coupled polar oscillators: a paradigm for artificial life. Theory Biosci. 142, 291–299 (2023). https://doi.org/10.1007/s12064-023-00401-4
6. Curran D., O'Riordan C. On The Design of An Artificial Life Simulator. National Univeristy of Ireland, Galway

Appendix: Division of Labor

	Task	Member
1	Cellular Automata, Validation	Kishore Shenoy
2	Agent Based Models	Atharva Bhalerao, Michele Orlandi
3	Search Models	Chaitra Sathe
4	Stochastic Analysis, Validation	Lukas Ott
		Michael Orlandi

In our collaborative efforts, we systematically distribute tasks to optimize the utilization of each team member's skills. More specifically, even though we separated the tasks as specified in the table before, while working on the project, the team members started to work on the task where the most effort was needed. Additionally, on the last day of the project, we collaboratively met at the Coda Building and brought the pieces together and debugged the code where there were some issues. Furthermore, we finalized the report and recorded the video. Overall, we think as a

team, even though we are at this point not fully satisfied with the results, the final outcome shows clearly what we have learned during the semester in the simulation and modeling course.